# DESIGN AND DEVELOPMENT OF MODELS FOR ANALYZING SOFTWARE EVOLUTION

By

**MEGHA UMMAT**

Roll No.: 2k13/Ph.D./COE/05

Under the guidance of

**Dr. Ruchika Malhotra**

Associate Professor, Discipline of Software Engineering,
Department of Computer Science & Engineering

Submitted in fulfillment of the requirements of the degree of
Doctor of Philosophy to the


DELHI TECHNOLOGICAL UNIVERSITY
(FORMERLY DELHI COLLEGE OF ENGINEERING)
SHAHBAD DAULATPUR, MAIN BAWANA ROAD, DELHI 110042

2019

# Declaration

I, **Megha Ummat**, Ph.D. student Roll No.: 2k13/Ph.D./COE/05, hereby declare that the thesis entitled **"Design and Development of Models for Analyzing Software Evolution"** which is being submitted for the award of the degree of Doctor of Philosophy in Computer Science & Engineering, is a record of bonafide research work carried out by me in the Department of Computer Science & Engineering, Delhi Technological University. I further declare that the work presented in the thesis has not been submitted to any University or Institution for any degree or diploma.

Date :

Place : New Delhi

Megha Ummat

meghakhanna86@gmail.com

Roll No.: 2k13/Ph.D./COE/05

Discipline of Software Engineering,

Department Of Computer Science & Engineering,

Delhi Technological University (DTU),

New Delhi -110042

# CERTIFICATE


**DELHI TECHNOLOGICAL UNIVERSITY**

(Govt. of National Capital Territory of Delhi)

BAWANA ROAD, DELHI - 110042

Date: _____

This is to certify that the work embodied in the thesis titled **"Design and Development of Models for Analyzing Software Evolution "** has been completed by **Ms. Megha Ummat** Roll No.: 2k13/Ph.D./COE/05 under the guidance of **Dr. Ruchika Malhotra** towards fulfillment of the requirements for the degree of Doctor of Philosophy of Delhi Technological University, Delhi. This work is based on original research and has not been submitted in full or in part for any other diploma or degree of any university.

Supervisor

**Dr. RUCHIKA MALHOTRA**

Associate Professor

Discipline of Software Engineering

Department of Computer Science & Engineering

Delhi Technological University, Delhi 110042

*"This thesis is dedicated to my dear grandmother Late Smt. Usha Khanna for being my eternal light. I will always miss you Dadi."*

# Acknowledgment

*"Guru Govind dono khade kake lagu paay, Balihari Guru aapki Govind diyo batay"* This saying by Sant Kabir enlightens one about the true meaning of a guide, someone without whose help, one is unable to cross the hurdles of life. I thank God for blessing me with a true guide, **Dr. Ruchika Malhotra**. Mam has paved the way for me to achieve my goals, has motivated me and has always pushed me to improve and hone my skills in the right manner for the ultimate aim. My association with Mam has truly been the sole reason for my existence in my professional domain, and it has not stopped at that. Mam has been a source of my motivation, a light in the darkness for me, both professionally and personally. I am deeply indebted to mam for choosing me as one of her students, for giving me the opportunity to learn under her kind guidance and for lending me support. This work would have never been possible without mam's trust in me and her guidance at each step. I thank mam from the core of my heart for being what she is and for making me what I am.

I would like to convey my sincere thanks to **Dr. Rajni Jindal**, HoD, Department of Computer Science & Engineering, Delhi Technological University for her cooperation while carrying out this research work.

At this moment of accomplishment, I am greatly indebted to my husband **Mr. Nikhil Ummat**, who has always been by my side and has unfailing supported me during my pursuit of a Ph.D degree. I would especially like to thank my father **Mr. Rakesh Khanna** who instilled in me that nothing is impossible if I set my mind for it. He has been a true motivator throughout. I would also like to thank my **mother, Mrs. Smriti Khanna** and **brother, Mr. Atul Khanna** for their unwavering support and confidence in all what I do, which is the reason for all my achievements. Lastly, my deep regards to my **parents-in-law** and **extended family** for their unfailing love, blessings and moral support.

**Megha Ummat**

# Abstract

Software systems are important business assets of any organization. However, in order to maintain the value of these assets, software evolution i.e. the process of planning and implementing change to the existing software systems is a crucial activity. One of the prime concerns while implementing changes is to maintain the quality of the software product as there are fewer resources and rigid deadlines, which may result in poor processes and software quality degradation. In such a scenario, the responsibility of a software practitioner is to envisage methods which provide good quality software with ideal resource usage at optimum costs. One such cost effective approach is to develop models for predicting change-prone parts of a software as these parts are considered as sources of changes and defects in a software. Detection of change-prone parts in the initial stages of software development lifecycle will help software developers in outlining competent resource usage during maintenance activities, planning remedial actions for software restructuring and implementing corrective actions for early removal of software defects.

Prediction of change-prone parts in an object-oriented software involves the use of various object-oriented metrics as predictor variables, which are representative of software characteristics such as size, coupling, cohesion and inheritance. Furthermore, we need a classification technique for developing an efficient prediction model which is able to distinguish between change-prone and not change-prone parts of a software. The various elements involved in the creation of software change prediction models need to be assessed and improved to yield efficient change-prediction

models.

This thesis verifies and validates the relationship between several object-oriented metrics and change-proneness attribute of an object-oriented class to develop effective prediction models. We also analyze the trends of object-oriented metrics in an evolving software in order to ascertain how the structural characteristics of a software change with its evolution. The thesis also evaluates the use of a specific set of process metrics, which are named as evolution-based metrics. These metrics encapsulate the evolution history of a class in an object-oriented software. Furthermore, the effectiveness of a combined set of object-oriented metrics and evolution-based metrics have also been investigated for determining the change-prone nature of a class in an object-oriented software.

Apart from predictor variables, the thesis also evaluates several categories of data analysis techniques, which can be used for developing software change prediction models. The investigated categories include statistical techniques and machine learning techniques, which have been used by several researchers in this domain. However, a new class of techniques i.e. search-based algorithms and their hybridized versions have recently gained popularity. We first review the capabilities, advantages and the experimental set-ups required to use this set of algorithms. Furthermore, we explore their capability for developing models which determine the change-prone nature of a class. The thesis also proposes a new set of classification algorithms based on ensemble methodology, using a search-based algorithm as a base-classifier. The proposed algorithms produce outputs by aggregating a number of constituent classifiers, which are fitness variants of the same base-classifier namely Constricted Particle Swarm Optimization. We also propose a unique classifier, which outputs the best classifier amongst an ensemble of classifiers for each data point (object-oriented class).

The thesis also evaluates the scenario when the historical data used for developing a change prediction model is imbalanced in nature. A dataset is said to be

of imbalanced nature, when the ratio of category of classes (change-prone and not change-prone) is disproportionate. In general, as the number of change-prone classes is few as compared to the number of not change-prone classes, effective learning is problematic. This is because the learning algorithm is provided with very few instances of change-prone classes, therefore, it is unable to learn their characteristics properly resulting in lower accuracy while determining change-prone classes. The thesis investigates the use of sampling methods and MetaCost learners for developing efficient change prediction models from imbalanced training data.

Apart from determining the change-prone nature of classes, it is also important to determine the impact of change in a software. We determine the change-impact of bug correction in a software i.e. the number of classes that would be affected when a specific software bug is corrected. Additionally, the thesis also proposes a categorization of software bugs into different levels on the basis of maintenance effort and change impact values in order to optimize maintenance resources.

# Contents

# List of Tables

# List of Figures

# List of Publications

**Papers Accepted/Published in International Journals**

1. Ruchika Malhotra and Megha Khanna, "Dynamic Selection of Fitness Function for Software Change Prediction using Particle Swarm Optimization", *Information and Software Technology*. Accepted and published online (Impact Factor: 2.627).

2. Ruchika Malhotra and Megha Khanna, "Particle Swarm Optimization-Based Ensemble Learning for Software Change Prediction", *Information and Software Technology*, vol. 102, pp. 65-84, October 2018 (Impact Factor: 2.627).

3. Ruchika Malhotra and Megha Khanna, "Threats to Validity in Search-Based Prediction Modeling for Software Engineering", *IET Software*, vol. 12, no. 4, pp. 293-305, August 2018 (Impact Factor: 0.733).

4. Ruchika Malhotra and Megha Khanna, "Prediction of Change Prone Classes using Evolution- based and Object-Oriented Metrics", *Journal of Intelligent and Fuzzy Systems*, vol. 34, pp. 1755-1766, March 2018 (Impact Factor: 1.261).

5. Ruchika Malhotra and Megha Khanna, "An Empirical Study to Evaluate the Relationship of Object-Oriented Metrics and Change Proneness", *International*

*Arab Journal of Information Technology*, vol. 15, no. 6, pp. 1016-1023, November 2018 (Impact Factor: 0.724).

6. Ruchika Malhotra and Megha Khanna, "An Empirical Study for Software Change Prediction using Imbalanced Data", *Empirical Software Engineering*, vol. 22, no. 6, pp. 2806-2851, December 2017 (Impact Factor: 3.275).

7. Ruchika Malhotra and Megha Khanna, "An Exploratory Study for Software Change Prediction in Object-Oriented Systems using Hybridized Techniques", *Automated Software Engineering Journal*, vol. 24, no. 3, pp. 673-717, September 2017 (Impact Factor: 2.625).

8. Ruchika Malhotra, Megha Khanna and Rajeev R. Raje "On the Application of Search-based Techniques for Software Engineering Predictive Modeling: A systematic Review and Future Directions", *Swarm and Evolutionary Computation*, vol. 32, pp. 85-109, February 2017 (Impact factor: 3.893).

9. Ruchika Malhotra and Megha Khanna, "The Ability of Search-Based Algorithms to Predict Change-Prone Classes", *Software Quality Professional Journal, ASQ*, vol. 17, no. 1, pp. 17-31, December 2014.

**Papers Accepted/Published in International Conferences**

10. Ruchika Malhotra and Megha Khanna, "Software Change Prediction using Voting Particle Swarm Optimization based Ensemble Classifier", *Genetic and Evolutionary Computation (GECCO 2017)*, pp. 311-312, Berlin, Germany, July 2017.

11. Ruchika Malhotra and Megha Khanna, "Common Threats to Software Quality Predictive Modeling Studies using Search-based Techniques", *International*

*Conference on Advances in Computing, Communications and Informatics (ICACCI 2016)*, pp. 568-574, Jaipur, Rajasthan, Septemebr 2016.

12. Ruchika Malhotra and Megha Khanna, "An Empirical Evaluation of Performance of Machine Learning Techniques on Imbalanced Software Quality Data", *Proceedings of 18th International Conference on Information Technology and Computer Science*, International Journal of Computer, Electrical, Automation, Control and Information Engineering, vol. 10, no. 4, pp. 562-570, Venice, Italy, April 2016.

13. Ruchika Malhotra and Megha Khanna, "Software Engineering Predictive Modeling using Search-based Techniques: Systematic Review and Future Directions", *North American Search Based Software Engineering Symposium (NASBASE 2015)*, pp. 1-16, Michigan, USA, February 2015.

14. Ruchika Malhotra and Megha Khanna, "Examining the Effectiveness of Machine Learning Algorithms for Prediction of Change Prone Classes", *International Conference on High Performance Computing and Simulation (HPCS 2014)*, pp. 635-642, Bologna, Italy, July 2014.

15. Ruchika Malhotra and Megha Khanna, "A New Metric for Predicting Software Change using Gene Expression Programming", *International Workshop on Emerging Trends in Software Metrics (WETSoM 2014)*, pp. 8-14, Hyderabad, India, June 2014.

**Papers Communicated in International Journals**

16. Ruchika Malhotra and Megha Khanna, "Software Change Prediction: A Systematic Review and Future Research Directions", *Knowledge-Based Systems*.

17. Ruchika Malhotra and Megha Khanna, "Analyzing Evolution Patterns of Object-Oriented Metrics: A Case Study on Android Software", *International Journal of Rough Sets and Data Analysis*.

18. Ruchika Malhotra and Megha Khanna, "On the Applicability of Search Based Algorithms for Software Change Prediction",*International Journal of System Assurance Engineering and Management*.

19. Ruchika Malhotra and Megha Khanna, "A Novel Framework for Software Bug Categorization using Change Impact Levels and Maintenance Effort", *Journal of Systems and Software*.

# Abbreviations

| | |
|---|---|
| **AB** | **Adaptive Boosting** |
| **ACDF** | **Aggregated Change Density normalized by Frequency of Changes** |
| **ACO** | **Ant Colony Optimization** |
| **AHF** | **Attribute Hiding Factor** |
| **AIF** | **Attribute Inheritance Factor** |
| **AIRS** | **Artificial Immune Recognition System** |
| **AMC** | **Average Method Complexity** |
| **AMSE** | **Adjusted Mean Square Error** |
| **ANA** | **Average Number of Ancestors** |
| **ANN** | **Artificial Neural Network** |
| **ASOF** | **Adaptive Selection of Optimum Fitness** |
| **ATAF** | **Aggregated Change Size normalized by Frequency of Changes** |
| **AUC** | **Area Under Receiver Operating Characteristic Curve** |
| **BG** | **Bagging** |
| **BMMRE** | **Balanced Mean Magnitude of Relative Error** |
| **BN** | **Bayesian Networks** |
| **BOC** | **Birth of a Class** |
| **Ca** | **Afferent Coupling** |
| **CAM** | **Cohesion Among Methods of a Class** |
| **CART** | **Classification and Regression Tree** |
| **CBM** | **Coupling Between Methods of a Class** |

| | |
|---|---|
| **CBO** | **Coupling Between Objects** |
| **CBR** | **Case Based Reasoning** |
| **CC** | **Changed Classes** |
| **Ce** | **Efferent Coupling** |
| **CCB** | **Change Control Board** |
| **CFS** | **Correlation-based Feature Selection** |
| **CHD** | **Change Density** |
| **CHO** | **Change Occurred** |
| **CIS** | **Class Interface Size** |
| **CK** | **Chidamber and Kemerer** |
| **CKJM** | **Chidamber and Kemerer Java Metrics** |
| **CLAMI** | **Clustering, Labeling, Metric selection and Instance Selection** |
| **CLG** | **Clonal Selection** |
| **CMS** | **Configuration Management System** |
| **COF** | **Coupling Factor** |
| **CPSO** | **Constricted Particle Swarm Optimization** |
| **CR** | **Cost Ratio** |
| **CS** | **Class Size** |
| **CSB** | **Changes Since Birth** |
| **CSBS** | **Changes Since Birth normalized by Size** |
| **CVS** | **Concurrent Versions System** |
| **DAC** | **Data Abstraction Coupling** |
| **DAM** | **Data Access Metric** |
| **DCC** | **Direct Class Coupling** |
| **DCRS** | **Defect Collection and Reporting System** |
| **DIT** | **Depth of Inheritance Tree** |
| **DSC** | **Design Size in Classes** |

| | |
|---|---|
| **DT** | **Decision Trees** |
| **DT-GA** | **Decision Trees with Genetic Algorithms** |
| **ELM-PLY** | **Extreme Machine Learning with Polynomial Kernel** |
| **EMRE** | **Magnitude of Relative Error Relative to the Estimate** |
| **FCH** | **First time Changes in a Class** |
| **FN** | **False Negative** |
| **FP** | **False Positive** |
| **FRCH** | **Frequency of Changes** |
| **GA** | **Genetic Algorithm** |
| **GA-ANN** | **Genetic Algorithm- Artificial Neural Network** |
| **GA-Int** | **Genetic Algorithm based classifier with Intervalar rules** |
| **GA-ADI** | **Genetic Algorithm based classifier with Adaptive Discretization Intervals** |
| **GA-SVM** | **Genetic Algorithm- Support Vector Machine** |
| **GEP** | **Gene Expression Programming** |
| **GFS-AB** | **Genetic Fuzzy System AdaBoost** |
| **GFS-LB** | **Genetic Fuzzy System LogitBoost** |
| **GFS-MLB** | **Genetic Fuzzy System Maxlogitboost** |
| **GFS-GP** | **Fuzzy Learning based on Genetic Programming** |
| **GFS-SP** | **GFS-GP with Grammar Operators and Simulated Annealing** |
| **GMDH** | **Group Method of Data Handling** |
| **GUI** | **Graphical User Interface** |
| **GP** | **Genetic Programming** |
| **HIEC** | **Hard Instance Ensemble Classifier** |
| **HBT** | **Hybridized Techniques** |
| **HIDER** | **Hierarchical Decision Rules** |
| **IC** | **Inheritance Coupling** |
| **IH-ICP** | **Information flow-based Inheritance Coupling** |

| | |
|---|---|
| **ICH** | **Information flow-based Cohesion** |
| **ICP** | **Internal Class Probability** |
| **ILP** | **Imbalanced Learning Problem** |
| **IM** | **Immunos** |
| **IQR** | **Inter Quartile Range** |
| **IR** | **Information Retrieval** |
| **KEEL** | **Knowledge Extraction based on Evolutionary Learning** |
| **K-NN** | **K-Nearest Neighbor** |
| **LAD** | **Least Absolute Deviation** |
| **LB** | **LogitBoost** |
| **LCA** | **Last Change Amount** |
| **LCD** | **Last Change Density** |
| **LCH** | **Last Time Changes in a Class** |
| **LCOM** | **Lack of Cohesion in Methods** |
| **LCC** | **Loose Class Cohesion** |
| **LCS** | **Learning Classifier System** |
| **LDA** | **Linear Discriminant Analysis** |
| **LOOCV** | **Leave-one-out Cross Validation** |
| **LR** | **Logistic Regression** |
| **LSD** | **Logarithmic Standard Deviation** |
| **MARE** | **Mean Absolute Relative Error** |
| **MAE** | **Mean Absolute Error** |
| **MFA** | **Method of Functional Abstraction** |
| **MHF** | **Method Hiding Factor** |
| **MIF** | **Method Inheritance Factor** |
| **ML** | **Machine Learning** |
| **MLP-BP** | **MultiLayer Perceptron with Backpropagation** |

| | |
|---|---|
| **MLP-CG** | **MultiLayer Perceptron with Conjugate Learning** |
| **MMRE** | **Mean Magnitude of Relative Error** |
| **MdMRE** | **Median Magnitude of Relative Error** |
| **MOA** | **Measure of Aggression** |
| **MOOD** | **Metrics for Object-Oriented Design** |
| **MOPSO** | **Multi-Objective Particle Swarm Optimization** |
| **MPC** | **Message Pass Coupling** |
| **MPLCS** | **Memetic Pittsburgh Learning Classifier System** |
| **MSE** | **Mean Square Error** |
| **MVEC** | **Majority Voting Ensemble Classifier** |
| **NB** | **Naive Bayes** |
| **NECM** | **Normalized Expected Cost of Misclassification** |
| **NIH-ICP** | **Non-Inheritance Information flow-based Coupling** |
| **NNEP** | **Neural Net Evolutionary Programming** |
| **NNGE** | **Non-nested Generalized Exemplars** |
| **NOA** | **Number of Operations Added by a subclass** |
| **NOC** | **Number of Children** |
| **NOH** | **Number of Hierarchies** |
| **NOM** | **Number of Methods** |
| **NOMA** | **Number of Object/Memory Allocation** |
| **NOO** | **Number of Operations Overridden** |
| **NOP** | **Number of Polymorphic Methods** |
| **NPM** | **Number of Public Methods** |
| **OO** | **Object-Oriented** |
| **PD** | **Probability of Detection** |
| **PF** | **Probability of False Alarm** |
| **PSO** | **Particle Swarm Optimization** |

| | |
|---|---|
| **PSO-LDA** | **Particle Swarm Optimization with Linear Discriminant Analysis** |
| **POF** | **Polymorphism Factor** |
| **PUNN** | **Product Unit Neural Network** |
| **QMOOD** | **Quality Model for Object-Oriented Design** |
| **QS** | **Quality Score** |
| **RBF** | **Radial Basis Function** |
| **REP** | **Reduces Error Pruning** |
| **RF** | **Random Forests** |
| **RFC** | **Response For a Class** |
| **RIPPER** | **Repeated Incremental Pruning to Produce Error Reduction** |
| **RMSE** | **Root Mean Square Error** |
| **ROC** | **Receiver Operating Characteristic Curve** |
| **RQ** | **Research Question** |
| **RRSE** | **Root Relative Square Error** |
| **SBA** | **Search-based Algorithms** |
| **SBC** | **Software Bug Categorization** |
| **SE** | **Standard Error** |
| **SEM** | **Standard Error of the Mean** |
| **SEPM** | **Software Engineering Predictive Modeling** |
| **SLOC** | **Source Lines Of Code** |
| **SLAVE** | **Structural Learning Algorithm in a Vague Environment with Feature Selection** |
| **SMOTE** | **Synthetic Minority Oversampling TEchnique** |
| **SIX** | **Specialization Index** |
| **SPM** | **Software Prediction Models** |
| **SUCS** | **SUpervised Classifier System** |
| **SVM** | **Support Vector Machine** |
| **TACH** | **Total Amount of Changes** |

| | |
|---|---|
| **TCC** | **Tight Class Cohesion** |
| **TFIDF** | **Term Frequency Inverse Document Frequency** |
| **TN** | **True Negative** |
| **TP** | **True Positive** |
| **TS** | **Tabu Search** |
| **UCC** | **Unchanged Classes** |
| **VAF** | **Variance Accounted For** |
| **WCD** | **Weighted Change Density** |
| **WFR** | **Weighted Frequency of Changes** |
| **WCH** | **Weighted Changes** |
| **WEKA** | **Waikato Environment for Knowledge Analysis** |
| **WMC** | **Weighted Methods of a Class** |
| **WVEC** | **Weighted Voting Ensemble Classifier** |
| **WVHIEC** | **Weighted Voting Hard Instance Ensemble Classifier** |
| **XCS** | **X-Classifier System** |

# Chapter 1

# Introduction

## 1.1 Introduction

Software is the heartbeat of modern day technology. The software industry directly or indirectly affects developments in all other fields, be it medical, engineering or any other. Thus, it is important to assess and strengthen the need for a good quality software product. This thesis is focuses on to the improvement of software quality by developing models for analyzing software evolution. Planning of resources and change management is crucial during software evolution to assure good quality software products and satisfied customers. Software change prediction models, which predict the change-prone classes in an Object-Oriented (OO) software are effective methods for managing critical software resources during its evolution.

Prediction models for determining change-prone classes are developed with the aid of classification techniques. Researchers have investigated the use of a number of statistical and Machine Learning (ML) techniques for developing software change prediction models [2–5]. But, more empirical studies are required to ascertain which techniques give the best results in a specific scenario and to ascertain if there ex-

ists a relationship between OO metrics and change-proneness attribute of a class. Moreover, the use of Search-based Algorithms (SBA), a sub-class of ML techniques is limited in the domain of change prediction. SBA are meta-heuristic techniques which search for an optimal or near-optimal solution amongst a large population of candidate solutions [6]. They are suitable for developing predictive models as they are robust and can easily handle noisy data. Also, they can search for effective solutions by modeling performance measures as "fitness functions" [7]. We first analyze the current state of literature, where SBA are used for ascertaining four specific software attributes namely software effort, defect-proneness, maintenance effort and change-proneness. Furthermore, the thesis also investigates the use of several SBA for developing software change prediction models. Another contribution of this thesis is an investigation of hybridized algorithms, which combine a search-based algorithm and an ML technique into a single approach for determining change-prone classes in an OO software.

The research community consistently explores new methods and techniques for developing better and effective prediction models. A promising approach for improvement of existing classifiers is ensemble methodology as it aggregates various individual classifiers to provide stable results. In this thesis, we have investigated the use of ensemble methodology by aggregating several fitness variants of a search-based algorithm i.e. Constricted Particle Swarm Optimization (CPSO) using weighted voting. To the best of our knowledge, no study has evaluated the use of ensembles of SBA in the domain of software change prediction. Furthermore, we have proposed a novel framework, Adaptive Selection of Optimum Fitness (ASOF), which predicts the best fitness variant of CPSO (amongst seven fitness variants) for a specific data point in order to determine its change-prone nature. The premise of selecting different fitness variants corresponding to each data point is that it may be the case that various subsets of a dataset may give best results with a different

fitness function as compared to the use of only one fitness function over the entire dataset. Thus, we propose using optimum fitness functions for different data points of a dataset rather than selecting just a uniform fitness function for the entire dataset.

Another contribution of this thesis is the investigation of techniques for developing effective software change prediction models from imbalanced training data. In practice, researchers might not be able to procure balanced training data with proportionate number of change-prone and not change-prone classes. The use of sampling techniques and metacost learners for developing software change prediction models solve the issue of obtaining impractical models yielded from imbalanced training data.

In order to manage bugs effectively and correct them, categorization of bugs is an essential activity. We are the first ones to categorize software bugs on the basis of its change impact i.e. the number of classes modified to correct a specific software bug. Additionally, we also categorize software bugs in accordance with the maintenance effort required to correct them and due to the combined effect of change impact and maintenance effort. It is important to categorize software bugs on the basis of maintenance effort and change impact so that a software developer can plan the bug fixing regime. Bugs that are categorized as "high" according to maintenance effort, should be allocated more maintenance resources for their correction as compared to "low" category and "moderate" category bugs. On the other hand, stringent regression testing should be performed for bugs which are categorized as "high" in accordance with their change impact as larger number of classes are affected while correcting the bug.

Apart from structural characteristics of a class, its evolution history is also a critical factor in determining its change-prone nature. The evolution history may incorporate how the class has evolved over all the previous versions of the software. We empirically validate the effectiveness of the evolution-based, OO metrics and their combined metrics suite to determine the best predictors of change-prone classes. We

also assess the evolution patterns of OO metrics in order to assess the changes in the internal structure of a software during its evolution.

This chapter gives an introduction of the basic concepts involved in the thesis and the motivation of the work. We first describe the concept of software quality and the various attributes involved in it (Section 1.2). Next, we define software evolution (Section 1.3) and how it is related to software quality (Section 1.4). A brief account of various software metrics is provided in the subsequent section (Section 1.5). We also discuss the need to develop prediction models for software evolution (Section 1.6). The remainder of the chapter discusses the state of existing literature (Section 1.7), objectives of the thesis (Section 1.8) and its organization (Section 1.9).

## 1.2 What is Software Quality?

According to the Institute of Electrical and Electronics Engineers (IEEE), software quality is defined as [8]:

- "The degree to which a system, component, or process meets specific requirements."
- "The degree to which a system, component, or process meets customer or user needs or expectations."

A software should meet all its desirable attributes, be it functional attributes such as response time, graphical user interface design etc. or non-functional attributes such as fault tolerance, extensibility etc. to be considered as a good quality software. If a software fulfills its functional requirements, it implies it is fit for its intended purpose. On the other hand, non-functional requirements are necessary to support the effective delivery of functional requirements.

Another aspect of quality relates itself to customer satisfaction. If a customer is not satisfied with a product, it cannot be considered as a good quality product. Delivery of a poor quality product might result in business losses and a bad reputation for the organization.

Software quality also determines the "quality of design" and the "quality of conformance" [9]. The quality of design relates to the adequacy of software design. An effective software design is important for developing a software with minimum defects and high customer satisfaction. The quality of conformance relates to the degree to which a software conforms to the developed design. Only adequately designing the software is of no use if the software product does not conform to the design blueprint.

### 1.2.1 Software Quality Attributes

Software quality can be determined in terms of various software quality attributes. Some of the important software quality attributes are as follows [8]:

- *Functionality*: It refers to the degree to which a software is fit for its intended purpose. It includes several sub-attributes like completeness, correctness, efficiency, traceability and security.

- *Usability*: It refers to the degree to which a software is easy to work with. It includes several sub-attributes like learnability, operability, user-friendliness, installability and satisfaction.

- *Testability*: It refers to the degree to which a software can be operated to indicate the faults. Verifiability and validatable are the sub-attributes incorporated in this attribute.

- *Reliability*: It refers to the degree to which a software is able to operate without any failures. It includes robustness and recoverability as sub-attributes.

- *Maintainability*: It refers to the degree to which software defects can be identified, software may be modified during maintenance and its quality can be improved. It includes agility, modifiability, readability and flexibility as sub-attributes.

- *Adaptability*: It refers to the degree to which a software can adapt to varying platforms and technologies. It includes portability and interoperability as sub-attributes.

Apart from the above mentioned quality attributes, there are many other software attributes related to the domain of software engineering. Developing models to determine these attributes is known as Software Engineering Predictive Modeling (SEPM). This thesis limits itself to four important software engineering attributes, which are discussed below:

- *Software Development Effort*: This attribute estimates the effort required to develop a specific software product [10]. The effort required can be approximated in terms of person hours or cost. Effort estimation is important as software project managers require important information from past projects to plan and analyze the effort required for project development [11]. Such knowledge is critical for efficient allocation of human resources, so that products can be delivered on time and within the planned budget.

- *Defect-proneness*: It is an attribute which encapsulates the probability of occurrence of a defect in a class/module, after the software has been released [12]. A class/module may be found as "defective" or "not defective" in the next version of a software product. Identification of defect-prone classes, is important

as the cost of correcting defects increases exponentially in later phases of the software development cycle [8]. Thus, it is important for researchers to eliminate defects as early as possible to ensure customer satisfaction.

- *Maintainability*: This quality attribute measures the ease with which a software class or module can be modified and measures the effort required to evolve a particular software class or module [8]. The effort required can be estimated in terms of Source Lines of Code (SLOC). Since, the maintenance phase absorbs 60 to 70% of the total project resources [8], it is essential for software managers to estimate maintenance effort in the early phases so that proper planning and allocation can be done.

- *Change-proneness*: This attribute encapsulates the probability of occurrence of a change in a class/module after the software has gone into operation [5]. Similar to defect-proneness, a class/module may be "change-prone" or "not change-prone" in the next release of the software product. The determination of change-prone nature of a class helps practitioners in efficient resource allocation. These classes need more attention in the early phases of development so that minimal changes get carried to the next stage. Such steps would ensure an efficient and maintainable software product.

Defect-proneness and change-proneness are binary attributes while maintainability and software development effort are continuous attributes. Another important software attribute is change impact levels. This attribute predicts the level of change impact of a software bug i.e. ("low", "moderate" or "high"), which is estimated by the number of classes impacted while correcting a software bug. It is important to assess the attribute in order to manage resources during bug correction and removal.

## 1.3 What is Software Evolution?

Software evolution is crucial as organizations have invested huge amounts of money in their software. These organizations are extremely dependent on these software systems for critical business processes. The Research Institute in Software Evolution defines it as:

*"the set of activities, both technical and managerial, that ensures that software continues to meet its organizational and business objectives in a cost effective way"*

Software evolution is an ongoing process which persists throughout the lifecycle of a product. Delivering a software system successfully to its customers only puts an end to the development process, however software evolution, which involves constantly changing the software after its deployment is inevitable. It is mandatory for a software system to change in order to remain useful. A change could be because of an existing defect, changes in user expectations, technological advancements or business changes [5]. Thus, change is important in order to consistently improve the performance of the software system.

Organizations spend a large amount of money in the maintenance and upkeep of their software systems as these systems are important assets for their growth and business. As a consequence, organizations invest more money and resources on maintaining existing systems than development of new systems. The costs involved in changing a software are a huge part of an organization's budget [8]. Since changing and maintaining a system is expensive, we need to manage and plan changes properly.

### 1.3.1 Software Evolution Cycle

In order to manage and implement a change, a systematic series of steps are followed [9]. These steps are depicted in Figure 1.1.

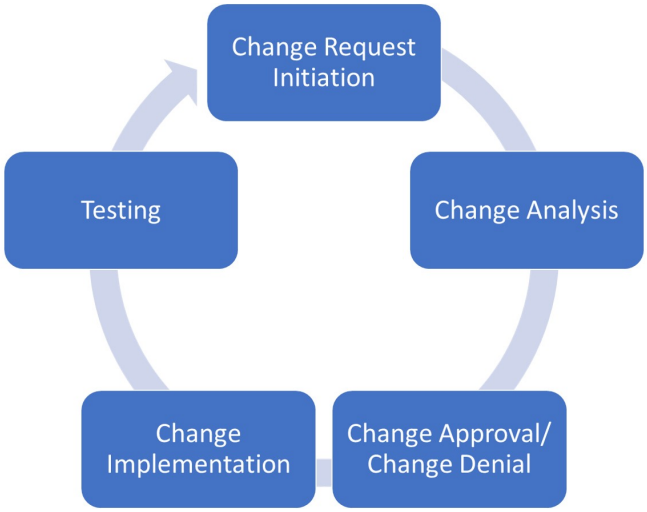


Figure 1.1: Software Evolution Cycle

Initially, a change is requested by a customer, management or any other stakeholder. The requested change is then analyzed by the Change Control Board (CCB) to evaluate the cost, time and resources required to implement it. The implications of the requested change are also assessed. If found suitable, the requested change is approved by the CCB and the developers are asked to implement it. On the other hand, if the change is not found appropriate, the CCB denies its implementation and the person who requested the change is notified. After a change is implemented, the software is validated to evaluate if it is in accordance with what was requested. Other portions of the software which are affected by the change are also investigated to assess if the change has not led to introduction of any new errors and the software is functioning appropriately.

## 1.4 Software Quality and Software Evolution

As already discussed, a software system continuously evolves. However, continuous changes in a software may lead to degradation of software quality. This is because

with continuous evolution, the size and complexity of the software consistently increases. It is difficult to maintain a larger and complex software as developers may not have sufficient resources to focus on all parts of the software. Moreover, a change may affect various portions of a software. It may be the case that a software developer may not be able to foresee all the implications of a change and new errors could be introduced in the software product, leading to a poor quality software product. Thus, it is important to focus on maintaining quality of a software product during its evolution.

Resource management and maintenance of software quality during evolution is an important area of research. Software managers are constantly looking for methods and practices which assure effective quality products. As software resources are a constraint, early prediction of problematic parts of a software is useful so that more resources can be allocated to such parts. The change-prone parts of a software can be considered as problematic as they are prime sources of tentative defects and probable changes in a software. Early identification of such parts may help software managers in effectively dispersing the allocated resources as they need larger number of resources than the other parts. Moreover, early error detection and thereby its removal from such parts of a software ensures cost-effective product management, as the cost to correct an error increases exponentially with each phase of the software development lifecycle [13]. Thus, early identification of change-prone parts of a software is an effective method for managing software quality during evolution.

## 1.5 Software Metrics

According to Tom DeMarco, “You can’t control what you can’t measure” [14]. Measuring structural characteristics of a software system is important to determine the quality of a software product. They can be used for continuous inspection of a soft-

ware to assist software developers in improving its quality. We can also use software metrics to develop quality models which predict important quality attributes such as defect-proneness and change-proneness.

Software metrics can be broadly categorized in accordance with the programming paradigm: a) the traditional programming paradigm, which primarily deals with algorithms and functional decomposition and b) the OO paradigm which focuses on identification of objects, their characteristics and method definitions. Since, the OO software paradigms are quite different from traditional procedural paradigms, it necessitates the need for a different set of metrics for OO software as traditional metrics fail to capture concepts like "inheritance" and "polymorphism" which are unique to OO paradigm. Henderson [15] states the following two differences in traditional and OO paradigms of software design:

- The coding and maintenance phases of the traditional paradigm require greater effort than OO paradigm.
- The OO paradigm focuses more on the earlier stages of software development (analysis and design).

### 1.5.1 Overview of existing OO Metric Suites

A number of OO metric suites have been proposed in the literature. Some of the most commonly used OO metric suites are discussed in this section.

Chidamber and Kemerer (CK) [16] introduced a metrics suite which consists of Weight Methods per Class (WMC), Depth of Inheritance Tree (DIT), Number Of Children (NOC), Coupling Between Object classes (CBO), Response For a Class (RFC) and Lack of Cohesion in Methods (LCOM) metrics. The CK metrics are analyzed to assess their usefulness for practicing managers and have been used successfully by a number of empirical studies [17].

Li and Henry [18] proposed a metric suite which consists of Data Abstraction Coupling (DAC), Message Pass Coupling (MPC) and Number of Methods (NOM). They also introduced two size metrics namely SIZE1 and SIZE2.

Lorenz and Kidd [19] proposed the following OO metrics (Pressman, 2000): Class Size metrics (CS), Number of Operations (methods) Overridden by a subclass (NOO), Number of Operations Added by a subclass (NOA), Number of Public Methods (NPM) and Specialization Index (SIX).

Metrics for Object Oriented Design (MOOD) [20] are used to measure OO programs based on of the following software quality indicators: Attribute Hiding Factor (AHF), Method Hiding Factor (MHF), Method Inheritance Factor (MIF), Attribute Inheritance Factor (AIF), Coupling Factor (COF), and Polymorphism Factor (POF).

Bieman and Kang [21] defined two cohesion metrics namely Tight Class Cohesion (TCC) and Loose Class Cohesion (LCC).

Briand et al. [22] gave a suite of 18 metrics that measured different types of interaction between classes. These metrics may be used to guide software developers about which type of coupling affects maintenance cost and reduces reusability. Two coupling metrics namely Afferent Coupling (Ca) and Efferent Coupling (Ce) were proposed by Martin [23].

Lee et al. [24] differentiate between inheritance-based and non-inheritance-based coupling by the corresponding measures: Non-Inheritance Information flow based Coupling (NIH-ICP) and Information flow-based Inheritance Coupling (IH-ICP). They also introduced Information flow-based coupling metric which was the sum of NIH-ICP and IH-ICP. In order to measure cohesion they also introduced information flow based cohesion (ICH) metric.

Bansiya and Davis [25] proposed the Quality Model for Object-Oriented Design (QMOOD) metrics suite, which includes Design Size in Classes (DSC), Number of Hierarchies (NOH), Average Number of Ancestors (ANA), Data Access Metric

(DAM), Direct Class Coupling (DCC), Cohesion among Methods of a Class (CAM), Measure of Aggression (MOA), Method of Functional Abstraction (MFA), Number of Polymorphic Methods (NOP), Class Interface Size (CIS) and Number of Methods (NOM).

Tang et al. [26] proposed Average Method Complexity (AMC), Inheritance Coupling (IC), Number of Object/Memory Allocation (NOMA) and Coupling Between Methods of a Class (CBM) metrics.

## 1.6 Developing Prediction Models for Software Evolution

In order to effectively manage the evolution of a software it is important to develop models which predict different aspects of software evolution. It is crucial for managers to identify and predict which parts of a software are likely to change and what effort would be expended in maintaining these parts. Software change prediction models aid software project managers in strategizing allocation of limited software resources such as time, cost and effort. A manager should allocate more resources to change-prone classes as they need to be stringently verified and tested to ascertain that changes have been incorporated efficiently and no new errors have been introduced [2, 3]. This would lead to cost effective and high quality products as we can estimate changes in the early phases of software development life cycle and develop plans for handling them effectively. Also, constant monitoring of change-prone classes results in detection of defects as early as possible so that corrective actions can be planned appropriately [5, 27, 28]. Software practitioners can also plan refactoring activities on the identified change-prone classes so that future changes to these classes may not cause ripple effects [1]. In this thesis, we develop models for

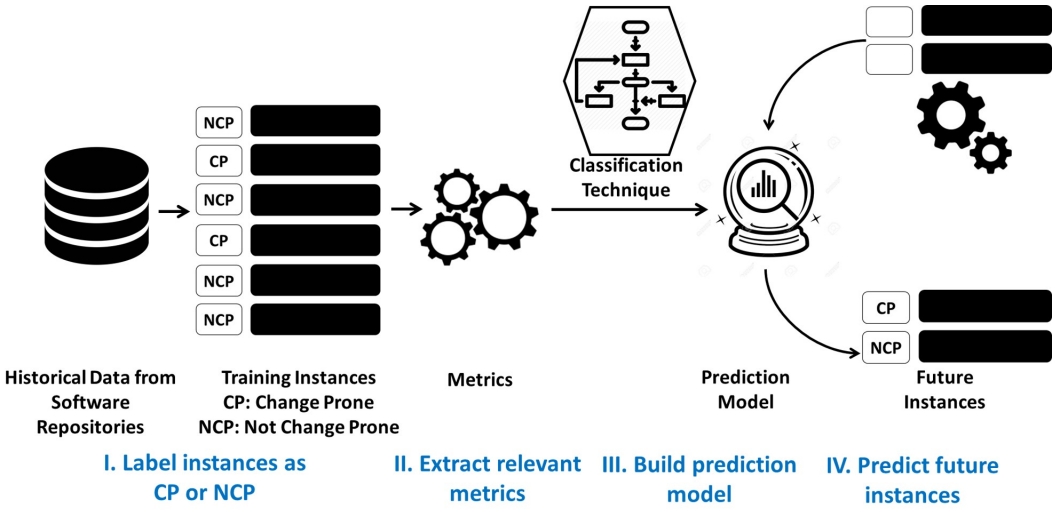


Figure 1.2: Framework of Prediction model for determining Change-prone classes

a) determining change-prone classes in an OO software and b) determining change impact of a software bug.

Predicting change-prone components involves understanding the relationship between the software system internal structural attributes and change. These structural attributes (metrics) can be quantified with the help of a number of software metrics (OO metrics) which are indicators of different attributes of a software like coupling, cohesion, polymorphism, size etc. Figure 1.2 depicts the framework of a change prediction model. First, historical data is extracted from software repositories. The extracted data is a set of data points which are change-prone (CP) or not change-prone (NCP). The data is used for training the model, where each data point contains a set of OO metrics which are predictors and a dependent variable (CP or NCP). Relevant metrics which are not redundant or noisy are extracted from the training data. The prediction models are developed with the aid of various classification techniques (statistical, ML, SBA, hybridized), which learn from historical software data and develop a prediction model. The developed model is validated and predicts the occurrence of change in future versions or yet unseen instances of the software product.

Evaluating the impact of a change is also critical, so that software practitioners can pay focused attention to parts which are impacted by a change. The impact of change of a software bug can be estimated by extracting the unstructured data present in the form of bug reports in the software repositories (Figure 1.3). Textual data from software bug reports can be extracted along with change history of previous modifications in the software. This change history incorporates the number of classes affected by a change and the number of lines of code required to make modifications. By analyzing the number of classes impacted while correcting a bug report, one can estimate its impact. Thus, the training data consists of the textual descriptions of bug reports and the category of change impact (low, medium or high) depending on the number of classes, impacted while correcting a specific bug report. The models developed are binary in nature, which predict if a new bug report would have a High impact (H) or a Not High impact (NH) as shown in Figure 1.3. An NH impact means either low impact or medium impact. Similarly, models for determining low impact or not low impact and moderate impact or not moderate impact can be developed.

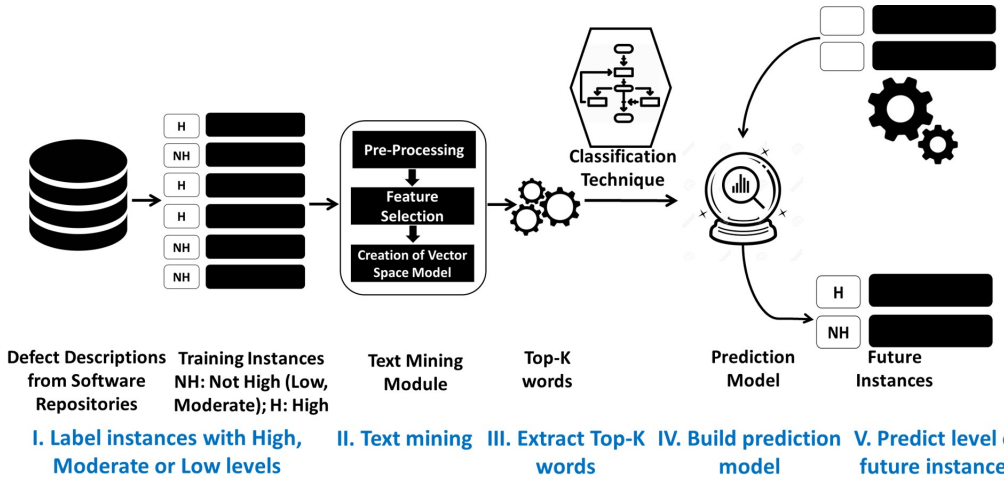


Figure 1.3: Framework of Prediction model for determining Change impact of a Software Bug

In order to mine data from textual descriptions, the training instances should be processed by the text-mining module. The text mining module consists of three main steps: pre-processing the training data, extracting relevant features and creation of a vector space model. Thereafter, Top-K words are extracted from bug descriptions and prediction model is developed with the help of a classification technique. The developed model can predict the impact of a new software bug on the basis of its textual description. Software Bug Categorization (SBC) models, which identify the level of a software bug on the basis of its bug reports in accordance with the maintenance effort required to correct them and its change impact can help software project managers in effective planning of limited resources.

## 1.7 Literature Survey

As discussed, the evolution of a software is critical for the success of a software product. Researchers and practitioners have been working hard to optimize the processes and other facets related to software evolution. A comprehensive study of the existing literature with respect to various elements and processes which require improvement during software evolution is crucial. This would aid in identification of gaps and provides motivation to work on different aspects of this area.

We first discuss the various software metrics that have been used for ascertaining change-prone classes. These metrics were either OO metrics or evolution-based metrics which have been proposed by Elish et al. [1]. We next discuss the models that have been developed in literature for optimizing software evolution. These models were further divided into two categories, one which focuses on identification of change-prone classes in a software and the other which focuses on identification of change impact of a change request. The software change prediction models have been further investigated in detail with respect to the categories of classification tech-

niques which have been used in literature, the investigation of ensemble methods for classification and the development of models from imbalanced training data.

### 1.7.1 Software Metrics

Software metrics encapsulate various characteristics of a software product such as its internal structure in terms of OO properties or evolution characteristics. These metrics are effective in determining the change-prone nature of classes.

**OO Metrics**

In order to develop software change prediction models, studies in literature have extensively used several OO metrics as predictors. A study by Zhou et al. [29] investigated the confounding effect of the size of a class on the predictive ability of OO metrics to determine change-prone nature of a class in an OO software. The study analyzed the effect of three size metrics on the capability of 55 OO metrics, which encapsulated the coupling, cohesion and the inheritance characteristics of a software. Their results were validated on two versions of the Eclipse software dataset and indicated the existence of the confounding effect of class size on the relationship between OO metrics and change-proneness. They found that class size tends to overestimate the existing relationship between OO metrics and change-proneness.

Lu et al. [28] investigated 17 inheritance metrics, 7 size metrics, 18 cohesion metrics and 20 coupling metrics to explore their ability to predict change-prone nature of an OO class. In order to do so, they used statistical meta-analysis techniques and evaluated the results on 102 software systems, developed in Java language. According to their results, the inheritance metrics were worst indicators of change in a class. The size metrics were found to have moderate capability followed by the coupling and cohesion metrics. Studies by Koru and Liu [2] and Koru and Tian [3] also assessed the effectiveness of several OO design metrics, which were representative

of size, coupling, inheritance and cohesion characteristics for ascertaining change-prone nature of a class. The open-source systems used by these studies included KOffce, Mozilla and Open Office.

Giger et al. [30] analyzed the capability of OO metrics proposed by Chidamber and Kemerer [16], along with centrality measures from social network analysis to predict software change. The models developed by them, successfully enlisted change-prone files in the order of their change-prone capability. Eski and Buzluca [27] validated CK metrics [16] and the QMOOD metrics suite [25] to detect change-proneness in three open-source software datasets. Their study confirmed the capability of the investigated metrics suites for determining change-prone classes of a software. Romano and Pinzger [4] evaluated the CK metrics suite [16], a set of metrics to estimate the complexity and usage of interfaces and two external cohesion metrics for ascertaining change-prone java interfaces. Their results indicate that the use of Interface Usage Cohesion metric may improve the prediction of change-prone java interfaces. Malhotra and Khanna [5] and Malhotra and Bansal [31] have also validated the use of several OO design metrics for predicting change-prone classes in open-source software. These studies confirm the predictive capability of the investigated OO metrics.

**Evolution-based Metrics**

Apart from OO metrics, certain studies assessed the effectiveness of evolution-based metrics, which encapsulate the history of changes in a class as predictors of software change. A study by Tsantalis et al. [32] proposed the prediction of probability of change in OO systems by proposing two main sources of change in a class: internal and external. The external sources of change, referred to those changes which were initiated in a class because of changes in other classes. These were categorized in accordance with three axes namely inheritance axis, reference axis and dependency axis. The internal sources of change referred to all possible sources of change

that originated from the class itself. The study validated their results on two datasets JMol and JFlex. It found that the proposed model improved the prediction accuracy for ascertaining change-prone classes.

Another key study, which took into account evolution characteristics of a class, while predicting change-proneness was conducted by Elish and Al-Khiaty [1]. They proposed sixteen evolution-based metrics, which encapsulated history of changes in a class and evaluated these to ascertain the change-prone nature of a class in the future versions of the software product. The study was conducted using two open-source datasets and ascertained the effectiveness of four possible scenarios of metric combinations for ascertaining an OO class's change-prone nature. These scenarios included the use of only evolution-based metrics, the use of only OO metrics, the combination of OO and evolution-based metrics and the use of Internal Class Probability (ICP) of change metric, proposed by Tsantalis et al. [32]. The findings of Elish and Al-Khiaty [1] indicated the superiority of the combined metrics suite, which consisted of both OO and evolution-based metrics for determining change-prone classes. However, more studies are needed to confirm their findings and provide generalized results.

### 1.7.2 Evolution-based Studies

The exponential growth in technology, presence of defects and expanding set of user requirements necessitates software evolution.

#### Software Change Prediction

We first summarize the various modeling techniques used in literature for identifying change-prone classes. Furthermore, we discuss ensemble methodology, which can be used to propose new modeling techniques in the domain of software change prediction. We also assess whether literature studies have investigated methods for effective learning through imbalanced training data in the domain of software change

prediction.

*Classification Techniques for Software Change Prediction*

A wide category of classification techniques are available. Studies in literature have investigated the use of traditional statistical techniques for determining software change. These include studies by Elish and Al-Khiaty [1], Zhou et al. [29] and Lu et al. [28]. Elish et al. [1] used Logistic Regression (LR) to create software change prediction models using two open-source datasets, VSSPlugin and Peersim, while Zhou et al. [29] and Lu et al. [28] used linear regression. However, the most popular category of methods seemed to be ML techniques. Studies by Koru and Tian [3] and Koru and Liu [2] assessed the use of tree-based models for ascertaining the change-prone nature of classes. Giger et al. [30] investigated the use of Bayesian Network (BN) and Neural networks on two open-source datasets developed in Java language to output files which can be potentially change-prone. Romano and Pinzger [4] also assessed the use of Support Vector Machine (SVM), apart from BN and Neural networks for determining change-prone Java interfaces. Sharma et al. [33] developed software change prediction models using Naive Bayes (NB), SVM, Decision Trees (DT), extreme learning machine and several other ML techniques and three ensemble classifiers. They also used the statistical technique, LR. Studies by Malhotra and Bansal [31] and Malhotra and Khanna [5] assessed several ML techniques such as Bagging (BG), Random Forests (RF), NB, Adaptive Boosting (AB), J48, LogitBoost (LB), etc., along with the statistical technique LR for evaluating change-prone nature of classes in an OO software. These studies indicated that the performance of ML techniques were found comparable to that of the statistical technique, LR.

Apart from the statistical and ML techniques, a new category of techniques, SBA have been assessed recently for ascertaining software change and modification effort. Azar [34] used Genetic Algorithm (GA) to determine "syntactic changes" in class methods or insertion/removal of class methods in an OO class. In another

study, Azar and Vybihal [35] investigated the capability of Ant Colony Optimization (ACO) to assess the same. A recent study by Bansal [36], analyzed the effectiveness of SBA and their hybridized counter-parts for determining change-prone classes in OO software. The above mentioned studies, support the use of SBA and their hybridizations in this domain. It may also be noted that their use has also been widely investigated in the domain of software defect prediction. Some of the prominent literature studies which assess the use of these algorithms for determining defect-prone parts of a software include De Carvalho et al. [37], Harman et al. [38], Ferrucci et al. [39], Xia et al. [40] and Hosseini et al. [41]. However, in order to draw generalized conclusions and to determine the effectiveness of wide category of SBA and their hybridizations, more empirical studies are required, specifically for determining change-prone classes in a software.

*Software Change Prediction using Ensemble Learners*

Ensemble learning is an effective methodology for improving the performance of individual classifiers, if the constituent classifiers are found to be accurate and diverse [42–44]. As discussed earlier, Malhotra and Khanna [5], Malhotra and Bansal [31] and Bansal [36] have investigated the use of ML ensemble classifiers like BG, AB, LB and RF. Sharma et al. [33] and Elish et al. [45] investigated three ensemble learning methodologies namely Best-in-Training, Majority Voting and Non-linear Ensemble Decision Tree Forest. However, researchers have not investigated the use of SBA as constituents of ensembles.

An interesting characteristic of SBA is the change in the results of prediction models with the use of different fitness functions [46–49]. Di Martino et al. [46] investigated the use of different fitness functions of GA, when GA was used for optimizing the parameters of SVM, while predicting defect-prone classes in a software. The study confirmed the change in the results of the classification model, with the change of fitness function. Another study by Ferrucci et al. [47] observed the vari-

ation in results due to change of fitness function, when Genetic Programming (GP) was used for estimating software effort. A study by Aslam [49] ascertained that the results of binary classification varied due to the choice of different fitness functions. Bhowan et al. [48] improved the results of GP on unbalanced datasets by proposing the change of fitness function. However, none of these studies ascertained the effect of variation of a fitness function in the domain of software change prediction.

Previous studies in the literature have used various approaches to combine multiple classifiers to correctly predict defect-prone classes in a software. Software defect prediction is a related area of change prediction. A study by Petric et al. [50] used four ML techniques which were aggregated using the weighted accuracy and diversity technique to obtain better performing defect prediction models. Panichella et al. [51] proposed an approach named as CODEP (COmbined DEfect Predictor) for combining several ML techniques to improve cross-project defect prediction. A study by Zhang et al. [52] combined six ML techniques using seven composite procedures to perform effective cross-project defect prediction. Aljamaan and Elish [53], Misirh et al. [54] and Laradji et al. [55] also combined multiple ML techniques for the purpose of software defect prediction. A study by Di Nucci et al. [56] proposed a framework for predicting a different ML technique for a particular instance of a dataset on the basis of its structural characteristics. However, there has been no study which predicts the optimum fitness variant of a search-based algorithm for a particular instance in the dataset.

*Software Change Prediction using Imbalanced Data*

Effective learning through imbalanced training data, where the number of classes belonging to a specific category are highly disproportionate as compared to the ones belonging to the other category is a well-recognized research problem. Various studies have addressed this problem for software defect prediction through the use of various sampling approaches, use of several cost-sensitive learners, use of ensemble

learners, use of active-learning and kernel-based methods and many other proposed approaches. We discuss some of these prominent studies in the area of defect prediction. Shatnawi [57] investigated a widely used oversampling method, Synthetic Minority Oversampling Technique (SMOTE) for developing defect prediction models from imbalanced data using three classifiers. Seliya and Khoshgoftaar [58] investigated six different cost-sensitive learning techniques, which were three cost-sensitive boosting techniques, meta-cost classifiers, weighted techniques and random undersampling on 15 software datasets for developing defect prediction models using imbalanced training data. Liu et al. [59] proposed a two-stage cost-sensitive learning technique for software defect prediction. The technique used cost-sensitive information at two stages, feature selection as well as classification. A study by Rodriguez et al. [60] compared cost-sensitive, sampling methods, hybrid techniques and ensembles to deal with imbalanced datasets. The study performed empirical validation using 12 imbalanced datasets. The results advocated the use of different pre-processing steps for dealing with imbalanced data to enhance the performance of software defect prediction models. Arar and Ayan [61] proposed a cost-sensitive neural network using artificial bee colony algorithm for developing effective defect prediction models from imbalanced data. Though, these studies investigated the use of various methods for imbalanced data learning, all of them developed software defect prediction models. A related study by Tan et al. [62] classified changes as "buggy" or "clean". The study used four sampling techniques namely SMOTE, resampling with replacement, resampling without replacement and spread subsample for building efficient models from imbalanced data. As it can be seen, there is a huge gap in literature with respect to analyzing approaches for imbalanced learning, while developing software change prediction models. Similar to defect prediction, the number of change-prone classes in the training data is much lower than not change-prone classes. Inappropriate learning from such data may lead to higher classification errors while determining

change-prone classes. Thus, there is an urgent need to evaluate various approaches for handling imbalanced data while developing software change prediction models.

**Software Change Impact**

Various studies in literature have investigated change impact analysis using static program analysis[63, 64], dynamic program analysis [65] or by using a hybrid of both [66]. Literature studies have also mined version histories to estimate the impact of a change. These version histories could be specific to the software [67] or could be independent of the software for which change impact analysis is to be done [68].

Another popular approach for investigating the change impact of an incoming change request has been the use of Information Retrieval (IR) techniques, which analyze the textual information available in the change request or in other program entities. A study by Canfora and Cerulo [69] analyzed the impact analysis of a change request by predicting a list of impacted files. IR algorithms were used to find the similarity between new change request and various source code entity descriptors. Antoniol et al. [70] mapped the maintenance request to its starting impact set using textual information present in source code and other high level documents of a software. Gethers et al. [71] proposed an integrated approach for change impact analysis which included mining of previous source code commits, dynamic analysis of execution data and IR from the textual data of the change request. A study by Zanjani et al. [72] used interaction as well as commit histories to find the change impact of an incoming change request using its textual description. Initially, a corpus of previously resolved change requests was built, which was queried with the incoming change request to obtained a ranked list of impacted source code entities. These studies indicate that the textual information of change requests can be successfully used as a predictor for estimating change impact sets.

Only very few studies have analyzed effort as an ordinal variable. Recent studies

by Jindal et al. [73, 74], categorized software bugs on the basis of maintenance effort required to correct them. They extracted relevant attributes from bug reports and classified bugs into four categories, viz., the ones requiring "very low", "low", "medium" and "high" maintenance effort for correction. Studies by Basgalupp et al. [75, 76] measured maintenance effort in terms of hours. However, they transformed these numeric values to nominal by transforming the data into low effort, medium effort and high effort equally. Balogh et al. [77] predicted "modification effort", which was analyzed as a function of weighted count of modifications and the net development time of these modifications. The dependent variable was then assigned low, medium and high values equally. However, to the best of our knowledge, there is no study in literature with respect to analyzing the change impact levels of a software change request or a bug report using IR techniques.

## 1.8 Objectives of the Thesis

### 1.8.1 Vision

Improving software quality by managing the software evolution process using efficient prediction models for ascertaining software quality attributes (change-proneness and change impact levels).

### 1.8.2 Focus

The focus of the thesis is to envisage methods which help in assessing and improving the various aspects involved in developing software change prediction models for determination of change-prone classes. The aspects analyzed in this work include (1) evaluation of existing diverse categories of classification techniques; (2) proposal

of new classification techniques based on ensemble methodology; (3) creation of efficient models from imbalanced datasets; (4) assessment of trends of various OO metrics in an evolving software and analysis of different set of predictor metrics, which encapsulate evolution characteristics. We also analyzed the change impact levels of software bugs using their textual descriptions.

### 1.8.3 Goals

A summary of the goals investigated in this work is provided below:

1. To evaluate and compare various data analysis techniques (statistical, ML, search-based and hybridized) for software change prediction.

    - With a large number of available classification techniques, which have different abilities and characteristics, it is important to conduct studies to evaluate their capability in the domain of software change prediction.
    - Such studies can be used by researchers and practitioners as guides for selection of an appropriate technique in a certain context.

2. To propose new modeling techniques based on ensemble methodology and ascertain their effectiveness for developing software change prediction models.

    - Modeling techniques based on ensemble methodology are known to develop stable and robust models, which improve the capability of their constituent algorithms. Such techniques can be used by practitioners and researchers to develop software change prediction models with improved accuracy as they are based on diverse constituent algorithms.

3. To investigate the use of methods for developing practical software change prediction models from imbalanced training data.

- In general, software datasets have low percentage of change-prone classes as compared to not change-prone classes. Such training sets when used for developing prediction models may produce models which are not able to correctly identify change-prone classes as they were deficient in the training data.
- Thus, studies should be conducted to examine methods that can lead to creation of effective models even from imbalanced data.

4. To perform software bug categorization and assign levels to bugs on the basis of their change impact values and/or the maintenance effort required to correct them.

    - Software testers and maintenance personnel need to prioritize resources as they are always a constraint. One method to do so is to assign levels to software bugs on the basis of their descriptions. These levels are allocated on various bug characteristics such as its change impact and the maintenance effort required while correcting a software bug. Resources can then be allocated on the basis of predicted levels.

5. To analyze the trends of OO metrics over various releases of an evolving software and to evaluate the use of evolution-based metrics as predictors for determining change-prone classes in a software.

    - OO metrics encapsulate various design characteristics of a class, such as its dependency on other classes, its cohesiveness, its size as well as its reusability. We need to conduct studies which analyze the progression of these metrics when a software evolves so that software practitioners can understand the effects of evolution on the software's internal structure and proper steps can be taken if the internal structure of a software degrades.

- Apart from OO metrics, other metrics which encapsulate evolution history of a class may be used as predictors for determining change. We need to perform experiments to evaluate the best set of predictors.

## 1.9 Organization of the Thesis

This section discusses the organization of the thesis. **Chapter 1** presents a basic introduction of the work and motivation of the thesis. **Chapter 2** describes the research methodology followed in the subsequent chapters. **Chapter 3** presents a review of existing studies in the domain of software change prediction, which identifies current trends and research gaps. **Chapter 4** presents the construction of software change prediction models using ML techniques. **Chapter 5** analyzes the capability of SBA for developing software change prediction models. Subsequently, **Chapter 6** analyzes the capability of Hybridized techniques (HBT) for developing models to ascertain the change-prone nature of classes and compares them with SBA and ML/statistical techniques. **Chapter 7** proposes four classification techniques based on ensemble methodology for software change prediction, while **Chapter 8** proposes a framework for dynamically allocating a fitness function to each data point in the training set. SBC models on the basis of a bug's change impact and maintenance effort have been evaluated in **Chapter 9**. **Chapter 10** discusses sampling methods and cost-sensitive learners for developing efficient change prediction models, when the training dataset is imbalanced. **Chapter 11** analyzes evolution-based metrics as predictors for determining software change. Finally, **Chapter 12** states the conclusion of the thesis.

**Chapter 1**: This chapter states the basic concepts of software quality, its various attributes and software evolution. The relationship between software quality and software evolution is also discussed. A literature survey of previous studies is also

included in the chapter. The chapter also states the objectives of the thesis.

**Chapter 2**: This chapter describes in detail the research methodology followed in the thesis. A brief description of the predictor variables, classification algorithms, validation methods, statistical tests and the datasets used in the work is provided. The pre-processing steps before model development are also explained. It also summarizes the performance measures used to evaluate the developed models.

**Chapter 3**: This chapter reviews 34 primary studies, which develop prediction models for ascertaining change-prone nature of a class or which determine change impact of an incoming software change request. Research Questions (RQs) have been formulated that summarize the empirical evidence with respect to predictors, experimental settings, categories of data analysis algorithms, predictive performance of ML techniques, statistical tests and possible threats to validity in these studies. The chapter also discusses the literature gaps in the domain.

**Chapter 4**: This chapter deals with construction of software change prediction models using 11 ML techniques (C4.5, RF, Multilayer Perceptron with Backpropagation (MLP-BP), MLP with Conjugate learning (MLP-CG), Group Method of Data Handling (GMDH), AB, LB, NB, BG, SVM, Classification and Regression Tree (CART)) on six open-source datasets. The results of ML techniques are also compared with the statistical technique, LR. The performance of the investigated techniques have been compared statistically using different performance measures.

**Chapter 5**: This chapter first reviews 91 primary studies from January 1992 to December 2017, which have used SBA in the domain of SEPM. The studies were restricted to four software attributes: software effort, defect-proneness, maintainability and change-proneness. The capability of SBA used in literature have been assessed and the chapter summarizes the trends with regard to the use of SBA for SEPM. Thereafter, the chapter analyzes the capability of several SBA (CPSO, Genetic Algorithm Based Classifier Algorithm with adaptive discretization intervals (GA-ADI),

Genetic Algorithm Based Classifier with Intervalar rules (GA-Int), Memetic Pittsburgh Learning Classifier System (MPLCS), X-Classifier System (XCS), SUpervised Classifier System (SUCS), Hierarchical Decision Rules (HIDER) and Gene Expression Programming (GEP)) for developing change prediction models. The results have been empirically validated using 14 open-source datasets (Eclipse, PMD, Subsonic, Simutrans, Frinika, Jmeter, Celestia, Glest, Apollo, AVISync, AOI, DSpace, DrJava and Robocode). The performance of the developed models have been assessed using G-mean1 and Balance performance measures. The capability of the investigated SBA is also compared with a traditional statistical method Linear Discriminant Analysis (LDA), and ML techniques (C4.5, SVM, CART and MLP-CG).

**Chapter 6**: This chapter evaluates the effectiveness of four hybridized versions of SBA (Decision Trees with Genetic Algorithms (DT-GA), Particle Swarm Optimization with Linear Discriminant Analysis (PSO-LDA), Neural Net Evolutionary Programming (NNEP) and Genetic Fuzzy System Logitboost (GFS-LB)) for constructing software quality models which predict the change-proneness attribute of a class. Six application packages of Android dataset have been used for empirical validation. Apart from evaluating their capabilities, the chapter also assesses the CPU time taken by these techniques. Furthermore, a comparison with 6 SBA and 5 ML/statistical techniques (C4.5, SVM, CART, ML-CG, LDA) is conducted on the basis of predictive capability and CPU time. The trade-off between CPU time and predictive performance of all the investigated techniques is also performed.

**Chapter 7**: This chapter proposes four classification algorithms based on ensemble methodology, namely, Majority Voting Ensemble Classifier (MVEC), Weighted Voting Ensemble Classifier (WVEC), Hard Instance Ensemble Classifier (HIEC) and Weighted Voting Hard Instance Ensemble Classifier (WVHIEC). The proposed classifiers were built by aggregating the votes of seven fitness-based CPSO classifiers. The votes of constituent classifiers were given appropriate weights based on their ac-

curate predictions or the ability to predict "hard instances". The capabilities of the four proposed classifiers were compared with their constituent classifiers and with four ML ensemble techniques (RF, BG, AB and LB). Ten open-source datasets have been used for empirical validation.

**Chapter 8**: This chapter proposes that different fitness functions might be used for various subsets of a dataset, rather than a uniform fitness function for the entire dataset, while using a search-based algorithm such as CPSO for predicting change-proneness. A novel framework named ASOF for predicting a dynamic fitness function for each instance of the dataset is validated. The framework predicts one amongst the seven investigated CPSO fitness variants, viz. accuracy based CPSO variant, precision based CPSO variant, G-mean1 based CPSO variant, G-mean2 based CPSO variant, Balance based CPSO variant, F-measure based CPSO variant or G-measure based CPSO variant for predicting change-prone nature of an OO class. The decision of the framework is based on structural characteristics of an OO class. The ASOF framework is empirically validated on 15 popular open-source datasets and the results are statistically evaluated. The results of the proposed framework are also compared with nine baseline techniques.

**Chapter 9**: This chapter proposes categorization of software bugs into three different levels (low, moderate and high) on the basis of two bug characteristics: maintenance effort and/or change impact. The chapter first ascertains the effectiveness of SBC models developed using 6 classification techniques (MLP-BP, LB, NB, BG, RF, LR) for categorizing software bugs on the basis of a) their maintenance effort; b) their change impact values and c) the product or maintenance effort and change-impact. Thereafter, the three categories of models are compared amongst themselves using Area Under Receiver Operating Characteristic Curve (AUC) and accuracy values.

**Chapter 10**: This chapter focuses on investigation of methods for developing effective software change prediction models from imbalanced datasets. The chap-

ter first evaluates the capabilities of three sampling methods (SMOTE, Resampling with replacement, Spread Subsample) and MetaCost learners when used with six ML techniques ( MLP-BP, NB, RF, AB, BG, LB) for dealing with unbalanced data. Thereafter, the best sampling method amongst the three investigated sampling methods is compared with MetaCost learners. The models are developed using both ten-fold cross-validation and inter-release validation on three Android application packages (Bluetooth, MMS, Calendar) and three Apache datasets (IO, Net, Log4j). The chapter evaluates the obtained results using traditional performance measures such as accuracy, sensitivity and precision as well as stable performance measures.

**Chapter 11**: This chapter first provides an overview of the trends of OO metrics corresponding to four dimensions viz. coupling, inheritance, size and cohesion. Thereafter, the chapter focuses on evaluating the best set of predictors for determining change-prone nature of classes in an OO software. The chapter ascertains the suitability of a new category of metrics, namely, evolution-based metrics proposed by Elish and Al-Khiaty [1]. It evaluates three other scenarios for prediction of change-prone classes, i.e. the use of only OO metrics, the use of combined metric suite which includes OO metrics and evolution-based metrics and the use of ICP metric proposed by [32]. The classification techniques used in the chapter include LR, MLP-BP, LB, AB, BG, RF and NB.

**Chapter 12**: This chapter summarizes the conclusion of the work performed and lists few directions for future work.

# Chapter 2

# Research Methodology

## 2.1 Introduction

In order to achieve our objectives and perform reliable experiments, we need to follow a systematic procedure. Research methodology is the well-defined sequence of steps that are required for conducting effective empirical experiments. This chapter explicitly states the overview of the research process and the methodology followed in the thesis. The chapter is organized as follows: Section 2.2 states the research process for conducting empirical experiments. Section 2.3 states the definition of the research problem, while section 2.4 states the literature survey conducted in order to provide an overview of the existing literature. Section 2.5 defines the dependent and independent variables used in the experiments conducted in subsequent chapters. Section 2.6 describes the functioning of the various data analysis techniques used in the thesis along with their parameter settings. Section 2.7 illustrates the process of empirical data collection while section 2.8 describes the data preprocessing steps. Section 2.9 discusses model development and validation. Thereafter, the performance measures (Section 2.10) and statistical tests (Section 2.11) are discussed.

## 2.2 Research Process

Research process systematically summarizes the steps conducted to investigate a research problem at hand. An illustration of the various steps of the research process, which are followed in the subsequent chapters of this thesis is provided in Figure 2.1. These steps are further elaborated into subsequent sections.

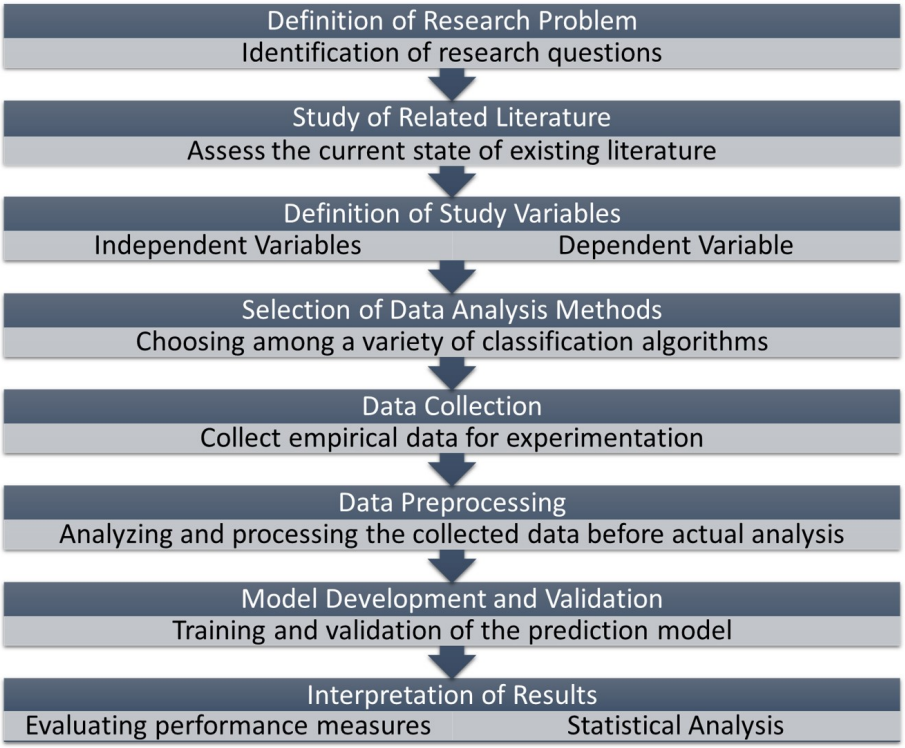


Figure 2.1: Research Process

## 2.3 Definition of Research Problem

To begin our research, we need to specifically state and define the research problem. The problem at hand is specified in the form of RQs. The main objective of con-

ducting a research experiment is to find answers to the investigated RQ's. The RQ's addressed in the current thesis are stated below:

1. What is the existing state (current trends and research gaps) of available literature studies in the domain of software change prediction?

2. What is the performance of different categories of classification techniques (statistical, ML, SBA and HBT) for developing software change prediction models?

3. Which methods can be used by the research community and software practitioners for developing effective change prediction models from imbalanced data?

4. What framework can be used to assign a change impact level to a software bug on the basis of its description?

5. What are the best set of predictors for determining the change-prone nature of a class in an OO software?

## 2.4 Literature Survey

A survey of existing related studies is essential to understand the problem. It also provides us with information about the extent to which the research problem has been investigated by previous studies. Various researchers in the past have developed models for predicting change-prone nature of a class [1–5, 27–31, 33–35, 78]. The studies successfully establish the relationship between OO metrics and change-proneness attribute of classes. Furthermore, the studies confirm the effective use of change prediction models in various commercial and open-source software datasets. Software practitioners can use these models for efficient resource allocation of limited project

resources such as its budget, effort and time to the identified change-prone classes so that good quality software products are delivered. Therefore, it has been established in literature that developing effective change prediction models is an essential and crucial activity for improving software quality.

## 2.5 Define Variables

In order to develop prediction models, we need to define two types of variables i.e. the dependent variable and the independent variable. The dependent variable (target variable) is the software attribute we would like to predict. It depends on the set of predictors (independent variables) and varies with the variation in the predictors. This thesis explores two dependent variables i.e. change-proneness and change impact levels. On the other hand, the independent variables are the predictors which are capable of predicting the dependent variable and are independent in nature, i.e. they are stand-alone and are not affected by the other factors/variables that we are analyzing. We intend to develop models which can study the change in the independent variables and predict the dependent variable. We explore three categories of independent variables in this thesis namely OO metrics, evolution-based metrics and set of relevant words from textual content of bug descriptions.

### 2.5.1 Object-Oriented Metrics

Relevant literature reveals wide use of OO metrics for developing models for defect prediction, maintainability prediction or change prediction. These metrics are representatives of various characteristics of a software such as coupling, reusability through inheritance, cohesion, size etc. as shown in Table 2.1. It is important to monitor and comprehend these metrics in order to gain a better understanding and

effectively administer and supervise a software product.

Table 2.1: OO Metrics depicting OO Characteristic

| OO Characteristic | Definition | OO Metrics |
|---|---|---|
| Coupling | Coupling represents the interdependence of a class on other classes. | CBO, RFC, Ca, Ce, IC, CBM |
| Cohesion | Cohesion signifies the strength of the relationship between a class's methods and its data members. | LCOM, CAM, LCOM3 |
| Inheritance | Inheritance signifies the characteristics of a descendant class by which it inherits properties of its ancestor class. | NOC, DIT, MFA |
| Size | This characteristic signifies the size of a class in terms of number of methods, number of source code lines and so on | WMC, NPM, SLOC, AMC |
| Composition | This characteristic represents the re-usability of a user-defined class as an instance of another class. It is commonly referred to as has-a relationship. | MOA |
| Encapsulation | This characteristic signifies the hiding of irrelevant information about an object in order to reduce complexity and increase efficiency. | DAC |

The OO metrics used in the thesis are summarized below:

- The CK metrics suite [16], as discussed in Chapter 1 consists of six OO metrics, WMC, NOC, DIT, CBO, LCOM and RFC. The CK metrics suite has been widely used for predictive modeling tasks in literature studies [1, 30, 79–82].

- We also analyze the QMOOD metrics suite, which includes MOA, DAM, MFA, NPM and CAM. This metrics suite has been previously validated by Eski and Buzluca [27] and Olague et al. [83] for change prediction and defect prediction tasks respectively.

- Metrics proposed by Martin [23] for coupling (Ca and Ce) are also analyzed in the thesis. Other metrics evaluated in the thesis are AMC, SLOC, LCOM3 (proposed by Henderson [15]), IC and CBM.

Table 2.2: Independent Variables

| OO Metric | Acronym | Definition | Source |
|---|---|---|---|
| Lack of Cohesion amongst Methods | LCOM | It represents a count of pairs of methods of a specific class that does not share any of the class's members and are hence not related. | [16] |
| Coupling Between Objects | CBO | It represents the number of coupled classes to a specific class. | [16] |
| Weighted Methods of a Class | WMC | It is the sum of all method complexities. A complexity value 1 is allocated to each method, thus it is a representative of number of a class's methods. | [16] |
| Response For a Class | RFC | It estimates the number of methods which respond if a specific class receives a message. | [16] |
| Depth of Inheritance Tree | DIT | It represents the maximum level of the class in the inheritance tree. | [16] |
| Number of Children | NOC | It counts the number of immediate subclasses. | [16] |
| Cohesion Amongst Methods | CAM | It estimates the connectivity amongst class methods on the basis of their parameter list. A summation of different parameter types used by all methods of a class is divided by the product of total count of methods of a class and the total number of different parameter types. | `http://www.scitools.com/features/metrics.php` |
| Lack of Cohesion amongst Methods 3 | LCOM3 | It is computed as: $\frac{(\frac{1}{v}\sum_{i=1}^{v}\lambda(v_i)) - m}{1-m}$ where m: no. of methods; v: no. of attributes; $\lambda$ (v): No. of methods that access variable v | [15] |
| Afferent Coupling | Ca | It represents the count of classes using a specific class (fan-in) . It can also be termed as export coupling. | [23] |
| Efferent Coupling | Ce | It represents the classes which are used by a specific class (fan-out). It can also be termed as import coupling. | [23] |
| Measure of functional Abstraction | MFA | It is computed as the ratio of inherited methods to the total number of accessible methods of a class. | `http://www.scitools.com/features/metrics.php` |

| OO Metric | Acronym | Definition | Source |
|---|---|---|---|
| Average Method Complexity | AMC | It computes the average number of java byte codes as a representative of method size. | http://www.scitools.com/features/metrics.php |
| Number of Public Methods | NPM | It counts the number of public methods of a class. | http://www.scitools.com/features/metrics.php |
| Source Lines of Code | SLOC | It counts the number of lines in the java binary code of the class. | http://www.scitools.com/features/metrics.php |

A detailed definition of these metrics is provided in Table 2.2. The primary reason for selection of these OO metrics is that they have been conventionally used and are widely accepted for predictive modeling tasks in the software engineering community [1, 27, 30, 79–83]. However, there have been only few studies which evaluate the change-proneness attribute on the basis of OO metrics. But, a related area with change-proneness prediction is prediction of defect-prone classes. According to a review conducted by Radjenovic et al. [84] on 106 defect prediction studies, the most commonly used OO metric suite was the CK metrics suite. Moreover, the study claims that the popularity of CK metric suite has been "evenly distributed over the years and there is no sign that this will change in future". Thus, this thesis also uses the CK metrics suite for determining change-prone classes. Moreover, an extensive study by Lu et al. [28] evaluated 62 OO metrics for determining change-prone classes and found coupling, cohesion and size metrics like CAM, LCOM, CBO and SLOC to be effective predictors of change. Eski and Buzluca [27] used CK as well as QMOOD metric suite for predicting change and found them to be efficient predictors of change. Therefore, we use an effective group of OO metrics for determining change-prone classes.

### 2.5.2 Evolution-based Metrics

Apart from OO metrics which are representative of the design of a class, other metric suites which quantify alternative dimensions may be useful for improving the accuracy of change prediction models. One such metrics suite is evolution-based metrics suite, which is proposed by Elish and Al-Khiaty [1]. The evolution-based metrics suite quantifies the release by release history of changes in a class. These metrics are representative of evolution characteristics of a class over all its previous releases and are important in order to understand the progression and change-prone nature of a class. The detailed description of these metrics is present in Chapter 11.

### 2.5.3 Bug Descriptions

In order to analyze the changes required to correct a specific software bug, we need to assess the corresponding bug descriptions. The bug descriptions can be extracted from bug reports. Software bug reports can be obtained from change logs, which are present in version control systems such as Git. The textual contents of the bug descriptions (unstructured data) are mined to retrieve the top-k words (structured data), which act as independent variables for analyzing the change impact of a bug. The process of extracting relevant words from textual descriptions of bug reports is described in Chapter 9.

### 2.5.4 Dependent Variables

In this thesis, we aim to develop models to predict two dependent variables: change-proneness and change impact levels.

- *Change-Proneness*: This attribute predicts whether a specific module/class of software will evolve, i.e., require changes after the software product goes into

its operational phase. It is a binary attribute which predicts whether a module/class is change-prone or not. In order to comprehend the change-prone nature of a class, we analyze two different versions of the same software, say an old version and a more recent new version. A class is termed as change-prone if certain SLOC have been added, deleted or modified in the corresponding class, in the new version of the software as compared to the old version, otherwise it is termed as not change-prone. Previous studies have also characterized the change-proneness dependent variable in the same way [5, 28]. Taking into account the change in SLOC is a practical measure for determining the binary variable change-proneness.

- *Change Impact Levels*: This attribute predicts the level of change impact introduced while correcting a software bug. The predicted level depends on the number of classes affected during correction of a software bug. It is an ordinal variable with three possible values: "low", "moderate" and "high". It should be noted that the number of classes impacted is a continuous variable. We convert this continuous variable into ordinal variable by binning the values into three equal sized categories according to their empirical distribution [73, 85].

## 2.6 Selection of Data Analysis Methods

In this thesis, we have used several statistical techniques, ML techniques, SBA and HBT for developing prediction models. Statistical techniques are traditional algorithms that model the underlying relationship between predictors and the target variable using mathematical equations. ML techniques learn from historical data to model the relationships between the independent and the dependent variables and predict the outcome of future instances. SBA are effective in solving optimization

problems. HBT combine the advantages both i.e. statistical / ML techniques along with SBA to provide a single algorithm for classification. Apart from these, we also proposed fitness-based ensemble classifiers. Table 2.3 states the techniques explored in the thesis belonging to each category.

Table 2.3: Data Analysis Techniques

| Category | Techniques |
|---|---|
| Statistical | LR, LDA |
| ML Techniques | Decision Trees (C4.5, CART), Ensemble Learners (BG, AB, LB, RF), Multilayer Perceptron (Back-propagation (MLP-BP) and Conjugate Learning (MLP-CG), GMDH), NB, SVM |
| SBA | CPSO, Genetic Algorithm based Classifier System (GA-Int and GA-ADI), HIDER, Learning Classifier Systems (XCS, SUCS, MPLCS), GEP |
| HBT | DT-GA, PSO-LDA, GFS-LB, NNEP |
| Fitness-based Ensemble Classifiers | MVEC, WVEC, HIEC, WVHIEC, ASOF |

We now state a brief description of all the data analysis techniques used in the thesis.

### 2.6.1 Logistic Regression

LR is used to estimate the percent of variance in the dependent variable due to the independent variables [86]. There are two types of LR analysis, a) Univariate & b) Multivariate. While the univariate LR analysis is used to find the relationship of each independent variable with the dependent variable, multivariate LR analysis is used to develop models using the complete set of independent variables. The multivariate LR analysis aids in construction of prediction models to determine the dependent variable. As multivariate LR models use a number of independent variables, there are two methods for selecting specific independent variables: forward selection and backward elimination [86]. The forward selection procedure selects a variable at each step to be included in the model. On the contrary, the backward elimination model

initially starts with all the variables, and then eliminates one variable at a time until the desired model is obtained. This thesis uses the backward elimination method. The formula for univariate LR analysis is defined as follows:

$$Odds = \frac{p}{1-p} \tag{2.1}$$

Here, *p* is the probability of a class being change-prone. The formula is derived as the simple logistic model is based on the linear relationship between a numerical predictor variable and the natural logarithm (ln) of an event (a class detected as change-prone).The equation for simple logistic model is defined as follows:

$$\ln(Odds) = \alpha + \beta x \tag{2.2}$$

Here, x is the predictor variable, $\alpha$ corresponds to y-intercept and $\beta$ corresponds to the slope. From (2.1) and (2.2), we get the univariate formula:

$$p = \frac{e^{f(x)}}{1 + e^{f(x)}} \tag{2.3}$$

where $f(x) = \alpha + \beta$. Hence, the univariate formula can be extended to the multivariate formula as follows:

$$p = \frac{e^{\alpha+\beta_1 x_1+\beta_2 x_2+...+\beta_n x_n}}{1 + e^{\alpha+\beta_1 x_1+\beta_2 x_2+...+\beta_n x_n}} \tag{2.4}$$

where, $x_i, 1 \leq i \leq$1 are predictor variables.
The following statistics are reported for each significant metric:

- Maximum likelihood estimation (MLE) and coefficients ($\beta_i$): MLE is a well-known method for estimating the coefficients of a model. The likelihood function (L) estimates the probability of observing the set of target values. The

MLE method aims to find the coefficient values, such that the log of likelihood function is maximized. A large coefficient value indicates higher impact of the predictors on the target variable [86].

- Odds Ratio (OR): It is computed using $\beta_i$. The formula is as follows:

$$OR = e^{\beta_i} \tag{2.5}$$

where $\beta$ is the coefficient. It is computed as the ratio of the probability of the occurrence of an event (a class is change-prone), over the probability of non-occurrence of an event. An 'odds ratio' with a value of three indicates that the target variable is increased three times when there is one unit increase in the predictor variable [86].

- Statistical significance (Sig.): It is a determinant of the significance of the coefficient. A larger significance value indicates lower impact of the predictor variables on the target variable. Statistical significance is computed using two-tailed p-value, obtained using the Wald test (W). It is the ratio of the coefficient of a predictor variable to its standard error (SE) [86].

### 2.6.2 Linear Discriminant Analysis

LDA is a technique which can be used for allocating classification labels to a set of instances by using a number of predictors. LDA uses training data to build a set of determinant functions, which are linear functions of predictor variables. The following equation depicts a determinant function ($d_i$):

$$d_i = w_{i1}x_1 + w_{i2}x_2 + ..... + w_{ij}x_j + b_i \tag{2.6}$$

Here $w_{ij}$ represent discriminant coefficients; $x_j$ represent the predictor variables, $b_i$ is a constant for discriminant function i; i = 1 to k for a k-class problem. The determinant functions are used for predicting the classification label of an unknown instance. For a k-class problem, i.e. a problem in which an instance can be allocated one of the k-classes; k determinant functions are computed for each instance with an unknown label. The determinant function of a class 'i', which obtains the highest value for the unknown instance is allocated as its label. In case of predicting change-prone nature of classes, the value for k is 2.

### 2.6.3 Multilayer Perceptron

Multilayer Perceptron (MLP) are artificial neural networks which map a set of inputs from the input layer to the desired set of outputs using a number of hidden layers [87]. They are feedforward networks which simulate the biological neurons. MLP training with backpropagation (MLP-BP) is the most common mechanism, where two passes take place i.e. forward and backward through the network. In the forward pass, the inputs are applied and the output is produced, which is the actual response of the network. During the process, the synaptic weights of the network are adjusted. The backward pass propagates the error signal through the network which is the difference of actual output and the desired output. The weights of the network are again readjusted so that the actual response becomes closer to the desired response. The parameter settings for MLP-BP used in the thesis are default parameter settings of the Waikato Environment for Knowledge Analysis (WEKA) tool [88]. A learning rate of 0.3, momentum of 0.2 and a validation threshold of 20 are the parameters used for MLP-BP.

MLP-CG is a feedforward neural network which also uses the backpropagation algorithm to adjust the weights. However, there is a slight modification. The direc-

tion of movement in the backpropagation algorithm is the negative of the gradient of error. Though, it is effective in obtaining minimum error, but fast convergence may not be possible.The conjugate gradient algorithm is used for performing search along conjugate directions. This leads to faster convergence as compared to backpropagation algorithm.

GMDH is a polynomial self-organizing network. The connections between the neurons and the number of layers in the GMDH network is not fixed and is optimized to achieve the best accuracy without overfitting. THE GMDH performs inductive procedure, which selects the best among complicated polynomial nodes by means of an external criterion. The process of finding the best solution in GMDH involves analyzing various subsets of the base function (also called partial models). There is a gradual increase in the number of these partial models to find a structure whose complexity is optimal and which satisfies the minimum value of the external criteria. This process is called self-organization of models. The parameters used for GMDH are 20 as the upper limit of network layers and a maximum polynomial order of 16.

### 2.6.4 Decision Trees

DT predict the label of instances by cataloging the instances down the tree, from the root node to the leaf node. An instance is traversed down the tree on the basis of values of its features (predictors). The leaf node of a decision tree provides the classification label of the instance. A DT identifies the most significant predictor that provides the best split on the basis of the training instances. Different decision trees vary on the basis of the splitting criteria they use. DT build accurate and efficient models which are easily interpretable. The DT can be easily converted into rules. This thesis uses two DT: C4.5 and CART. The splitting criteria used by C4.5 is gain ratio and the splitting criteria used by CART is Gini index.

The parameter settings for C4.5 decision tree method include confidence factor of 0.25, two instances per leaf and prune factor as true. The parameter settings for CART involves using a maximum tree depth of 90.

### 2.6.5 Ensemble Learners

Ensemble learners use an aggregation of multiple models to output correct and robust responses. This thesis uses four ML ensembles: BG, AB, LB and RF. All four techniques create several modified training samples from the original training data to create several models. The final output is produced after aggregating the results of all the created individual models.

BG, also known as bootstrap aggregating is a technique which constantly improves the developed classification models by creating a number of versions of the training set. Breiman [89] proposed creating bootstrap duplicates of the training data. The training sets are created with replacement. Therefore, a new function is trained for each one of the training set. In order to predict a class, the result of majority is output. The parameter settings for BG technique include Reduces Error Pruning (REP) tree as the classifier, a bag size percent of 100 and 10 iterations.

AB is an ensemble technique which uses the concept of boosting. The algorithm provides a weighted sum of various weak learners to construct an output which is of an efficient boosted classifier. The initial step involves providing equal weights to all the classifiers [88]. However, with each iteration, the weights are adjusted in such a manner that the algorithm focuses on the hard to learn examples to improve the efficiency of the boosted classifier. The parameter settings for AB used in the thesis are decision stump classifier, 10 iterations and a weight threshold of 100.

LB is another boosting method. The LB technique uses AB technique for additive model and applies the cost function of the LR technique [90]. The LB parameters

used in the thesis are 10 iterations, 100 as weight threshold, decision stump classifier and a likelihood threshold of $-1.79$.

RF is a collection of decision trees at the training stage [91]. It predicts the label of a particular instance by taking the mode of all the individual decision trees which are created randomly. This property helps in overcoming the overfitting characteristic of a DT. We used a RF of 100 decision trees in this work.

### 2.6.6 Naive Bayes

NB is a probabilistic classifier which captures the relationship between the predictors and the target variable. It is based on the assumption that various predictors are conditionally independent and uses Bayes theorem [92]. If X is the set of predictor variables and Y is the target variable, Bayes theorem can be defined by the following formula:

$$P(Y|X) = \frac{P(X|Y)P(Y)}{P(X)} \tag{2.7}$$

Here, $P(X|Y)$ denotes the posterior probability or class conditional probability. It represents the probability that X predictors are observed given that the target class is Y. $P(Y)$ is the prior probability of Y, which denotes any previous knowledge about the chance that class Y is observed. $P(X)$ represents the evidence i.e. the given set of predictors. During the training phase of NB, one needs to learn all posterior probabilities for all possible combinations of predictors (X) and target (Y). This information is extracted from the training data. With the aid of these posterior probabilities, an instance with unknown label may be classified by finding the target class Y', which would maximize the posterior probability $P(Y'|X')$. For example, if we are given a test record with binary dependent variable: "change-prone" and "not change-prone", we compute the posterior probabilities of $P(\text{“}Change - prone\text{”}|X)$

and $P(\text{“}Not\ Change - prone\text{”}|X)$, based on the information extracted from training data. In case the value of $P(\text{“}Change - prone\text{”}|X)$ is greater than the value of $P(\text{“}Not\ Change - prone\text{”}|X)$, the record is classified as “Change-prone” or else as “Not Change-prone”. The parameter settings used for NB techniques is kernel estimator and supervised discretization.

### 2.6.7 Support Vector Machine

SVM is a classifier used for predicting the label of binary target variable. Each data point is represented by an n-dimensional vector. A data point could belong to only one of the possible target class. A linear SVM separates the two categories of data points with the help of a hyperplane [93]. However, it may be noted that there may be many hyperplanes that correctly segregate a given group of training data points. In such a scenario, the SVM technique chooses the hyperplane which achieves the maximum separation by selecting the hyperplane which has the largest margin. The “margin” is defined as the summation of the shortest distance from the separating hyperplane to the nearest data point of both categories. Such a hyperplane is chosen as it is likely to generalize better and correctly predict “unseen” data points (test data points).

It may be the case that the data points may not be “linearly-separable”. In such a scenario, kernel function may be used to separate non-linear data. The kernel function aids the transformation of data into a higher dimensional space to make the separation easy. This thesis uses the polynomial kernel function for separating data. The parameter settings for SVM method include a $c$ value of 100, an $\epsilon$ value of 0.001, degree of 1, $\gamma$ value of 0.01 and $v$ value of 0.1.

### 2.6.8 Constricted Particle Swarm Optimization

CPSO is a variant of the Particle Swarm Optimization (PSO) algorithm. PSO is based on the premises that an intelligent optimization solution can be achieved by collective behavior without any centralized control [94]. Thus, PSO looks for solutions in a distributed manner.

A PSO algorithm needs to keep track of three global conditions, i.e., a) the target or the function, which needs to be optimized; b) the global best ($g_{best}$), which represents the best value obtained by any particle in the solution space so far; and c) the stopping criteria, which states the number of iterations after which the algorithm should stop, if the target value could not be attained. Also, each particle in the PSO algorithm would contain a) the required data, which is representative of a possible solution; b) the velocity value, which indicates the extent to which the data can be modified; and c) the best solution i.e. the best value of the particle obtained so far ($p_{best}$). Thus, a PSO algorithm surveys the search space looking for the most favorable region.

The velocity ($v_t$) of each particle at time $t$ is updated in each iteration in accordance with $p_{best}$, $g_{best}$, current position ($x_{t-1}$) and current velocity ($v_{t-1}$). However, it is important to avoid search space explosion while looking for the optimum solution. Clerc and Kennedy [95] suggested that the use of proper constriction coefficients in order to do so. Thus, proper use of constriction coefficients ($\delta$) of the PSO algorithm, i.e. the use of CPSO can help in controlling the exploration versus exploitation bias. Apart from the constriction coefficient, the algorithm uses two other parameters: cognitive parameter ($c_1$) and social parameter ($c_2$). The cognitive parameter is responsible for updating a particle's position in accordance with its local best, while the social parameter is responsible for updating a particle's position in accordance with the global best. The following equations are representative of the update in

velocity of a particle and the position of the particle:

$$v_t = \delta(v_{t-1} + c_1(p_{best} - x_{t-1}) + c_2(g_{best} - x_{t-1})) \tag{2.8}$$

$$x_t = x_{t-1} + v_t \tag{2.9}$$

This work uses the following CPSO parameters for model development: 25 particles, 0.1 as convergence radius, 2.05 as maximum weights for $c_1$ and $c_2$, a maximum of 0.1 uncovered instances, 0.73 as $\delta$, 0.1 as the threshold for indifference and a convergence platform of width 30. The fitness function used by the CPSO classifier is the product of sensitivity and specificity. These are performance measures which are explained in Section 2.10.

### 2.6.9 Genetic Algorithm based Classifier System

Genetic Algorithm Based Classifier System (GAssist) uses GA for evolution of individuals which are representatives of the complete solution [96]. A chromosome in a GA is representative of a classification rule, which is codified in the form of a bit array. The genes of a chromosome are representative of attributes.

In order to represent real valued attributes, one needs to perform discretization using proper number of intervals. Hence, the bits represent a discretization interval. The traditional method used in GA is a set of rules where the antecedent is defined by a prefixed finite number of intervals to handle real-valued attributes. Hence, the capability of these systems is dependent on the right selection of the intervals. Therefore, we require a rule representation with adaptive discrete intervals. In GA-ADI, the intervals are split and merged throughout the evolution process. The fitness evaluation of individuals is done according to the proportion of correctly classified instances. On the contrary, GA-Int uses intervalar rules.

The parameters used for GA-ADI and GA-Int in this work include 500 iterations, 4,5,6,7,8,10,15,20,25 as intervals for uniform discretization, 2 strata, 12 minimum rule deletions and 4 as size penalty of minimum rules. The fitness function for both the algorithms was the square of accuracy. The accuracy performance measure is defined in Section 2.10.

### 2.6.10 Hierarchical Decision Rules

HIDER performs multiple runs of an evolutionary algorithm to formulate a set of rules. Once the rules are defined, they are used to classify new instances [97]. It may be noted that one must apply rules in the order they are obtained. Hence, HIDER produces a hierarchical set of rules. In accordance with the hierarchy, an instance whose label is unknown will be classified by the $k^{th}$ rule only when the conditions of k-1 preceding rules are not matched by the instance. The rules are obtained sequentially until the space is totally covered. When a new rule is formed, all training instances that match the antecedent of the rule are deleted. The initial rules obtained by the HIDER algorithms are the ones which cover more instances. Thereafter, the number of predictors needed to test the remaining set of rules decreases. This thesis uses the following parameter settings for HIDER: 100 as population size, 100 generations, a mutation probability of 0.5, an extreme mutation probability of 0.05, a crossover percentage of 80 and a penalty factor of 1. The fitness function used by HIDER is defined as follows:

$$Fitness\ Function = 2(Num_1 - Error(r)) + G(r) + Coverage(r) \qquad (2.10)$$

Here, $Num_1$ represents the total number of instances processed by the rule, $Error$ denotes the error produced when an instance is not in the same class but is covered

by the rule. $Coverage$ denotes the search space which is covered by a rule.

### 2.6.11 Learning Classifier Systems

A Learning Classifier System (LCS) is an adaptive system that is trained to perform the best action given a set of inputs. While "input" is the set of predictors in the classification context, "actions" corresponds to the predicted class label. An LCS consists of several "condition-action rules". For a particular set of input, an LCS may find several rules which match the current input. Some of the matched rules might be advocating a specific class label for the input, while other matched rules might advocate some other class label. Thus, LCS computes, for each possible class label, an average of the predictions of the classifiers advocating that class label, and then chooses the label with the largest average. The prediction average is weighted by its fitness. When the LCS predicts the class label with the largest average prediction, the environment returns some amount of payoff (P). This payoff might be used to alter the predictions in the training stage of LCS. Besides its prediction, each classifier maintains an estimate (Q) of the error of its predictions. The fitness of LCS is adjusted by moving it closer to the inverse of error. With each iteration, high fitness classifiers are reproduced over less accurate ones and the "offspring" are modified by genetic operators such as mutation and crossover.

XCS uses a niche genetic algorithm with reinforcement learning where an appropriate reward is given for each action by the system. The algorithm evolves as a population of classifiers, where each classifier consists of a rule and parameters for estimating the quality of the rule [98]. The GA in XCS is applied to the action sets, rather than over all the population. First, it selects two parents from the actual action set with probability proportional to fitness. Then, the parents are crossed and mutated. The resulting offspring are introduced into the population. The parameter

settings used for XCS is this thesis are 1,00,000 explores, a population size of 6,400, a crossover probability of 0.8, two-point crossover type, a mutation probability of 0.04, free mutation, roulette wheel selection, $\delta = 0.1$, $\theta_{mna} = 2$, GA subsumption as true, $\theta_{sub} = 50.0$, $\theta_{ga} = 50.0$, $\alpha = 0.1$, $\beta = 0.2$, $\theta_{del} = 50.0$, 0.4 as size of tournament, 10.0 as initial prediction, 0.0 as initial prediction error, 0.25 as reduction of prediction error, 0.01 as initial fitness value, $\mu = 10.0$, $r = 1.0$, $m = 0.1$. The fitness function used by XCS is the inverse function of the error which can be calculated according to the following formula [99]:

$$Error = 1/Reward(|Reward - Pred_{cl}| * Prob_{cl}(c) + |\theta - Pred_{cl}| * (1 - Prob_{cl}(c))) \tag{2.11}$$

Here $Pred_{cl}$ denotes the prediction of the classifier and $Prob_{cl}(c)$ denotes the probability of correct classification.

SUCS is designed for supervised environments. For SUCS, the training is performed where each training instance is associated with a class. This is different from XCS where reinforcement learning is performed. When an instance with input x is presented during training, a match set is formed which consists of those classification rules whose conditions match with the input x. All the classifiers in the match set are divided into sets $C$ or $!C$. All the classifiers in the match set which predict the correct known class for input x (provided during training) are allocated to set $C$, and all others to $!C$. In the testing phase, when an instance with input x is presented, the system needs to predict the associated class. The predicted class is computed by the weighted vote of all the classifiers in the match set. The weights are allocated in accordance with fitness. The GA in SUCS is only applied to the correct set $C$ . SUCS selects two classifiers from set $C$ with probability proportional to fitness and applies crossover and mutation [98]. The parameter settings used for SUCS are same as the parameter settings for XCS. The fitness function for SUCS is as follows, where $v$ is

a constant:

$$Fitness\ Function = (\frac{Number\ of\ Correct\ Classifications}{Number\ of\ matches\ for\ a\ rule})^{v} \tag{2.12}$$

A Pittsburg approach involves rule sets rather than individual rules. Memetic refers to a combination of population based global search (an evolutionary algorithm) along with cultural evolution in the search cycle (local refinement). MPLCS hybridizes GAssist with local search algorithms such that it edits the classical rule set wise operator so as to obtain the smallest set of rules that yield the maximum accuracy while training [100]. The rule set wise operator evaluates all the candidate rules, selects the parent rules and generates the offspring. This operator is integrated in the crossover stage of GAssist. The parameters used for MPLCS in this work are 750 iterations, 0.05 as local search probability, 0.1 as rule set-wise crossover probability, 4 as the size of penalty rules and 5 rule ordering repetitions. The fitness function for MPLCS is defined as follows:

$$Fitness\ Function = Exception\ Bits + Wt. * Theory\ bits \tag{2.13}$$

Here, $Wt.$ assigns the weight for adjustment of exception and theory bits, $Theorybits$ symbolize the length of all classification rules which are alive and $Exception\ bits$ symbolize all examples which are wrongly classified or not classified at all.

### 2.6.12 Gene Expression Programming

According to Candida Ferreira, the GEP algorithm is similar to GA in terms of finding an efficient solution in a vast search space of candidate solutions [101]. Both the algorithms constantly improve their solutions with the help of various operators like mutation, selection, recombination etc. However, there is a dissimilarity in the rep-

resentation of the candidate solutions for both the algorithms, while both the GA as well as the GEP algorithm use chromosomes whose length is fixed throughout, in the GEP algorithm the chromosomes are later represented as optimal expression trees [101]. Ferreira adapted this system in order to incorporate a diverse range of mutation, crossbreeding, transposition and recombination operators whose application does not inhibit efficient translation of the chromosome into an accurate expression tree. Thus, the advantage of the GEP algorithm is two-fold, a) at the initial stage chromosomes are simple, linear and small objects which are easily handled while performing various transitions and operations b) performing any operation on the chromosome would lead to an accurate and precise expression tree [101]. These specific traits make the GEP algorithm highly adaptable, efficient and fast.

A gene in the GEP algorithm consists of two parts, head and tail. Figure 2.2 demonstrates a GEP gene for the expression ((m + n) / (o * r)). The gene expression tree can be encoded in Karva language by listing the nodes from the highest level to the lowest level and from left to right [101]. The gene in Figure 2.2 can be encoded as "/+*mnor", where "/+*" is the head part and "mnor" is the tail part. A GEP chromosome is composed of one or more genes of equal length. A linking function is used to combine genes if required.

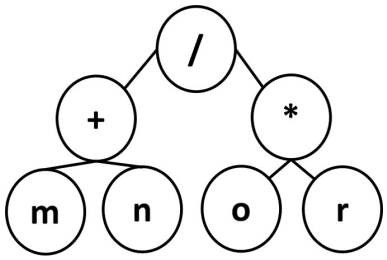


Figure 2.2: GEP Gene

The GEP algorithm starts with an arbitrary generation of the chromosomes. Each chromosome of the population is evaluated by ascertaining its fitness function. The

best candidates are selected according to their fitness by performing the roulette wheel selection. After selecting the good chromosomes, we perform reproduction by applying the GEP operators (replication, mutation, transposition and recombination) to produce new candidates with better traits. Thereafter, we prepare the chromosomes for the next generation and repeat the steps until a fixed number of iterations or until we get the best solution from the population.

This thesis uses GEP with the following parameters: a population size of 50, 4 genes per chromosome, a gene head length of 8, 500 simplification generations, addition as linking function, maximum generations of 2000, 0.04 as mutation rate, 0.1 as transposition rate, 0.3 as recombination rate and a fitness function of "number of correct hits with penalty".

### 2.6.13 Decision Trees with Genetic Algorithm

DT algorithms have a bias towards generality i.e. they are well suited for formulating rules that cover a large number of instances (large disjunct), but does not form appropriate rules that are suitable for covering small number of instances (small disjunct). On the other hand, GA are robust and flexible search algorithms which tend to cope with attribute interaction better than most rule induction algorithms. GA do not get trapped in local minima and are more suitable for finding rules that cover small number of instances.

DT-GA is a hybridized technique that combines the advantages of the base algorithms. Although each rule that covers small number of instances covers just a few examples, the set of all such rules may cover a large number of examples. Hence, it is important that both types of rules, those which cover large instances and those which cover small instances should not be ignored.

DT-GA performs training in two phases. In the first phase, C4.5 algorithm is run

and the resultant tree is converted into a set of rules. The second phase uses a GA to discover rules covering small number of instances [102]. This thesis uses DT-GA parameter settings as 2 instances per leaf, a confidence of 0.20, 10 as the threshold for considering small disjuncts, 50 generations for GA, 200 chromosomes in the population, a crossover probability of 0.7 and a mutation probability of 0.01. The fitness function is the product of Sensitivity and Specificity performance measures which are explained in Section 2.10.

### 2.6.14 Particle Swarm Optimization with Linear Discriminant Analysis

PSO-LDA is a hybridized SBA, which uses PSO for feature selection and LDA for model development. The reason behind such a hybridization is that LDA often suffers from the small sample size problem when the number of dimensions of the data is much greater than the number of data points. PSO is used to select the beneficial features and to enhance the classification accuracy of LDA. PSO is a powerful meta-heuristic technique which is capable of dealing with two conflicting objectives, i.e. maximizing the classification performance and minimizing the number of features. Hence it is capable of efficient feature selection.

The architecture of PSO-LDA first involves data pre-processing. Normalization is performed and the range of each predictor is scaled to [0,1]. In case, an instance has missing values, it is removed. Thereafter, each particle in the PSO algorithm is representative of the solution i.e. the selected subset of predictors. The training dataset along with the selected predictors is used in building the LDA classifier model. After training the discriminant functions with the help of training data, the classification accuracy is computed using the testing data. The local best and global best is computed for each particle on the basis of fitness values. In case the termination criteria

are fulfilled, the algorithm stops otherwise the next iteration continues.

The parameters used for PSO-LDA are 400 as the upper limit of iterations, 150 non-improving iterations, 0.8 as cognition learning factor, 1.2 as social learning factor, an inertia weight of 0.5 and 15 particles. The fitness function is the accuracy performance measure, which is discussed in section 2.10.

### 2.6.15 Genetic Fuzzy System LogitBoost

GFS-LB is a hybridized technique which uses Genetic Fuzzy Rule Learning for generating classification rules and the LB technique for combining the set of weak rules into a strong classifier. Fuzzy rule base can be compared to a weighted combination of weak hypotheses. LB combines low quality classifiers with a voting scheme to produce a classifier better than any of its components.

A fuzzy rule assigns a class label to an instance with a specific confidence. Given a particular set of predictors, a fuzzy rule outputs the class label and a number representing the degree of confidence of the classification. While boosting fuzzy rules, a single fuzzy rule can be fit on a set of weighted examples. The algorithm is repetitively performed for each rule in the base. The LB algorithm computes the number of votes each rule is assigned and recomputes the weight of each instance if a new rule is added to the base.

The Fuzzy classifier rules are developed using genetic learning in three stages. The first stage is called “fuzzy rule generation” in which a population of several candidate fuzzy rules evolve [103]. These rules correctly classify the training instances. Out of these rules, the one with the largest covering degree is added to the intermediate rule base. In the second stage, all the training examples are re-weighted in accordance with how well they are classified according to the new rule. The process is repeated in iterations until the desired number of rules are obtained or a specific

classification accuracy has been obtained.

The parameters used for GFS-LB in this thesis are 5 labels and 25 rules in the base. The fitness of a fuzzy rule is computed as the squared error between the desired output and the logistic transform of the classifier's output. A detailed description is provided by [103].

### 2.6.16 Neural Net Evolutionary Programming

NNEP is a hybrid technique which uses an evolutionary algorithm to design the structure and optimize the weights of a product unit. A Product Unit Neural Network (PUNN) is a feed-forward neural network which computes the weighted product, where each input is raised to a power determined by a variable weight [104]. Such units can learn polynomial terms. However, the error surface of PUNN is extremely convoluted as small changes in exponents might result in large changes in the total error surface. Hence, training of such networks becomes difficult.

Classical neural network training algorithms assume a fixed architecture. However, it is very difficult to know beforehand what the most suitable structure of the network for a given problem will be. An evolutionary algorithm can be used to design a nearly optimal neural network architecture of PUNN's because error surface associated with neural networks has numerous local optima and plateaus. This justifies the use of an evolutionary algorithm to design the topology of the network and to train its corresponding weights [105]. The evolutionary process determines the number of basis functions, associated coefficients and corresponding exponents in the model for PUNN's. The evolutionary algorithm begins a search with the initial population of PUNNs. In each iteration, the population is subject to population-update along with the use of replication and mutation operators.

The parameters used by NNEP are 6 hidden nodes and 200 generations. The

fitness function used by NNEP is as follows, where $l(\theta)$ is the Hessian matrix for error function:

$$Fitness\ Function = \frac{1}{1 + l(\theta)} \tag{2.14}$$

### 2.6.17 Fitness-based Ensembles

SBA search for an optimum solution with the help of a fitness function. These functions dictate the search process of an SBA. However, literature studies [46–49] have ascertained that a variation in fitness function varies the result of the developed model. As ensemble learning techniques are robust and effective, we propose an ensemble of different fitness-variants of a search-based algorithm. The detailed description of the proposed fitness-based ensembles is provided in Chapters 7 and 8.

## 2.7 Empirical Data Collection

Data for empirical validation may be collected from industrial software, open-source software or academic systems. Over the last few years, there has been a paradigm shift, where the most common software datasets used for empirical validation have been open-source datasets. The primary reasons for their popularity are ease of availability, cost-effectiveness, abundant support and ease of customization. This thesis evaluates several open-source datasets, whose source code could be easily downloaded. We first describe the process of data collection and list the datasets used in the thesis along with their characteristics.

We followed four main steps for data collection which are depicted in Figure 2.3. In order to collect data, we need two releases of a software, a previous release and a recent release. The source code of the releases was downloaded from either `http://googlesource.com` or `http://sourceforge.net`. The four steps are

explained in detail below:

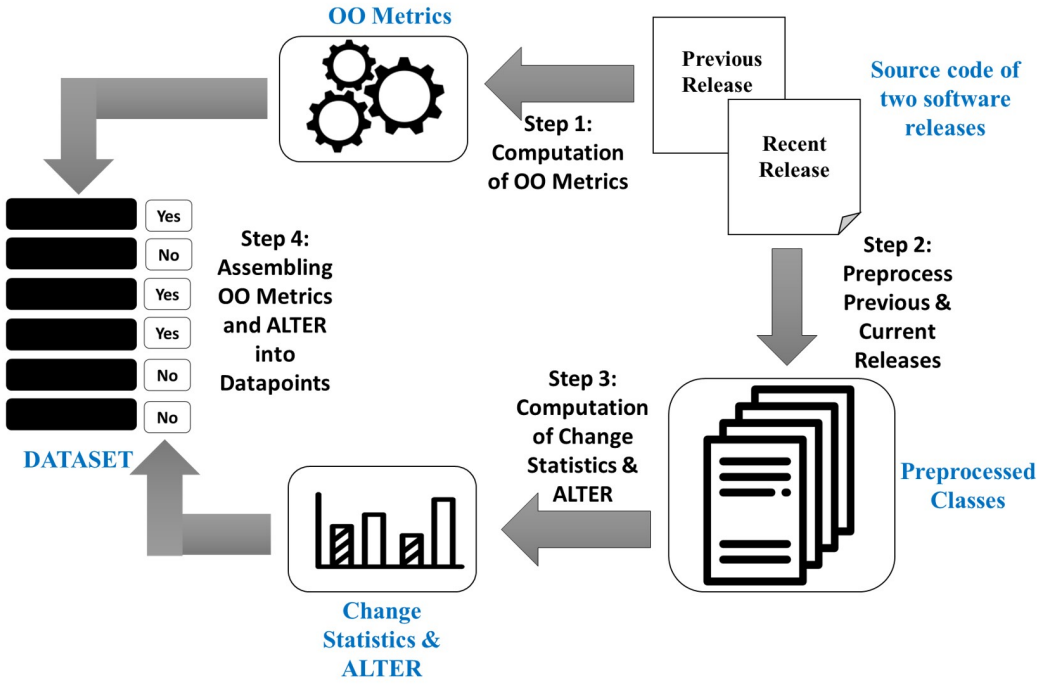


Figure 2.3: Procedure for Data Collection

1. *Computation of OO Metrics*: OO metrics are extracted from the previous release of the software. As already discussed, these metrics are representative of various OO characteristics like size, cohesion, inheritance etc. The metrics are computed with the help of two basic tools, i.e. either Understand for Java or Chidamber and Kemerer Java Metrics (CKJM) tool. The Understand for java tool is available at `https://scitools.com` and the CKJM tool can be downloaded from `http://gromit.iiar.pwr.wroc.pl/p_inf/ckjm/metric.html`. It may be noted that while the Understand for Java tool computes metrics from source code, the CKJM computes metrics from bytecode of compiled Java files. The tools may give metrics at method level or for anonymous classes. However, we remove these metrics to obtain

only "class level" metrics.

2. *Preprocess Previous and Recent Releases*: In this step, we preprocess both the releases of the software to extract the common classes in both releases. All classes which were present in the previous release but were deleted in the recent release and all the classes which were newly added in the recent release and were not present in the previous release are removed. Each common class extracted in this step will represent a data point.

3. *Computation of Change Statistics and ALTER*: For each common class extracted in the previous step, we compute change statistics. Change statistics include the SLOC inserted, SLOC deleted and SLOC modified in a class when it progressed from previous release to recent release. We also compute the total SLOC changes according to the following rules [5, 29]:

   - Each inserted SLOC or deleted SLOC is counted as one SLOC change.
   - Each modified SLOC is counted as two SLOC changes, a deletion followed by an insertion.

   After computing the total SLOC changes, we derive the ALTER variable. It is a binary variable with two outcomes: "Yes" and "No". An ALTER variable is designated as "Yes", if the total SLOC changes are greater than zero, otherwise it is designated as "No". In other words, an ALTER variable as "Yes" signifies a "change-prone" class and an ALTER variable as "No" signifies a "not change-prone" class. The computation of change statistics is done with two different methods, which are stated as follows:

   - With the aid of Defect Collection and Reporting System (DCRS) Tool [106], which was developed by undergraduate students of Delhi Technological University. The tool was developed in Java language and is

used for collecting data from open-source repositories which use GIT as the version control system. The DCRS tool analyzes the source code of an open-source software by extracting change-logs. A change record contains change information such as an identifier, commit timestamp, description of the change and a file listing of all the files which are modified including the lines of changed code. Change statistics are extracted from change logs. It may be noted that the DCRS tool also incorporates CKJM tool for OO metrics computation.

- With the aid of "diff" command for differentiating between source files and computing the change statistics. Common class files i.e. the ones which have the same name in both the releases are compared with "diff" command. As a result of the command change statistics are returned. In order to use "diff" command, Configuration Management System (CMS) tool was used [107]. It may be noted that if change statistics were computed using CMS tool, we used Understand for Java tool for OO metrics computation.

4. *Assembling OO metrics and ALTER into Data points*: In the final step, we combine the OO metrics computed in Step 1 and the ALTER variable computed in Step 3 to formulate data points for each common class. A data point is the culmination of OO metrics and change statistics for a specific class.

This thesis uses several open-source software datasets, which belong to varied domains. The primary objective for selecting datasets from varied domains was to obtain generalized results. Moreover, the open-source nature of the datasets increases the replicability of the results. Appendix A.1 states the various datasets along with the analyzed versions, the number of classes and the percentage of changed classes in each dataset. It also states the function of software datasets. It may be noted

that three software datasets (Celestia, Glest and Simutrans) were developed in C++ language, while all other datasets were developed in Java programming language.

## 2.8 Data Preprocessing

We first analyze the descriptive statistics of various metrics of each dataset. Next, we identify the outliers in each corresponding dataset. It is important for effective model development to remove all the outliers from the data. Also, it is important to eliminate noisy and redundant features to develop a good prediction model using a feature selection method.

### 2.8.1 Descriptive Statistics

We report the descriptive statistics of each dataset corresponding to each OO metric. This step helps in evaluating the characteristics of each dataset. We report the following descriptive statistics:

- *Minimum (Min.)*: This statistic reports the minimum value obtained by any class for a specific metric.
- *Maximum (Max.)*: This statistic reports the maximum value obtained by any class for a specific metric.
- *Mean*: This statistic reports the average value obtained by all the classes in a particular dataset for a specific metric. It is a measure of central tendency which is computed by dividing the total value of all the elements in the dataset by the total number of elements in the dataset.
- *Median*: This statistic reports the midpoint of the frequency distribution of all

the classes in a particular dataset for a specific metric. It is a better representative of central tendency than mean, if the dataset has an outlier.

- *Standard Deviation (SD)*: It is a measure of dispersion in the dataset. A low value for standard deviation indicates that data points lie close to the mean. However, a higher value of standard deviation indicates that the data points are dispersed and spread out.

The datasets analyzed in the thesis (mentioned in Appendix A.1) are grouped into three categories ("small", "medium" and "large") according to the number of data points they contain. Datasets containing up to 200 data points are allocated to "small-sized" category, those having data points in the range 201-500 are allocated as "medium-sized", while those having data points greater than 500 are designated as "large-sized" datasets. We summarize the descriptive statistics observed for each category of datasets by stating the range of values for each statistic in Appendix A.2 (Table A.2-A.4). Thereafter, we summarize the following observations after analyzing the descriptive statistics:

- The class size of different datasets ranged from 2-1,972 SLOC for small-sized datasets, 1-8,8858 SLOC for medium-sized datasets, 1-6,764 SLOC for large-sized datasets.

- The median DIT and NOC values indicate that the inheritance attribute was not much used in the software systems (Median DIT: 0.00-2.00 (for all datasets), Median NOC: 0.00 (small-sized), 0.0-4.0 (medium-sized) and 0.0-3.0 (large-sized)).

- The maximum LCOM values in the datasets was high (upto 11,628 for small-sized, upto 31,375 for medium-sized, upto 8,128 for large-sized) indicating low cohesion in software datasets.

### 2.8.2 Outlier Analysis

According to Barnett and Lewis [108], "an outlying observation, or outlier, is one that appears to deviate markedly from other members of the sample in which it occurs". A critical step in data preprocessing involves outlier analysis. In order to provide unbiased results, we effectively identified and removed all the outliers from each dataset respectively. The outliers were identified using Inter Quartile Range (IQR) filter of the WEKA tool [88], which is based on the IQR metric. The IQR metric is computed as the difference between the upper quartile ($Q_3$) and the lower quartile ($Q_1$). A data point will be considered an outlier, if any of its independent variable value is an outlier i.e. if the value of any of the OO metric mentioned is an outlier. The computation of IQR metric and outliers is defined as follows:

$$IQR = Q_3 - Q_1 \tag{2.15}$$

An OO class is considered an outlier if any one of the following conditions hold for any of the class's independent variable ($x's$). Here, Extreme Values Factor (EVF) is 6.0 and Outlier Factor (OF) is 3.0, the default values in WEKA tool.

$$Q_3 + OF * IQR < x \leq Q_3 + EVF * IQR \tag{2.16}$$

$$Q_1 - EVF * IQR \leq x < Q_1 - OF * IQR \tag{2.17}$$

### 2.8.3 Correlation based Feature Selection

The final step of this phase involves use of Correlation based Feature Selection (CFS) method in order to eliminate noisy and redundant independent variables from each dataset. According to Hall [109], the CFS method helps in identifying those independent variables which are highly correlated with the change (i.e. the dependent

variable) in the class but not amongst each other. The set of independent variables extracted after application of the CFS method are important as they help in reducing the dimensionality of datasets. The CFS method evaluates an exhaustive possible combination of OO metrics, to select the best subset of OO metrics for change prediction in each corresponding dataset. It may be noted that there is no single best technique for selecting features in a dataset [110, 111]. A study by Hall and Holmes [111], evaluated six attribute selection methods on 15 datasets and confirmed that though there is no best method for attribute selection, the CFS method shows good results. Furthermore, an extensive review conducted by Malhotra [17] of 64 defect prediction studies from 1991 to 2013 points out that CFS was the most commonly used feature selection in those studies. Since a large number of studies in literature have successfully applied the CFS method for feature selection [5, 37, 112] while developing predictive models using ML techniques, we use this method.

## 2.9 Model Development and Validation

This thesis develops software change prediction models in order to identify change-prone nature of classes in future and yet unseen releases of the software. The prediction models learn from historical data of a project to identify which classes are likely to change in future releases. The models are developed using supervised learning and use various data analysis techniques as discussed in Section 2.6. The training data for the model consists of both the independent variables and the class labels ("change-prone" or "not change-prone"). The data analysis technique helps in learning the model. After the model is constructed, it is validated. The validation data is such that only the value of independent variables are provided to the model and the model is supposed to predict a label. The predicted label is matched with the actual label to ascertain the performance of the model. There are various ways to validate

a predicted model. The validation methods used in the thesis are described in the subsections below.

### 2.9.1 Ten-fold Cross Validation

This is a type of within project validation as the training data as well as the testing data is derived from within the project. This validation method functions by performing division of all the data points of a specific dataset into ten subsets [113]. The method then performs ten iterations by using nine subsets for training and the tenth subset for validation until we use each of the ten subsets once for validation purpose. We use ten-fold cross validation technique as it reduces validation bias [37, 114]. Figure 2.4 depicts the ten-fold cross-validation method.

### 2.9.2 Inter-release Validation & Cross-project Validation

Inter-release validation and Cross-project validation are termed as external project validation as the data used for training is obtained from a different release/ software and the data used for testing is obtained from some other release or some other software project. For developing models using inter-release validation, we train the model using a specific release of the dataset and validate the model on another release of the same dataset. It should be noted that outliers are only removed from the training datasets while developing inter-release validation models. In case of cross-project validation, the training data belongs to a specific dataset, while the testing data is extracted from altogether a different project. This approach is cost-effective as it leads to availability of large validation data where validation data specific to the project may be limited or scarce in nature [115, 116]. Figure 2.5a depicts the process of inter-release validation while Figure 2.5b depicts cross-project validation.

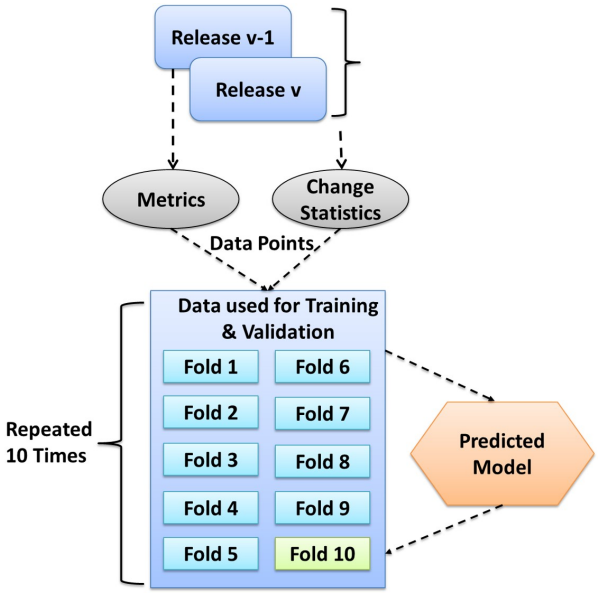


Figure 2.4: Ten-fold Cross Validation

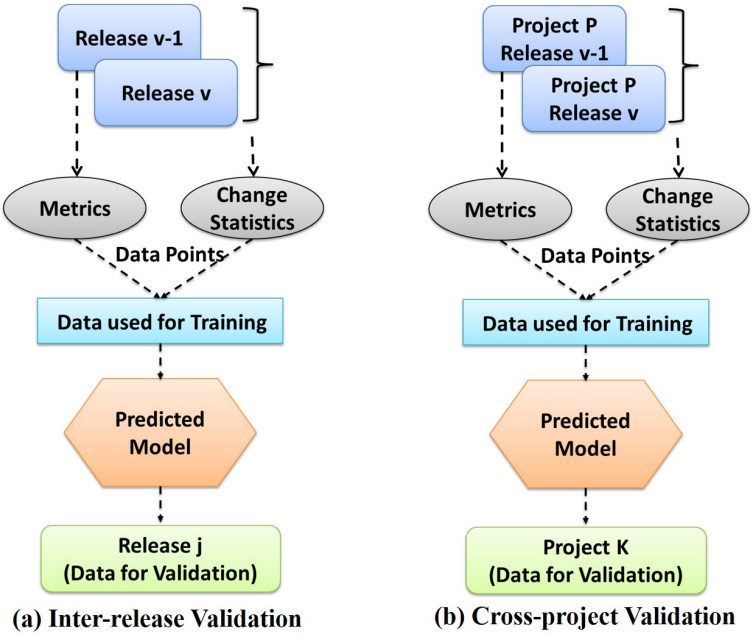


Figure 2.5: External Validation

## 2.10 Performance Measures

The performance of prediction models with binary outcomes can be evaluated with the help of 2 X 2 confusion matrix shown in Table 2.4.

Table 2.4: Confusion Matrix

| | | **Predicted** | |
|---|---|---|---|
| | | **Change-prone** | **Not Change-prone** |
| **Observed** | **Change-prone** | True Positive (TP) | False Negative (FN) |
| | **Not Change-prone** | False Positive (FP) | True Negative (TN) |

Various performance measures can be defined in terms of this binary variable confusion matrix. The matrix has four terms: True Positives (TP) that determine the count of change-prone classes which are correctly identified by the predictor; True Negatives (TN) that determine the count of not change-prone classes which are correctly identified by the predictor; False Positives (FP) that account for incorrectly identified change-prone classes that are actually not change-prone in nature and False Negatives (FN) that account for incorrectly identified not change-prone classes that are actually change-prone in nature.

The various performance measures used in this thesis are defined as follows:

- *Accuracy*: It is defined as the percentage of correct predictions of change-prone

and not change-prone classes.

$$Accuracy = \frac{TP + TN}{TP + FP + FN + TN} * 100 \tag{2.18}$$

- *Sensitivity*: It is defined as the percentage of correctly predicted change-prone classes amongst actual change-prone classes. It is also commonly known as Recall or True Positive Rate (TPR) or Probability of Detection (PD).

$$Sensitivity = \frac{TP}{TP + FN} * 100 \tag{2.19}$$

- *Specificity*: It is defined as the percentage of correctly predicted not change-prone classes amongst actual not change-prone classes. It is also commonly known as True Negative Rate.

$$Specificity = \frac{TN}{TN + FP} * 100 \tag{2.20}$$

- *Precision*: It is defined as the percentage of correct change-prone classes amongst total predicted change-prone classes.

$$Precision = \frac{TP}{TP + FP} * 100 \tag{2.21}$$

- *False Positive Rate*: It is defined as the percentage of not change-prone classes that are incorrectly predicted as change-prone amongst actual not change-prone classes. It is also commonly known as Probability of False Alarm (PF).

$$PF = \frac{FP}{TN + FP} * 100 \tag{2.22}$$

- *G-Mean1*: It is defined as the geometric mean of sensitivity and specificity .

$$G - Mean1 = \sqrt{\frac{TP}{TP + FN} * \frac{TN}{TN + FP}} \tag{2.23}$$

- *G-Mean2*: It is defined as the geometric mean of sensitivity and precision .

$$G - Mean2 = \sqrt{\frac{TP}{TP + FN} * \frac{TP}{TP + FP}} \tag{2.24}$$

- *G-Mean3*: It is defined as the geometric mean of positive accuracy (Precision) as well as negative accuracy (Negative Predictive Value).

$$G - Mean3 = \sqrt{\frac{TP}{TP + FP} * \frac{TN}{TN + FN}} \tag{2.25}$$

- *F-measure*: It is defined as the harmonic mean of sensitivity and precision.

$$F - measure = 2 * \frac{Sensitivity * Precision}{Sensitivity + Precision} \tag{2.26}$$

- *G-measure*: It is defined as the harmonic mean of sensitivity and 100-PF.

$$G - measure = 2 * \frac{Sensitivity * (100 - PF)}{Sensitivity + (100 - PF)} \tag{2.27}$$

- *Balance*: It depicts the Euclidean distance between a pair of (Sensitivity, PF) to that of an optimal value of Sensitivity =1 and PF=0.

$$Balance = 1 - \sqrt{\frac{(0 - (\frac{PF}{100})^2)(1 - (\frac{Sensitivity}{100})^2)}{2}} \tag{2.28}$$

- *AUC*: Receiver Operating Characteristic (ROC) represents a plot between 1-specificity values on the x-axis and recall values on the y-axis. A model is considered favorable if it has higher area under the ROC curve. AUC is robust in handling skewness in class distributions and the unequal cost of misclassification errors [117].

A good change prediction model will have higher values for each of the discussed performance measures. Previous studies in literature have criticized the use of traditional performance evaluators like accuracy and precision when evaluating models using imbalanced datasets [110, 118, 119] . However, a number of studies in literature have discussed the favorability of using other robust performance measures for assessing models developed using imbalanced data. Studies by Harman et al. [38], Kubat and Matwin [120] and He and Garcia [118] support the use of G-Mean1 as a performance measure for imbalanced datasets. Certain other studies by Menzies et al. [121] and Li et al. [122] advocate Balance performance measure as an efficient evaluator. Moreover, studies by Lessmann et al. [123], Shatnawi [57] and He and Garcia [118] propose the use of ROC analysis, where AUC can be used as an effective indicator for evaluating models developed using imbalanced data.

## 2.11 Statistical Analysis of Results

In order to assess and validate various hypothesis of the thesis, we use Friedman statistical test which is followed by Wilcoxon signed rank post-hoc test. These tests are non-parametric and thus can be effectively used without much inconvenience as they do not depend on a number of assumptions of underlying data such as data normality, homogeneity of variances etc., which are mandatory for parametric tests [124]. Moreover, as reported by Lessmann et al. [123], very few studies evaluate the

results statistically while comparing various developed models. Thus, it is important to conduct statistical analysis to strengthen the conclusion validity of the study.

### 2.11.1 Friedman Test

Friedman test is used to rank the performance of k techniques over multiple datasets [125]. It is based on the assumption that the performance measures of techniques computed over different datasets are independent of each other. The Friedman test hypothesis can be stated as follows:

- *Null Hypothesis ($H_0$)*: The performance of different techniques is not statistically different from each other.
- *Alternate Hypothesis ($H_a$)*: The performance of different techniques is significantly different from each other.

The Friedman test is based on chi-square statistic ($\chi^2$), which can be computed as follows:

- *Step 1*: For a specific dataset, sort the performance values of all the techniques in descending order. Allocate ranks to each technique on the basis of performance on the specific dataset. Rank '1' is designated to a technique with the best performance and rank 'k' is designated to the technique with worst performance. In case two techniques have equivalent performance on the dataset, assign average of the ranks that would have been assigned to the techniques.
- *Step 2*: Compute the total of ranks allocated to each technique on all the datasets. The total ranks allocated to each technique is denoted by $R_1, R_2, R_3, .....R_k$.

- *Step 3*: Compute $\chi^2$ statistic according to the following formula:

$$\chi^2 = \frac{12}{nk(k+1)} \sum_{i=1}^{k} R_i^2 - 3n(k+1) \tag{2.29}$$

Here, $R_i$ is the rank total of $i^{th}$ technique and $n$ is the number of total datasets. The degrees of freedom of Friedman test is $k-1$. If the computed Friedman statistic is in the critical region, we reject the null hypothesis and conclude that the performance of different techniques is significantly different from each other. We compute Friedman test statistic at $\alpha = 0.05$.

### 2.11.2 Wilcoxon Signed Rank Test

This test is used either as a post-hoc test after the results of Friedman test are found significant or as an independent test to compare the pairwise performance of two techniques. The test is applicable only in the case when two different techniques are evaluated on the same set of datasets [126]. The Wilcoxon test hypothesis can be stated as follows:

- *Null Hypothesis ($H_0$)*: The performance of the two compared techniques is not statistically different from each other.
- *Alternate Hypothesis ($H_a$)*: The performance of the two compared techniques is significantly different from each other.

In order to conduct the test, we first compute the differences among the related pair of values of both the techniques. The resulting differences are ranked on their absolute values. The ranks are allocated in accordance with the following rules.

- Remove the pairs where the difference amongst both the techniques is zero. The number of reduced pairs is denoted as $n_r$.

- Assign a rank of '1' to the smallest absolute difference and so on to all the $n_r$ pairs.
- In case of a tie, an average of tied ranks is allocated.

Next, two variables $R^+$ and $R^-$ are computed. $R^+$ is computed as the sum of ranks where the difference is positive (i.e. the first technique outperforms the second technique), while $R^-$ is computed as the sum of ranks where the difference is negative (i.e. the second technique outperforms the first technique). Next, we compute the Z statistic as follows:

$$Z = \frac{Q - \frac{1}{4}n_r(n_r+1)}{\sqrt{\frac{1}{24}n_r(n_r+1)(2n_r+1)}} \tag{2.30}$$

Here Q is the minimum of $R^+$ and $R^-$. If the Z statistic is in the critical region with a specific level of significance, then we reject the null hypothesis. This means that the performance of the two compared techniques are significantly different from each other. We compute Wilcoxon test statistic (Z) at $\alpha = 0.05$. The Wilcoxon test was performed with Bonferroni correction to remove family-wise error. With Bonferroni correction, a p-value is considered significant only if it is less than $c$ ($\alpha$ value divided by the total number of comparisons performed).

In order to evaluate the practical significance of obtained results, we also report the effect size of the Wilcoxon test for significant cases. The effect size was computed according to the following formula [127]:

$$Effect\ Size = \frac{Z}{\sqrt{2 * Number\ of\ Matched\ Pairs}} \tag{2.31}$$

As indicated by Cohen [128], an effect size value of 0.1 is considered small, 0.3 is considered medium and 0.5 is assumed to be large.

# Chapter 3

# Software Change Prediction: A Systematic Review

## 3.1 Introduction

Change is crucial in any software to modify and upgrade it according to changing requirements and technological advancements. It is important for software practitioners to analyze the impact of a proposed change or predict change-prone classes in order to efficiently plan resource allocation during testing and maintenance phases of a software. Moreover, correct identification of change-prone classes in the early phases of software development life cycle helps in developing cost-effective, good quality and maintainable software. Developers become aware of the parts of a software product which would require more effort and thus can optimize the available resources to the fullest.

The existing literature with respect to software change prediction may be broadly categorized as a) development of prediction models which ascertain the change-prone nature of a class/module or b) the assessment of change impact of an incoming soft-

ware change request. We need to systematically summarize and review the current state of existing literature in order to ascertain the current trends in this domain. In order to do so, we perform a systematic review of existing studies in the domain of software change prediction. We investigate the following RQ's:

- RQ1: Which predictors are useful for developing software change prediction models?
- RQ2: What experimental settings are used while developing software change prediction models?
  - RQ2.1: Which techniques have been used for feature selection or dimensionality reduction while developing software change prediction models?
  - RQ2.2: What are the characteristics of datasets used for developing software change prediction models?
  - RQ2.3: What are the various validation methods used for developing software change prediction models?
  - RQ2.4: Which performance measures have been used for developing software change prediction models?
- RQ3: What are the various categories of data analysis techniques used for developing software change prediction models?
  - RQ3.1: Which is the most popular category of data analysis technique used for developing software change prediction models?
  - RQ3.2: What are the various ML techniques used for developing software change prediction models?
- RQ4: What is the predictive performance of ML techniques used for developing software change prediction models?

- RQ5: What is the comparative performance of ML techniques for developing software change prediction models?
- RQ6: Which statistical tests have been used for validating the results of software change prediction models?
- RQ7: What threats to validity exist while developing software change prediction models?
  - RQ7.1 What are the various categories of threats which exist while developing software change prediction models?
  - RQ7.2 What steps are required to mitigate the threats while developing software change prediction models?

The review would also help in identification of research gaps and will provide future guidelines to researchers and practitioners. The aim of the review is to systematically summarize the empirical evidence reported in literature with respect to various metrics, datasets, data analysis techniques, performance measures, validation methods and statistical tests used for software change prediction.

The chapter is organized as follows: Section 3.2 describes the review procedure and the stages involved in conducting the review. Section 3.3 states the review protocol. Section 3.4 states the answers to each of the investigated RQ. Finally, section 3.5 discusses the future directions. The results of the chapter are communicated as [129]

## 3.2 Review Procedure

According to the guidelines advocated by Kitchenham et al. [130], a review is conducted in three fundamental stages. These stages are reportedly planning, conducting

and reporting. The foremost step of the planning stage is to evaluate the necessity of the review. As already discussed, the aim of the review was to assess and summarize the empirical evidence in the domain of software change prediction. It intends to provide an overview of existing literature in the domain and would scrutinize possible future directions. Once the need of the review is assessed, the planning stage involves formation of RQs. Thereafter, a review protocol is formulated. The protocol includes a detailed search strategy. The search strategy consists of the list of possible search databases one intends to scrutinize, the search string and the criteria for including and excluding the extracted studies. Apart from the search strategy, the protocol also includes the criteria for assessing the quality of the candidate studies, the procedure for collecting the relevant data from the primary studies and synthesis of the collected data. The second stage involves the actual execution of the review protocol. In this stage, all the relevant literature studies are extracted, scrutinized and the relevant data is obtained. The final stage of the review reports the results of the investigated RQs. The RQs are answered on the basis of the data extracted from primary studies.

## 3.3 Review Protocol

The review protocol includes the search strategy, inclusion and exclusion criteria and the quality criteria for assessing the collected candidate studies. The following sections describe the review protocol followed.

### 3.3.1 Search Strategy

In order to design our search terms, we divided the explored RQs into comprehensive logical units. Moreover, terms were identified from paper titles, keywords and abstracts. Thereafter, all equivalent terms and synonyms were compiled using Boolean

OR, while distinguishable search terms were aggregated using Boolean AND. The period of the search was chosen from January 2000 to December 2017. The search-string for extracting candidate studies is as follows:

(“software product” OR “open source project” OR “software application” OR “software system” OR “software quality”) AND (“change” OR “evolution” OR “maintenance”) AND (“impact” OR “prediction” OR “proneness” OR “classification” OR “classifier” OR “empirical” OR “request”) AND ( “machine learning” OR “statistical” OR “search-based” OR “evolutionary” OR “data analysis”)

We searched a number of prominent search-databases such as SCOUPUS, Wiley, SpringerLink, IEEExplore, and ACM digital library. We also searched the reference lists of the extracted studies. As a result of this comprehensive effort, we identified 79 relevant studies. These studies were then subjected to the inclusion and exclusion criteria, as indicated in 3.3.2.

### 3.3.2 Inclusion and Exclusion Criteria

We use the following inclusion and exclusion criteria for selecting or rejecting a study based on the RQs. After applying the inclusion and exclusion criteria, we get 37 candidate studies.

***Inclusion Criteria***

All studies which determine the binary change-proneness attribute of a class/module or determine class stability with the aid of software metrics were included. We also included studies which used IR techniques for analyzing the change impact of a change request. Literature studies which reported and compared various data analysis techniques amongst themselves for developing software change prediction models were also included.

***Exclusion Criteria***

Studies which were based on predicting other dependent variables such as fault-proneness, maintenance effort, maintainability, change-count, amount of changes etc. were excluded. Also, studies which predict ordinal dependent variables for change-proneness such as low, medium, high etc. were not included as a part of the review. Review studies, poster papers, PhD dissertations and studies with little or no empirical analysis were excluded. In case a conference paper was extended in a journal, only the journal version of the paper was included. Studies which used only design patterns or code smells for determining change-prone nature of a class /module were removed.

Also, studies which assessed the change-impact using IR techniques from only the source code and not the change request were removed. Studies which used IR techniques for only bug-localization were excluded.

### 3.3.3 Quality Criteria

It is important to analyze the significance and contribution of each selected study in answering the various RQs. The 37 candidate studies were further assessed to evaluate their quality, according to the quality questions presented in Table 3.1. Each candidate study was given a Quality Score (QS) by aggregating the grades scored by a specific study on the basis of the 10 quality questions stated in Table 3.1. For each quality question, a study could be allocated three possible scores of 0 (No), 0.5 (Partly) and 1(Yes). Table 3.1 also lists the percentage of candidate studies acquiring a grade of "Yes", "Partly" and "No". All the studies whose QS was less than 5 (50% of the total quality score) were rejected. We rejected three studies [131–133]. After this step, a total of 34 literature studies were selected, which were termed as primary studies of our review. Relevant data pertaining to RQs was extracted from these

studies and the obtained results are reported in Section 3.4.

Table 3.1: Quality Questions

| Q# | Quality Questions | Yes | Partly | No |
|---|---|---|---|---|
| Q1 | Are the objectives of the research /research questions clear and concise? | 100% | 0% | 0% |
| Q2 | Are the predictor variables clearly defined and described? | 76% | 19% | 5% |
| Q3 | Are the number and magnitude of datasets analyzed suitable? | 70% | 27% | 3% |
| Q4 | Does the study use feature selection /dimensionality reduction techniques? | 32% | 8% | 60% |
| Q5 | Are the data analysis techniques clearly defined and described? | 57% | 11% | 32% |
| Q6 | Is there any comparative analysis amongst various models /techniques? | 73% | 8% | 19% |
| Q7 | Are the performance measures clearly specified? | 81% | 14% | 5% |
| Q8 | Did the study perform statistical hypothesis testing? | 46% | 3% | 51% |
| Q9 | Does the study validate the proposed approach? Are the validation methods appropriate? | 75% | 3% | 22% |
| Q10 | Is there a description of threats to validity of research? | 41% | 8% | 51% |

Table 3.2: Primary Studies with Quality Score

| Study No. | Study | QS | Study No. | Study | QS |
|---|---|---|---|---|---|
| CP1 | Liu & Khoshgoftaar 2001 [134] | 6.5 | CP18 | Malhotra & Khanna 2015 [135] | 8.5 |
| CP2 | Khoshgoftaar et al. 2003 [136] | 6.5 | CP19 | Bansal 2017 [36] | 9.5 |
| CP3 | Tsantalis et al. 2005 [32] | 8 | CP20 | Catolino et al. 2017 [137] | 8 |
| CP4 | Sharafat & Tahvildari 2008 [138] | 5.5 | CP21 | Elish et al. 2017 [139] | 8 |
| CP5 | Azar 2010 [34] | 6.5 | CP22 | Kumar et al. 2017a [33] | 8 |
| CP6 | Han et al. 2010 [140] | 6 | CP23 | Kumar et al. 2017b [141] | 9 |
| CP7 | Azar & Vybihal 2011 [35] | 7.5 | CP24 | Kumar et al. 2017c [142] | 7 |
| CP8 | Eski & Buzluca 2011 [27] | 5 | CP25 | Malhotra & Jangra 2017 [143] | 9 |
| CP9 | Lu et al. 2011 [28] | 7 | CP26 | Yan et al. 2017 [144] | 9.5 |
| CP10 | Romano & Pinzger 2011 [4] | 8 | CI1 | Antoniol et al. 2000 [70] | 5.5 |
| CP11 | Giger et al. 2012 [30] | 8 | CI2 | Canfora & Cerulo 2005a [145] | 5 |
| CP12 | Elish et al. 2013 [1] | 9.5 | CI3 | Canfora & Cerulo 2005b [146] | 6 |
| CP13 | Malhotra & Khanna 2013 [5] | 9 | CI4 | Canfora & Cerulo 2006 [69] | 6 |
| CP14 | Malhotra & Bansal 2014 [147] | 6 | CI5 | Gethers et al. 2012 [71] | 8 |
| CP15 | Malhotra & Khanna 2014 [148] | 9 | CI6 | Asl & Kama 2013 [149] | 6.5 |
| CP16 | Marinescu 2014 [150] | 7 | CI7 | Dit et al. 2014 [151] | 5 |
| CP17 | Elish et al. 2015 [45] | 6 | CI8 | Zanjani et al. 2014 [72] | 9.5 |

Table 3.2 lists all the primary studies with a specific allocated study number and its QS. All change-proneness prediction studies are allocated a number prefixed by ”CP” and all software change impact studies are allocated a number prefixed by ”CI”. According to quality analysis, the top scoring studies were CP12, CP19, CP26 and CI8. Also, the most popularly cited studies were CI3, CI5, CP10 and CP13.

## 3.4 Review Results and Discussion

This section states the results of the review. It may be noted that 53% of the 34 primary studies were published in various conferences and symposiums, while 44% of the studies were extracted from peer-reviewed journals, one study was published as a technical report. Table 3.3 states the most popular journal and conference venues. It may be noted that 76% of the studies developed change-proneness prediction models. Eight studies assessed the software change impact of a change request.

Table 3.3: Summary of Top Venues

| Name | Venue Type | Studies(%) |
|---|---|---|
| International Conf. on Mining Software Repositories | Conference | 9% |
| International Conf. on Advances in Computing, Communication and Informatics | Conference | 9% |
| International Conf. on Software Engineering | Conference | 6% |
| Information and Software Technology | Journal | 6% |
| Conf. means Conference | | |

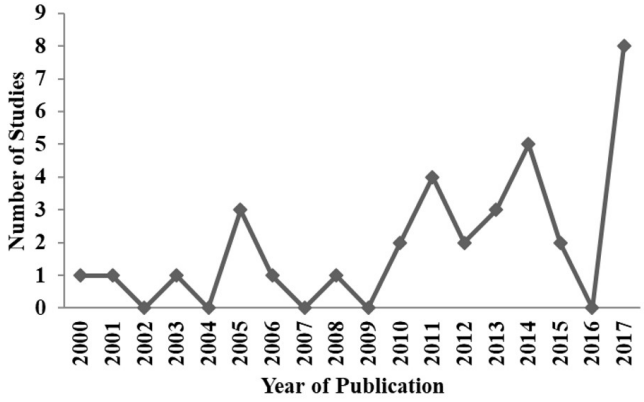


Figure 3.1: Year-wise Distribution of Primary Studies

A year-wise distribution of all the 34 primary studies is given in Figure 3.1. According to the figure, the most number of studies were published in 2017. Also, there is a surge in the number of studies since the year 2009.

We state the predictors, datasets, data analysis techniques, performance measures,

validation method and the statistical test used with respect to each of the 34 primary studies in Appendix B.1.

### 3.4.1 Results specific to RQ1

This RQ determines the various predictors which have been used in literature for developing software change prediction models. The predictors with respect to each study are summarized in Appendix B.1.

*Change-proneness*: An analysis of change-proneness prediction literature studies reveals that both product as well as process metrics have been used for developing prediction models. Figure 3.2 depicts a pie chart which states the percentage of change-proneness prediction studies using a specific category of metrics. According to the figure, it can be seen that 88% of the studies used product metrics. These metrics are generally source code design metrics, which depict the structural characteristics of a class such as its size, inheritance, cohesiveness etc. Many such metric suites which characterize the OO properties of class have been proposed in literature such as CK metrics suite [16], QMOOD metrics suite [25], Lorenz & Kidd metrics suite [19], Li & Henry metrics suite [18] and many others. We found that the CK metrics suite was the most commonly used OO metrics suite in literature studies. Apart from the CK metric suite, the SLOC metric (a measure of class size) has also been frequently used. Only 12% of change-proneness prediction studies used process metrics. While CP12 and CP21 used evolution-based metrics which characterize the evolution history of a class, CP20 used metrics which attempt to capture the complexity of the development process. It may be noted that CP12 and CP21 advocated the combination of both process as well as product metrics for determining change-prone nature of a class.

We also analyzed the granularity level over which these metrics were collected.

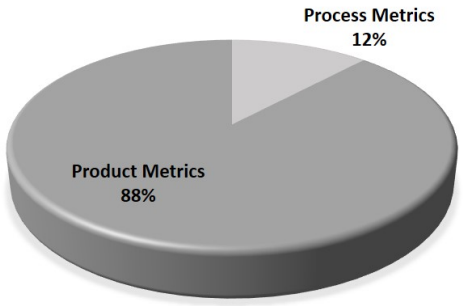


Figure 3.2: Product and Process Metrics Distribution

Three studies (CP1, CP2 and CP11) collected file-level metrics, one study each collected OO metrics at interface level (CP10) and method level (CP4). However, all other studies analyzed class-level metrics. It may also be noted that certain studies (CP5, CP7, CP9, CP22, CP23, CP24), analyzed a large number of OO metrics with respect to different dimensions (cohesion, coupling, size and inheritance) in order to obtain generalized results.

*Change Impact*: We extracted eight primary studies from literature which have attempted to determine the change impact of a software change request using IR techniques. An analysis of these studies indicate that IR techniques may be applied to a variety of textual data in the software repositories, be it source code, requirements and design documents or other free text for change impact analysis. Apart from this, studies have also proposed using commit histories, developer interactions and dynamic execution traces to aid the determination of impacted entities by a change request.

### 3.4.2 Results specific to RQ2

This RQ explores the various experimental settings i.e. the feature selection or dimensionality reduction methods, the characteristics of datasets used for empirical

validation, the validation methods and the performance measures used by software change prediction studies.

***Feature Selection & Dimensionality Reduction Techniques (RQ2.1)***

Literature studies use feature selection or dimensionality reduction techniques to aid the development of effective software change prediction models. We analyzed these studies to determine the most commonly used methods.

*Change-proneness*: An analysis of 26 primary studies which developed change-proneness prediction models revealed that only 46% of them used either a feature selection or a dimensionality reduction technique. The commonly used techniques and the corresponding studies which use them are listed in Table 3.4.

Table 3.4: Feature Selection /Dimensionality Reduction Techniques

| Feature Selection /Dimensionality Reduction | Study Numbers |
|---|---|
| CFS | CP13, CP15, CP18, CP19, CP25 |
| Univariate Analysis | CP22, CP23, CP25 |
| Principal Component Analysis (PCA) | CP12, CP21 |
| Multivariate Regression with forward and backward selection | CP3, CP13, CP23 |

According to Table 3.4, the most commonly used feature selection technique was CFS. Apart from the techniques listed in Table 3.4, other literature studies used several other miscellaneous methods (Best-first search (CP12), Gain Ratio (CP22), T-test (CP23), GA (CP24), Metric Violation Score (CP26)). Apart from feature selection, several studies performed correlation analysis to investigate whether the predictors used are correlated with change-proneness attribute (CP8, CP9, CP10, CP11, CP12, CP13, CP19, CP22, CP23).

Certain studies in literature reported specific OO metrics as significant predictors of change-prone nature of a class. These metrics were selected after application of feature selection or dimensionality reduction techniques. Since, in RQ1 we reported that majority of studies used the CK metrics suite and the SLOC metric, we state the studies which report these metrics as significant indicators of change-proneness

(Table 3.5). According to the table, a majority of the studies reported metrics which characterize size attribute (SLOC & WMC) and the ones which characterize coupling attribute (CBO & RFC) as significant indicators of change-proneness. Moreover, it may be noted that the inheritance attribute metric, DIT was only reported significant by one study and there was no study which indicated NOC as significantly related to change-proneness. These findings are similar to those of Lu et al. [28].

Table 3.5: Significant OO Metrics reported in Literature

| Metric Acronym | OO Attribute | Study Numbers |
|---|---|---|
| SLOC | Size | CP4, CP8, CP15, CP18, CP19, CP22, CP25 |
| WMC | Size | CP8, CP15, CP18, CP21, CP23, CP25 |
| CBO | Coupling | CP8, CP15, CP18, CP21, CP22, CP23, CP25 |
| RFC | Coupling | CP8, CP13, CP15, CP19, CP21, CP22 |
| LCOM | Cohesion | CP21, CP22 |
| DIT | Inheritance | CP21 |

*Change Impact*: In terms of change impact, only one literature study (CI8) reported singular value decomposition for dimensionality reduction. Thus, we could not get enough literature data to draw any conclusive results.

***Dataset Characteristics (RQ2.2)***

In order to perform empirical validation, literature studies have used a number of datasets. This question explores the characteristics of these datasets which includes their nature (public /private), size, percentage of change and other attributes. Software datasets used by literature studies can be broadly categorized into public /open-source datasets or private /commercial datasets. Table B.1 (Appendix B.1) depicts the datasets used in each primary study.

*Change-proneness*: We categorized the datasets in change-proneness prediction studies and found that only 8% of these studies used commercial /private datasets. All other change-proneness prediction studies used open-source datasets, which are publicly available. This trend was observed as commercial datasets are difficult to obtain. Therefore, researchers tend to validate their results on datasets that are open-

source and easily available in software repositories.

We also investigated the language used to develop the datasets, which are used for empirical validation for change-proneness prediction. Only three studies (CP1, CP2, CP18) used datasets developed using the C++ language. CP15 used four datasets developed in Java language and two datasets developed in C++ language. It may be noted that all other studies used datasets developed in Java language.

The datasets used in literature for change-proneness prediction are of varying sizes and with different percentage of change-prone classes. For each study, we analyzed the minimum and maximum size of datasets in terms of number of data points (Table 3.6). We also state the minimum and maximum percentage of change in the datasets used by these studies (Table 3.6).

It is also important to evaluate whether the datasets used for developing models are imbalanced in nature. As already discussed, a dataset is said to be imbalanced if it has a disproportionate number of change-prone and not change-prone classes. We state the number of datasets which were found to be imbalanced for a specific study (Table 3.6). As it is more important to determine the change-prone classes correctly, one should have sufficient number of change-prone classes in a dataset. We term a dataset as imbalanced if it has less than 40% of change-prone classes. Table 3.6 depicts all these details of the datasets for each specific study. Studies from which relevant information could not be extracted are not shown in the table.

According to the information shown in Table 3.6, the size of datasets used in literature for change-proneness prediction varies from 18-2,845 data points. Therefore, literature studies have analyzed small sized, moderately sized and large-sized datasets for developing change-proneness prediction models. It may also be noted that the percentage of change found in these datasets varies from 1% - 97%. However, in a majority of the studies 25%-100% of datasets analyzed were imbalanced in nature. Researchers should take active steps to develop effective prediction models

from such imbalanced datasets.

Table 3.6: Study-wise Details of Datasets

| Study Number | Number of Data Points | | Percentage of Change | | Number of Imbalanced Datasets (%) |
|---|---|---|---|---|---|
| | Minimum | Maximum | Minimum | Maximum | |
| CP1 | —- | 1,211 | —- | 24% | 1 (100%) |
| CP2 | —- | 1,211 | —- | 24% | 1 (100%) |
| CP3 | 58 | 169 | 25% | 50% | 1 (50%) |
| CP4 | —- | 58 | —- | 25% | 1 (100%) |
| CP5 | 18 | 2,737 | —- | —- | —- |
| CP6 | 44 | 62 | —- | —- | —- |
| CP7 | 18 | 958 | —- | —- | —- |
| CP8 | 38 | 693 | —- | —- | —- |
| CP9 | 38 | 2,845 | —- | —- | —- |
| CP10 | 25 | 165 | —- | —- | —- |
| CP11 | 98 | 788 | —- | —- | —- |
| CP12 | 36 | 170 | 4% | 91% | 10 (50%) |
| CP13 | 254 | 657 | 10% | 52% | 2 (67%) |
| CP14 | 607 | 2,786 | 1% | 97% | 3 (25%) |
| CP15 | 108 | 510 | 18% | 66% | 3 (33%) |
| CP17 | 36 | 60 | —- | —- | —- |
| CP18 | 108 | 510 | 45% | 66% | None (0%) |
| CP19 | 685 | 756 | 24% | 33% | 2 (100%) |
| CP21 | 36 | 170 | 4% | 78% | 5 (38%) |
| CP23 | 1,507 | 1,524 | 7% | 16% | 5 (100%) |
| CP24 | 83 | 1,943 | 30% | 68% | 3 (30%) |
| CP25 | 348 | 434 | 4% | 30% | 2 (100%) |
| CP26 | 53 | 3,150 | 8% | 94% | 8 (57%) |
| Note: ”—-” indicates the corresponding information was not found in the study. | | | | | |

*Change Impact*: With respect to change impact studies, the most commonly used datasets are stated in Table 3.7. Only 25% of the studies (CI1, CI6) used private /commercial datasets. All other studies used publicly available open-source datasets which were extracted from software repositories.

Table 3.7: Datasets used for developing Change Impact Models

| Dataset name | Study Numbers |
|---|---|
| Firefox | CI2, CI3, CI4 |
| ArgoUML | CI4, CI5, CI7 |
| KCalc | CI3, CI4 |
| Kpdf | CI3, CI4 |
| Kspread | CI3, CI4 |

Figure 3.3 depicts the number of studies which extracted datasets from specific

software repositories for developing change impact models. These repositories were Bugzilla, CVS and Subversion (SVN).

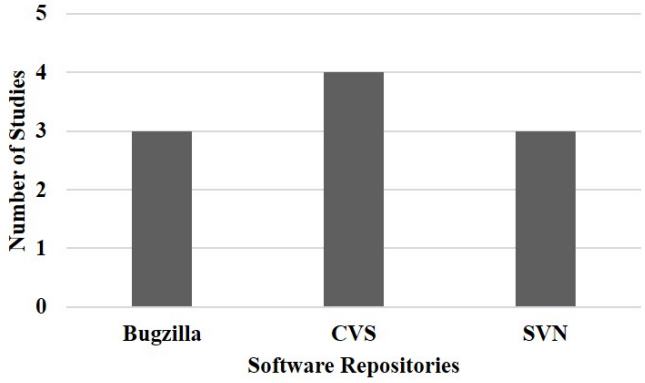


Figure 3.3: Software Repositories used for extracting Change Impact Data

***Validation Methods (RQ2.3)***

Studies in literature have used various validation methods for developing software change prediction models which can be broadly categorized into within-project validation methods and cross-project validation methods. Models developed using within-project validation use training and testing data of the same software project. The model is generally trained using the data obtained from the previous versions of the same project and is validated on the later versions. On the contrary, in cross-project validation, the prediction model is trained using data from one project (say Project A) and is validated on another project (Project B). Cross-project validation is useful in case historical data of the same software project is not available. The validation methods used by a specific study are mentioned in Table B.1 (Appendix B.1).

*Change-proneness*: Figure 3.4 depicts the most commonly used validation methods in change-proneness prediction studies. An analysis of the figure reveals that majority of studies used within-project validation. Within-project validation can be

performed using either hold-out validation, K-fold cross validation or Leave-one-out Cross Validation (LOOCV), which are described below:

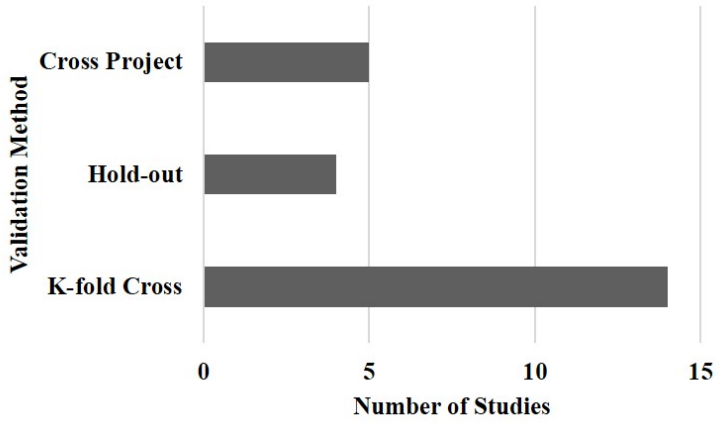


Figure 3.4: Validation Methods in Literature Studies

- *LOOCV*: For a dataset having N instances, this method requires N iterations. In each iteration, all data points except one is used for training the model. The remaining data point is used for validating the developed model. It is ensured that all data points are used at least once for validating the developed model. Only one study (CP17) used LOOCV.

- *K-fold Cross Validation*: In this method, the whole dataset is divided into K parts. It is ensured that the partitions are nearly equal in size. The rest of the process is similar to ten-fold cross validation method explained in Section 2.9.1. The most frequently used value for K is 10. Only one literature study used K=20 (CP23). It may be noted that K-fold cross-validation is the most popular method for validating change-proneness prediction models.

- *Hold-out Validation*: This method randomly partitions the available data points into testing and training. One of the most common ratio used for partitioning

is 75:25. In such a case, 75% of data points are used while training and the developed model is validated on the remaining 25% of data points. However, the method has high variability due to random division of training and test sets. The points which make the training and test sets may affect the performance of the developed model. Only four studies used hold-out validation.

Apart from within-project validation, cross-project validation was used by five change-proneness prediction literature studies (CP14, CP18, CP24, CP25, CP26). We found that k-fold cross validation is the most popular validation method as it averages the results obtained over several partitions, and reduces variance. As a result, the data is not sensitive to partitioning as in the case of hold-out validation. This gives an accurate estimate of the performance of the developed model.

*Change Impact*: Literature studies which proposed models to assess the change impact of a change request using IR techniques did not specifically mention the validation method. Only CI8 mentioned that it validated the proposed approach by using 90% of the data for training and 10% for testing (Hold-out validation). It may be noted that the other studies also performed validation as they mentioned the use of datasets but the method used by them is not specified.

***Performance Measures (RQ2.4)***

The developed software change prediction models in literature studies are assessed using various performance measures. This RQ investigates the most commonly used performance measures. The performance measures used by each specific study are stated in Table B.1 (Appendix B.1).

*Change-proneness*: Figure 3.5 states the most commonly used performance measures in change-proneness prediction studies. The definitions of these measures have already been stated in Chapter 2. According to the figure, the most commonly used measure is accuracy. However, in case of imbalanced datasets, accuracy is not an

appropriate measure [110, 118, 152] . Even if the percentage of correctly predicted change-prone classes are very few, accuracy values can be high as the performance measure is not sensitive to class distributions. On the contrary, the AUC measure is effective as it takes into account both recall and 1-specificity. Researchers should use an appropriate performance measure to yield unbiased results. Selection of an appropriate performance measure is vital to strengthen the conclusion validity of the study. Apart from the measures shown in Figure 3.5, there were several other performance measures (Type I error, Type II error, Overall misclassification error, False positive ratio, False negative ratio, Goodness of fit, J-index, G-measure, G-mean, Change cost, cost ratio), which were used by only one or two studies.

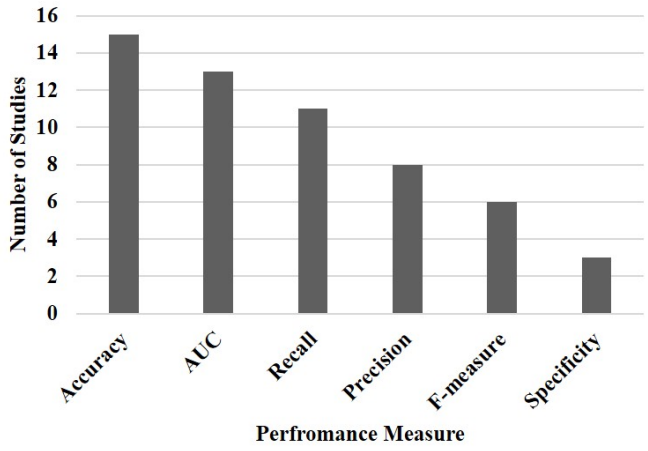


Figure 3.5: Performance measures in Literature Studies

*Change Impact*: Recall and Precision along with their variants have been used by 75% of literature studies which determine the change impact of a change request. The definition of Recall and Precision in terms of IR is stated below:

- *Recall*: It is defined as the ratio of correctly predicted impacted documents over the number of all impacted documents for a change request in the document set.

- *Precision*: It is defined as the ratio of correctly predicted impacted documents over the number of all predicted documents.

### 3.4.3 Results specific to RQ3

Prediction models are developed with the help of data analysis techniques. The data analysis techniques used by each study is stated in Table B.1 (Appendix B.1). These techniques can be broadly categorized into statistical or ML. We first investigate the most popular category of techniques for developing software change prediction models.

***Popular Category of Data Analysis Techniques (RQ3.1)***

Certain studies in literature used only a specific category of techniques i.e. only statistical or only ML, while certain others used more than one category of techniques.

*Change-proneness*: Figure 3.6 depicts the number of change-proneness prediction studies using the various categories of techniques. A new category of technique i.e. ensemble techniques were used by certain studies (CP17, CP22, CP24), which were ensemble of several base learning techniques. For instance, CP17 used an ensemble of SVM, MLP, LR, GP and K-means techniques which were aggregated using majority voting. According to figure 3.6, ML techniques are the most popular category of techniques, followed by the statistical techniques. The advantage of ML techniques over statistical techniques is that they do not require any prior assumptions about the underlying relationships between predictors and the target variable. Out of the 26 studies, which predict change-prone classes /modules, three studies did not use any specific technique but predicted classes using a certain set of equations (CP4), by using a combined rank list (CP8) or by using random effect meta-analysis model (CP9).

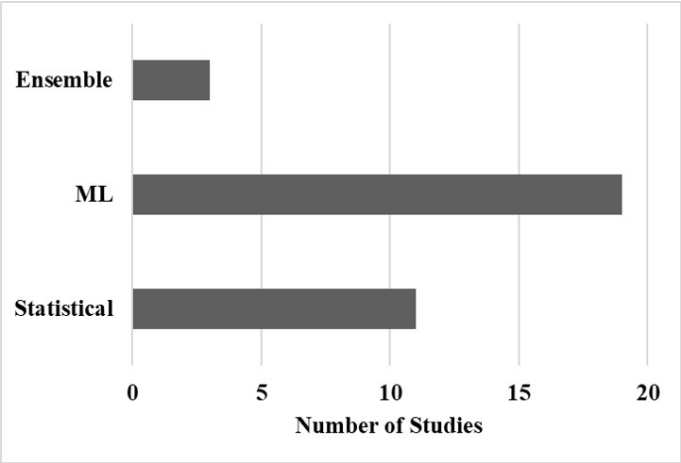


Figure 3.6: Categories of Techniques

*Change Impact*: Table B.1 (Appendix B.1) lists the data analysis techniques used by each study for determining the change-impact of a software change request. Only three studies listed their data analysis techniques. CI6 used Breadth-first search, CI7 used association rules and CI8 used K-nearest Neighbor (K-NN) algorithm. Due to the scarcity of studies, we could not obtain any conclusive results with respect to the popularity of data analysis techniques.

***ML Techniques used for Software Change Prediction (RQ3.2)***

With respect to change-proneness prediction studies, ML techniques can be further divided into several categories. It may be noted that only one change-impact study used an ML technique i.e K-NN.

The division of ML techniques has been done according to various ML categories discussed by Malhotra [17]. Table 3.8 states the various sub-categories of ML techniques which are used by change prediction studies. These sub-categories are DT, Bayesian algorithms, SVM, ML Ensembles, Neural Networks and SBA. Other remaining algorithms were grouped into a miscellaneous category.

Table 3.8: Sub-categories of ML Techniques

| Sub-category | ML Techniques |
|---|---|
| DT | C4.5, J48 |
| Bayesian | NB, BN |
| SVM | SVM, SVM with Linear Kernel, SVM with Polynomial Kernel, SVM with Sigmoid Kernel, Least-Square SVM |
| Neural Networks | MLP, Radial Basis Function (RBF), GMDH, Extreme ML (Linear, Polynomial and RBF kernels) |
| ML Ensemble | RF, AB, BG, LB |
| SBA | GP, Decision Tree-GP, ACO, Artificial Immune Recognition System (AIRS), Immunos99, Clonal Selection (CLG), CPSO, HIDER, MPLCS, SUCS, Fuzzy Learning based on Genetic Programming Grammar Operators and Simulated Annealing (GFS-SP), Fuzzy Learning based on Genetic Programming (GFS-GP), Genetic Fuzzy System Adaboost (GFS-AB), GFS-LB, Genetic Fuzzy System Maxlogitboost (GFS-MLB), NNEP, Structural Learning Algorithm in a Vague Environment with Feature Selection (SLAVE) |
| Miscellaneous | K-NN, K-Means, Non-nested Generalized Exemplars (NNGE) |

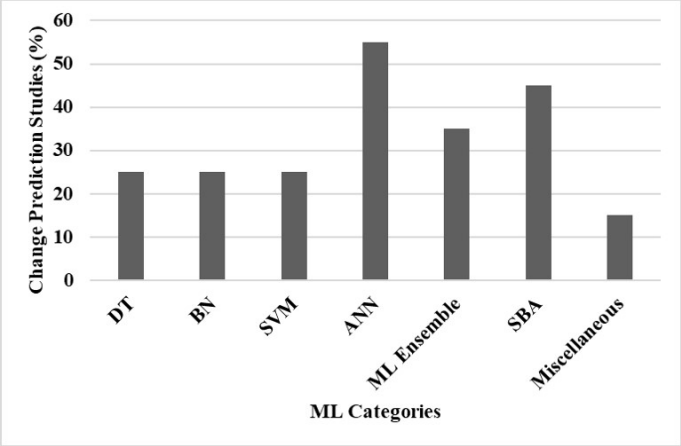


Figure 3.7: Sub-categories of ML Techniques

We further analyzed the percentage of primary studies which used a specific category of ML techniques amongst the studies which used an ML technique for software change prediction (Figure 3.7). It may be noted that neural networks is the most popular category of ML techniques which are used by 55% of studies. Neural networks are capable of modeling complex non-linear relationships and are adaptive in nature making them suitable for change prediction tasks. The next popular category of tech-

niques is SBA, used by 45% of studies. It is a subclass of ML techniques, which have recently gained popularity. SBA are self-optimizing techniques, which are capable of dealing with noisy and imprecise data. ML ensemble techniques, which form several classification models using variants of training set and use voting scheme to combine these models are the next popular category of techniques used by 35% of studies.

### 3.4.4 Results specific to RQ4

The predictive capability of various ML techniques investigated in the literature should be assessed so as to ascertain the effectiveness of change-proneness prediction models developed using them. In order to do so, we state the values of popular performance measures of the developed software change prediction models. However, we need to generalize our results and avoid any bias. This was done by reporting the results of models developed by those techniques which were validated by using at least three different datasets and by at least two of the primary studies. This would forbid a technique which exhibits exceptional performance only in a certain study or only by using certain datasets to be declared as a superior one. We analyze the statistics in accordance with the datasets. However, it may be the case that the performance of a technique varies due to its application on a specific dataset. Thus, we remove outlier values in accordance with the investigated datasets. We also report the median values to reduce biased results. The following rules were observed while extracting various statistics. The rules are chosen so that optimum values attained by a technique may be reported. This is important as the performance of an ML technique is dependent on its internal parameter settings.

- If a specific study develops models on the same dataset more than once with different experimental settings, we choose the best performance measure values obtained by the technique.

- In case there is more than one study which develops models using the same dataset and the same technique, we use the best of performance measure value reported in all the studies.

*Change-proneness*: According to RQ2.4, the most commonly used performance measures by change-proneness prediction studies are accuracy and AUC. Various datasets have been used in literature for developing software change prediction models which are stated in Table B.1 (Appendix B.1). Figure 3.8 depicts the dataset-wise outliers of different ML techniques with respect to accuracy measure. According to the figure, the Jmeter dataset was an outlier for both BG and RF techniques, showing lower accuracy values than all other investigated datasets. Two outliers were found with respect to MLP technique (Hibernate and Junit) and one with respect to C4.5 technique (PeerSim). Figure 3.9 depicts the dataset-wise outliers of different ML techniques with respect to AUC measure. According to the figure, AB, BG and MLP techniques were found to have one outlier each depicting lower AUC values.

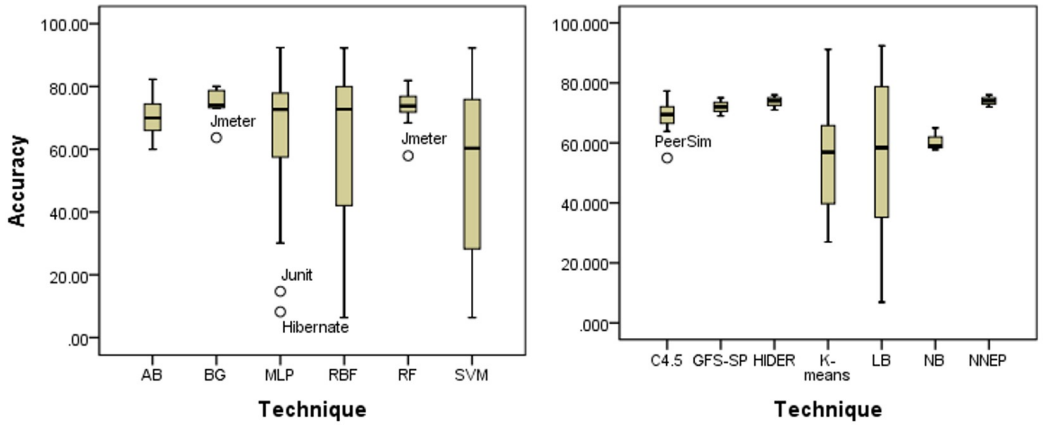


Figure 3.8: Dataset-wise Accuracy Outliers of ML Techniques

A good change prediction model exhibits higher values of accuracy and AUC measures. Table 3.9 and 3.10 presents the comparative results of the change prediction models developed using ML techniques for the accuracy and AUC measure

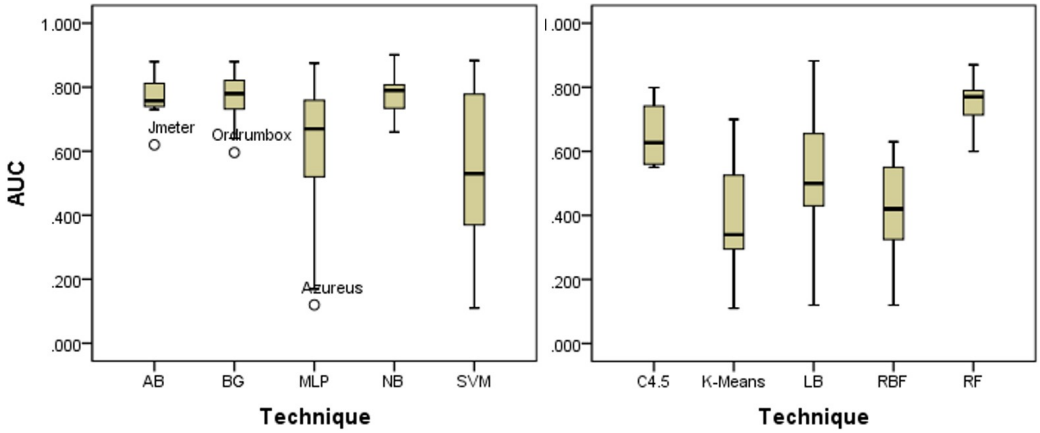


Figure 3.9: Dataset-wise AUC Outliers of ML Techniques

respectively. The tables report the minimum, maximum, mean, median and standard deviation values along with the count of datasets from which the statistics were extracted, after removing the outliers.

As depicted in Table 3.9, the majority of ML techniques (except K-means, LB and SVM) depicted mean accuracy values in the range 60-75%. The RF technique depicted the best mean accuracy value of 74.59%. With respect to median accuracy values, the best median values were depicted by NNEP, RF, BG and HIDER techniques of 74% each. As depicted in Table 3.10, with respect to AUC, the majority of ML techniques (except K-means, LB, RBF and SVM) depicted a mean AUC value in the range 0.63-0.79. The AB technique depicted the highest mean AUC value of 0.79. The best median AUC values were depicted by NB and BG techniques of 0.79 each, followed closely by the RF technique (0.77). These results indicate effectiveness of ML techniques in determining change-prone nature of classes/ modules.

Table 3.9: Accuracy Results of ML Techniques for Change-proneness Models

| ML Tech. | Count | Minimum | Maximum | Mean | Median | S.D. |
|---|---|---|---|---|---|---|
| AB | 9 | 60.00 | 82.28 | 70.69 | 70.00 | 7.61 |
| BG | 7 | 72.96 | 80.05 | 76.00 | 74.00 | 3.06 |
| C4.5 | 11 | 63.86 | 77.33 | 70.98 | 70.80 | 3.19 |
| GFS-SP | 3 | 69.00 | 75.00 | 72.00 | 72.00 | 3.00 |

| ML Tech. | Count | Minimum | Maximum | Mean | Median | S.D. |
|---|---|---|---|---|---|---|
| HIDER | 3 | 71.00 | 76.00 | 73.67 | 74.00 | 2.52 |
| K-Means | 16 | 26.99 | 91.17 | 54.51 | 56.91 | 19.35 |
| LB | 16 | 6.92 | 92.38 | 55.16 | 58.43 | 26.87 |
| MLP | 31 | 8.22 | 92.38 | 66.71 | 73.33 | 19.40 |
| NB | 3 | 57.77 | 65.00 | 60.59 | 59.00 | 3.87 |
| NNEP | 3 | 72.00 | 76.00 | 74.00 | 74.00 | 2.00 |
| RF | 15 | 68.42 | 81.85 | 74.59 | 74.00 | 3.64 |
| RBF | 24 | 6.38 | 92.29 | 62.40 | 72.75 | 25.51 |
| SVM | 17 | 6.38 | 92.29 | 53.15 | 60.33 | 29.07 |
| Tech. indicates Technique; S.D. indicates Standard Deviation | | | | | | |

Table 3.10: AUC Results of ML Techniques for Change-proneness Models

| ML Tech. | Count | Minimum | Maximum | Mean | Median | S.D. |
|---|---|---|---|---|---|---|
| AB | 7 | 0.73 | 0.88 | 0.79 | 0.76 | 0.06 |
| BG | 11 | 0.64 | 0.88 | 0.78 | 0.79 | 0.07 |
| C4.5 | 4 | 0.55 | 0.80 | 0.65 | 0.63 | 0.12 |
| K-Means | 16 | 0.11 | 0.70 | 0.40 | 0.34 | 0.17 |
| LB | 19 | 0.12 | 0.88 | 0.51 | 0.50 | 0.21 |
| MLP | 34 | 0.17 | 0.88 | 0.63 | 0.69 | 0.19 |
| NB | 13 | 0.66 | 0.90 | 0.78 | 0.79 | 0.06 |
| RBF | 16 | 0.12 | 0.63 | 0.41 | 0.42 | 0.16 |
| RF | 11 | 0.6 | 0.87 | 0.75 | 0.77 | 0.07 |
| SVM | 25 | 0.11 | 0.88 | 0.57 | 0.53 | 0.25 |
| Tech. indicates Technique; S.D. indicates Standard Deviation | | | | | | |

It may be noted that the RF, BG, AB techniques belong to the ensemble category of ML techniques. Therefore, their effective predictive capability is a result of aggregation of results of several base models. This leads to stable and robust models. It may also be noted that the SBA (NNEP and HIDER) also exhibit good accuracy results. SBA are effective in optimizing the accuracy of the developed change prediction models. This category of ML techniques needs to be further explored as their results are promising. The statistics reported in Table 3.9 and 3.10 reveal that the use of ML techniques for change-proneness prediction tasks should be encouraged as they yield effective results.

*Change Impact*: As only one study used an ML technique for determining the change impact of change request, we did not yield conclusive results with respect to predictive capability of ML techniques for predicting change impact.

### 3.4.5 Results specific to RQ5

This section compares the performance of different ML techniques amongst themselves and with statistical techniques used for developing software change prediction models. The comparative performance was evaluated dataset-wise and we followed the same rules as discussed in RQ4. Furthermore, Wilcoxon signed rank test was performed with $\alpha$ =0.05 to statistically evaluate the comparison results.

*Change-proneness*: We compared the performance of 12 ML techniques namely, MLP, BG, AB, RF, RBF, SVM, C4.5, K-Means, LB, HIDER, GFS-SP and NNEP amongst each other and with LR. LR is chosen as it is the most common statistical technique used in software change prediction literature. The other ML techniques were chosen as they are commonly used in literature and sufficient data could be collected for comparison purposes of these techniques with other ML techniques (i.e., used by at least two primary studies and on three datasets).

Table 3.11 and Table 3.12 reports the results of the Wilcoxon signed rank test when different techniques are compared amongst each other and with the LR technique according to accuracy and AUC performance measures respectively. The symbols used in the table represent whether the performance of the technique stated in the row is significantly superior ($\uparrow *$), significantly inferior ($\downarrow *$), superior but not significantly ($\uparrow$), inferior but not significantly ($\downarrow$) or equivalent ($=$), when compared with the technique stated in the column. According to Table 3.11, the MLP technique shows significantly better performance than LR, SVM and C4.5 techniques in terms of accuracy measure. The performance of MLP technique is equivalent to K-means technique and worse but not significantly, when compared with the AB technique. MLP's accuracy performance is better than RF, RBF and LB techniques but not significantly.

Table 3.11: Wilcoxon Test Results of ML Techniques Comparison on Accuracy measure

| Technique | MLP | BG | AB | RF | LR | RBF | SVM | C4.5 | K-Means | LB | HIDER | GFS-SP | NNEP |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| MLP | —- | ↓ ∗ | ↓ | ↑ | ↑ ∗ | ↑ | ↑ ∗ | ↑ ∗ | = | ↑ | NC | NC | NC |
| BG | ↑ ∗ | —- | ↑ | ↑ | ↑ | NC | NC | NC | NC | NC | NC | NC | NC |
| AB | ↑ | ↓ | —- | = | ↑ | NC | NC | NC | NC | ↑ | ↓ | ↓ | ↓ |
| RF | ↓ | ↓ | = | —- | ↑ ∗ | ↓ | NC | NC | NC | NC | NC | NC | NC |
| RBF | ↓ | NC | NC | ↑ | NC | —- | ↑ | ↑* | ↓ | NC | NC | NC | NC |
| SVM | ↓ ∗ | NC | NC | NC | ↓ | NC | —- | NC | ↓ | ↓* | NC | NC | NC |
| C4.5 | ↓ ∗ | NC | NC | NC | NC | ↓ ∗ | NC | —- | NC | NC | NC | NC | NC |
| K-Means | = | NC | NC | NC | NC | ↑ | ↑ | NC | —- | NC | NC | NC | NC |
| LB | ↓ | NC | ↓ | NC | NC | NC | ↑ ∗ | NC | NC | —- | NC | NC | NC |
| HIDER | NC | NC | ↑ | NC | NC | NC | NC | NC | NC | NC | —- | ↑ | ↓ |
| GFS-SP | NC | NC | ↑ | NC | NC | NC | NC | NC | NC | NC | ↓ | —- | ↓ |
| NNEP | NC | NC | ↑ | NC | NC | NC | NC | NC | NC | NC | ↑ | ↑ | —- |

"↑ ∗" means significantly better results and "↑" means better but not significant results
"↓ ∗" indicates significantly worse results and "↓" means worse but not significant results
"=" indicates the equivalent performance; "NC" indicates requisite comparison data could not be extracted
"—-" indicates the techniques cannot be compared with itself

Table 3.12: Wilcoxon Test Results of ML Techniques Comparison on AUC measure

| Technique | MLP | NB | SVM | BG | AB | RF | LR | LB | RBF | C4.5 | K-Means |
|---|---|---|---|---|---|---|---|---|---|---|---|
| MLP | —- | ↓ | ↑ | ↓ | ↓ | ↓ | ↓ | ↓ | ↑ | ↑ | ↑ |
| NB | ↑ | —- | ↓ | NC | NC | NC | NC | NC | NC | NC | NC |
| SVM | ↓ | ↑ | —- | NC | NC | NC | ↓ | NC | ↓ | NC | ↓ |
| BG | ↑ | NC | NC | —- | ↑ | ↑ | ↑ | ↑ | NC | NC | NC |
| AB | ↑ | NC | NC | ↓ | —- | ↓ | ↑ | = | NC | NC | NC |
| RF | ↑ | NC | NC | ↓ | ↑ | —- | ↑ | ↑ | NC | NC | NC |
| LB | ↑ | NC | NC | ↓ | = | ↓ | ↓ | —- | NC | NC | NC |
| RBF | ↓ | NC | ↑ | NC | NC | NC | NC | NC | —- | NC | ↓ |
| C4.5 | ↓ | NC | NC | NC | NC | NC | NC | NC | NC | —- | NC |
| K-Means | ↓ | NC | ↑ | NC | NC | NC | NC | NC | ↑ | NC | —- |

Symbols same as Table 3.11

The Wilcoxon test results according to AUC measure depicted in Table 3.12 show that there was no technique which was significantly better or worse than the other compared technique. The BG and RF techniques showed better AUC performance than four other techniques including LR but not significantly. The MLP technique also depicts better AUC values than four other compared techniques.

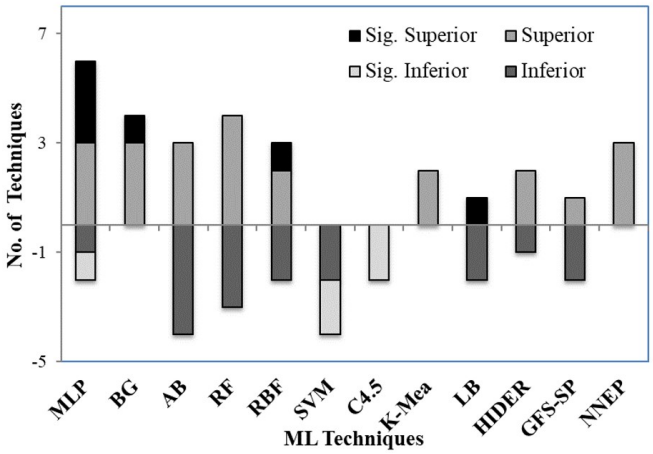


Figure 3.10: Wilcoxon Test results of ML Techniques Comparison based on Accuracy measure

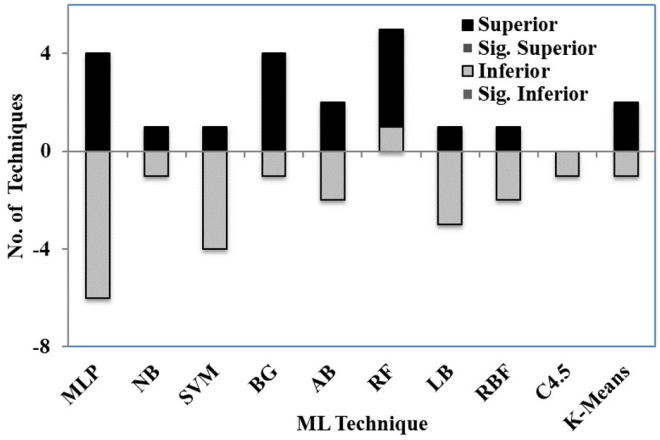


Figure 3.11: Wilcoxon Test results of ML Techniques Comparison based on AUC measure

Figure 3.10 and 3.11 summarizes the comparison results stated in Table 3.11 and

3.12 respectively. According to Figure 3.10, the BG, MLP, K-Means and NNEP seems to be the best ML techniques according to accuracy measure as they are better than most other compared techniques. SVM and C4.5 seem to be the worst techniques on the basis of accuracy. Similarly, according to Figure 3.11 the RF technique seems the best according to AUC values as it depicts better AUC values than all other compared techniques. The BG and K-means techniques also show effective AUC values as compared to other techniques. The C4.5 technique seems to show the worst comparative performance on the basis of AUC.

It may be noted from the results of Table 3.11 and 3.12 that four ML techniques depicted better accuracy results than the statistical technique, LR. Only one technique (SVM) showed poor accuracy results than LR. With respect to AUC, three ML techniques were found better than LR. This indicates that the performance of ML techniques is comparable to that of the statistical technique, LR. However, more studies need to be conducted for an extensive comparison of various ML techniques with that of LR.

Also, as a number of columns in Table 3.11 and 3.12 have the value "NC", where sufficient data is not available in literature for comparing the performance of ML techniques, more studies are needed which compare the performance of different ML techniques with each other on the basis of different performance measures.

*Change Impact*: As there was only one study which used an ML technique for determining the change impact, we could not evaluate the comparative performance of ML techniques in this domain.

### 3.4.6 Results specific to RQ6

Statistical verification of a study's results is important in order to yield reliable conclusions. However, only 47% of primary studies used statistical tests for validating

their results. The tests used by each study are listed in Table B.1 (Appendix B.1). These tests can be broadly categorized as parametric tests or non-parametric tests.

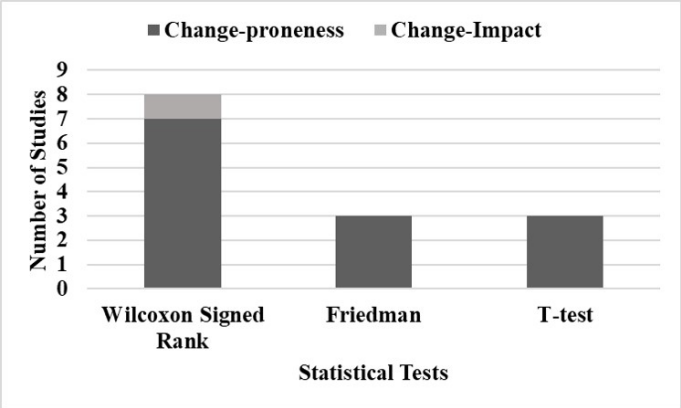


Figure 3.12: Statistical Tests used in Literature Studies

*Change-proneness*: Fourteen primary studies which predicted change-prone nature of a class /module statistically validated their results. Out of these 14 studies, 71% of studies used non-parametric tests, while the others used parametric tests. This is because parametric tests require complete information about the population and require specific assumptions, such as the data normality to be true before the application of these tests. Although, these properties make parametric tests more powerful, they are difficult to use as compared to the non-parametric tests which do not require any assumptions with respect to the population and are simple and easy to understand. Figure 3.12 states the number of studies using the most commonly used statistical tests. These tests were the Wilcoxon signed rank test, Friedman test and T-test. The most popular test was Wilcoxon signed rank test. Certain other tests (ANOVA, Nemenyi, Proportion, Cliff's) were used by one study each.

*Change Impact*: Only two studies statistically evaluated their results, while determining the change impact of a software change request using IR techniques. While one study used the parametric test ANOVA (CI8), the other used non-parametric Wilcoxon signed rank test (CI5).

### 3.4.7 Results specific to RQ7

This RQ states the various probable threats to empirical studies which develop software change prediction models using data analysis techniques. There are various probable sources of threats to a study which mitigate the validity of the obtained results. A researcher should carefully analyze the probable sources of threats in the design phase of an experiment so that proper experimentation set up can be provided and should also list these threats in the study so that the readers can comprehend the limitations of the research results. We extracted the threats from the primary studies of the review, which have a separate section for "Threats to Validity" or "Limitations".

***Categories of Threats (RQ7.1)***

The probable threats to software change prediction studies are categorized into four dimensions namely conclusion validity threats, internal validity threats, construct validity threats and external validity threats. Table 3.13 states the various threats corresponding to each category along with the studies which state them. It may be noted that we state only those threats which are mentioned in at least two or more primary studies. Threats specific to a study's experimental design are omitted to yield unbiased results.

Table 3.13: Threats to Validity in Software Change Prediction Studies

| Threat No. | Category | Threat Description (Study Numbers) |
|---|---|---|
| T1 | Conclusion | Absence of appropriate statistical tests for validating study's results. (CP10, CP16, CP20) |
| T2 | Internal | Omittance of other important variables that can act as predictors or may affect the predictors. (CP3, CP11) |
| T3 | Internal | Does not account for the confounding effect of other variables such as class size or other project and human factors (such as developer experience, application domain etc.) on the relationship between dependent and independent variables. (CP9, CP10, CP26, CI8) |
| T4 | Internal | Does not account for the "causal effect" of the independent variables on the dependent variable. (CP12, CP13, CP15, CP19, CP25) |

| Threat No. | Category | Threat Description (Study Numbers) |
|---|---|---|
| T5 | Internal | Does not account for different rules or thresholds for computing the dependent and the independent variables. (CP9, CP10, CP20, CP25, CP26) |
| T6 | Construct | The type of change i.e. whether it is corrective, adaptive, perfective or preventive is not taken into account. (CP9, CP12, CP13, CP19, CP25, CP26) |
| T7 | Construct | OO metrics may not be accurate representatives of the OO concepts they propose to measure. (CP9, CP12, CP13, CP15, CP19, CP25) |
| T8 | Construct | Independent variables (OO metrics) may not be correctly collected. (CP9, CP11, CP12, CP13, CP16, CP19) |
| T9 | External | Obtained results may be specific to a certain domain i.e. all validated datasets belonging to the same domain. (CP3, CP10, CP11, CP12, CP15, CP20, CP22, CI5, CI6) |
| T10 | External | Obtained results may not be validated on datasets of appropriate size or number of datasets. (CP9, CP10, CP12 CP13, CP15, CP16, CP19, CP20, CP25, CP26, CI8) |
| T11 | External | Obtained results may not be easily replicated. (CP10, CP16) |
| T12 | External | Obtained results may not be validated on industrial datasets. (CP13, CP15, CP23) |
| T13 | External | Obtained results may not be validated on datasets developed using different programming languages. (CP15, CP16, CP20, CP25, CP26, CI5) |

As stated in Table 3.13, we found one conclusion validity threat, four internal validity threats, three construct validity threats and five external validity threats. It may be noted that T11 was also referred to as "Reliability threat" in two studies.

***Mitigation of Threats (RQ7.2)***

This RQ explores how the various threats identified in RQ7.1 are addressed by the primary studies. We state the steps suggested by primary studies to mitigate the corresponding threats in Table 3.14. The table states the mitigation of only those threats, whose mitigation could be extracted from primary studies. The threats which were only mentioned in the "Threats to Validity" section of primary studies (T2 and T6), but could not be mitigated by the study or whose mitigation was not suggested are not stated in the table.

Researchers should incorporate these steps (Table 3.14), while designing the experimental set-ups of their study in order to ensure reliable results. Also, several studies should be performed with different size, category, domain and other dataset

characteristics to obtain generalized results in the domain of software change prediction.

Table 3.14: Mitigation of Threats to Validity in Software Change Prediction Studies

| Threat No. | Threat Mitigation |
|---|---|
| T1 | The results of a study should be validated using proper statistical tests. In case the underlying data does not fulfill the assumptions of a parametric statistical test, non-parametric statistical tests may be used. |
| T3 | The confounding effect of variables may be evaluated by first building a univariate regression model of the confounding variable C on each independent variable X. Thereafter, subtract the predicted values by the regression model from X to obtain a new variable X'. Now, C does not have a confounding effect on X'. |
| T4 | Controlled experiments should be carried out where only one specific predictor variable should be varied while keeping all other variables constant to determine the "causal" effect of predictor variables. |
| T5 | Additional thresholds or rules may be used to determine the impact of these on dependent and the independent variables. |
| T7 | OO metrics which are commonly used in literature and have been validated by previous studies may be used. |
| T8 | The tools used for collecting independent and dependent variables should be manually verified to ascertain their correctness. The use of public datasets, which have been verified by previous studies also mitigate the threat. |
| T9 | The results should be validated on datasets belonging to different domains. |
| T10 | The results should be validated on datasets of appropriate size and on an appropriate number of datasets. |
| T11 | The use of open-source datasets enhances the replicability of the study. Furthermore the tools used to implement the approach should be available. The steps conducted in the experiment should be clearly presented to ease replicated experiments. |
| T12 | The results should be validated on industrial datasets or datasets whose characteristics are similar to industrial datasets. |
| T13 | The results should be validated on datasets developed using different programming languages. |

## 3.5 Discussion & Future Directions

An extensive systematic review was performed to analyze the current state of existing literature in the domain of software change prediction and to further identify

research gaps in this domain. We selected 34 primary studies to answer the various RQs. 26 of these studies developed change-proneness prediction models, while eight others attempt to determine the change impact of a software change request using IR techniques. After taking into account the result discussions specific to each RQ in section 3.4, we propose the following future directions.

- The product metrics especially the CK metrics suite have been widely used in literature studies for developing models which predict change-prone nature of a class. However, the use of process metrics and the combination of product and process metrics is limited in this domain. Studies should be conducted to assess the capability of only process metrics as well as both process and product metrics as predictors of software change.

- Majority of studies in literature have analyzed OO metrics at class level. More studies should be performed which evaluate OO metrics at method level.

- There are very few (only eight) studies which determine the change impact of a software change request using IR techniques. Furthermore, no study has evaluated the change impact levels (low, moderate or high) of a software change request using IR techniques. More studies should be conducted by researchers to determine change impact or change impact levels of a software change request using IR techniques.

- Feature selection /dimensionality reduction techniques have been used by only 46% of change-proneness prediction studies or only 12% of change-impact studies. More studies should examine significant predictors using feature selection techniques in order to develop effective prediction models.

- Most of the datasets used by change-proneness studies or change impact studies were open-source in nature. However, more studies should be conducted to

validate commercial datasets to yield practical and generalized results.

- It was observed that 25% - 100% of datasets in a majority of the change-proneness studies were imbalanced in nature (had less than 40% of changed classes). Researchers in future should evaluate methods to develop effective models from imbalanced datasets as correct identification of change-prone classes is crucial. This would aid developers in prioritizing their resources effectively during maintenance and testing phases of a software development lifecycle.

- Within project validation has become a common standard while validating software change-proneness prediction models. However, studies in the future should explore more use of cross-project validation. Other validation methods such as inter-release validation and temporal validation may also be investigated in this domain. Furthermore, change impact studies should properly state and clarify the validation techniques for easy replication and verification.

- Apart from accuracy, the use of AUC is prominent in literature for evaluating change-proneness prediction models. Performance measures such as AUC, G-mean and Balance should be used by researchers in future as they give a realistic estimate of the performance of models which are developed from imbalanced datasets.

- It was observed that a majority of studies used ML techniques and these techniques are effective in the domain of software change prediction. However, more studies should be conducted which assess and compare the effectiveness of statistical and ML techniques for software change prediction. Also, more researchers should explore the use of ensemble of techniques as an alternative to other data analysis techniques for developing software change prediction

models.

- It was found that few SBA (HIDER and NNEP) exhibited effective accuracy results. However, more studies which investigate the effectiveness of SBA in the domain of software change prediction are required to yield conclusive results about their capability. Studies should be conducted to evaluate the effectiveness of SBA and compare their performance with other established ML and statistical techniques.

- The results indicate that a majority (51%) of studies did not use any statistical tests for verifying the obtained results. Hence, future studies should statistically evaluate the significance of their results.

- Researchers should take into account the various possible "Threats to Validity", while designing their experiments to yield effective and reliable results.

# Chapter 4

# Analyzing Software Change in Open-source projects using Machine Learning Techniques

## 4.1 Introduction

Managing change in the early stages of a software development lifecycle is an effective strategy for developing a good quality software at low costs. In order to manage change, software practitioners use software quality models which can efficiently predict change-prone classes and hence guide developers in the appropriate distribution of limited resources. However, to develop change prediction models, it is important to evaluate the existence of a relationship between OO software metrics of a class and its change-proneness attribute. This chapter conducts experiments to validate this relationship and identifies OO metrics, which are significant indicators of change in a class. Furthermore, the chapter examines the effectiveness of eleven ML techniques

for developing software change-proneness prediction models on six OO software datasets. We also compare the performance of ML techniques with the widely used statistical technique, LR. This chapter investigates the following research questions:

- RQ1: Does the relationship between OO metrics & change-proneness attribute of a class exists?
- RQ2: Are ML techniques effective in ascertaining change-prone classes in an OO software?
- RQ3: What is the comparative performance of ML and statistical techniques for developing change prediction models?

The motivation for RQ1 is the importance of developing software change prediction models. It is necessary for a software system to change progressively in order for it to remain useful and functional during its entire lifecycle. Therefore, a primary challenge for software practitioners is to understand and maintain these changing software systems effectively. Software change prediction models which identify change-prone classes of a software would help in efficient maintenance of such systems as more resources may be assigned to change-prone classes. This would help in the development of better quality software products as these classes are probable sources of defects and enhancements. Thus, stringent testing activities on such classes would result in less number of changes and defects in the future. Literature studies develop software change prediction with the help of various OO metrics [15, 16, 20] which measure inheritance, coupling, cohesion, size and other OO attributes. We need to first validate the relationship between OO metrics and software change and thereafter determine OO metrics which are significant predictors of change-prone classes.

ML techniques have been successfully applied in various other domains of software engineering like defect prediction [17, 37, 114], effort prediction [10, 153] and

maintainability prediction [45, 154]. Thus, RQ2 intends to analyze the effectiveness of these techniques for the determination of change-prone classes. Also, different ML techniques work differently and may yield contrasting results on various software datasets. Thus, it is important to evaluate a number of ML techniques for the task of software change prediction. This chapter investigates RF, BG, AB, LB, C4.5, CART, NB, MLP-BP, MLP-CG, GMDH and SVM techniques for determining change-prone classes in an OO software.

RQ3 is motivated by the fact that the basic structure and functioning of ML techniques is quite different from the traditional statistical techniques. Also, as noted in Chapter 3, more studies are required which compare the effectiveness of ML and statistical techniques for determining change-prone classes. Thus, we statistically compare the performance of ML and the statistical technique, LR in the domain of software change prediction.

This chapter is organized as follows: Section 4.2 states the research background and the research methodology. Section 4.3 states the results of the chapter. Section 4.4 discusses the response to the investigated RQ's, while section 4.5 states the key findings of the chapter. The results of the chapter are published in [155, 156].

## 4.2 Research Background & Methodology

This section first states the variables used in the study and then explains the research methodology. The research methodology followed in the chapter is diagrammatically represented in Figure 4.1.

Six open-source datasets are collected using the data collection procedure explained in Chapter 2 (Section 2.7). The descriptive statistics of the collected datasets are analyzed and outlier analysis is done. Univariate analysis is conducted in order to find the relationship between each OO metric and change-proneness attribute of

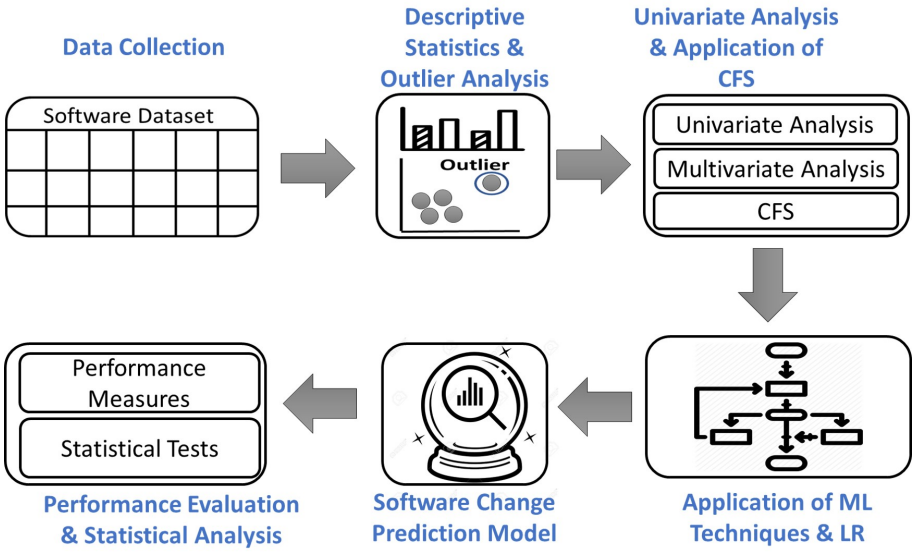


Figure 4.1: Research Methodology of Assessing Prediction Models

a class for each dataset. Multivariate LR analysis is conducted to analyze the combined effect of independent variables on change-prone nature of classes. Redundant and noisy variables are eliminated using CFS method. The metrics found significant (with univariate analysis and CFS) are used to develop models which predict change-prone nature of a class. We use ML techniques and the statistical technique, LR and the models are developed using ten-fold cross validation. The performance of the developed models are assessed using AUC, G-Mean1 and Balance measures. The developed models are compared statistically with Friedman test and Wilcoxon test.

### 4.2.1 Independent and Dependent Variables

The independent variables used in the study are six CK metrics [16] and the SLOC metric. The CK metric suite comprises of six popularly used metrics (RFC, DIT, NOC, WMC, LCOM and CBO). For detailed description of these OO metrics refer to section 2.5.1 (Chapter 2). The choice of CK metrics suite has been motivated by

the fact that it has been widely used for predictive modeling tasks [1, 17, 30, 80, 81] in literature. The dependent variable analyzed in this chapter is change-proneness.

### 4.2.2 Data Collection

The chapter uses six open-source datasets, as there has been a paradigm shift where open-source software have been highly appreciated by both the developers and users. Developers appreciate open-source software because of the large community support, while the users appreciate it because of its ability to customize features. Thus, it is important to examine the effectiveness of change-proneness prediction models on open-source datasets to support and enhance their development process and quality. The six open-source datasets of the chapter are AOI (2.7-2.9.2), Apollo (0.1-0.2), AviSync (1.1-1.2), DrJava (r4668-r5686), DSpace (1.6.0-1.8.1) and Robocode (1.7.2.2-1.7.4.4). The details of these datasets can be referred from Appendix A.

### 4.2.3 Descriptive Statistics and Outlier Analysis

The descriptive statistics of each of the six investigated datasets for each independent variable were analyzed as mentioned in Chapter 2 (Section 2.8.1).

We also analyzed the outliers using IQR filter of WEKA tool [88]. We found 24 outliers in AOI dataset, 13 in Apollo dataset, 2 in AviSync dataset, 30 in DrJava dataset, 12 in DSpace dataset and 18 in Robocode dataset. These outliers were removed before further analysis.

## 4.3 Result Analysis

This section states the univariate LR results, multivariate LR results, feature selection results and the ten-fold cross validation results of the developed change-proneness

prediction models.

### 4.3.1 Univariate Analysis

The results of univariate LR analysis are depicted in Table 4.1. It provides the statistical significance (p-value) for each metric in all the six datasets. All the metrics with a significance value of less than 0.05 are shown in bold and are significantly related to change-proneness in the corresponding dataset.

Table 4.1: Univariate LR Results

| Dataset | CBO | NOC | RFC | SLOC | DIT | LCOM | WMC |
|---|---|---|---|---|---|---|---|
| AOI | **< 0.001** | 0.194 | **< 0.001** | **< 0.001** | **0.049** | **< 0.001** | **< 0.001** |
| Apollo | **< 0.001** | 0.905 | **0.001** | 0.102 | 0.938 | **0.022** | **0.002** |
| AviSync | **0.005** | 0.399 | 0.090 | **0.003** | **0.005** | 0.602 | **0.014** |
| DrJava | **< 0.001** | 0.307 | 0.094 | **< 0.001** | **0.002** | **< 0.001** | **< 0.001** |
| DSpace | **< 0.001** | 0.074 | **0.001** | **< 0.001** | 0.328 | **< 0.001** | **< 0.001** |
| Robocode | **< 0.001** | 0.965 | 0.519 | **< 0.001** | **0.004** | **< 0.001** | **< 0.001** |

In AOI dataset, all metrics except NOC were found significant for predicting change-prone nature of classes. For Apollo dataset, four out of seven metrics were found significantly related to change-proneness at a threshold level of 0.05. However, the NOC metric, the SLOC metric and the DIT metric were found insignificant on the basis of univariate analysis for Apollo dataset. The univariate results of AVISync dataset show CBO, SLOC, DIT and WMC as significant metrics at a threshold value of 0.05. The metrics which were insignificant for the AVISync dataset were NOC, RFC and LCOM. The metrics found significant in DrJava dataset were CBO, SLOC, DIT, LCOM and WMC. For DSpace dataset, the univariate results indicated that the NOC and DIT metrics are insignificant. The NOC and RFC metrics were found insignificant in the Robocode dataset.

The cumulative results indicate that CBO and WMC are significant indicators of change-prone nature of a class as they were found significant in each of the six

datasets. The SLOC and LCOM metrics were chosen as significant predictors in five out of six datasets. The NOC metric is not a significant change predictor as it was not selected in any of the six datasets.

### 4.3.2 Multivariate LR Analysis

A multivariate LR analysis is used to analyze the combined effect of OO metrics on the change-proneness of a class. Multicollinearity depicts the extent to which the effect of a variable can be predicted by other variables in the analysis [82]. The conditional number for the models on all the datasets is below 30 indicating tolerable multicollinearity. We use backward elimination method for the generation of multivariate LR model. Tables 4.2-4.7 provide the coefficient (B), standard error (SE), statistical significance and odds ratio for the metrics which are included in the multivariate model for each of the six datasets used in the chapter.

Table 4.2 indicates that CBO, SLOC and DIT metrics are included in the multivariate change prediction model of AOI dataset. According to Table 4.3, only the CBO metric was selected for inclusion in the multivariate change-proneness model for Apollo dataset. However, the multivariate LR model on the AVISync dataset included DIT and WMC for the model development as shown in Table 4.4.

Table 4.2: Multivariate LR Results of AOI (Backward LR)

| Metric Name | B | SE | Significance | Odds Ratio |
|---|---|---|---|---|
| CBO | 0.076 | 0.022 | 0.001 | 1.079 |
| SLOC | 0.003 | 0.001 | $< 0.000$ | 1.003 |
| DIT | -0.354 | 0.176 | 0.045 | 0.702 |
| Constant | -1.482 | 0.292 | $< 0.001$ | 0.227 |

Table 4.3: Multivariate LR Results of Apollo (Backward LR)

| Metric Name | B | SE | Significance | Odds Ratio |
|---|---|---|---|---|
| CBO | 0.117 | 0.029 | $< 0.001$ | 1.124 |
| Constant | -1.706 | 0.241 | $< 0.001$ | 0.182 |

Table 4.4: Multivariate LR Results of AviSync (Backward LR)

| Metric Name | B | SE | Significance | Odds Ratio |
|---|---|---|---|---|
| DIT | -0.478 | 0.255 | 0.061 | 0.620 |
| WMC | 0.083 | 0.033 | 0.007 | 1.086 |
| Constant | -0.567 | 0.178 | 0.404 | 0.567 |

Table 4.5: Multivariate LR Results of DrJava (Backward LR)

| Metric Name | B | SE | Significance | Odds Ratio |
|---|---|---|---|---|
| CBO | 0.160 | 0.036 | $< 0.001$ | 1.173 |
| LCOM | 0.013 | 0.003 | $< 0.001$ | 1.013 |
| WMC | 0.026 | 0.010 | 0.008 | 1.027 |
| Constant | -1.636 | 0.215 | $< 0.001$ | 0.195 |

Table 4.6: Multivariate LR Results of DSpace (Backward LR)

| Metric Name | B | SE | Significance | Odds Ratio |
|---|---|---|---|---|
| CBO | 0.253 | 0.067 | $< 0.001$ | 1.287 |
| SLOC | 0.019 | 0.005 | $< 0.001$ | 1.019 |
| LCOM | 0.016 | 0.005 | 0.003 | 1.016 |
| WMC | -0.055 | 0.029 | 0.059 | 0.946 |
| Constant | -2.188 | 0.371 | $< 0.001$ | 0.112 |

Table 4.7: Multivariate LR Results of Robocode (Backward LR)

| Metric Name | B | SE | Significance | Odds Ratio |
|---|---|---|---|---|
| CBO | 0.048 | 0.028 | 0.081 | 1.050 |
| SLOC | 0.003 | 0.001 | 0.040 | 1.003 |
| LCOM | 0.014 | 0.005 | 0.011 | 1.014 |
| Constant | -2.428 | 0.322 | $< 0.001$ | 0.088 |

Table 4.5 states that CBO, LCOM and WMC metrics were selected using Backward LR for developing multivariate change-proneness prediction model on DrJava dataset. Multivariate LR results (Table 4.6) on DSpace dataset indicates inclusion of CBO, SLOC, LCOM and WMC metrics. According to Table 4.7, CBO, SLOC and LCOM metrics were selected for inclusion in the multivariate change-proneness prediction model for Robocode dataset.

### 4.3.3 CFS Results

The results on each dataset after application of CFS method are depicted in Table 4.8. The CBO metric was selected in all the six datasets. In five out of six datasets, WMC and SLOC metrics were selected for developing change-proneness prediction models. The LCOM metric was selected by three datasets. The RFC metric was selected by both Apollo and DrJava datasets. The NOC and DIT metrics were selected by only one dataset each. This indicates less use of the inheritance metrics (NOC and DIT) while developing models for predicting change-prone classes. The results of CFS are different for each dataset as the selected metrics are dependent on dataset characteristics and the change statistics. Similar results have been observed by previous literature studies [5, 36, 37].

Table 4.8: Metrics selected after application of CFS

| Dataset | Metrics Selected |
|---|---|
| AOI | CBO, SLOC, LCOM, WMC |
| Apollo | CBO, RFC, SLOC |
| AviSync | CBO, DIT, WMC |
| DrJava | CBO, NOC, RFC, SLOC, LCOM, WMC |
| DSpace | CBO, SLOC, WMC |
| Robocode | CBO, SLOC, LCOM, WMC |

### 4.3.4 Ten-Fold Cross Validation Results

Software change-proneness prediction models were developed using eleven ML techniques (RF, BG, AB, LB, C4.5, CART, NB, MLP-BP, MLP-CG, GMDH and SVM) and the statistical technique, LR. The specific details of each ML technique are mentioned in Chapter 2 (Section 2.6). It may be noted that the OO metrics selected after application of the CFS method are used for developing change-proneness prediction models using ML techniques. The metrics which were significant with univariate analysis were selected for model prediction using the LR method.

Table 4.9 states the ten-fold cross validation results using AUC measure on the six datasets used in the chapter. The technique whose model exhibited the best AUC result in each dataset is depicted in bold.

Table 4.9: AUC Results using Ten-fold Cross Validation

| ML Technique | AOI | Apollo | AviSync | DrJava | DSpace | Robocode |
|---|---|---|---|---|---|---|
| RF | **0.82** | **0.76** | 0.80 | 0.77 | 0.79 | 0.73 |
| BG | 0.79 | 0.64 | 0.76 | 0.78 | 0.82 | 0.74 |
| AB | 0.75 | 0.71 | **0.84** | 0.75 | 0.84 | 0.69 |
| LB | 0.77 | 0.71 | 0.81 | 0.77 | **0.83** | 0.70 |
| C4.5 | 0.59 | 0.58 | 0.72 | 0.72 | 0.79 | 0.71 |
| CART | 0.69 | 0.56 | 0.69 | 0.70 | 0.78 | 0.68 |
| NB | 0.71 | 0.64 | 0.75 | 0.77 | 0.78 | **0.75** |
| MLP-BP | 0.75 | 0.63 | 0.78 | **0.80** | 0.82 | 0.72 |
| MLP-CG | 0.72 | 0.67 | 0.75 | 0.79 | 0.81 | 0.75 |
| GMDH | 0.73 | 0.67 | 0.55 | 0.78 | 0.82 | 0.74 |
| SVM | 0.79 | 0.74 | 0.72 | 0.79 | 0.82 | 0.67 |
| LR | 0.76 | 0.67 | 0.73 | 0.79 | 0.82 | 0.74 |

According to Table 4.9, the models developed using the RF technique exhibited the best AUC values of 0.82 and 0.76 in AOI and Apollo datasets respectively. The models developed using the AB technique and the LB technique with AUC values of 0.84 and 0.83 respectively, gave the best AUC values in AviSync and DSpace datasets. The model developed using the MLP-BP technique with an AUC value of 0.80 is the best one for DrJava dataset, while the model developed using the NB technique exhibited the highest AUC value of 0.75 on the Robocode dataset. It may be noted that though the models developed using the LR technique exhibited competitive AUC values, they were not the best on either of the six datasets. The AUC values exhibited by a majority of models developed using the ML techniques were in the range of 0.60-0.84. This indicates an effective performance of ML techniques for developing change-proneness prediction models.

The G-Mean1 and Balance values of ten-fold cross validation results of change-proneness prediction models developed using the ML techniques and the LR technique are depicted in Table 4.10 and Table 4.11 respectively. The models depicting

the best performance measure values in each dataset are depicted in bold. According to the tables, the models developed using the RF technique exhibited the highest G-Mean1 values on AOI, Apollo and AviSync datasets and the highest Balance values on AOI and Apollo datasets.

Table 4.10: G-Mean1 Results using Ten-fold Cross Validation

| ML Technique | AOI | Apollo | AviSync | DrJava | DSpace | Robocode |
|---|---|---|---|---|---|---|
| RF | **0.72** | **0.60** | **0.72** | 0.72 | 0.71 | 0.57 |
| BG | 0.68 | 0.48 | 0.69 | 0.72 | 0.75 | **0.61** |
| AB | 0.52 | 0.40 | **0.72** | 0.68 | 0.75 | 0.60 |
| LB | 0.62 | 0.39 | 0.68 | 0.72 | 0.75 | 0.59 |
| C4.5 | 0.40 | 0.21 | 0.61 | 0.71 | 0.75 | 0.53 |
| CART | 0.58 | 0.31 | 0.63 | 0.69 | 0.76 | 0.60 |
| NB | 0.57 | 0.45 | 0.59 | 0.59 | 0.65 | 0.53 |
| MLP-BP | 0.57 | 0.24 | 0.60 | **0.74** | **0.78** | 0.57 |
| MLP-CG | 0.53 | 0.31 | 0.62 | **0.74** | 0.76 | 0.59 |
| GMDH | 0.54 | 0.38 | 0.42 | 0.73 | 0.75 | 0.53 |
| SVM | 0.50 | 0.00 | 0.58 | 0.73 | 0.77 | 0.44 |
| LR | 0.55 | 0.29 | 0.62 | 0.72 | 0.51 | 0.48 |

Table 4.11: Balance Results using Ten-fold Cross Validation

| ML Technique | AOI | Apollo | AviSync | DrJava | DSpace | Robocode |
|---|---|---|---|---|---|---|
| RF | **70.11** | **58.39** | 71.07 | 71.86 | 70.16 | 55.03 |
| BG | 64.47 | 47.17 | 66.78 | 71.53 | 74.27 | 56.81 |
| AB | 49.16 | 41.26 | **72.09** | 68.38 | 74.73 | **57.88** |
| LB | 58.75 | 41.08 | 67.35 | 71.54 | 74.07 | 55.46 |
| C4.5 | 41.14 | 32.34 | 60.24 | 70.57 | 73.69 | 51.53 |
| CART | 56.85 | 36.41 | 63.20 | 69.35 | 75.55 | 57.52 |
| NB | 54.22 | 44.96 | 57.82 | 55.45 | 61.92 | 50.39 |
| MLP-BP | 53.81 | 33.35 | 57.40 | 73.26 | 77.22 | 52.77 |
| MLP-CG | 50.59 | 36.44 | 57.84 | **74.11** | 75.79 | 54.82 |
| GMDH | 51.68 | 40.28 | 43.29 | 72.55 | 73.01 | 50.59 |
| SVM | 48.01 | 29.29 | 55.06 | 71.25 | 76.71 | 43.54 |
| LR | 52.05 | 35.42 | 60.81 | 71.80 | **77.33** | 46.81 |

According to Table 4.10, other models which exhibited the highest G-Mean1 values on other datasets were developed using BG, AB, MLP-BP and MLP-CG techniques. The best G-Mean1 values on all the datasets were in the range 0.60-0.78, which demonstrates the capability of ML techniques for developing change-proneness prediction models. With respect to Balance values (Table 4.11), apart

from the model developed using the RF technique (AOI and Apollo datasets) and the AB technique (AviSync and Robocode), the models developed using the MLP-CG and LR techniques exhibited highest Balance values on DrJava and DSpace datasets. Apart from Apollo dataset, majority of models developed using the ML techniques on other datasets exhibited Balance values in the range of 50-70%.

### 4.3.5 Friedman Test Results

We analyzed a number of change-proneness prediction models using eleven ML techniques and the LR technique on six open-source datasets. Though, there is a difference in the performance of the models developed using various techniques, we need to assess whether the difference is statistically significant. In order to do so, we perform Friedman statistical test at $\alpha = 0.05$. The details of the test are mentioned Chapter 2. The Friedman test was conducted based on the AUC values, G-Mean1 values and Balance values of each model respectively.

Table 4.12: Friedman Ranking of ML Techniques based on AUC Values

| ML Technique | Mean Rank |
|---|---|
| RF | 4.50 |
| LB | 4.67 |
| LR, BG | 5.00 |
| AB | 5.33 |
| SVM | 5.50 |
| MLP-CG | 5.67 |
| MLP-BP | 5.92 |
| GMDH | 7.00 |
| NB | 7.75 |
| C4.5 | 10.17 |
| CART | 11.50 |

Table 4.12 states the mean ranks obtained by each technique after application of Friedman test on AUC values. According to the results, the models developed using the RF technique exhibited best AUC values as the RF technique obtained a mean rank of 4.50. The second rank was given to the LB technique, which was closely

followed by the LR and BG techniques. The CART technique was designated as the worst technique with a mean rank of 11.50. The Friedman statistic value with eleven degrees of freedom was calculated as 25.67. Furthermore, a p-value of 0.007 indicates that the results are true with a confidence interval of 95%. Thus, we reject the null hypothesis of the Friedman test which states that all ML techniques and the LR technique are behaviorally same. The techniques are significantly different in their behavior. The results also indicate that certain ML techniques like RF and LB perform better than the statistical technique LR.

We also computed Friedman test on G-Mean1 and Balance values. However, the Friedman test results using these performance measures were not found significant.

### 4.3.6 Wilcoxon Test Results

As Friedman test results using AUC performance measure were found significant, we conducted Wilcoxon signed rank post-hoc test to statistically compare the pairwise performance of ML techniques and the statistical technique, LR.

Table 4.13: Wilcoxon Test Results based on AUC Values

| Compared Pair | Wilcoxon Test Output |
|---|---|
| LR vs RF | ↓ |
| LR vs LB | ↓ |
| LR vs BG | ↓ |
| LR vs AB | ↓ |
| LR vs SVM | ↓ |
| LR vs MLP-CG | ↑ |
| LR vs MLP-BP | ↑ |
| LR vs GMDH | ↑ |
| LR vs NB | ↑ |
| LR vs C4.5 | ↑ |
| LR vs CART | ↑ |
| ↑: Not significantly better; ↓: Not significantly poor | |

The null hypothesis of Wilcoxon signed rank test is that the performance of LR does not differ significantly when compared with the other investigated ML tech-

niques (RF, BG, AB, LB, C4.5, CART, NB, MLP-BP, MLP-CG, GMDH and SVM), using AUC values. The test was conducted at $\alpha = 0.05$. The results of the test are depicted in Table 4.13. The Wilcoxon output in the table is either stated as "↑" or "↓". "↑" indicates better results of the LR technique when compared with the stated ML technique. "↓" indicates poor results of the LR technique when compared with the stated ML technique. According to the results in Table 4.13, five ML techniques (RF, LB, BG, AB and SVM) were found superior to LR, but not significantly. Other compared ML techniques were found inferior, but again not significantly. Therefore, the investigated hypothesis is accepted. This indicates that the performance of ML techniques were comparable to the statistical technique, LR.

## 4.4 Response to RQ's

This section discusses the answers to the investigated RQ's of the chapter.

***Response to RQ1***

OO metrics which are representative of various OO attributes like coupling, cohesion, size etc. can be effectively used for predicting change-prone classes. This chapter has evaluated a number of change prediction models which can efficiently predict change-prone classes of a software product with OO metrics as the independent variables. Moreover, we conducted univariate LR analysis and applied CFS method to select OO metrics which are efficient predictors of change. Our results indicate that the CBO metric, the SLOC metric and the WMC metric are good indicators of change as these metrics were selected by a majority of the datasets using both univariate LR analysis and the CFS method. While CBO is a coupling metric, SLOC and WMC are size metrics. We also found that the inheritance metrics, DIT and NOC were not efficient change predictors since NOC was not selected by any

dataset using univariate LR analysis and DIT was selected in only one dataset using the CFS method. Similar results have been shown by Lu et al. [28], where they found that the inheritance metrics were least effective while predicting change. Thus, the experiments conducted in the chapter ascertains the existence of the relationship between OO metrics and change-proneness attribute of a class.

***Response to RQ2***

This chapter evaluates a number of change prediction models which were developed using the ML techniques (RF, BG, AB, LB, C4.5, CART, NB, MLP-BP, MLP-CG, GMDH and SVM). The performance capability of the models were evaluated with a number of performance measures i.e. AUC, G-Mean1 and Balance. A majority of the models developed using the ML techniques yield good results i.e. AUC values ranging from 0.60-0.84, G-Mean1 values ranging from 0.50-0.78 and Balance values ranging from 40-74%. These results show that the ML techniques can be effectively used for developing good change prediction models. These models can be used by the software industry and researchers to identify change-prone classes in the early phases of software development life cycle. Determining change-prone classes would help in planning effective resource usage during the maintenance and testing phases of a software. It would further help in developing a better quality software by rigorous verification activities of these change-prone classes.

***Response to RQ3***

The results of change prediction models developed using ML techniques and the LR technique were found comparable. However, in certain cases, the results of the models developed using ML techniques were better than change prediction models developed using the LR technique. According to Balance values of the developed change prediction models depicted in Table 4.11, only in the case of DSpace dataset, the LR technique exhibited the best Balance values as compared to the investigated

ML techniques. In all other seventeen cases (Table 4.9-4.11), where the best model according to a specific performance measure (AUC, G-Mean1 or Balance) was analyzed, an ML technique exhibited the best performance. This indicates the superiority of ML techniques for developing change-proneness prediction models.

In order to compare the performance of the ML techniques with the LR, we employed Friedman statistical test. The test ranks all the techniques (eleven ML and one statistical) using performance measure values of the developed prediction models on the six open-source datasets used in the chapter. The Friedman test results using G-Mean1 and Balance values were found insignificant. This indicates comparable results of the models developed using the LR technique and the ML techniques. However, the Friedman results using the AUC measure were found significant. The best rank according to AUC values was allocated to the RF technique, followed by the LB technique. Both RF and LB are ensemble techniques which provide stable and effective results by training on different samples and then aggregating the outputs. The results of the Friedman statistical test were true for a confidence interval of 95%.

Furthermore, we conducted Wilcoxon signed rank test to pairwise compare the performance of the LR technique with other investigated ML techniques for developing change prediction models. It was found that no pairwise comparison was found significant. Though, RF, LB, BG, AB and SVM models were found better than LR, but the statistical results were not significant. Similarly, the performance of all other ML techniques was found inferior but not significantly. Thus, we reiterate that the performance of the ML techniques is comparable to that of the LR technique for developing change-proneness prediction models.

## 4.5 Discussion

The aim of the chapter was to evaluate eleven ML techniques for their effectiveness in predicting change-prone classes of an OO software. The empirical validation was done using six open-source datasets. We further analyze and compare the performance of the ML techniques with the statistical technique LR using the Friedman and Wilcoxon signed rank statistical tests. Thus, the significance of this chapter includes evaluation of an extensive set of ML techniques for change-proneness prediction problem in OO datasets and the statistical comparison amongst all the ML techniques and the statistical technique, LR. The important findings of the chapter are as follows:

1. The chapter ascertains that there exists a relationship among OO metrics and change-proneness attribute of a class. The CBO metric, the SLOC metric and the WMC metric are efficient predictors of change and can be used for determining change-prone classes in an OO software.

2. The change prediction models developed using different ML techniques exhibited mean AUC values in the range 0.68-0.78, mean G-Mean1 values in the range 0.51-0.67 and mean Balance values in the range 54-66% over all the investigated datasets. This indicates their effectiveness in the domain of software change prediction. Researchers and practitioners may use ML techniques for identifying change-prone classes.

3. The performance of ML techniques were comparable and in some cases better than statistical technique, LR, though not significantly. Thus, ML algorithms are competitive with the LR technique while predicting software change and can be used by the software industry to effectively plan resource allocation and develop good quality software products.

# Chapter 5

# Analysis of Search-based Algorithms for Software Change Prediction

## 5.1 Introduction

SEPM involves the construction of models, with the help of software metrics, for estimating quality attributes. Recently, the use of SBA have gained importance as they help the developers and project managers in the identification of optimal solutions for developing effective prediction models. SBA are meta-heuristic procedures, which are capable of identifying an optimized solution from a large search space consisting of potential solutions. The search process in a search-based algorithm is guided by a fitness evaluator which ascertains the appropriateness of a specific solution [6]. The application of SBA for predictive modeling has been advocated by Harman and Jones [157] and Harman [158], as these techniques are efficient in balancing constraints and conflicts. Moreover, they are also efficient in handling noisy, partially inaccurate and incomplete datasets. Other advantages of SBA include their simple problem solving approach and robustness [158, 159]. These algorithms avoid getting

trapped in local optima and conduct the global search efficiently. Harman and Clark [160] have argued that the performance measures, such as the classification accuracy, can be used by SBA as fitness functions and hence, can be used to create software prediction models (SPM).

Given the newly identified relationship between the SBA and predictive modeling and the various advantages of SBA, this chapter has two main objectives:

1. Systematically study empirical evidence reported in literature about the use of SBA for the development of SPM.

2. Evaluate the effectiveness of SBA for developing prediction models to determine change-prone classes in a software.

In order to achieve the first objective, we conduct an extensive review on the use of SBA for SEPM, with a specific focus on: (1) software development effort estimation, (2) defect-proneness prediction, (3) maintainability prediction, and (4) change-proneness prediction. These attributes were selected, as according to a survey of empirical studies by Briand and Wust [161], some of the most common attributes investigated in empirical studies were defect-proneness, the effort for various developmental activities and the number of changes or defects. However, to structure our work, we selected two binary outcome variables (defect-proneness and change-proneness) and two continuous outcome variables (development effort and maintenance effort). It may be noted that effort, both before software development and during maintenance is a crucial attribute which needs to be estimated by software project managers so that proper planning and resource allocation can be performed. The improper estimation may lead to unnecessary delays and dissatisfied customers. Similarly, defects and changes need to be investigated so that project managers can identify weak components of a software. Such weak components may be restructured or allocated more effort for developing effective software products. We evalu-

ate SBA reported in the literature, for the four mentioned SEPM tasks, over a time period from January 1992 to December 2017 by conducting a systematic review of 91 primary studies. Moreover, we also explore effective experimental setups and methods followed in literature for using SBA in SEPM domain.

The review was conducted to summarize and assess empirical evidence associated with primary studies regarding: (1) the context of SBA in SEPM; (2) the validation techniques, number of runs to account for the stochastic nature and the performance measures used for SEPM using SBA; (3) the use of different fitness functions for SEPM; (4) the performance capabilities of SBA for SEPM; (5) the comparative predictive capabilities of SBA and the ML techniques; (6) the use of different tests to statistically validate the comparative performance of various techniques and, (7) the threats to validity encountered while using SBA for SEPM.

The second objective is addressed by performing an empirical experiment which assesses the use of SBA for determining change-prone classes. As indicated in Chapter 3, SBA exhibited effective results while developing change-proneness prediction models. Moreover, we found that more empirical studies are required, which assess and compare the effectiveness of SBA in this domain with other techniques. Thus, we compare the performance of SBA with ML and statistical techniques. The performance of eight SBA (CPSO, GA-ADI, GA-Int, GEP, HIDER, MPLCS, SUCS, XCS), 4 ML techniques (SVM, C4.5, CART, MLP-CG) and 1 statistical technique (LDA) is evaluated on fourteen open-source datasets for developing models which predict change-prone classes. It is essential to evaluate a number of algorithms as some algorithms work well on or are well suited to certain datasets while they may not yield good results on other datasets. Furthermore, it enhances the generalizability of the experiment's results. We also statistically evaluate and compare the performance of the investigated techniques using Friedman and Wilcoxon tests.

It may be noted that we would address results specific to the first objective as "re-

view results" and the results specific to the empirical experiment (second objective) as "experimental results".

This chapter is organized as follows: Section 5.2 states the review background and its results. It includes the details of the review and states the results specific to each investigated RQ in the review. It also summarizes the current trends and provides future guidelines to researchers. Section 5.3 states the experimental design and framework for conducting the empirical experiment. Section 5.4 states the results of the empirical experiment and analyzes them. Finally, Section 5.5 states the discussion of the chapter results. The results of the chapter are published in [162–164].

## 5.2 Review Background & Results

This section first states the background of the review i.e. the investigated RQ's, review procedure and the primary studies. Thereafter, the subsequent subsections discuss the results corresponding to each RQ investigated in the review.

### 5.2.1 Review Background

In order to conduct the review, we follow a similar review procedure as discussed in Chapter 3 (Section 3.2). The RQ's investigated in the review are mentioned below:

- RQ1: What is the context of SBA in SEPM?
- RQ2: What experimental settings are used for SEPM, when SBA are used?
  - RQ2.1: Which type of validation is performed for SEPM using SBA?
  - RQ2.2: How many runs/executions have been performed in the experiment to account for the stochastic nature of SBA for SEPM?

  - RQ2.3: Which performance measures are used by experiments for analyzing the developed SPM?

- RQ3: Which fitness functions have been used by SBA for SEPM?
- RQ4: What is the predictive performance of a specific search-based algorithm for SEPM tasks?
- RQ5: What is the comparative performance of the models predicted using SBA and models predicted using ML techniques for SEPM?
- RQ6: Which statistical tests have been used to compare the predictive ability of different SBA for SEPM?
- RQ7: What threats to validity exist in the application of SBA to SEPM? How to mitigate the identified threats?

The search-string for extracting candidate studies is as follows:

(“software product” OR “open-source project” OR “software application” OR “software system”) AND (“defect” OR “error” OR “fault” OR “bug” OR “effort” OR “cost” OR “change” OR “maintenance effort” OR “maintainability”) AND (“estimation” OR “prediction” OR “proneness” OR “classification” OR “classifier” OR “assessment” OR “empirical”) AND ( “search-based” OR “meta-heuristic” OR “evolutionary” OR “multi-objective”) AND (“local search” OR “tabu search” OR “hill climbing” OR “simulated annealing” OR “genetic programming” OR “gene expression programming” OR “genetic algorithm” OR “ant colony optimization” OR “particle swarm optimization” OR “differential evolution” or “evolutionary programming” OR “artificial immune system” OR “cuckoo search” OR “artificial bee colony” OR “harmony search” or “hybrid” OR “memetic” OR “teaching-learning-based optimization”)

We searched a number of prominent search-databases such as SCOUPUS, Wiley, SpringerLink, IEEExplore, and ACM digital library. Furthermore, important search-based avenues that include a repository of Search Based Software Engineering publications (`http://crestweb.cs.ucl.ac.uk/resources/sbse_repository/repository.html`) and proceedings of conferences such as Genetic and Evolutionary Computation (GECCO) and Symposium on Search-Based Software Engineering (SSBSE) were also explored. The period of the search of candidate studies was chosen as January 1992 to December 2017 as the use of a first search-based algorithm for a software engineering application was done in 1992 [165].

A total of 169 candidate studies were collected, which were first subjected to inclusion and exclusion criteria, mentioned in Appendix C.1. After the application of the criteria, a total of 112 studies were included. These studies were further assessed to evaluate their quality, according to the quality questions stated in Appendix C.2. Each literature study was given a QS by aggregating the grades scored by a specific study on the basis of these 16 quality questions. For each quality question, a study could be allocated three possible scores of 0 (No), 0.5 (Partly) and 1 (Yes). All the studies with QS $< 8$ (50% of the total QS) were rejected. After this step, a total of 91 literature studies were selected, which were termed as primary studies of our review (Table 5.1). The details of data collection procedure are listed in Appendix C.3. Relevant data pertaining to RQs was extracted from these studies and the obtained results are reported in subsequent sections.

Table 5.1 lists all the 91 primary studies with a specific allocated study number and its QS. It may be noted that 47% of the studies developed effort estimation models (denoted by ES), 41% of the studies developed defect prediction models (denoted

Table 5.1: Primary Studies with Quality Score

| Study No. | Study | QS | Study No. | Study | QS |
|---|---|---|---|---|---|
| ES1 | Dolado & Fernandez 1998 [166] | 11.5 | DS4, CS2 | Khoshgoftaar et al. 2003 [136] | 11.5 |
| ES2 | Dolado 2000 [167] | 13 | DS5 | Liu et al. 2004 [168] | 9.5 |
| ES3 | Shukla 2000 [169] | 12.5 | DS7 | De Carvalho et al. 2008 [170] | 14 |
| ES4 | Burgess & Lefley 2001 [171] | 13 | DS8 | Tsakonas & Dounias 2008 [172] | 10 |
| ES5 | Dolado 2001 [173] | 11 | DS9 | Vandecruys et al. 2012 [174] | 12 |
| ES6 | Kirsopp et al. 2002 [175] | 12.5 | DS10 | Catal & Diri 2009 [176] | 14 |
| ES7 | Shan et al. 2002 [177] | 12.5 | DS11 | Singh et al. 2009 [178] | 11 |
| ES8 | Lefley & Shepperd 2003 [179] | 12 | DS12 | Afzal 2010 [180] | 13.5 |
| ES9 | Regolin et al. 2003 [181] | 11.5 | DS13 | De Carvalho et al. 2010 [37] | 15 |
| ES10 | Lokan 2005 [182] | 11.5 | DS14 | Jin et al. 2010 [183] | 10 |
| ES11 | Huang & Chiu 2006 [184] | 12 | DS15 | Liu et al. 2010 [185] | 13 |
| ES12 | Sheta 2006 [186] | 9 | DS16 | Pendharkar 2010 [187] | 10.5 |
| ES13 | Chiu & Huang 2007 [188] | 12 | DS17 | Chiu 2011 [189] | 10.5 |
| ES14 | Ahmed et al. 2008 [190] | 8.5 | DS18 | Di Martino et al. 2011 [46] | 11.5 |
| ES15 | Braga et al. 2008 [191] | 11.5 | DS19 | Yu 2012 [192] | 12 |
| ES16 | Huang et al. 2008 [193] | 11.5 | DS20 | Rodriguez et al. 2012 [194] | 11.5 |
| ES17, DS6 | Tsakonas & Dounias 2008 [195] | 11 | DS21 | Sarro et al. 2012 [196] | 13.5 |
| ES18 | Ferrucci et al. 2009 [197] | 13.5 | DS22 | Can et al. 2013 [198] | 8.5 |
| ES19 | Li et al. 2009a [199] | 11 | DS23 | Canfora et al. 2013 [200] | 15 |
| ES20 | Li et al. 2009b [201] | 10.5 | DS24 | Abaei & Selamat 2014 [202] | 12 |
| ES21 | Tsakonas & Dounias 2009 [203] | 12 | DS25 | Harman et al. 2014 [38] | 12.5 |
| ES22 | Ferrucci et al. 2010a [204] | 14 | DS26 | Li et al. 2014 [205] | 8.5 |
| ES23 | Ferrucci et al. 2010b [47] | 13.5 | DS27 | Malhotra 2014 [206] | 13 |
| ES24 | Ferrucci et al. 2010c [207] | 12.5 | DS28 | Arar & Ayan 2015 [61] | 14.5 |
| ES25 | Oliveira et al. 2010 [153] | 14.5 | DS29 | Jin & Jin 2015 [208] | 14 |
| ES26 | Sheta et al. 2010 [209] | 9 | DS30 | Abdi et al. 2015 [210] | 15 |
| ES27 | Alaa and Al-Afeef 2010 [211] | 11 | DS31 | Afzal & Torkar 2016 [212] | 14.5 |
| ES28 | Aljahdali 2010 [213] | 10 | DS32 | Kumar & Rath 2016 [214] | 11 |
| ES29 | Chavoya et al. 2010 [215] | 11.5 | DS33 | Ryu & Baik 2016 [216] | 14 |
| ES30 | Araujo al. 2012 [217] | 11 | DS34 | Xia et al. 2016 [40] | 16 |
| ES31 | Sarro et al. 2012 [218] | 13 | DS35 | Ferrucci et al. 2017 [39] | 16 |
| ES32 | Bardsiri et al. 2013 [219] | 12.5 | DS36 | Hosseini et al. 2017 [41] | 15.5 |
| ES33 | Barros et al. 2013 [220] | 14 | DS37 | Mausa & Grbac 2017 [221] | 14 |
| ES34 | Corazza et al. 2013 [222] | 16 | MS1 | Baqais et al. 2014 [223] | 8 |
| ES35 | Minku & Yao 2013a [224] | 14 | MS2 | Malhotra & Chug 2014 [225] | 10.5 |
| ES36 | Minku & Yao 2013b [226] | 13 | MS3 | Kumar et al. [227] | 10.5 |
| ES37 | Dan 2013 [228] | 8 | MS4 | Jain et al. 2016 [229] | 10 |
| ES38 | Bardsiri et al. 2014 [230] | 14.5 | CS3 | Azar 2010 [34] | 13 |
| ES39 | Azzeh et al. 2014 [231] | 9.5 | CS4 | Azar & Vybihal 2011 [35] | 11.5 |
| ES40 | Sarro et al. 2016 [232] | 15.5 | CS5 | Marinescu 2014 [150] | 10.5 |
| ES41 | Benala & Mall 2017 [233] | 14.5 | CS6 | Malhotra & Khanna 2014 [148] | 13.5 |
| ES42 | Murrillo-Morera et al. 2017 [234] | 15 | CS7 | Malhotra & Khanna 2015 [135] | 12 |
| ES43 | Wu et al. 2017 [235] | 9.5 | CS8 | Elish et al. 2015 [45] | 9 |
| DS1 | Hochman et al. 1996 [236] | 9 | CS9 | Bansal 2017 [36] | 13.5 |
| DS2 | Hochman et al. 1997 [237] | 13 | CS10 | Kumar et al. 2017 [141] | 11 |
| DS3, CS1 | Liu et al. 2001 [134] | 12 | | | |

by DS) and 11% of the studies developed models to predict change-proneness (denoted by CS). Only four studies developed maintainability prediction models (denoted by MS). Three studies assessed more than one software attribute. According to Table 5.1, the best studies according to QS were ES34, DS34 and DS35. Researchers might refer to these studies for designing effective experimental setups while using SBA for SEPM. Also, the most popularly cited studies were ES4, DS10, ES20, ES16, ES40 and DS34. A year-wise distribution of all the primary studies is discussed Appendix C.4.

### 5.2.2 Results specific to RQ1

It is important to first observe the various SBA used by literature studies for SEPM and categorize them. Furthermore, the most prevalent category of SBA along with the characteristics of SBA, which make them suitable in developing SEPM needs to be assessed.

***Categories of SBA used for SEPM***

SBA which are used for developing SEPM can be categorized into four broad categories: Local Search, Evolutionary, Swarm Intelligence and Hybrid.

A local search technique starts its search from a specific candidate solution and explores only its neighbourhood to search for an optimal solution. Thus, the fitness function of these techniques is locally optimized indicating the name local search. In order to overcome local optima, these techniques may perform many solution restarts. On the contrary, evolutionary techniques simulate biological evolution mechanisms. They search for optimum solutions by application of various genetic operators such as selection, reproduction, mutation and crossover on candidate solutions. Thus, the population of solutions keeps evolving leading to globally optimal or near-optimal solutions.

Swarm intelligence techniques are modeled on the basis of collective behaviour of individual agents (candidate solutions) without any centralized control. An intelligent solution is outcome by an interaction of these individual agents amongst each other and with the provided environment. These agents follow a specific set of rules and simulate natural behaviours such as bird flocking, immune systems, etc. Hybrid category of techniques encompass SBA, which combine more than one approach into a single unit. For instance, a search-based approach such as GA may be combined with an ML technique such as Artificial Neural Network (ANN). The prime motive of such approaches is to aggregate the strengths of both the constituent techniques to provide a better approach.

Appendix C.5 states the various SBA used by the primary studies and the study numbers that use them according to these categories. It may be noted from the table that a wide variety of hybrid SBA are possible, which have been used by various studies in the domain of SEPM.

***Popularity of different SBA used for SEPM***

Figure 5.1 depicts the percentage of studies which used a specific category of SBA for effort estimation, defect prediction, maintainability prediction or change prediction. We summarize the following trends from the figure:

- A majority of effort estimation studies (58%) used the evolutionary techniques followed by the hybrid techniques (16%). The most commonly used evolutionary technique was GP, which was closely followed by GA.

- The most commonly used category of SBA for defect prediction were the hybrid techniques (43%). In addition, swarm intelligence techniques were used in 27% of defect prediction studies. Evolutionary algorithms (mainly GP and GA) were explored in 30% of the defect prediction studies.

- Maintainability prediction studies used either hybrid (75%) or evolutionary (25%) techniques.

- A majority of change prediction studies (41%) evaluated the use of evolutionary techniques such as GP and GA. Only 38% of the studies used swarm intelligence techniques for developing models. Other studies used hybrid techniques.

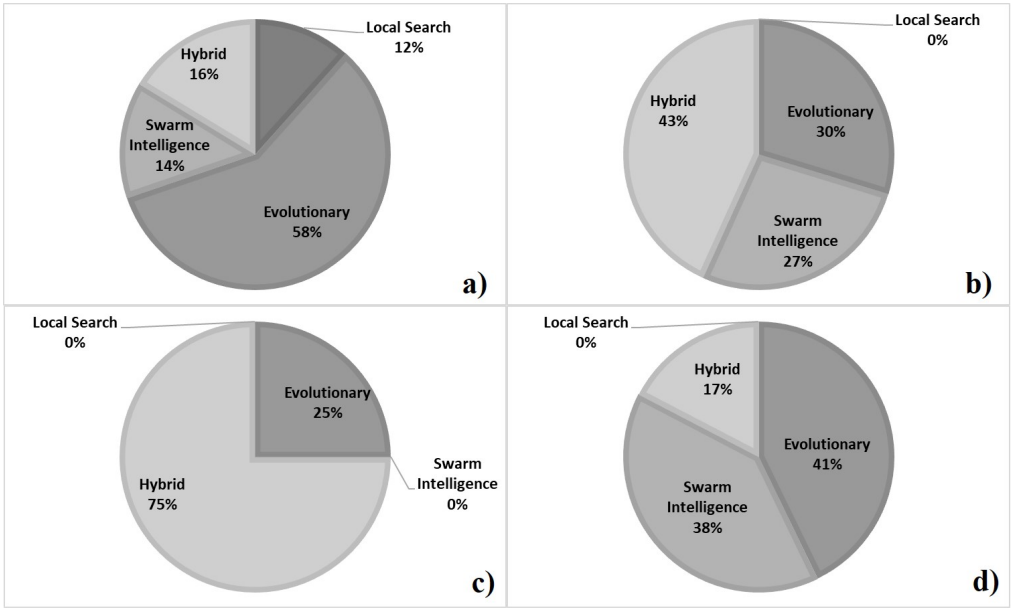


Figure 5.1: Distribution of studies based on SBA categorization (a) Effort estimation (b) Defect prediction (c) Maintainability prediction (d) Change prediction

The observations indicate that the evolutionary category of techniques are the most popular ones. Their favourableness is attributed to their simplicity and adaptability with which they can incorporate change according to variation in environment. Moreover, such techniques can be easily modeled to optimize more than one fitness function, generating globally good quality acceptable solutions. The GP technique is commonly used for developing effort estimation models, as it does not make pre-assumptions about the distribution of underlying data and can efficiently operate on a

small portion of training set to generate effective rules [177, 183, 220]. Also, GP can take into account whether the source of data is internal to an organization or outside the organization by using appropriate multipliers, thus, building effective prediction models [47].

The next most popular category of techniques was Hybrid. As discussed above, these techniques encapsulate the strengths of their constituent techniques to provide effective approaches for SEPM . Swarm intelligence techniques are efficient, as they are based on collective behaviour and are easy to implement. However, they have not been extensively explored for SEPM. These trends also show restricted use of local search techniques for SEPM.

The local search techniques were the least popular of all, as they were only explored in effort estimation studies. Local search technique such as Hill climbing needs many restarts to find globally optimal solutions [6]. The local search technique, Tabu Search (TS) though effective, requires the definition of problem-specific memory-based strategies which are efficient. More advanced techniques came up which could be the reasons for the unpopularity of these local search techniques.

***Favourable characteristics of SBA***

The general characteristics of SBA which make them suitable in the domain of SEPM have been extracted from primary studies, which are mentioned as follows:

- SBA are typically suitable in scenarios when there is a large population of candidate solutions (ES33). This is because they are capable of searching multiple regions of the search space in parallel for ascertaining optimum solutions (ES25, DS15). This characteristic also ensures global optimum solutions as candidate solutions at various regions of the search space are explored simultaneously, thus, avoiding local optima.

- SBA employ various performance measures as fitness functions to evaluate the

quality of candidate solutions. This specific characteristic make SBA ideal for developing SPM as they can find a good solution by optimizing these performance measures (ES23, ES31, ES34, DS31, DS25).

- Many SBA such as GA, GP and PSO do not require any pre-assumptions about the training data. This characteristic makes them suitable candidates for developing prediction models from historical training data, as the data does not need to follow specific assumptions (ES11, ES25, ES42, DS29, DS31).

- SBA are robust in nature and are effective in scenarios when inaccurate or noisy training data is available (ES40, DS37, CS9).

- While developing SPM, researchers may need to optimize various conflicting constraints to guarantee an effective solution. SBA with multiple objectives are ideal for such situations (ES31, DS13, DS30, DS33).

- Hybrid SBA combine the advantages of all their constituent approaches into one. Such approaches may provide better model accuracy, faster convergence, comprehensibility etc. An ideal hybridization would be that of a specific ML technique and a search-based algorithm. For instance, in Genetic Algorithm-Support Vector Machine (GA-SVM), GA may be used for tuning the parameters of an ML technique such as SVM, where SVM is used for developing a prediction model. Thus, such strengthened hybridized SBA are ideal for SEPM (DS34, CS9).

### 5.2.3 Results Specific to RQ2

In order to effectively develop a SPM, it is important to analyze and explore various experimental settings such as validation techniques, the number of runs to address the stochastic nature of SBA and the performance measures used to assess the developed

SPM. This RQ explores the most commonly used experimental settings described in the literature for SEPM using SBA.

***Validation Technique (RQ2.1)***

In order to develop a predictive model and assess its effectiveness, it is crucial to use an efficient validation method. The most common validation methods used by the primary studies were LOOCV, K-fold cross validation, hold-out cross validation and cross-project validation. The K-fold cross validation is the most frequently used method, which was used in about 52% of primary studies.

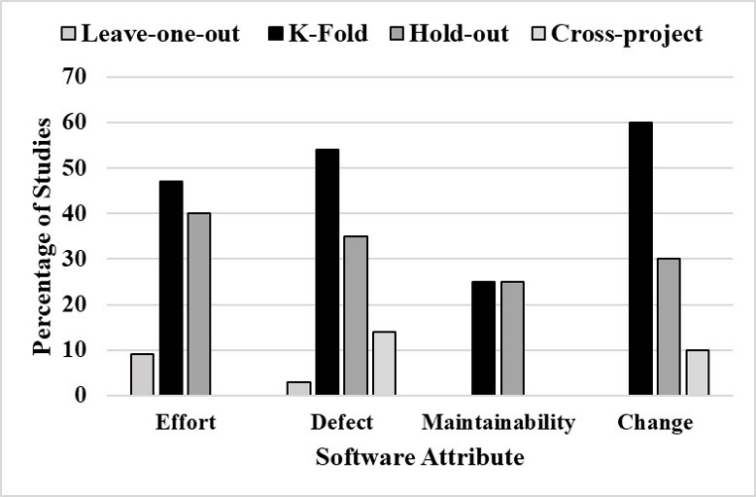


Figure 5.2: Validation Methods in Literature Studies using SBA

Figure 5.2 depicts the distribution of primary studies using different validation methods, according to a specific quality attribute. These attributes are described below:

*Effort Estimation*: The hold-out validation method was used in 40% of the effort estimation studies, while the LOOCV method was used by 9% of the effort estimation studies. Around 47% of the effort estimation studies used the K-fold cross validation with K=3 or K=10.

*Defect Prediction*: As shown in Figure 5.2, 54% of defect prediction studies used

K-fold cross validation method. The K value was taken as 10 in all the defect prediction studies except DS17 (K=3). Also, 35% of defect prediction studies use hold-out validation. Only one study used the LOOCV method. It may be noted that 14% of defect prediction studies also used cross-project validation.

*Maintainability Prediction*: Around 25% of maintainability prediction studies used the ten-fold cross validation and LOOCV methods respectively. Other maintainability prediction studies did not mention their validation method.

*Change Prediction*: The majority (60%) of the change prediction studies used K-fold cross validation technique (K=10). Three change prediction studies used hold-out validation while only one change prediction study used cross-project method.

Although, the hold-out validation technique is used in a large number of studies, the most commonly used technique according to Figure 5.2 is ten-fold cross-validation. This is because the hold-out validation technique exhibits higher variance in results. For example, if we have a single hold-out set with 25% of data for testing and 75% for training, the test set is relatively small and would lead to a lot of variations in results due to different partitions of data for training and test sets. Since ten-fold cross validation averages the results obtained over ten partitions, it reduces variance and the data is not sensitive to partitioning. This gives an accurate estimate of performance. However, a better scenario would be to perform repeated ten-fold cross validation where the data is re-partitioned for every round to increase the number of estimates, before an average is obtained. The use of cross-project should also be encouraged as it supports the generalizability of results.

***Runs to account for stochastic nature of SBA (RQ2.2)***

All the SBA are stochastic in nature. According to Ali et al. [238], it is essential to perform 10 or more runs in order to effectively address the stochastic nature of the SBA. Out of 91 primary studies, 62% of the studies explicitly mentioned that they

performed multiple runs for accounting for the stochastic nature of the SBA.

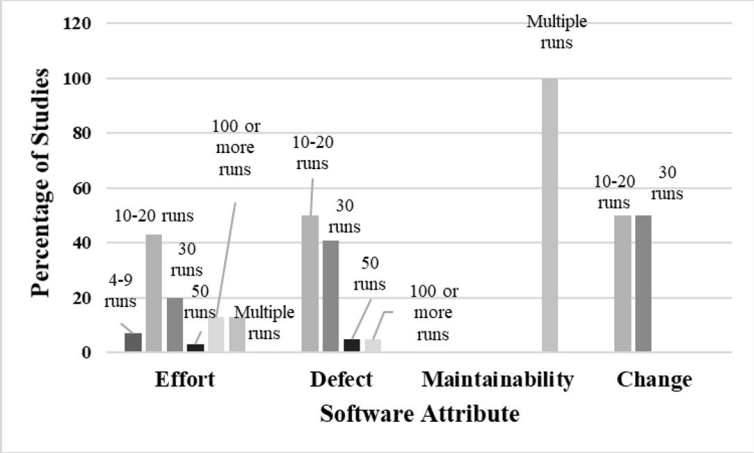


Figure 5.3: Distribution of primary studies according to number of runs

Figure 5.3 shows the percentage distribution of primary studies using different number of runs to account for the stochastic nature of SBA. A detailed description of the usage of different number of runs is as follows:

*Effort Estimation*: Around 70% of the effort estimation studies mentioned that they accounted for the stochastic nature of SBA by performing multiple runs. Out of these 70% studies, a majority (43%) of the studies performed 10-20 runs and 20% of these studies performed 30 runs. Only one effort estimation study performed 50 runs. 7% of the effort estimation studies, which try and account for the stochastic nature of SBA, used only 4-9 runs and 13% of these studies used 100 or more runs.

*Defect Prediction*: 59% of defect prediction studies mentioned that they performed multiple runs to account for the stochastic nature of SBA. Out of these, 59% of the studies, 41% of the studies used 30 runs and 50% of the studies used 10-20 runs. 50 runs and 100 or more runs were performed by one defect prediction study each.

*Maintainability Prediction*: Only one maintainability prediction study mentioned that it accounted for the stochastic nature of SBA by performing multiple runs. How-

ever, it did not mention the number of runs it performed.

*Change Prediction*: Only 40% of change prediction studies reported that they performed multiple runs. Out of these studies, 50% of the studies conducted 10-20 runs, and the remaining 50% conducted 30 runs.

It should be noted that 38% of the primary studies do not explicitly mention whether they performed multiple runs to account for the stochastic nature of SBA or not.

***Performance Measures (RQ2.3)***

The performance and estimation accuracy of different SBA can be evaluated with the help of various performance measures, which are used to assess the effectiveness of the SPM developed by these techniques. Figure 5.4 shows the most frequently used performance measures in effort estimation, defect prediction, maintainability prediction and change prediction studies along with the percentage use of these measures. A detailed description of the usage of different performance measures for each quality attribute is as follows:

*Effort Estimation*: Most of the effort estimation studies (81%) used the Mean Magnitude of Relative Error (MMRE) as a performance evaluator for effort estimation models. Around 67% of effort estimation studies used Pred (25) for assessing the performance of the developed effort estimation models. Other commonly used performance evaluators were the Median Magnitude of Relative Error (MdMRE), Magnitude of Relative Error (MRE), Mean Square Error (MSE) and coefficient of regression ($R^2$). Around 47% of the effort estimation studies also used other miscellaneous performance evaluators such as the Adjusted Mean Square Error (AMSE), Balanced Mean Magnitude Relative of Error (BMMRE), Root Relative Square Error (RRSE), Magnitude of Relative Error Relative to the Estimate (EMRE), and Variance Accounted For (VAF) amongst others. All these performance measures were used in

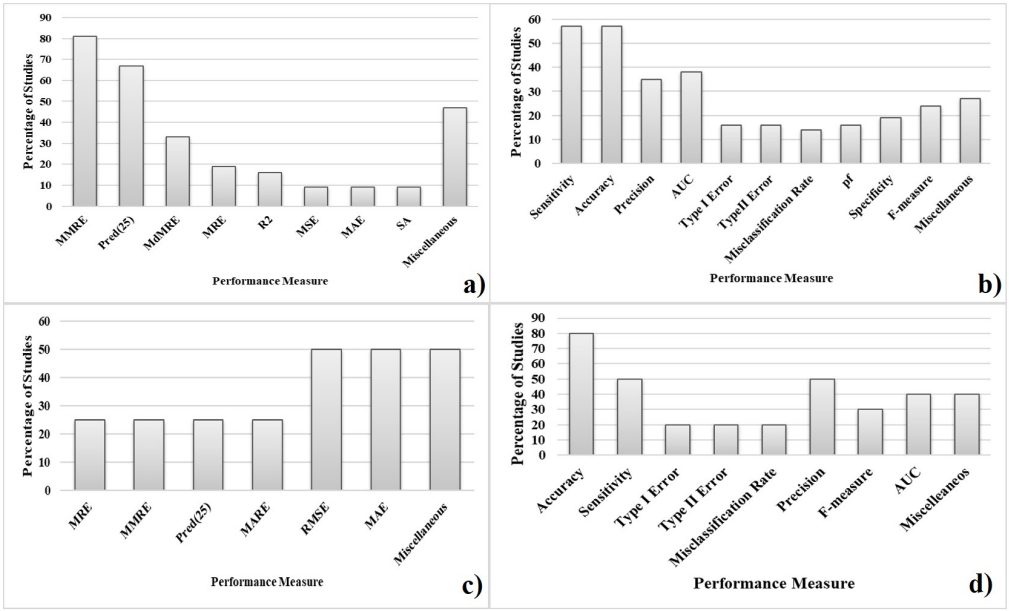


Figure 5.4: Performance measures for (a) Effort estimation (b) Defect prediction (c) Maintainability prediction (d) Change prediction

very few studies and were thus clubbed into the miscellaneous category.

*Defect Prediction*: The most commonly used performance evaluators in defect prediction studies were sensitivity and accuracy, used in 57% of studies each. Precision and AUC performance measures were used in 35% and 38% of the studies respectively. Type I and Type II errors, misclassification rate, specificity, F-measure, and PF were other commonly used performance measures. Some other performance measures grouped under the miscellaneous category were G-mean, weighted accuracy, support, coverage, Balance, and Normalized Expected Cost of Misclassification (NECM) amongst others.

*Maintainability Prediction*: As there are only four maintainability prediction studies, it is difficult to find the most commonly used performance measures. Some of the used performance measures are MRE, MMRE, Pred(25), Mean Absolute Relative Error (MARE), Root Mean Square Error (RMSE) and Mean Absolute Error

(MAE). Pred(30) and Standard Error of the Mean (SEM) were grouped under the miscellaneous category performance evaluators.

*Change Prediction*: Accuracy was the most commonly used performance evaluator in change prediction studies (80%). Other frequently performance measures were sensitivity, precision, F-measure, AUC, Type I and Type II error rates and misclassification rate. The G-measure, G-mean, j-index and specificity were grouped under the miscellaneous category of performance evaluators for change prediction models.

It can be seen that as effort estimation and maintainability prediction are regression problems, they use a number of common performance measures such as MMRE and Pred(25). While estimating effort, the most popular performance measures have been different variants of relative errors such as MMRE, Pred (25) etc. However, these measures have been recently criticized for their biased nature [239, 240]. Use of performance measures such as standardized accuracy [241], which provides an unbiased and realistic estimate of the developed effort estimation model should be encouraged.

Also, it may be noted that Accuracy is a traditional performance measure used by defect prediction as well as change prediction studies. It is easy to compute using a confusion matrix. Apart from accuracy, certain other performance measures such as sensitivity, precision, F-measure, specificity and PF can also be computed using a confusion matrix. However, the use of accuracy and precision has been discouraged while evaluating a binary variable as the class imbalance problem needs to be considered [118, 119, 242]. Recent studies advocate the use of stable performance measures such as the AUC, and the G-Mean1 for evaluating models trained by imbalanced datasets [17, 38, 121].

### 5.2.4 Results specific to RQ3

SBA evaluate the goodness of a number of solutions in order to search for an optimal or near optimal solution. The fitness function is used to evaluate the efficiency of a specific solution. Different studies use different fitness functions for evaluation that are generally based on performance measures. This section gives a brief description of the most commonly used fitness functions for effort estimation, defect prediction, maintainability prediction and change prediction studies. 79% of primary studies explicitly stated the fitness functions used for evaluating the goodness of a solution. We summarize the trends of the evaluated fitness functions in primary studies with respect to each investigated software attribute.

*Effort Estimation*: Figure 5.5 depicts the percentage of effort estimation studies using a specific fitness function. According to the figure, MSE and MMRE were the most commonly used fitness functions - 26% and 30% respectively. Other fitness functions used were the MdMRE, VAF, Least Absolute Deviation (LAD) and certain combinations of MMRE and Pred (25). A few studies (7%) used multi-objective fitness functions to evaluate a solution. For example, Logarithmic Standard Deviation (LSD), MMRE and Pred (25) were used collectively to evaluate the fitness of a solution. Some studies used other miscellaneous functions, such as Pred (25) (ES12, ES18) and MAE (ES37) amongst others.

*Defect Prediction*: Figure 5.6 shows the percentage of studies using a particular fitness function for defect prediction. According to the figure, 16% of defect prediction studies use accuracy and its variants as fitness functions. The misclassification cost and its variants are also used for ascertaining the goodness of a defect prediction model in 22% of the studies. The F-measure was used as fitness function in 14% of studies. The multi-objective function of specificity and sensitivity, and the product of specificity and sensitivity were each used by 8% of defect prediction studies. Other

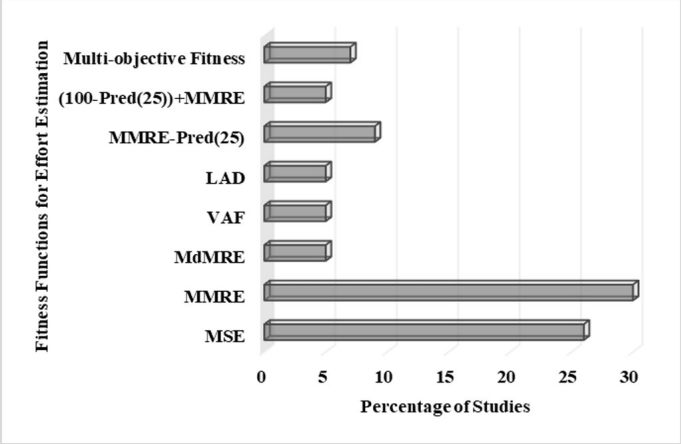


Figure 5.5: Fitness functions of Effort estimation studies

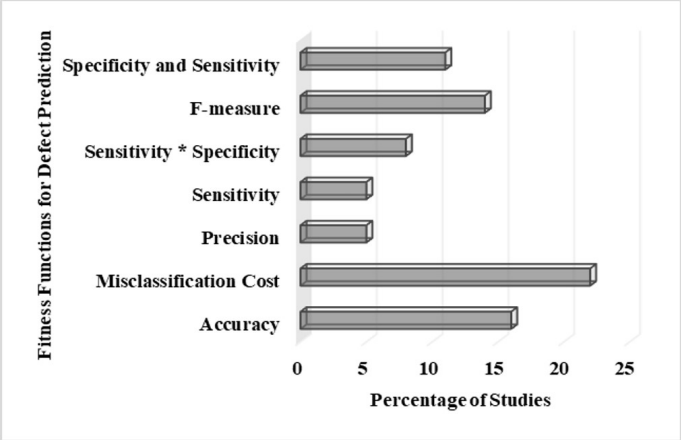


Figure 5.6: Fitness functions of Defect prediction studies

fitness functions used for defect prediction studies were precision and sensitivity. A number of defect prediction studies also used other miscellaneous fitness functions such as the product of F-measure and G-Mean1, only G-Mean1, the mean absolute percentage error, and the lift.

*Maintainability Prediction*: Only one maintainability study (MS3) specified its fitness function which was the inverse of RMSE.

*Change Prediction*: Seven change prediction studies (70%) clearly stated the fitness functions used by them while developing change prediction models. CS1 used

the misclassification costs for fitness purpose, while the CS2 used a multi-objective fitness function which includes the tree size along with the misclassification cost for evaluation. CS3 used Accuracy, J-index and average number of rules per rule set as fitness criteria. CS4 used a function which reported the proportion of correct predictions to incorrect predictions. CS5 used the absolute difference between actual and predicted outcome as fitness. CS9 employed various fitness functions for the nine SBA used in the study. CS10 also specified its fitness function.

It can be seen from the above discussion that a wide variety of performance measures have been employed as fitness functions for developing various SPM. However, a researcher should explore a number of different performance measures for the same search-based algorithm as fitness functions as a change in fitness function may lead to a significant change in results [47]. Moreover, only 8% of primary studies used multi-objective fitness functions; the use of multi-objective fitness functions should be explored as it helps in creating balanced and stable models by optimizing several constraints simultaneously.

### 5.2.5 Results specific to RQ4

The performance of different SBA is assessed by reporting the values of various performance measures to ascertain their usefulness for developing SPM. To derive generalized results, we follow the same rules as mentioned in section 3.4.4 (Chapter 3). We report the results dataset-wise after removing outliers (Appendix C.6).

*Effort Estimation*: As discussed in Section 5.2.3, MMRE and Pred (25) are the most commonly used performance measures for assessing effort estimation models. A good effort estimation model will have low MMRE values and high Pred (25) values. In order to analyze the predictive performance of different SBA, we extracted the values of MMRE and Pred (25) on different datasets used in different effort es-

timation studies. Some of the most commonly used effort estimation datasets were academic projects by Dolado, Desharnais dataset, Finnish dataset, NASA software projects, ISBSG dataset, COCOMO dataset and IBM DP dataset amongst many others. Figure C.3 (Appendix C.6) illustrates the outliers, associated with these two performance evaluators, for different SBA when applied on different datasets. These outliers were removed before further analysis.

Table 5.2: Results of SBA for Effort estimation models

| SBA | Count | Performance Measure | Min. | Max | Mean | Median | S.D. |
|---|---|---|---|---|---|---|---|
| TS | 4 | MMRE | 0.21 | 0.75 | 0.48 | 0.47 | 0.22 |
| | 4 | Pred(25) | 14.00 | 72.00 | 39.00 | 35.00 | 24.34 |
| PSO | 7 | MMRE | 0.01 | 0.64 | 0.40 | 0.39 | 0.22 |
| | 6 | Pred(25) | 38.00 | 69.00 | 51.33 | 49.50 | 12.19 |
| GA | 10 | MMRE | 0.05 | 0.69 | 0.34 | 0.33 | 0.22 |
| | 10 | Pred(25) | 30.00 | 98.00 | 73.53 | 77.50 | 22.66 |
| GP | 10 | MMRE | 0.09 | 1.58 | 0.54 | 0.48 | 0.38 |
| | 16 | Pred(25) | 32.40 | 94.40 | 60.57 | 63.25 | 22.95 |
| GA-SVM (Linear) | 6 | MMRE | 0.09 | 0.66 | 0.32 | 0.29 | 0.21 |
| | 6 | Pred(25) | 56.25 | 94.44 | 75.53 | 74.21 | 16.93 |
| GA-SVM (RBF) | 6 | MMRE | 0.09 | 0.45 | 0.28 | 0.30 | 0.14 |
| | 6 | Pred(25) | 66.67 | 94.44 | 79.11 | 74.80 | 11.92 |
| GA-ANN (RBF) | 5 | MMRE | 0.12 | 0.33 | 0.23 | 0.19 | 0.09 |
| | 6 | Pred(25) | 61.67 | 94.44 | 77.75 | 76.74 | 14.14 |

Table 5.2 describes the count of datasets, minimum (Min.), maximum (Max.), mean, median and standard deviation (S.D.) values of the MMRE and Pred (25) performance measures, after removing the outliers in order to reduce bias. According to the table, the Genetic Algorithm- Artificial Neural Network (GA-ANN) technique gave the best average MMRE value of 0.23, while the GA-SVM (RBF) gave the best Pred (25) value of 79.11. The GP technique gave the worst MMRE average score of 0.54, while the TS technique gave the worst Pred (25) value of 39. It may also be noted that, the mean MMRE values of all the reported techniques range between 0.23-0.54. Also the mean Pred (25) values of the techniques, apart from the TS technique, range between 50%-80%. The poor results of the TS technique could be

due to its local nature. Thus, the use of SBA is encouraging as the reported ranges are close to the acceptable level (Pred (25) $\geq$ 75% and MMRE $\leq$ 0.25) for effort estimation models as stated by Conte et al. [243].

*Defect Prediction*: As investigated, accuracy and sensitivity are the most commonly used performance measures for defect prediction studies. However, we also use AUC, as it is a stable performance measure for evaluating models developed using imbalanced datasets. A number of datasets were used for developing defect prediction models which included the telecommunication dataset, NASA datasets (CM1, PC1, PC2, PC3, PC4, KC1 with modules, KC1 with classes, KC2, JM1, KC3, MW1) which are publicly available in the PROMISE repository and various open-source datasets such as Lucene, Ant, Camel, Tomcat, Jedit etc. Appendix C.6 presents the outliers with respect to different SBA, when reported on different datasets.

Table 5.3: Results of SBA for Defect prediction models

| SBA | Count | Performance Measure | Min. | Max | Mean | Median | S.D. |
|---|---|---|---|---|---|---|---|
| MOPSO | 5 | Accuracy | 77.79 | 84.81 | 81.60 | 80.90 | 3.05 |
| | 6 | AUC | 0.72 | 0.85 | 0.78 | 0.78 | 0.05 |
| AIRS1 | 6 | Accuracy | 71.67 | 89.40 | 81.20 | 81.46 | 6.36 |
| | 4 | Sensitivity | 22.40 | 40.20 | 30.63 | 29.95 | 9.20 |
| | 8 | AUC | 0.56 | 0.73 | 0.62 | 0.59 | 0.07 |
| AIRS2 | 7 | Accuracy | 72.93 | 91.37 | 81.58 | 80.70 | 7.25 |
| | 7 | AUC | 0.54 | 0.72 | 0.60 | 0.58 | 0.06 |
| AIRS2P | 7 | Accuracy | 71.98 | 91.86 | 82.42 | 82.02 | 7.27 |
| | 7 | AUC | 0.54 | 0.71 | 0.61 | 0.58 | 0.06 |
| CLG | 7 | Accuracy | 69.42 | 92.31 | 82.53 | 82.50 | 7.71 |
| | 6 | AUC | 0.49 | 0.62 | 0.53 | 0.52 | 0.05 |
| IM1 | 5 | Accuracy | 59.17 | 69.74 | 64.41 | 64.82 | 4.23 |
| | 7 | AUC | 0.64 | 0.73 | 0.69 | 0.70 | 0.03 |
| IM2 | 5 | Accuracy | 70.91 | 93.06 | 84.15 | 84.55 | 7.61 |
| | 7 | AUC | 0.50 | 0.74 | 0.57 | 0.50 | 0.11 |
| GP | 6 | Accuracy | 67.26 | 78.30 | 71.59 | 71.31 | 4.34 |
| | 6 | Sensitivity | 59.67 | 75.44 | 68.79 | 69.30 | 6.10 |
| GEP | 3 | Accuracy | 69.30 | 99.11 | 85.84 | 89.10 | 15.17 |
| | 3 | Sensitivity | 11.11 | 75 | 42.03 | 40.00 | 31.09 |
| | 3 | AUC | 0.55 | 0.77 | 0.67 | 0.69 | 0.11 |
| GA-SVM | 9 | Accuracy | 35.00 | 89.00 | 59.93 | 61.50 | 16.52 |
| | 9 | Sensitivity | 30.00 | 100.00 | 71.31 | 76.00 | 25.46 |
| MOPSO: Multi-objective particle Swarm Optimization; IM: Immunos | | | | | | | |

A good defect prediction model exhibits higher values of accuracy, sensitivity or AUC measures. Table 5.3 presents comparative results of the defect prediction models using the selected performance measures, after removing the outliers. It reports the minimum (Min.), maximum (Max.), mean, median and standard deviation (S.D.) values. According to the table, the GEP, GA-SVM and the Multi-Objective Particle Swarm Optimization (MOPSO) techniques gave the best results using to the mean accuracy (85.84%), the mean sensitivity (71.31%) and the mean AUC values (0.78) respectively. Most of SBA (except Immunos1 (IM1) and GA-SVM) gave good performance scores for the accuracy measure in the range 70%-85%. However, it can be noted that only the performance of the models developed by the MOPSO and the GEP techniques gave an acceptable value for the AUC measure in the range 0.7-0.8. Since MOPSO uses a multi-objective fitness function, it provides effective results for developing efficient defect prediction models.

*Maintainability Prediction*: Different SBA, such as the GA-ANN, NNEP and other evolutionary fuzzy rule learning and evolutionary symbolic regression techniques were used in different maintainability studies. Thus, we could not report the cumulative statistics for these techniques. The MMRE values, for maintainability prediction models developed using SBA, ranged between 0.25-0.39; while their Pred (25) values ranged up to 60%. These values indicate an acceptable performance of the SBA for developing maintainability prediction models.

*Change Prediction*: For change prediction studies, different studies used different SBA, such as the GA, GP, ACO, AIRS, IM99 and MPLCS among others. Hence, the statistics for the majority of techniques could not be reported. Only three SBA (GFS-SP, HIDER and NNEP) were found to be evaluated in two studies and more than three datasets. The predictive capability of these techniques are already discussed in Section 3.4.4 (Chapter 3). Their mean accuracy values ranged from 72%-74%. Most of the studies which evaluated SBA for change prediction found them to be

appropriate and their use was encouraged for SEPM.

### 5.2.6 Results specific to RQ5

This RQ investigates the comparative performance of different SBA amongst each other and with other ML techniques used for developing SPM. It should be noted that while comparing the performance of SBA, we analyzed the results dataset-wise. Moreover, the rules for selection of techniques are same as in section 5.2.5. Wilcoxon signed rank test with $\alpha = 0.05$ was conducted to statistically evaluate the comparison results.

*Effort Estimation*: We analyzed the performance of six SBA, namely, PSO, GA, GP, GA-SVM with Linear Kernel (GA-SVM (Lin)), GA-SVM with RBF kernel (GA-SVM (RBF)) and GA-ANN amongst themselves, and six other ML techniques ANN, CART, Case Based Reasoning (CBR), SVM with Linear kernel (SVM-Lin), SVM with RBF Kernel (SVM-RBF) and BG) for developing effort estimation models, according to the MMRE values.

These techniques were selected as they are commonly used in literature and sufficient data could be collected for comparison purposes for these techniques with other SBA . Figure 5.7 summarizes the comparison results. According to the figure, the GP technique, performed worse than all the techniques except two (ANN and CART). The GA technique performed better than 7 other techniques. However, the results were significant in only one case. Similarly, the PSO technique performed better than two other compared techniques. The results show good performance of the GA for developing effort estimation models as it was found better than most of the other compared techniques. This could be due to strengths of GA technique which include no assumptions about the underlying data, does not get trapped in local minima and performs a simplified automatic search [153, 184, 186, 188]. The PSO technique is

also effective, because as compared to GA it has less number of parameters which need adjustment. Similarly, the results of other SBA were also encouraging.

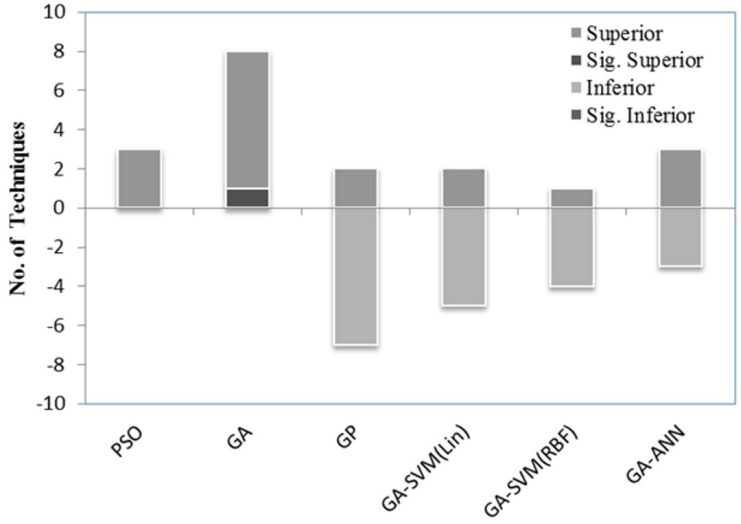


Figure 5.7: Wilcoxon test results of SBA comparison based on MMRE values

*Defect Prediction*: We analyzed the performance of nine SBA (MOPSO, AIRS1, AIRS2, AIRS2P, CLG, IM1, IM2, GP and GA-SVM) amongst themselves and with eight other ML techniques (BN, NB, RF, ANN, Repeated Incremental Pruning to Produce Error Reduction (RIPPER), C4.5, SVM, and Nearest Neighbor Algorithm with Non-nested Generalized Exemplars (NNGE)) for developing defect prediction models. Again, these techniques were chosen as they were evaluated by at least two primary studies on three datasets. We compared the defect prediction models based on accuracy and AUC values. The sensitivity values were not considered as very few techniques (only three) could be compared using the sensitivity values.

Figure 5.8 summarizes the comparison results. As seen in this figure, the IM1 and GP were the worst SBA according to the accuracy values. On the other hand, the IM2 and CLG were the worst SBA according to the AUC values. Our results are in accordance with [202] where Immunos algorithms were not found effective for

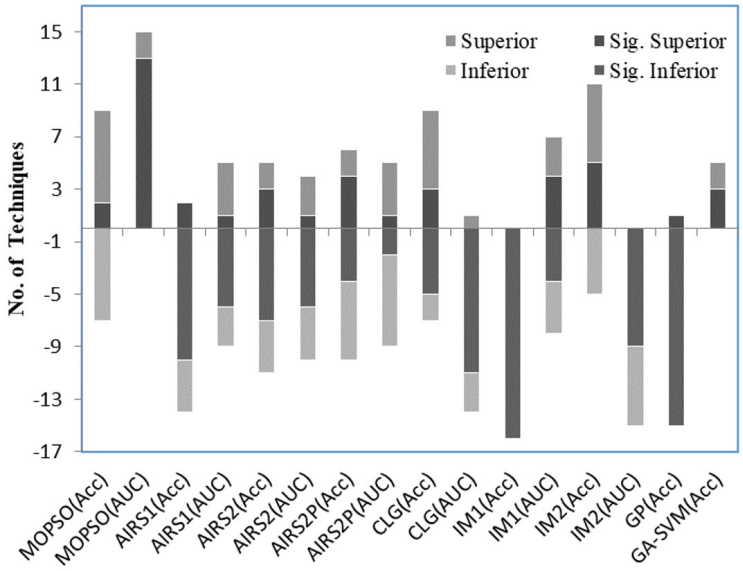


Figure 5.8: Wilcoxon test results of SBA comparison based on MMRE values

large datasets. Since the use of accuracy has been criticized in the literature due to the imbalanced nature of the datasets [118], we advocate the results obtained using the AUC technique. Thus, the MOPSO technique is the best search-based algorithm investigated in literature for developing defect prediction models, when evaluated using AUC values.

*Maintainability Prediction*: As there are very few studies for maintainability prediction, we did not have sufficient data to statistically compare the results of different techniques. MS2 compared two evolutionary fuzzy algorithms with DT, ANN and SVM and found the evolutionary fuzzy algorithms to be significantly better than most of the compared techniques. The study MS3 compared the neuro-genetic approach with the BN, ANN, regression tree and many other ML techniques. The approach was found promising. MS4 compared an evolutionary technique for maintainability prediction with Decision Table, BN, Radial Basis Network and sequential minimal optimization.

*Change Prediction*: As there were few change prediction studies, it was difficult

to collect sufficient data for effective comparison. However, only Hider, GFS-SP and NB were compared with AB as depicted in section 3.4.5 (Chapter 3). Though, these SBA were found poorer to AB, the results were not significant.

Certain primary studies which reported a comparative analysis of SBA in the domain of software change prediction are discussed. CS2 compared a standard GP approach with GP-based decision trees for developing change prediction models and found that the GP-based decision trees are better. Study CS3 compared the GA technique with the C4.5 technique to develop change prediction models and evaluated it on both balanced and unbalanced datasets. Their results indicate that the GA technique outperformed the C4.5 technique. CS4 statistically compared the ACO technique with the C4.5 technique for the change prediction and concluded that the ACO technique provided significantly better results. CS6 evaluated the capability of three swarm intelligence algorithms namely AIRS, IM99 and CLG with five ML techniques. However, the ML techniques were found superior to the investigated SBA (AIRS, IM99, CLG). Study CS7 compared six SBA, i.e., CPSO, HIDER, SUCS, NNEP, MPLCS and GFS-SP with a number of ML techniques, such as ANN, BG and RF, using the Friedman test. Although, the best performing technique was BG, the SBA showed promising results in developing change prediction models. CS9 compared nine SBA with four ML techniques (AB, LB, NB, BN) using accuracy and G-Mean3 performance measures. The study statistically evaluated the results of a hybridized search-based algorithm PSO-LDA as best when compared with ML techniques using Wilcoxon signed rank test.

### 5.2.7 Results specific to RQ6

It is important to statistically validate the comparative results of different techniques in order to provide an effective support to the reported conclusions [244]. 51% of the

primary studies used statistical tests for comparison amongst different techniques. Table 5.4 shows primary studies that have used various statistical tests.

Table 5.4: Statistical Tests for comparison of SBA

| Statistical Test | Study Identifier |
|---|---|
| T-test | ES2, ES3 |
| | DS2, DS12, DS15, DS19, DS32 |
| | MS2 |
| | CS6, CS10 |
| Wilcoxon Signed Rank Test | ES18, ES22, ES23, ES24, ES31, ES34, ES35, ES36, ES38, ES40, ES41, ES42 |
| | DS7, DS13, DS23, DS25, DS30, DS34, DS35, DS36 |
| | CS4, CS9 |
| Friedman Test | ES33, ES36, ES41 |
| | DS9, DS28, DS30 |
| | MS4 |
| | CS7 |
| ANOVA Test | ES10, ES29 |
| Mann-Whitney U-test | ES25 |
| Nemenyi Test | ES33 |
| | DS36 |
| McNemar's Test | DS17 |
| Kruskal Wallis Test | DS31, DS36 |
| Proportion Test | CS5 |
| Wilcoxon Rank Sum Test | DS33 |

A brief description of the use of different statistical tests is as follows:

*Effort Estimation*: Around 51% of effort estimation studies used a statistical test for comparison amongst various techniques. The majority of the effort estimation studies used the Wilcoxon signed rank test followed by the Friedman test, T-test and ANOVA test. Some other tests used for statistical comparison in effort estimation studies were the Mann-Whitney U-test and the post-hoc Nemenyi test.

*Defect Prediction*: 49% of defect prediction studies used statistical validation for comparing different techniques. Eight defect prediction studies used the Wilcoxon signed rank test, 5 studies used the T-test and three defect prediction study used the Friedman test for statistical comparative analysis. Other used statistical tests were McNemar's test, Nemenyi test, Kruskal-Wallis test and Wilcoxon rank sum test.

*Maintainability Prediction*: Two maintainability prediction studies used statisti-

cal tests (T-test and Friedman test) for effective comparison of different techniques.

*Change Prediction*: 60% of change prediction studies used statistical tests for a comparative analysis amongst different techniques. Two change prediction studies each used the T-test and the Wilcoxon signed rank test. One change prediction study each used the Friedman test and proportion test.

It may be observed that the majority of the studies use non-parametric statistical test such as the Wilcoxon signed rank test, Friedman test, Mann-Whitney U test, Proportion test, Kruskal-Wallis test and Nemenyi test as compared to parametric tests (T-test, ANOVA test and McNemar's test). This trend is similar to the one as observed in section 3.4.6 (Chapter 3), because parametric tests require stringent assumptions before their application. The trends, reported in the primary studies, also show that the T-test and the Wilcoxon signed rank test are the most commonly used tests in literature, i.e., most of the studies try and evaluate pairwise comparisons amongst different techniques. However, as the T-test requires large sample sizes and normal distribution, most studies use the Wilcoxon test for pairwise statistical comparisons if the differences amongst pairs is non-normal. It should also be noted that the Wilcoxon test can be used as a post-hoc test after the application of the Friedman test. This increases the number of studies, which choose it for the purpose of statistical analysis.

### 5.2.8 Results specific to RQ7

This RQ focuses on the threats to validity involved in the development of SPM when SBA are used. We list the probable sources of threats on four possible dimensions, namely conclusion, internal, construct and external. These threats were primarily extracted from the "Threats to Validity" sections of the primary studies which report them. It may be noted that a study may report threats which are specific to a study's

design. Thus, to avoid this bias in reporting of threats, we list threats which are reported by more than one study. Furthermore, we also list the remedial actions extracted from primary studies to mitigate the identified threats.

*Conclusion Validity*: These threats include all possible sources which may lead to improper conclusions, such as an incorrect association between predictor and outcome variables [9]. It is also termed as statistical conclusion validity. All the possible conclusion validity threats along with their mitigation, which are extracted from primary studies are mentioned in Appendix C.7 (Table C.3). The appendix also states the studies (SS) which mention the threats.

The various threats to conclusion validity include missing statistical verification of results, use of improper statistical test without verifying its underlying assumptions, not accounting for the randomness of SBA and validation bias. Another important threat to SEPM studies is the absence of baseline benchmark or expert evaluation. It is extremely important in prediction studies to compare any new proposed technique with respect to baseline benchmarks [245, 246]. Whigham et al. [246] pointed out that there are many previous studies which propose complex estimation approaches without comparing them with the basic ones. They found some of these proposals to be in fact comparable with basic techniques, thus necessitating a baseline comparison.

*Internal Validity*: All possible sources (extraneous variables) which may lead to a change in the outcome variable but are not themselves predictor variables may indicate a threat to internal validity [247]. All the identified internal validity threats, with their corresponding mitigation are mentioned in Appendix C.7 (Table C.4). They include an absence of proper tuning of internal parameters of SBA, not performing data cleaning steps and attribute selection and not accounting for the confounding effect of extraneous variables.

According to Arcuri et al. [248], parameter tuning has a critical impact on the per-

formance of the SBA. Before developing models, a researcher should first check the training data for any noise or inconsistency. Another possible threat is the existence of noisy and superfluous attributes. Researchers have ascertained that better prediction capability is achieved if such input attributes have been eliminated [109]. Lastly, though not specifically related to the application of SBA, but all empirical studies should mention extraneous attributes which could falsify the relationship between independent and dependent variables. The table also states the remedies corresponding to each threat, which are extracted from primary studies.

*Construct Validity*: Threat to construct validity exists, if the measures or metrics adopted by the study do not effectively represent the concepts they symbolize [9]. The various construct validity threats include the improper selection of independent variables and performance measures which pose a gap between theoretical and actual concepts and improper data collection. The threats are listed in Table C.5 (Appendix C.7) along with the studies that mention them and their corresponding remedies.

It is essential that the chosen independent variables are effective symbolizers of the actual metrics which are capable of predicting the dependent variables. Improper or incorrect independent variables will lead to erroneous models. Similarly, though a wide variety of performance measures are available to ascertain the effectiveness of the developed SPM, it is important to choose an effective indicator which is unbiased and provides a realistic estimate. Lastly, human errors in data collection may lead to misleading results if improper data is used for model training and evaluation. Thus, this threat should be accounted.

*External Validity* : It assesses whether the results of the study are valid in scenarios that are not evaluated for in the conducted study. It assesses the generalizability of the results [247]. The external validity threats are reported in Appendix C.7 (Table C.6), with their supporting studies (SS) and mitigation solution. Use of non-industrial, small sized and small number of datasets or mono-language datasets are

the various sources which affect the external validity of the experiment. Furthermore, if a study uses datasets extracted from software which belongs to a similar domain, a threat to external validity exists. In case a study does not provide enough details for replication, there is a possible threat to study's external validity.

### 5.2.9 Analysis of Review Results

This section states the current trends found with respect to the application of SBA for SEPM along with future guidelines to the researchers interested in the domain.

*Current Trends*: The trends with respect to the each investigated RQ is stated as follows:

- According to the studies that we analyzed, the evolutionary techniques were the most commonly used techniques for effort estimation and change prediction. However, the hybrid SBA were the most commonly used techniques for the defect prediction and maintainability prediction tasks. The trends indicate a limited use of local search techniques for SEPM, which could be because the local search techniques require considerable programming expertise, large set up time, and are not appropriate for goals involving a large number of variables.

- The most commonly used validation method, for SEPM using SBA, was the K-fold cross validation with a value of K=10. This method produces accurate estimates of a model's performance with low variance in results. A number of studies, which accounted for the stochastic nature of SBA by performing multiple runs, performed the number of runs between 10-20.

- In general, the most commonly evaluated and traditional performance measure for the quality attributes which are binary in nature is accuracy. However, there is a shift towards other stable performance measures such as the AUC, and the

G-Mean1. For variables which are continuous in nature, it is common to use estimates of errors such as the MMRE, and the Pred (25) for evaluating the performance of the developed model.

- The MSE and MMRE were the most commonly used fitness functions in effort estimation studies. Similarly, the accuracy and its variants as well as various combinations of Type I and Type II errors were the most commonly used fitness functions in defect prediction studies. However, different change prediction studies used different fitness functions for developing change prediction models. It is important for researchers to appropriately select a fitness function for achieving optimum results. This can be done by performing repeated studies which optimize different fitness functions for developing effective SPM and comparing their results in order to choose an optimum fitness function. Also, the use of multi-objective fitness functions should be explored as they help to balance out various constraints. This is possible by maximizing or minimizing more than one objective while developing SPM.

- For effort estimation tasks, a majority of SBA show an acceptable level of predictive capabilities with the MMRE values ranging between 0.23 - 0.54 and Pred (25) values ranging between 50 and 80. The predictive capability of SBA, such as MOPSO, is good for defect prediction with accuracy values ranging between 70%-85% and the AUC values ranging between 0.7 and 0.8. The MMRE values, for maintainability prediction models developed using SBA, were in the range of 0.25-0.39 and the Pred(25) values were up to 60%. SBA exhibited an acceptable performance for change prediction models as their mean accuracy values ranged between 72% - 74%.

- The results of the Wilcoxon test indicate that the GA technique outperforms a majority of other techniques when used for effort estimation tasks. However,

for the defect prediction tasks, the MOPSO technique works well. It outperformed most of the other SBA and the ML techniques when used with a majority of the datasets for defect prediction. Certain studies compared SBA with other techniques for developing maintainability prediction and change prediction models. The results of these studies support the use of SBA, as they performed better than the other compared techniques for developing these models in a majority of the studies. These results encourage the use of SBA as they are robust, provide global optimum results and easily adapt to changing circumstances.

- Statistical tests were used by 51% studies for the comparison amongst different techniques. The most frequently used test for comparing SPM using SBA was the Wilcoxon signed rank test due to its non-parametric nature.

- We identified 18 threats, which exist in studies which use SBA for SEPM. These threats were categorized into four dimensions viz. conclusion (5 threats), internal (4 threats), construct (3 threats) and external (6 threats). General good practices such as evaluating the statistical validity of the obtained results, use of appropriate number and size of datasets for empirical validation and use of other effective design parameters in a study can help in mitigation of most of the identified threats. Therefore, researchers should first properly design their experimental setups to reduce probable sources of threats.

*Future Guidelines*: Based on the results of the review, studies in the future should incorporate the following guidelines for using SBA in the domain SEPM:

- As indicated in the results, only 5% of the total primary studies analyzed the predictive capability of local SBA in the domain of SEPM. Therefore, there is an urgent need for future studies to evaluate the effectiveness of these techniques in this domain.

- According to the quality analysis, 20% of the studies did not completely state the parameter settings and fitness functions used for developing models using SBA. The specification of parameters and fitness function are very important to perform repeated and replicated studies. Thus, future studies should clearly describe the experimental settings in order to enable effective repeated application of the work.

- According to the results, 44% of the studies do not specify whether they performed multiple runs to account for the stochastic nature of SBA or did not perform appropriate number of runs. Future studies should account for the randomness of SBA by performing appropriate number of runs in order to produce accurate and repeatable results.

- Recent studies also advocate the use of other effective performance evaluators like Standardized accuracy for continuous variables and AUC, G-Mean1, etc. for binary variables. These performance measures are termed as stable since they provide reliable results even when imbalanced software quality data is available. Thus, future studies should incorporate multiple and stable performance evaluators for developed models.

- According to the results, 11% of the studies do not compare the capabilities of SBA with ML techniques and other SBA. Also, 28% of the studies does not perform any baseline comparisons. However, there is an urgent need for more studies to assess and evaluate the performance of SBA amongst each other, with well-established ML techniques and to conduct baseline comparisons of proposed SBA. Although, our results advocate the use of SBA for SEPM, comparative analysis is important to establish generalized results about their effectiveness.

- As a large number of studies (49%) did not use any statistical test for comparative analysis, future studies should statistically evaluate the significance of their results.

## 5.3 Experimental Design & Framework

This section discusses the experimental design for performing the empirical experiment, which evaluates the performance of eight SBA for developing change-proneness prediction models. The models developed by SBA are also compared with those developed using 4 ML techniques and a statistical technique. This section explains the variables used, the framework of the experiment, a description of the datasets along with data collection method and the performance measures which are used to validate the experimental results.

### 5.3.1 Independent and Dependent Variables

Similar to the experiments performed in Chapter 4, seven OO metrics were used as the independent variables, which include the CK metrics suite [16] and the SLOC metric. A detailed description of these metrics can be obtained from section 2.5.1 (Chapter 2). The dependent variable of the experiment is change-proneness.

### 5.3.2 Framework of the Experiment

The development of a prediction model involves a number of steps as discussed in this section:

1. *Data Collection*: In order to perform the experiment, we validate fourteen open-source datasets. These datasets include the six open-source datasets used

in Chapter 4 along with eight others. The datasets investigated in the chapter are AOI (2.7-2.9.2), Apollo (0.1-0.2), AviSync (1.1-1.2), Celestia (1.4.1-1.6.1), DrJava (r4668-r5686), DSpace (1.6.0-1.8.1), Eclipse (2.0-2.1), Frinika (0.2.0-0.6.0), Glest (1.0.10-3.2.2), Jmeter (2.8-2.9), PMD (3.9-4.3), Robocode (1.7.2.2-1.7.4.4), Simutrans (111.3-112.3) and Subsonic (2.0-4.6). The characteristics of each dataset can be referred from Appendix A.1.

2. *Feature Selection*: All the datasets obtained from step 1 undergo feature selection in order to obtain the most correlated features with the dependent variable i.e. change-proneness. Feature selection is advantageous as it decreases dimensionality, leads to a reduction of execution time and boosts the accuracy of prediction. We use the CFS method for doing so.

3. *Training & Validation*: Training and validation are two important processes for the establishment of a prediction model. The training process incorporates a classification technique (statistical, ML or SBA) to identify classification rules which can effectively distinguish between change-prone and not change-prone classes. The set of these rules forms the prediction model. The validation process tests these rules to predict and identify change-prone and not change-prone classes. The models were developed using eight SBA (CPSO, GA-ADI, GA-Int, GEP, HIDER, MPLCS, SUCS, XCS), 4 ML techniques (SVM, C4.5, CART, MLP-CG) and 1 statistical technique (LDA). The parameter settings and fitness functions of each of the investigated technique may be referred from Chapter 2 (Section 2.6). In order to validate the developed model, we perform ten-fold cross validation.

4. *Evaluation of Model's Performance*: In order to assess the performance of the prediction model we evaluate four performance measures namely specificity,

sensitivity, G-Mean1 and Balance. According to Ali et al. [238], multiple iterations (more than ten) are crucial for effectively handling the stochastic nature of SBA. Therefore, the study develops change prediction models using 30 runs and assesses the median values of the performance measures achieved over 30 runs. Reporting median values of 30 runs is a common practice in search-based software engineering literature. A similar practice has been followed by Hosseini et al. [41], Ferrucci et al. [47], Ryu and Baik [216], Harman et al. [38] and Canfora et al. [200]. Furthermore, the developed models are statistically assessed using Friedman test and post-hoc Wilcoxon signed rank test.

It may be noted that the different techniques investigated in the experiment are simulated in KEEL (Knowledge Extraction based on Evolutionary Learning) tool. We used the default parameter settings of the KEEL tool, `www.keel.es` for each technique. These parameter settings are mentioned in Section 2.6 (Chapter 2). Arcuri and Fraser [249] advocate the use of default parameter settings as parameter tuning is an expensive process. Moreover, parameter tuning may not yield significant result improvement in all cases. Thus, the use of default parameters is a practical choice.

## 5.4 Experimental Results & Analysis

This section discusses the experimental results which evaluate the use of SBA for determining change-prone classes and compares their results with other ML and statistical techniques.

### 5.4.1 CFS Results

Table 5.5 states the CFS results on eight datasets. It may be noted that we only state the results of those eight datasets, which were not used in Chapter 4. The CFS results

of the other six datasets used in Chapter 4 may be referred from Table 4.8 (Chapter 4).

Table 5.5: Metrics selected after application of CFS

| Dataset | OO Metrics Selected |
|---|---|
| Celestia | RFC, SLOC |
| Eclipse | CBO, SLOC, LCOM, RFC |
| Frinika | CBO, RFC, SLOC |
| Glest | RFC, SLOC, WMC |
| JMeter | CBO, NOC, SLOC |
| PMD | CBO, RFC, LCOM, SLOC |
| Simutrans | CBO, RFC, SLOC |
| Subsonic | CBO, RFC, SLOC |

After analyzing the results obtained on application of the CFS method on different datasets (Table 4.8 & Table 5.5) analyzed in this chapter, it was found that the CBO metric and the SLOC metric are the most commonly selected OO metrics which are highly correlated with change. Also, the RFC metric can be considered as a secondary indicator of change as it was selected by the CFS method in 9 out of 14 software datasets.

### 5.4.2 Ten-Fold Cross Validation Results

The sensitivity and specificity values obtained by change prediction models are mentioned in Table 5.6. The values shown in the table are the median values of all the 30 runs executed for a specific technique on a particular dataset. According to the table, the sensitivity values of the change prediction models of the Celestia dataset ranged from 60.14%-93.24% and the specificity values ranged from 26.42%-75.47%. The range of sensitivity and specificity values achieved on the DrJava dataset was 62.31%-76.88% and 61.58%-84.73% respectively. The change prediction models developed on the DSpace dataset exhibited sensitivity and specificity values in the range 78.05%-87.80% and 51.75%-75.44% respectively. The majority of change

prediction models developed on Simutrans, PMD and Glest datasets exhibited sensitivity values in the range 47.62%-73.59%, 73.75%-88.42% and 70.42%-87.32% respectively.

However, low sensitivity (0%-45%) values and high specificity (78%-98%) values were obtained in a majority of cases for change prediction models developed using Apollo, Robocode and AOI datasets. This was observed as the datasets only had 27%-30% change and thus the developed models could not learn well to accurately predict change-prone classes. Similarly, high sensitivity (86.17%-100%) and low specificity (0%-62.07%) values were observed on Subsonic dataset as only 24% of not change-prone classes were available while training on the dataset. Therefore, prediction models could not effectively learn the characteristics of not change-prone classes. It can be seen from Table 5.6 that the specificity and sensitivity values obtained by SBA models are comparable to that of ML models and LDA models.

Table 5.7 reports the median values (30 runs) of G-Mean1 (GM1) and Balance (Bal.) performance measures obtained by change prediction models developed on all the 14 datasets using all the investigated techniques. The table highlights the technique which achieves the best performance in a particular dataset. According to the table, the model which achieved the best values on the AOI dataset was developed using the MLP-CG technique and exhibited a G-Mean1 value of 0.71 and a Balance value of 67.49. The models developed by MPLCS, SUCS and CPSO also gave very good results with G-Mean1 and Balance values in the range 0.61-0.62 and 58.66-59.54 respectively. The MLP-CG technique also depicted best results on Apollo dataset (G-Mean1: 0.59, Balance: 56.71). These results were followed by the MPLCS technique (G-Mean1: 0.56, Balance: 53.42). Both CPSO and GA-Int

Table 5.6: Sensitivity & Specificity Median values of Ten-fold cross validation models

| Dataset/ SBA | AOI | | Apollo | | AviSync | | Celestia | | DrJava | | DSpace | | Eclipse | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *Sens.* | *Spec.* | *Sens.* | *Spec.* | *Sens.* | *Spec.* | *Sens.* | *Spec.* | *Sens.* | *Spec.* | *Sens.* | *Spec.* | *Sens.* | *Spec.* |
| **CPSO** | 46.56 | 79.54 | 26.09 | 77.05 | 70.37 | 73.91 | 85.81 | 28.30 | 63.16 | 72.41 | 86.59 | 54.39 | 83.28 | 55.27 |
| **GA-ADI** | 40.46 | 91.09 | 23.19 | 91.80 | 51.85 | 73.91 | 68.92 | 59.91 | 71.36 | 66.50 | 82.32 | 69.30 | 76.22 | 69.19 |
| **GA-Int** | 32.06 | 92.08 | 30.43 | 86.89 | 62.96 | 84.78 | 83.11 | 55.66 | 69.35 | 73.89 | 82.93 | 66.67 | 83.83 | 60.54 |
| **GEP** | 33.59 | 92.41 | 7.25 | 93.44 | 51.85 | 84.78 | 75.68 | 61.32 | 71.86 | 68.97 | 89.63 | 63.16 | 63.23 | 80.64 |
| **HIDER** | 28.24 | 94.06 | 8.70 | 97.81 | 37.04 | 91.30 | 81.42 | 50.00 | 67.34 | 73.89 | 82.32 | 67.54 | 81.57 | 61.87 |
| **MPLCS** | 43.51 | 88.78 | 34.78 | 90.71 | 40.74 | 82.61 | 62.16 | 73.58 | 72.86 | 67.00 | 85.37 | 65.79 | 76.32 | 69.19 |
| **SUCS** | 43.13 | 87.30 | 14.49 | 92.35 | 44.44 | 80.43 | 69.59 | 59.43 | 64.32 | 73.89 | 86.59 | 63.16 | 77.85 | 66.22 |
| **XCS** | 35.11 | 91.75 | 2.94 | 96.72 | 55.56 | 89.96 | 89.19 | 29.72 | 71.57 | 73.89 | 87.80 | 51.75 | 93.04 | 29.79 |
| **SVM** | 29.72 | 94.06 | 0.00 | 100.00 | 37.04 | 91.30 | 72.30 | 46.23 | 62.31 | 84.73 | 78.05 | 75.44 | 71.63 | 74.67 |
| **C4.5** | 16.79 | 97.69 | 4.35 | 97.27 | 48.15 | 78.26 | 78.38 | 69.81 | 68.84 | 72.41 | 85.37 | 65.79 | 82.23 | 62.95 |
| **CART** | 42.75 | 78.88 | 10.14 | 96.17 | 70.37 | 56.52 | 60.14 | 75.47 | 76.88 | 61.58 | 81.10 | 71.05 | 79.17 | 64.99 |
| **MLP-CG** | 54.96 | 90.76 | 40.58 | 85.25 | 51.85 | 60.87 | 75.00 | 46.23 | 70.35 | 71.43 | 87.80 | 66.67 | 80.84 | 63.92 |
| **LDA** | 29.01 | 92.41 | 11.59 | 95.08 | 37.04 | 82.61 | 93.24 | 26.42 | 67.84 | 70.94 | 80.49 | 74.56 | 75.73 | 67.25 |
| **Dataset/ SBA** | **Frinika** | | **Glest** | | **Jmeter** | | **PMD** | | **Robocode** | | **Simutrans** | | **Subsonic** | |
| | *Sens.* | *Spec.* | *Sens.* | *Spec.* | *Sens.* | *Spec.* | *Sens.* | *Spec.* | *Sens.* | *Spec.* | *Sens.* | *Spec.* | *Sens.* | *Spec.* |
| **CPSO** | 75.20 | 48.76 | 84.51 | 32.43 | 80.50 | 39.26 | 91.51 | 17.44 | 40.58 | 78.68 | 10.82 | 91.40 | 88.30 | 24.14 |
| **GA-ADI** | 57.09 | 72.73 | 70.42 | 75.68 | 76.25 | 44.45 | 77.22 | 61.63 | 39.13 | 92.39 | 64.50 | 77.78 | 90.43 | 55.17 |
| **GA-Int** | 62.99 | 66.94 | 77.46 | 70.27 | 85.50 | 37.78 | 79.54 | 69.77 | 44.93 | 89.85 | 65.37 | 74.55 | 92.55 | 37.93 |
| **GEP** | 71.65 | 64.46 | 78.87 | 78.38 | 87.00 | 33.33 | 83.40 | 63.95 | 31.38 | 90.36 | 61.47 | 79.57 | 86.17 | 55.17 |
| **HIDER** | 67.72 | 66.94 | 87.32 | 59.46 | 94.50 | 26.67 | 85.33 | 63.37 | 17.39 | 96.45 | 61.90 | 76.70 | 91.49 | 31.03 |
| **MPLCS** | 56.69 | 71.49 | 71.83 | 75.68 | 77.50 | 43.70 | 79.54 | 62.21 | 39.13 | 89.85 | 64.07 | 79.21 | 86.17 | 62.07 |
| **SUCS** | 62.99 | 70.25 | 81.69 | 67.57 | 78.00 | 44.45 | 81.47 | 72.09 | 37.68 | 87.82 | 50.22 | 84.23 | 87.23 | 48.28 |
| **XCS** | 70.87 | 56.20 | 95.77 | 14.87 | 93.50 | 8.89 | 87.60 | 53.49 | 42.03 | 90.36 | 60.61 | 80.65 | 100.00 | 0.00 |
| **SVM** | 53.54 | 76.86 | 80.28 | 91.89 | 95.50 | 5.93 | 73.75 | 73.26 | 20.29 | 95.43 | 54.11 | 86.02 | 91.49 | 24.14 |
| **C4.5** | 70.87 | 63.64 | 77.46 | 67.57 | 90.50 | 27.41 | 80.69 | 69.77 | 31.88 | 88.32 | 64.50 | 72.76 | 88.30 | 48.28 |
| **CART** | 79.53 | 38.84 | 83.10 | 59.46 | 90.50 | 29.63 | 88.42 | 57.56 | 42.03 | 84.26 | 73.59 | 62.37 | 90.43 | 34.48 |
| **MLP-CG** | 59.06 | 66.95 | 80.28 | 75.68 | 79.00 | 39.26 | 76.83 | 73.26 | 37.68 | 93.91 | 65.37 | 78.14 | 89.36 | 44.83 |
| **LDA** | 51.97 | 77.69 | 95.77 | 2.70 | 93.50 | 10.37 | 76.83 | 65.12 | 23.19 | 94.92 | 47.62 | 88.53 | 100.00 | 0.00 |

Sens. indicates Sensitivity & Spec. indicates Specificity

Table 5.7: G-Mean1 & Balance Median values of Ten-fold cross validation models

| Dataset/ SBA | AOI | | Apollo | | AviSync | | Celestia | | DrJava | | DSpace | | Eclipse | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *GM1* | *Bal.* | *GM1* | *Bal.* | *GM1* | *Bal.* | *GM1* | *Bal.* | *GM1* | *Bal.* | *GM1* | *Bal.* | *GM1* | *Bal.* |
| **CPSO** | 0.61 | 59.54 | 0.45 | 45.27 | **0.72** | **72.09** | 0.49 | 48.32 | 0.68 | 67.46 | 0.69 | 66.38 | 0.68 | 66.23 |
| **GA-ADI** | 0.60 | 57.36 | 0.46 | 45.38 | 0.62 | 61.28 | 0.64 | 63.64 | 0.69 | 68.83 | 0.76 | 74.95 | 0.73 | 72.48 |
| **GA-Int** | 0.54 | 51.63 | 0.51 | 49.94 | **0.73** | **71.69** | 0.68 | 66.45 | 0.72 | 71.53 | 0.74 | 73.52 | 0.71 | 69.85 |
| **GEP** | 0.56 | 52.73 | 0.26 | 34.25 | 0.66 | 64.29 | 0.68 | 67.69 | 0.70 | 70.38 | 0.75 | 72.94 | 0.72 | 71.44 |
| **HIDER** | 0.52 | 49.11 | 0.29 | 35.42 | 0.58 | 55.06 | 0.64 | 62.53 | 0.71 | 70.43 | 0.75 | 73.86 | 0.71 | 70.06 |
| **MPLCS** | 0.62 | 59.28 | 0.56 | 53.42 | 0.58 | 56.33 | 0.69 | 68.51 | 0.70 | 69.79 | 0.75 | 73.69 | 0.73 | 72.52 |
| **SUCS** | 0.61 | 58.66 | 0.37 | 39.30 | 0.60 | 58.35 | 0.64 | 64.15 | 0.69 | 68.74 | 0.74 | 72.28 | 0.72 | 71.44 |
| **XCS** | 0.57 | 53.83 | 0.17 | 31.33 | 0.70 | 67.25 | 0.50 | 49.31 | **0.73** | **72.71** | 0.67 | 64.81 | 0.53 | 50.11 |
| **SVM** | 0.50 | 48.01 | 0.00 | 29.29 | 0.58 | 55.06 | 0.58 | 57.23 | 0.73 | 71.25 | 0.77 | 76.71 | **0.73** | **73.10** |
| **C4.5** | 0.40 | 41.14 | 0.21 | 32.34 | 0.61 | 60.24 | **0.74** | **73.74** | 0.71 | 70.57 | 0.75 | 73.69 | 0.72 | 70.94 |
| **CART** | 0.58 | 56.85 | 0.31 | 36.41 | 0.63 | 62.80 | 0.67 | 66.90 | 0.69 | 68.29 | 0.76 | 75.55 | 0.72 | 71.20 |
| **MLP-CG** | **0.71** | **67.49** | **0.59** | **56.71** | 0.56 | 56.13 | 0.59 | 58.07 | 0.71 | 70.89 | 0.77 | 74.90 | 0.72 | 71.11 |
| **LDA** | 0.52 | 49.51 | 0.33 | 37.39 | 0.55 | 53.81 | 0.50 | 47.75 | 0.69 | 69.35 | **0.77** | **77.33** | 0.71 | 71.17 |
| **Dataset/ SBA** | **Frinika** | | **Glest** | | **Jmeter** | | **PMD** | | **Robocode** | | **Simutrans** | | **Subsonic** | |
| | *GM1* | *Bal.* | *GM1* | *Bal.* | *GM1* | *Bal.* | *GM1* | *Bal.* | *GM1* | *Bal.* | *GM1* | *Bal.* | *GM1* | *Bal.* |
| **CPSO** | 0.60 | 58.88 | 0.53 | 50.98 | 0.56 | 55.00 | 0.40 | 41.31 | 0.57 | 55.36 | 0.31 | 36.65 | 0.46 | 45.72 |
| **GA-ADI** | 0.64 | 63.21 | 0.73 | 72.92 | 0.58 | 57.25 | 0.69 | 68.45 | 0.60 | 56.62 | 0.71 | 70.39 | 0.71 | 67.59 |
| **GA-Int** | 0.65 | 64.91 | 0.74 | 73.62 | 0.57 | 54.82 | 0.74 | 74.19 | **0.64** | **60.40** | 0.70 | 69.61 | 0.59 | 55.80 |
| **GEP** | **0.68** | **67.86** | 0.79 | 78.62 | 0.54 | 51.97 | 0.73 | 71.94 | 0.54 | 51.35 | 0.71 | 69.16 | 0.69 | 66.83 |
| **HIDER** | 0.68 | 67.74 | 0.74 | 70.72 | 0.50 | 48.00 | 0.74 | 72.10 | 0.41 | 41.53 | 0.69 | 68.42 | 0.53 | 50.86 |
| **MPLCS** | 0.63 | 62.56 | 0.74 | 74.43 | 0.58 | 57.13 | 0.70 | 69.61 | 0.59 | 56.36 | 0.71 | 70.65 | **0.73** | **71.45** |
| **SUCS** | 0.64 | 63.17 | 0.76 | 74.58 | **0.58** | **57.45** | **0.77** | **76.31** | 0.58 | 55.10 | 0.65 | 63.07 | 0.65 | 62.33 |
| **XCS** | 0.63 | 62.80 | 0.37 | 39.72 | 0.29 | 35.41 | 0.68 | 65.96 | 0.62 | 58.44 | 0.70 | 68.96 | 0.00 | 29.29 |
| **SVM** | 0.64 | 63.30 | **0.86** | **84.92** | 0.24 | 33.4 | 0.74 | 73.50 | 0.44 | 43.54 | 0.68 | 66.08 | 0.47 | 46.02 |
| **C4.5** | 0.67 | 67.05 | 0.72 | 72.07 | 0.50 | 48.23 | 0.75 | 74.64 | 0.53 | 51.13 | 0.69 | 68.36 | 0.65 | 62.50 |
| **CART** | 0.56 | 54.40 | 0.70 | 68.94 | 0.52 | 49.79 | 0.71 | 68.89 | 0.60 | 57.52 | 0.68 | 67.49 | 0.56 | 53.18 |
| **MLP-CG** | 0.62 | 62.42 | 0.78 | 77.86 | 0.56 | 54.56 | 0.75 | 74.98 | 0.59 | 55.72 | **0.71** | **71.04** | 0.63 | 60.27 |
| **LDA** | 0.64 | 62.55 | 0.16 | 31.14 | 0.31 | 36.46 | 0.71 | 70.39 | 0.47 | 45.57 | 0.65 | 62.08 | 0.00 | 29.29 |
| GM1 indicates G-Mean1 & Bal. indicates Balance | | | | | | | | | | | | | | |

obtained optimum results on AviSync dataset with G-Mean1 values of 0.72 and 0.73 respectively and Balance values of 72.09 and 71.69 respectively. The change prediction models developed using the C4.5 technique obtained the best results on Celestia dataset. The next best results on Celestia dataset were given by the MPLCS technique (G-Mean1: 0.69, Balance: 68.51), followed by the GEP and GA-Int techniques. For DrJava dataset, the XCS technique gave optimum results with a G-Mean1 value of 0.73 and a Balance value of 72.71. The LDA technique obtained the best results on only one dataset i.e. DSpace with a G-Mean1 value of 0.77 and a Balance value of 77.33. The SVM technique obtained the best results on both Eclipse and Glest datasets. These results were closely followed by those obtained by the GA-ADI technique (G-Mean1: 0.73, Balance: 71.25) on the Eclipse dataset.

As depicted in Table 5.7, the best prediction models on Frinika were developed using the GEP technique (G-Mean1: 0.68, Balance: 67.86). Similarly, the best change prediction models on Robocode and Subsonic datasets were developed using the GA-Int technique and the MPLCS technique respectively. The SUCS technique obtained the best results on both Jmeter and PMD datasets with G-Mean1 values of 0.58 and 0.77 respectively. Though, an ML technique (MLP-CG) obtained the best change prediction results on Simutrans dataset (G-Mean1: 0.71, Balance: 71.04), these results were closely followed by MPLCS, GA-ADI, and GEP techniques.

According to the results in Table 5.7, it was observed that the model giving the best validation results in seven of the fourteen software datasets was the one developed using a search-based algorithm. Moreover, wherever models developed by the SBA did not give the best validation results, they performed fairly well and the results were competent to those obtained by ML and the statistical techniques. Moreover, the classification models developed using the SBA gave good G-Mean1 and Balance values. These results clearly assert effective use of SBA for predicting change-prone classes in a software.

### 5.4.3 Friedman Test Results

In order to ascertain the cumulative performance of different techniques used in the chapter on the fourteen datasets, we use Friedman test to assign ranks to each technique on the basis of median values (obtained on 30 runs) of the G-Mean1 and Balance performance measures. Table 5.8 states the average rankings of all the techniques as obtained by Friedman test in brackets. According to Table 5.8, the MPLCS technique was declared as the best technique with a ranking of 4.96 and 4.82 respectively with G-Mean1 and Balance performance measures. The second rank was obtained by the GA-Int and MLP-CG techniques according to Balance values and vice-versa according to G-Mean1 values. The worst performing technique was LDA according to the Friedman test results.

Table 5.8: Friedman Ranking of SBA based on G-Mean1 & Balance Values

| SBA ranks using G-Mean1 | SBA ranks using Balance |
|---|---|
| MPLCS (4.96) | MPLCS (4.82) |
| MLP-CG (5.14) | GA-Int (5.14) |
| GA-Int (5.18) | MLP-CG (5.21) |
| GA-ADI (5.32) | GA-ADI (5.29) |
| GEP (5.61) | GEP (5.75) |
| SUCS (6.29) | SUCS (6.25) |
| C4.5 (6.89) | C4.5 (7.04) |
| CART (7.54) | CART (7.50) |
| SVM (7.68) | SVM (7.75) |
| HIDER (7.89) | HIDER (8.39) |
| XCS (9.04) | XCS (8.89) |
| CPSO (9.57) | CPSO (9.14) |
| LDA (9.89) | LDA (9.82) |

The Friedman statistic has 12 degrees of freedom and was computed as 34.83 with a p-value of less than 0.001 when Friedman test was conducted on G-Mean1 values. Similarly, a chi-square value of 32.66 and a p-value of 0.001 was obtained when Friedman test was conducted on Balance values. This means that the Friedman statistic is true at a significance level of 0.05 and there are considerable differences

amongst various techniques. Thus, we reject the null hypothesis of the Friedman test and conclude that the techniques have different behavioral attributes and perform significantly different from each other for developing change prediction models.

### 5.4.4 Wilcoxon Test Results

As the results obtained by Friedman test were significant, thus we need to ascertain the pairwise differences amongst the performances of different techniques for developing change prediction models. In order to do so, we used Wilcoxon signed rank test with Bonferroni correction. The median values of 30 runs of G-Mean1 and Balance measures of all the investigated datasets were compared and the test was performed at a significance level of 0.05. The null hypothesis of the Wilcoxon test states that the performance of change prediction model developed using the MPLCS technique does not differ significantly from the performance of the change prediction models developed using other investigated techniques, when evaluated using the G-Mean1 or Balance performance measures.

Table 5.9: Wilcoxon Test Results

| Compared Pair | Using G-Mean1 | Using Balance |
|---|---|---|
| MPLCS vs CPSO | ↑ (0.006) | ↑ (0.008) |
| MPLCS vs GA-ADI | ↑ (0.222) | ↑ (0.187) |
| MPLCS vs GA-Int | ↑ (0.833) | ↑ (0.638) |
| MPLCS vs GEP | ↑ (0.593) | ↑ (0.307) |
| MPLCS vs HIDER | ↑ (0.045) | ↑ (0.026) |
| MPLCS vs SUCS | ↑ (0.258) | ↑ (0.096) |
| MPLCS vs XCS | ↑ (0.028) | ↑ (0.022) |
| MPLCS vs SVM | ↑ (0.116) | ↑ (0.096) |
| MPLCS vs C4.5 | ↑ (0.208) | ↑ (0.208) |
| MPLCS vs CART | ↑ (0.035) | ↑ (0.301) |
| MPLCS vs MLP-CG | ↓ (0.875) | ↓ (0.826) |
| MPLCS vs LDA | ↑ (0.006) | ↑ (0.006) |
| ↑: Not significantly better; ↓: Not significantly poor | | |

Table 5.9 states the results obtained by the Wilcoxon test. The table uses two symbols: "↑" indicates MPLCS is better than the other compared technique, but not

significantly. "↓" indicates MPLCS is worse than the other compared technique, but not significantly. According to the results of the Wilcoxon test shown in Table 5.9, it can be clearly seen that the MPLCS technique is better than all the other investigated techniques except the MLP-CG technique. However, these results were not significant in any case. Thus, we accept the null hypothesis and conclude that the results of MPLCS technique are comparable to other investigated techniques.

### 5.4.5 Analysis of Experiment's Results

The results stated in previous sections indicate good performance of SBA for development of classification models which identify change-prone classes in a software dataset. We also analyzed the average of median G-Mean1 and average of median Balance values obtained by the investigated techniques on all the datasets. The best average G-Mean1 value was obtained by the MPLCS technique (a search-based algorithm) with a value of 0.67. The performance of the MPLCS technique was closely followed by the GA-Int and the MLP-CG technique, with average G-Mean1 values of 0.66 each. It may be noted that the average G-Mean1 values of the other SBA were found comparable with the other evaluated ML techniques. The worst performance in terms of average G-Mean1 values was depicted by the LDA and the XCS techniques.

The average Balance values again established the superiority of the MPLCS technique (Mean Balance: 65.41) along with the MLP-CG technique (Mean Balance: 65.15). The GA-ADI (Mean Balance: 64.31), GA-Int (Mean Balance: 64.85), SUCS (Mean Balance: 63.21) and HIDER (Mean Balance: 59.70) techniques also show comparable mean Balance values with the other investigated ML techniques. The worst performers were LDA and XCS techniques. These results support the application of SBA for predicting change-prone classes.

The results of Friedman test indicate the MPLCS technique as the best for developing change prediction models followed by the GA-Int and the MLP-CG techniques. However, the worst rank was achieved by the LDA technique. The MPLCS technique is effective as its fitness function is based on the minimum description length principle [250]. This principle helps in developing effective rule sets for classification which are aimed at maximizing the accuracy of the classifier along with a decrease in complexity of the rules formed. Thus, this technique provides effective change prediction models in most of the datasets. Furthermore, according to the results of the Wilcoxon test, the change prediction models developed using the MPLCS technique were better than all the investigated techniques (except MLP-CG), but not significantly. This indicates that the MPLCS technique is effective and comparable to all the other techniques explored in the study. In fact, it yields better results in some cases when compared with other techniques.

## 5.5 Discussion

The goal of the chapter was two-fold: a) to systematically summarize empirical evidence with respect to the use of SBA for SEPM in literature studies and b) to ascertain the predictive capability of SBA for developing change-proneness prediction models.

In order to assess empirical evidence reported in literature, we conducted a systematic review of 91 primary studies which use SBA for predictive modeling in the domain of software effort estimation (43 studies), defect-proneness prediction (37 studies), maintainability prediction (4 studies) and change-proneness prediction (10 studies). After analyzing the 91 primary studies, we conclude that SBA can be used in three ways for SEPM: a) for performing feature selection, b) for deciding the weights, structure or parameters of other techniques, or c) for developing predictive models. The results of the review advocate the use of SBA for SEPM tasks. Af-

ter taking into account the current trends (Section 5.2.9), we propose the following future work for researchers who intend to use SBA for SEPM:

- Future work should explore and assess SBA for developing software effort estimation, defect prediction and especially maintainability prediction and change prediction models as we found only limited studies which use SBA in these domains. However, as the results of the review encourage the use of SBA, a large number of studies should be carried out for ascertaining the capability of these techniques.

- Apart from prediction accuracy, future work should focus on other aspects for evaluation of SBA. These aspects may include the cost effectiveness of the developed models, their comprehensibility, and their generalization capability. These evaluators would help in an effective and thorough evaluation of the capabilities of any search-based algorithm. Future work should focus on developing intuitive models for predicting various software attributes using SBA.

- Future work should evaluate the run-time consumed by SBA for SEPM tasks and explore alternatives for reducing it. Recent studies [251, 252] propose the use of parallel or cloud-based search-based software engineering for effective implementation of SBA. These implementations may lead to significant reduction in the running time for model development. Thus, active exploration and evaluation of such alternatives which will overcome the shortcomings of SBA and can aid in their efficient application to SEPM tasks should be undertaken.

- Future work should incorporate a number of other validation techniques such as inter-release validation, cross-project validation or temporal validation, which reduce validation bias and evaluates the developed models in real and practical scenarios.

- The fitness function of a search-based algorithm plays a crucial role in its performance. A change in fitness function leads to variation in results. Thus, proper specification of fitness functions is important for researchers to perform repeated studies. Also, future studies should evaluate different fitness functions and especially explore multi-objective fitness functions to determine their suitability and achieve optimum results.

- Future work should focus on the careful design of experiments which use SBA for SEPM tasks by taking into account the various threats to the validity to the results of the study. These threats include conclusion, internal, construct and external validity threats. Such a practice would lead to an effective application of SBA for predictive modeling tasks and would yield practical and reliable results which can be used by the software industry.

In order to investigate the applicability of SBA for software change prediction we performed an empirical study using fourteen open-source datasets. We used an effective experimental setup for our experiment by (a) investigating eight SBA (CPSO, GA-ADI, GA-Int, GEP, HIDER, MPLCS, SUCS, XCS) and comparing their results with 4 ML and one statistical technique; (b) completely specifying the parameter settings and fitness functions of the investigated techniques to aid replicability; (c) reporting median values after performing 30 runs to account for the stochastic nature of SBA; (d) using stable and robust performance measures (G-Mean1 and Balance) in order to yield reliable results; (e) using fourteen datasets belonging to different domains and of appropriate size, which were developed using Java and C++ languages to increase the generalizability of the obtained results and (f) statistically evaluating the results using Friedman and Wilcoxon tests to strengthen the conclusion validity. This experimental design was in line with the recommended guidelines obtained by the review (Section 5.2.9).

The primary results of the empirical study indicate that the MPLCS technique, a search-based algorithm outperformed all the other investigated statistical techniques, ML techniques and SBA as it secured the highest rank after application of Friedman statistical test. The validation results obtained by the MPLCS technique were closely followed by the GA-Int technique and the MLP-CG technique. Furthermore, the result of Wilcoxon test indicate that the performance of the MPLCS technique was better than the majority of the other techniques explored in the study, though not significantly. The superior performance of the MPLCS technique could be attributed to its effective fitness function which tries to balance the complexity as well as the accuracy of the rule set obtained for classifying change-prone classes.

The results clearly indicate better or in some cases comparable performance of SBA when compared with statistical and ML techniques. Hence, this empirical experiment approves of development of prediction models using SBA for identification of change-prone classes in OO systems. Such models can be effectively used and adapted by the software industry for application on current OO software projects, so that change-prone classes can be effectively determined.

# Chapter 6

# Software Change Prediction using Hybridized Techniques

## 6.1 Introduction

A wide array of classification techniques are available to develop effective software quality models which include statistical techniques, ML techniques and SBA amongst many others. Lately, a number of researchers in literature have advocated the use of SBA in the domain of SEPM [6, 157–159, 253, 254]. These techniques help in the identification of optimal solutions for a specific problem by testing the goodness of a large number of possible solutions. As indicated in Chapter 5, an extensive review of literature studies which use SBA for SEPM provides evidence of their effectiveness in this domain. Although, the results of Chapter 5 indicate that the evolutionary category of SBA are the most popular ones, there is an urgent need to evaluate the capabilities of HBT in the domain of SEPM. These techniques are another category of SBA, which combine SBA with ML techniques into a single approach. HBT may produce better results as they combine the advantages of both ML

techniques and SBA [255]. Incorporating the use of SBA as part of a larger system, may also lead to improvement in predictive performance and the time required for model development. Grosan and Abraham [256] recognized the importance of HBT and stated that their use would pave the way for generating optimal solutions. In predictive modeling, the combination of ML techniques and SBA may increase the speed of convergence of models under development. These characteristics of HBT motivate us to analyze their effectiveness in the domain of software change prediction. Furthermore, we also compare the performance of HBT with other categories of SBA, ML techniques and statistical techniques for predicting change-prone classes.

In order to investigate the effective application of SBA and HBT for determination of change-prone classes, we follow a generalized and repeatable approach in this chapter, similar to the one followed in Chapter 5. The approach includes providing the complete description of parameter settings and fitness functions and accounting for the stochastic nature of these techniques using multiple (30) runs. Moreover, the chapter provides statistical evidence for comparison amongst different techniques, which strengthens the obtained results. Furthermore, we evaluate both the CPU time as well as the predictive performance of a specific technique for model development. Hence, the chapter explores a trade-off between both these dimensions as they are critical parameters for a technique's selection. We compare and analyze the capability of 15 techniques (4 HBT, 6 SBA and 5 ML/statistical techniques) in this chapter for developing change-proneness prediction models. The HBT investigated in the chapter are NNEP, GFS-LB, DT-GA and PSO-LDA. Five of the SBA analyzed in the chapter were evolutionary techniques (XCS, SUCS, MPLCS, GA-ADI and HIDER), while one of them was a swarm intelligence technique (CPSO). We compared the performance of these HBT and SBA with the statistical technique LDA and four ML techniques (SVM, CART, MLP-CG and C4.5).

We investigate the following RQs in this chapter:

- RQ1: What is the comparative performance of HBT with SBA, SBA with ML /statistical techniques and HBT with ML/ statistical techniques for prediction of change-prone classes?

- RQ2: Which is the best hybridized technique for development of change prediction models? Which techniques perform statistically better or worse than the best chosen hybridized technique?

- RQ3: What is the comparative CPU time taken by change prediction models developed using different HBT vs SBA, SBA vs ML/ statistical techniques and HBT vs ML/ statistical techniques?

- RQ4: Is there any trade-off between the CPU time and predictive performance of different HBT, SBA and ML/ statistical techniques?

RQ1 aims to provide empirical evidence about the predictive capability of different SBA, HBT and ML/statistical techniques for the development of efficient change prediction models. The predictive capability of each category of techniques is compared with each other. We perform empirical validation on six application packages of open-source Android dataset.

We investigate RQ2 as it is important to statistically assess and establish results, which support and advocate the use of an effective technique for the development of change prediction models. We use Friedman statistical test on Balance, G-Mean1 and G-Mean3 values, in order to determine the technique exhibiting best predictive capabilities. Furthermore, we perform pairwise comparisons of all the other techniques with the best technique. This helps in evaluating which pairs of techniques are statistically significantly different than the chosen best technique using Wilcoxon signed rank test with Bonferroni correction.

The motivation for RQ3 is that CPU time for model prediction and validation is a

crucial factor for the selection of an appropriate technique. Thus, we evaluate various techniques according to the CPU time consumed for model prediction and validation.

We investigate RQ4 as an effective technique would be the one that exhibits good predictive capability and simultaneously takes low CPU time for model prediction and validation. A technique, which performs very well but consumes a lot of CPU time, may be less favorable over a moderately performing technique that takes very low CPU time.

The chapter is organized as follows: Section 6.2 states the experimental design, while Section 6.3 states the data preprocessing results. Section 6.4 states the results specific to each RQ. A comparison of previous chapter results and the results of previous literature studies is presented in Section 6.5. Section 6.6 provides a discussion of the overall findings of the chapter. The results of the chapter are published in [257].

## 6.2 Empirical Research Framework

We first present the various design considerations of the chapter followed by the experimental design.

### 6.2.1 Independent and Dependent Variables

The independent variables used in the chapter are 18 OO metrics. These OO metrics have been successfully used in previous studies for predictive modeling tasks [1, 27, 45, 79, 83] and are thus effective. The OO metrics include the CK metrics suite [16] (DIT, NOC, RFC, WMC, CBO and LCOM), five metrics of the QMOOD metrics suite [25] (MOA, DAM, MFA, CAM and NPM), afferent and efferent coupling (Ca & Ce) metrics [23] along with AMC, SLOC, LCOM3 [15], IC and CBM metrics.

The dependent variable analyzed in this chapter is change-proneness. The detailed description of dependent and independent variables may be referred from Chapter 2.

### 6.2.2 Empirical Data Collection

This chapter analyzes six packages of Android OS, a popular mobile operating system for empirical validation. Around 70% of smart phones in the market use Android as a preloaded OS. The open-source nature of Android increases the replicability of the study. Android forms a unique niche in the operating system market, but the results of the chapter are also generalizable to iOS, Windows and Unix as they belong to the domain of operating systems. Moreover, the wide use of Android aids the external validity of the results as they can be effectively used in similar scenarios. The data used in the chapter is collected by analyzing two versions of Android system: Ice Cream Sandwich and Jelly beans.

It was noted that the source code for Android OS was available under many application packages, rather than a single package. The various available application packages include packages for a number of functions such as libraries, kernel, or native applications. We investigated Bluetooth (4.3.1-4.4.2), Contacts (4.3.1-4.4.2), Calendar (4.3.1-4.4.2), Gallery2 (4.2.2-4.3.1), MMS (2.3.7-4.0.2) and Telephony (4.4.2-4.3.1) application packages. The six Android packages investigated were selected as a representative of Android application as they contained an appropriate number of data points i.e. classes which are required for developing effective change prediction models [1]. Secondly, the selected application packages had sufficient and generally varying degree of percentage of change-prone classes i.e. $19\% - 63\%$, which would help in developing appropriate prediction models and ascertaining the accuracy of these models when developed using varying degrees of change [1]. Certain other application packages such as “Desk Clock” and “Email” were also evaluated

as candidates for the study between the two Android OS versions but were not found suitable as they did not meet the above criteria. The details of the six application packages can be obtained from Appendix A. These datasets were collected with the aid of the DCRS tool.

### 6.2.3 Experimental Design

Figure 6.1 depicts the diagrammatic representation of the experimental design used in the chapter, which is described in detail as follows:

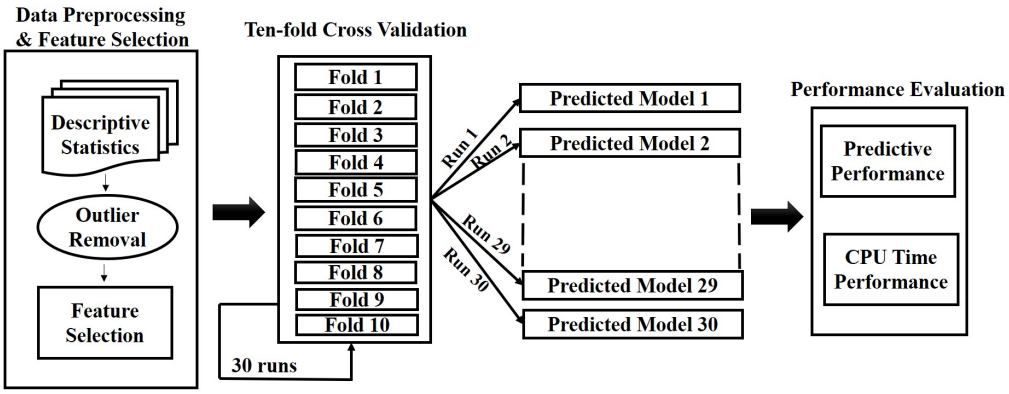


Figure 6.1: Experimental Design for Evaluating Software Change Prediction Models

- *Data Preprocessing & Feature Selection*: We first analyze the descriptive statistics of OO metrics of each dataset. Next, all the outliers of each of the six datasets were identified and removed using IQR filter. Finally, as suggested by Hall [109], we use the CFS method to effectively select the best predictor metrics out of the eighteen OO metrics used in the chapter.

- *Model Prediction & Validation*: Next, we develop change prediction models using all the techniques on the six datasets using ten-fold cross validation method. The method was selected as it reduces validation bias [37, 114]. We

run each technique 30 times in order to account for the randomness of SBA and HBT [38, 200].

- *Performance Evaluation*: The results of change prediction models developed by various techniques are evaluated using recall, PF, Balance, G-Mean1 and G-Mean3 measures. An extensive review by Malhotra [17] reveals that a large number of studies (67% and 32% respectively) report recall and PF values instead of reporting the actual number of FPs and FNs. In order, to make our results comparable, we use the above performance measures. Furthermore, it is also important to account for imbalance in a dataset, when evaluating the prediction model which learns from it. Therefore, we use G-Mean1, G-Mean3 and Balance measures as they are stable and robust. They yield a reliable estimate of the model's performance. We further use the Friedman and Wilcoxon signed rank test with Bonferroni correction in order to statistically analyze the results.

  We also evaluate the CPU time of a particular technique as it is an important indicator for evaluating the effectiveness of a technique. The CPU time is compared with the predictive capability of a technique in order to analyze the trade-off between both of them.

We used the KEEL tool `www.keel.es` for simulation of the 15 techniques investigated in the chapter. In order to effectively tune the parameters of each technique used in the chapter, a number of variations of different parameters were investigated before selecting the optimum parameters for each technique relative to Android datasets. The parameter settings for each of the investigated SBA and HBT are mentioned in Table 6.1. The fitness functions used for SBA and HBT were the default ones, which are mentioned in Chapter 2.

Table 6.1: Parameter Settings for SBA & HBT

| SBA/ HBT | Parameter Settings |
|---|---|
| GFS-LB | Number of label: 5; Number of rules: 25. |
| DT-GA | Instances per leaf: 2; Confidence: 0.20; Threshold for considering small disjunct: 10; Number of total generations for GA: 50; No. of chromosomes in the population: 200; Crossover Probability: 0.7; Mutation Probability: 0.01. |
| NNEP | Hidden Nodes: 6; Transfer: Product Unit; Generations: 200. |
| PSO-LDA | Maximum Number of Iterations: 400; Number of non-improving iterations: 150; Cognition learning factor: 0.8; Social learning factor: 1.2; Inertial weight (w): 0.5; Number of particles: 15. |
| HIDER | Population Size: 100; Number of Generations: 150; Mutation Probability: 0.3; Cross Percent: 70; Extreme Mutation Probability: 0.06; Prune Examples Factor: 0.05; Penalty Factor: 2; Error Coefficient: 0. |
| GA-ADI | Number of Iterations: 500; Number of strata: 2; Rule deletion min rules: 14; Size penalty min rules: 4; Number of intervals for uniform discretization: 4, 5, 6, 7, 8, 10, 15, 20, 25. |
| CPSO | No. of Particles: 25; Convergence Radius: 0.1; Weights Upper limit: 1.05; Max Uncovered Instances: 0.1; Indifference Threshold: 0.2; Constriction Coefficient: 0.83; Convergence Platform Width: 35. |
| MPLCS | No. of Iterations: 700; Size of Penalty Rules: 3; Probability of local search: 0.03; Probability of Rule Set-Wise (RSW) Cross-over: 0.2; No. of parents in RSW Crossover: 10; No. of Rule Ordering Repetitions: 5. |
| XCS | Number of explores: 1,00,000; Population size: 6,400; $\alpha = 0.1$; $\beta = 0.2$; $\delta = 0.1$; $\mu = 10.0$; $\theta_{ga} = 50.0$; $\theta_{mna} = 2$; $\theta_{del} = 50.0$; initial prediction: 10.0; initial fitness: 0.01; initial prediction error: 0.0; prediction error reduction: 0.25; GA subsumption: false; Type of Selection: Roulette Wheel Selection; Tournament size: 0.4; Type of Crossover: 2 point; Crossover Probability: 0.8; Type of Mutation: Free; Mutation probability: 0.03; $r = 1.0$; $m = 0.1$. |
| SUCS | Number of explores: 70,000; Population Size: 8,400; $\alpha = 0.1$; $\beta = 0.2$; $\delta = 0.1$; $\mu = 10.0$; $\theta_{del} = 50.0$;initial prediction: 10.0; initial fitness: 0.01; initial prediction error: 0.0; prediction error reduction: 0.25; GA subsumption: true; $\theta_{sub} = 500.0$; Type of Selection: Tournament; Tournament size: 0.4; Type of Crossover: 2 point; Crossover Probability: 0.8; Type of Mutation: Free; Mutation probability: 0.06; $\theta_{ga} = 50.0$; $r = 0.6$; $m = 0.1$. |

### 6.2.4 Hypothesis Evaluation using Statistical Tests

In order to ascertain the best hybridized technique (RQ2), we investigate the following hypothesis using Friedman test and Wilcoxon signed rank test.

***Hypothesis investigated using Friedman Test***

Friedman test is used to test the following hypothesis (H0, H1 and H2). The predictive capability of all the 15 investigated techniques are assessed using Balance,

G-Mean1 and G-Mean3 values in order to determine the best hybridized technique.

*Alternate Hypothesis H0/ H1/ H2*: The developed models for software change prediction using the various investigated techniques of the chapter are significantly different from each other with respect to Balance/ G-mean1/ G-Mean3 values.

***Hypothesis investigated using Wilcoxon Signed Rank Test***

In case the results of Friedman test are found significant, depicting a specific technique X with the best predictive capability, we conduct a post-hoc Wilcoxon test with Bonferroni correction. Pairwise comparisons are performed amongst all the investigated techniques and technique X (best technique) using Balance, G-Mean1 and G-Mean3 performance measures. The investigated hypothesis is stated as follows:

*Alternate Hypothesis H3/ H4/ H5*: The change prediction models developed using technique X are significantly different with respect to Balance/ G-Mean1/ G-Mean3 values from the change prediction models developed using technique A.

In the above hypothesis, technique A represents all the other investigated techniques except X (best ranked technique according to Friedman test). In case, the PSO-LDA technique is the best technique established by the results of the Friedman test, then technique A represents the other 14 techniques investigated in the chapter apart from the PSO-LDA technique.

## 6.3 Results and Analysis

This section describes the descriptive statistics, outlier removal and feature selection results. Thereafter, we state the results specific to each RQ and their corresponding answers.

### 6.3.1 Descriptive Statistics & Outlier Removal

We analyzed the descriptive statistics of all the datasets used in the chapter. We found that the inheritance attribute was not much used in the investigated datasets as the mean NOC and mean DIT metric values are low. The range of mean NOC values was 0.64-1.00 and that of mean DIT value was 0.00-0.37. However, the datasets were found to exhibit high mean values for LCOM in the range of 104.74-709.05.

The Bluetooth application package was found with 7 outliers. Similarly, the Contacts, Calendar, Gallery2, MMS and Telephony datasets were detected with 12, 6, 43, 22 and 37 outliers respectively, which were removed before further analysis.

### 6.3.2 CFS Results

As explained in experimental design, we use the CFS method for reducing the dimensionality of the input features except for model development using the PSO-LDA technique. This is because the feature selection is already incorporated in the PSO-LDA technique. Table 6.2 reports the metrics which were selected after application of the CFS method on each dataset. As the SLOC metric and the CAM metric are selected in five and four datasets respectively, they seem to be highly effective in predicting change-prone classes.

Table 6.2: Metrics selected after application of CFS

| Dataset | Metrics Selected |
|---|---|
| Bluetooth | SLOC, CAM, WMC, RFC |
| Contacts | SLOC, DIT, NPM, CBO |
| Calendar | CBO, Ce |
| Gallery2 | SLOC, Ce, LCOM3, MOA, CAM |
| MMS | SLOC, DAM, LCOM3, MOA, CAM, AMC |
| Telephony | SLOC, MFA, WMC, LCOM3, CAM |

### 6.3.3 Results specific to RQ1

The predictive capability of various techniques in the domain of software change prediction is assessed by analyzing the performance of ten-fold cross validation models developed using them. Figures 6.2-6.6 represent the median values of performance measures (Recall, PF, Balance, G-Mean1 and G-Mean3) of 30 runs executed using a specific technique. Each bar in the figures corresponds to the performance measure value of a specific technique (color grouped according to the category of the technique) on a specific dataset. It may be noted that the dark gray bars depict the HBT, the light gray bars depict SBA and the black colored bars are representative of ML/statistical techniques.

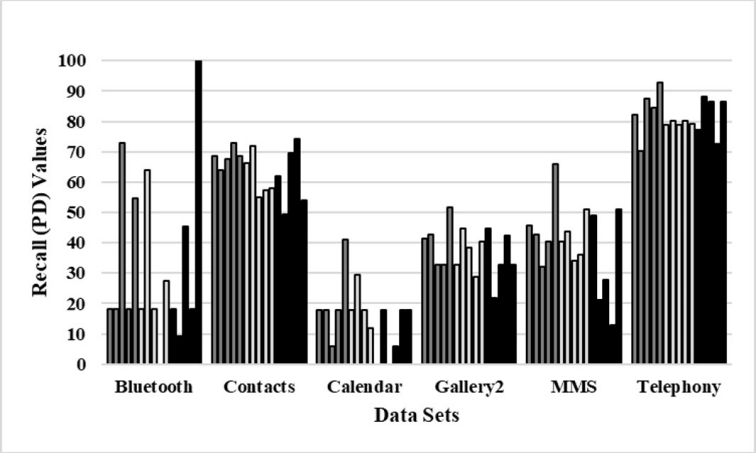


Figure 6.2: Median Recall Values on 30 Runs of different techniques

The recall and PF values obtained by change prediction models developed by the investigated techniques are illustrated in Figures 6.2-6.3. The change prediction models on Contacts dataset obtained recall values in the range 50%-75%. Similarly, the range of recall values on Calendar, Gallery2 and MMS datasets range from 0%-42%, 21%-52% and 12%-65% respectively. It may be noted that high recall values (72%-92%) were obtained on Telephony dataset. An analysis of Figure 6.2 depicts

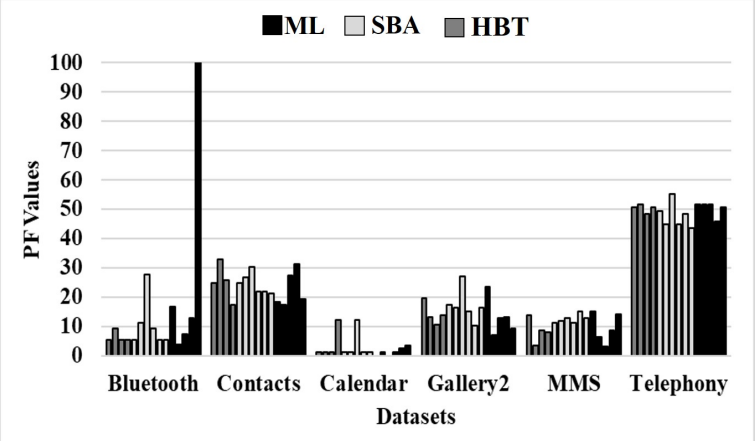


Figure 6.3: Median PF Values on 30 Runs of different techniques

that the LDA technique obtained the best recall value on Bluetooth dataset, while the CART technique obtained the best recall value on Contacts dataset. However, the PSO-LDA technique, a hybridized technique obtained the best recall values on four datasets (Calendar, Gallery, MMS and Telephony). An analysis of Figure 6.3 depicts PF values in the range 3%-100% on Bluetooth dataset, 17%-27% on Contacts dataset, 0%-12% on Calendar dataset, 6%-27% on Gallery2 dataset, 3%-15% on MMS dataset and 43%-51% on Telephony dataset. It was noted that SBA showed the best PF values in four of the six investigated datasets.

Figure 6.4 illustrates the median Balance values of 30 runs for the change prediction models developed using each investigated technique using the six datasets of the chapter. The change prediction models developed by the PSO-LDA technique exhibited the best cumulative Balance results. The median Balance values of the PSO-LDA technique were in the range of 57% to 74% (Bluetooth: 67%, Contacts: 74%, Calendar: 57%, Gallery2: 64%, MMS: 75% and Telephony: 64%). We also analyzed the Balance values for each category of techniques and found them to be in the range of 40% to 80% for a majority of HBT, 40% to 70% for a majority of SBA and 40% to 75% for ML /statistical techniques. According to the figure, the

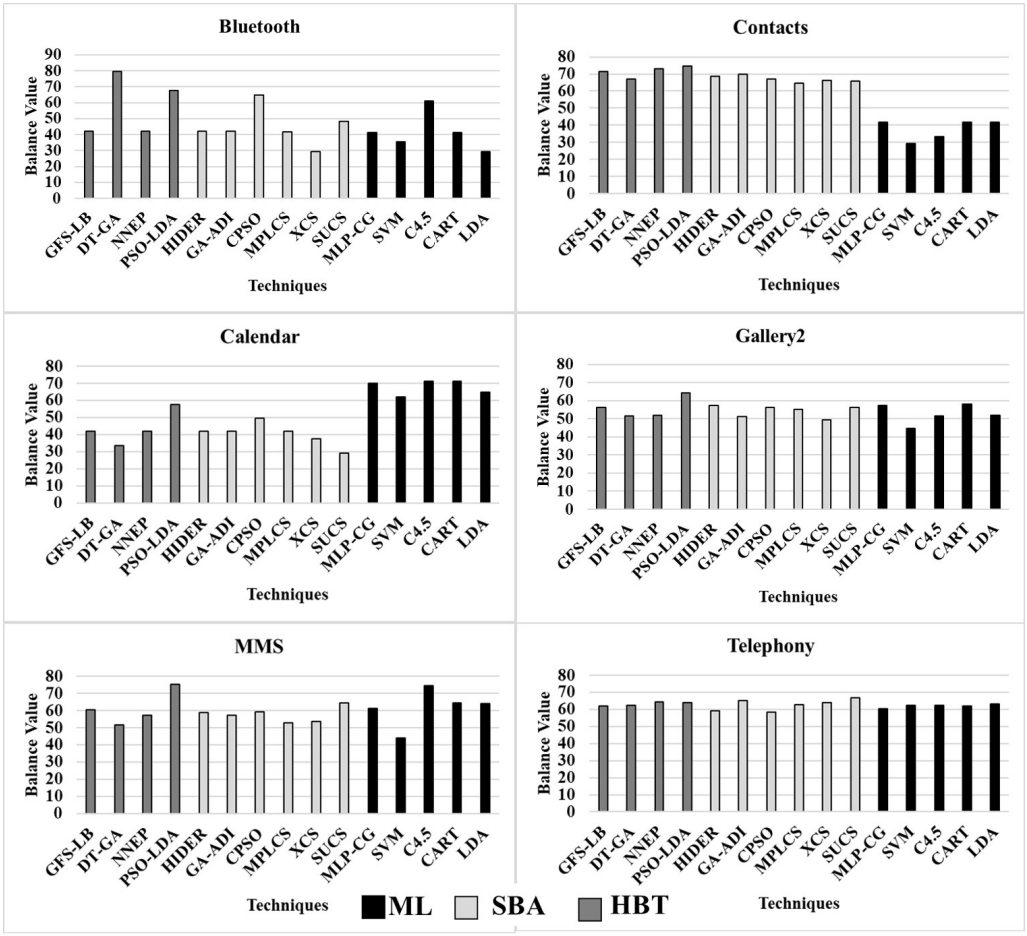


Figure 6.4: Median Balance Values of different techniques

Balance values of HBT were found comparable to that of SBA, since both these sets of techniques are capable of accomplishing a global search for optimum solution by avoiding local maxima. While ML techniques such as CART and C4.5 exhibited good Balance values, other ML techniques such as SVM exhibited low Balance values than the HBT and SBA in four of the investigated datasets.

Figure 6.5 presents the median values of G-Mean1 measure for 30 runs on each investigated dataset of the chapter. Though, the DT-GA technique depicted the best G-Mean1 values on Bluetooth dataset (0.81), for all the other datasets, the models

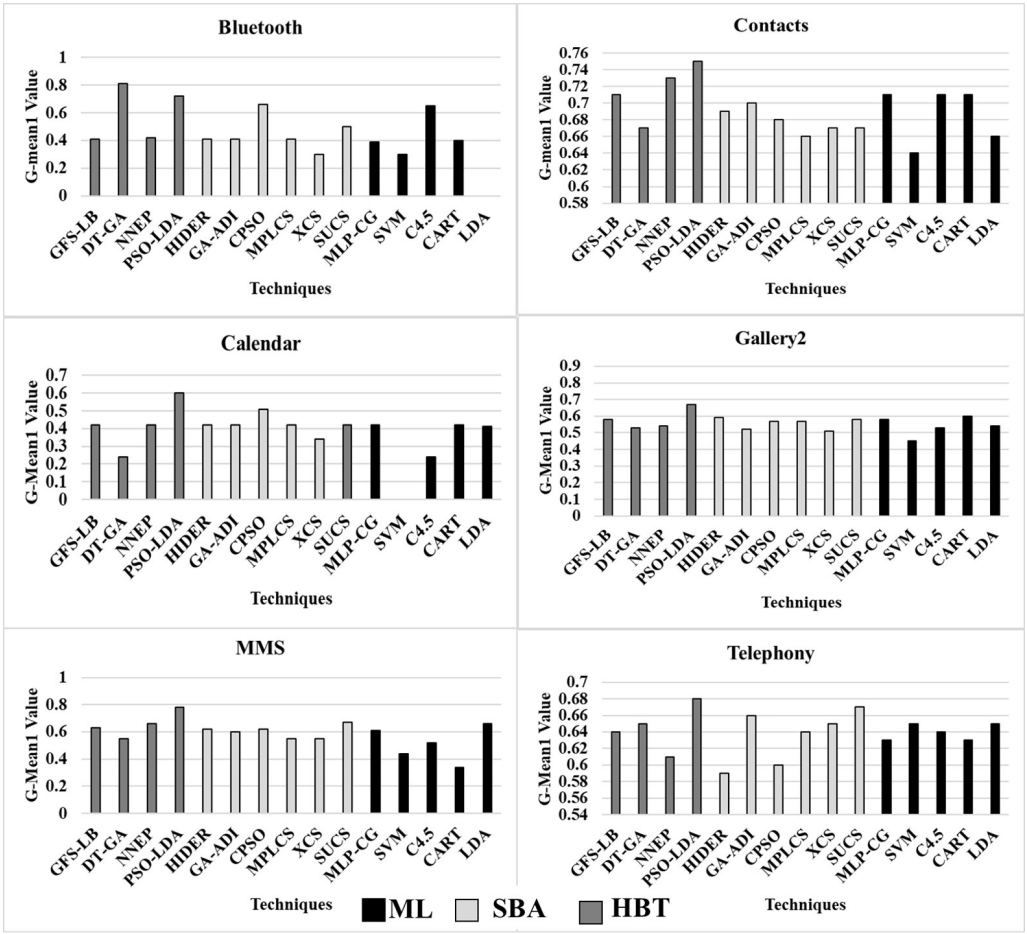


Figure 6.5: Median G-Mean1 Values of different techniques

developed using the PSO-LDA technique depicted the best G-Mean1 values (Contacts: 0.75, Calendar: 0.60, Gallery2: 0.67, MMS:0.78, Telephony: 0.68). Certain techniques such as LDA on Bluetooth dataset and the SVM technique on Calendar dataset, showed extremely poor G-Mean1 values as it was not defined on them. The average of G-Mean1 values on all the datasets ranged from 0.41-0.56 for ML /statistical techniques, 0.50-0.61 for SBA and 0.56-0.70 for HBT. It can be seen that the performance of HBT were in most cases better than or equivalent to ML/statistical techniques or SBA. The SVM technique depicted the worst average G-Mean1 values

in the majority of the datasets.

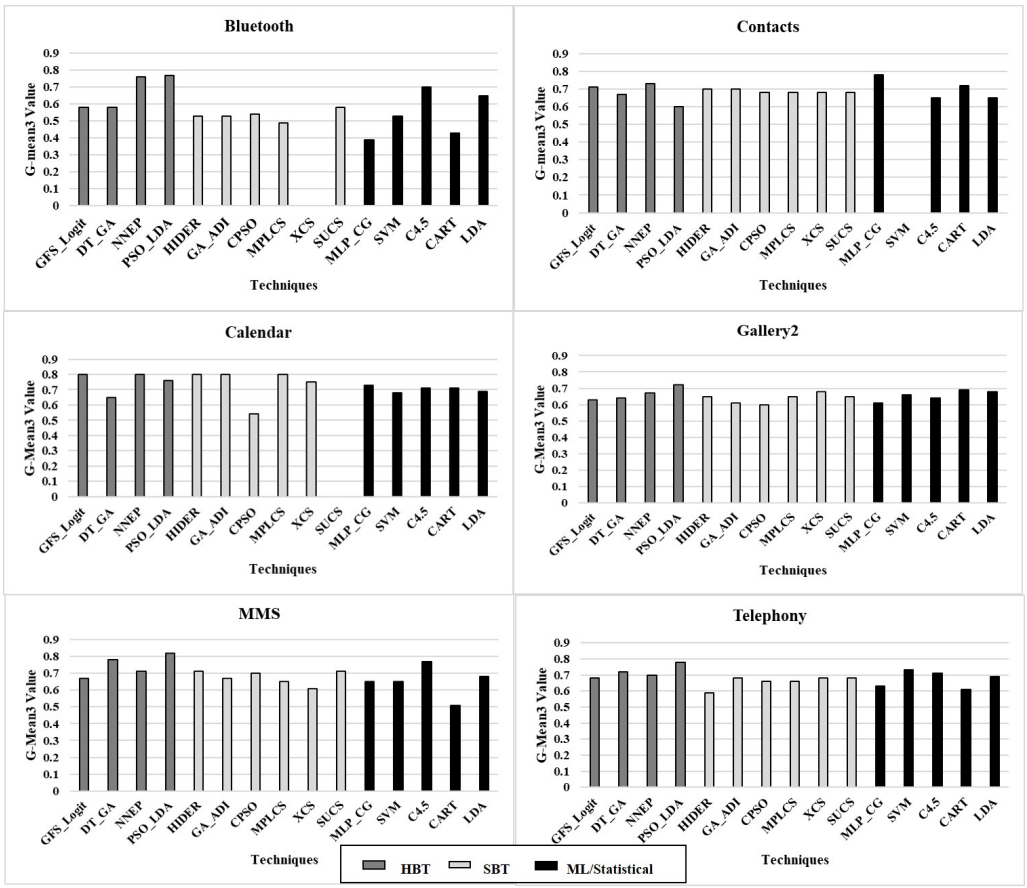


Figure 6.6: Median G-Mean3 Values of different techniques

The median G-Mean3 values of the models developed using the various techniques are depicted in Figure 6.6. It was found that the G-Mean3 values of the change prediction models developed using most of the HBT and SBA were in the range of 0.50 to 0.80, while that of ML /statistical techniques were in the range of 0.40 to 0.78. The models developed by the PSO-LDA technique exhibited the best cumulative results, which were followed by the models developed by the NNEP technique. All HBT exhibited better or equivalent performance in terms of G-Mean3 values, when compared with the SBA and ML /statistical techniques.

***Analysis of RQ1 Results***

An analysis of the results discussed above indicates that the HBT depict better predictive capabilities than SBA for developing change prediction models on the investigated datasets. Similarly, models developed by HBT like PSO-LDA, GFS-LB and DT-GA outperform the models developed using the investigated ML /statistical techniques. SBA were found to exhibit the best PF values on four of the six investigated datasets. On comparing the change prediction models developed using the SBA and ML/ statistical techniques, we found SBA such as GA-ADI, MPLCS and CPSO developed better models than the investigated ML techniques. As the search of SBA is global, they inherently avoid getting trapped in local maxima, thereby attaining effective solutions.

The trends signify that the predictive capability of HBT are advantageous for developing change prediction models. A hybridized technique benefits from the favorable properties of all its constituent techniques to improve the developed models. For example, the DT-GA technique incorporates the effective pruning of DT along with search process of the GA, which helps it to avoid local maxima and perform a global search.

***Answer to RQ1***

The predictive performance of the investigated techniques in terms of the various performance measures (Recall, PF, Balance, G-Mean1 and G-Mean3) signify that the models developed by most of the HBT are superior to the ones developed by the SBA and the ML /statistical techniques. However, while comparing SBA and ML/statistical techniques, the SBA developed better change prediction models. As HBT are influenced by the advantages of both its constituent techniques, the models developed using them outperform the other investigated techniques.

### 6.3.4 Results specific to RQ2

The best hybridized technique was ascertained by Friedman test (hypothesis stated in Section 6.2.4) [125]. As the test computes the mean ranks of techniques over all the investigated datasets, we look for the technique obtaining the lowest Friedman rank as the best one. The degrees of freedom for Friedman test is 14. It was evaluated on the median Balance, G-Mean1 and G-Mean3 values of 30 runs for all the techniques on all the six datasets. The hypothesis was evaluated at $\alpha = 0.05$ level.

Table 6.3: Friedman Ranking

| **Based on Balance Values** | | | | | |
|---|---|---|---|---|---|
| Technique | Mean Rank | Technique | Mean Rank | Technique | Mean Rank |
| PSO-LDA | 1.83 | CPSO | 7.33 | MPLCS, LDA | 9.50 |
| NNEP | 6.00 | C4.5 | 7.42 | DT-GA | 9.58 |
| GFS-LB | 6.07 | GA-ADI | 7.67 | XCS | 11.17 |
| CART | 6.67 | HIDER | 8.00 | SVM | 13.42 |
| SUCS | 7.08 | MLP-CG | 8.17 | | |
| **Based on G-Mean1 Values** | | | | | |
| Technique | Mean Rank | Technique | Mean Rank | Technique | Mean Rank |
| PSO-LDA | 1.17 | GA-ADI | 7.75 | C4.5 | 9.25 |
| SUCS | 5.42 | HIDER | 7.83 | LDA | 9.50 |
| NNEP | 5.75 | MLP-CG | 8.00 | MPLCS | 9.92 |
| GFS-LB | 6.25 | CART | 8.83 | XCS | 11.00 |
| CPSO | 6.92 | DT-GA | 9.08 | SVM | 13.33 |
| **Based on G-Mean3 Values** | | | | | |
| Technique | Mean Rank | Technique | Mean Rank | Technique | Mean Rank |
| PSO-LDA | 2.83 | DT-GA | 8.00 | SVM | 9.58 |
| NNEP | 3.67 | HIDER | 8.08 | XCS | 9.75 |
| C4.5 | 6.17 | GA-ADI | 8.58 | MPLCS | 10.08 |
| LDA | 6.17 | SUCS | 8.75 | MLP-CG | 10.33 |
| GFS-LB | 7.33 | CART | 9.50 | CPSO | 11.17 |

Table 6.3 states the Friedman mean ranks obtained by each technique. According to the table, the PSO-LDA technique obtained the best rank in all the three cases. Though, NNEP received second rank using Balance and G-Mean3 values, it ranked third using G-Mean1 values indicating its effectiveness. The Friedman test results are significant at $\alpha = 0.05$ as the obtained p-value is 0.01 using Balance values and 0.03 using G-Mean1 and G-Mean3 values each. The Friedman test statistic was

calculated as 28.30 using Balance values. Similarly, the test statistic was computed as 33.23 and 24.55 using G-Mean1 and G-Mean3 values respectively. Thus, we accept alternate hypothesis H0, H1 and H2, which means that change prediction models developed using different techniques perform significantly different from each other when evaluated using the Balance, G-mean1 and G-Mean3 measures.

As the Friedman test yields significant results, showing PSO-LDA as the best technique, we performed post-hoc Wilcoxon signed rank test with Bonferroni correction to evaluate pairwise differences amongst the performances of different techniques at significance level $\alpha = 0.05$. The set of hypothesis evaluated using Wilcoxon test is stated in section 6.2.4. Here, technique X corresponds to the PSO-LDA technique. Figures 6.7, 6.8 and 6.9 report the pairwise comparison results of Wilcoxon test after Bonferroni correction using the Balance, G-Mean1 and G-Mean3 values respectively. The bars represent the number of datasets which achieve different significance levels. The different colors of the bars indicate different significance levels. It should be noted that the maximum number of datasets achieving a specific significance level would be six as we use six datasets in the chapter. The symbol "Sig. Superior" indicates technique X is significantly superior to technique A on a dataset D, while the symbol "Sig. Inferior" indicates that technique A significantly outperforms technique X on a dataset D. "Not Sig. Superior" symbol indicates technique X outperforms technique A on dataset D but not significantly. Similarly, "Not Sig. Inferior" indicates technique A outperforms technique X on dataset D but not significantly. The Wilcoxon comparison was performed by comparing the performance measures values of 30 runs.

According to Figure 6.7, the PSO-LDA technique significantly outperforms all the other techniques on all the datasets except in five cases. Thus, the results of the Wilcoxon test with Bonferroni correction accept the alternate hypothesis H3.

According to figure 6.8, the post-hoc Wilcoxon test using G-Mean1 values indi-

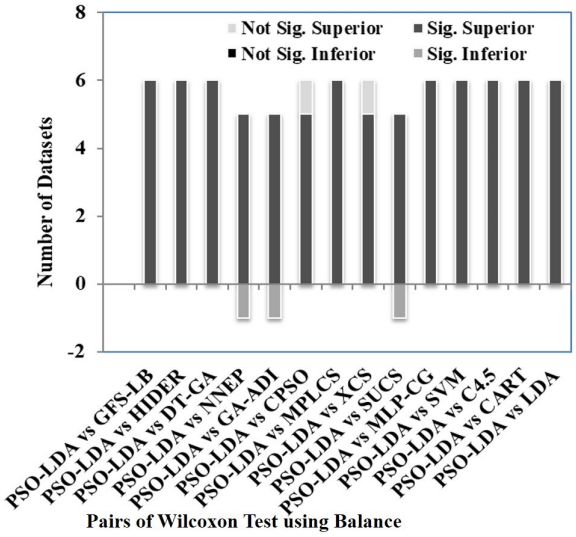


Figure 6.7: Wilcoxon Test Results using Balance values

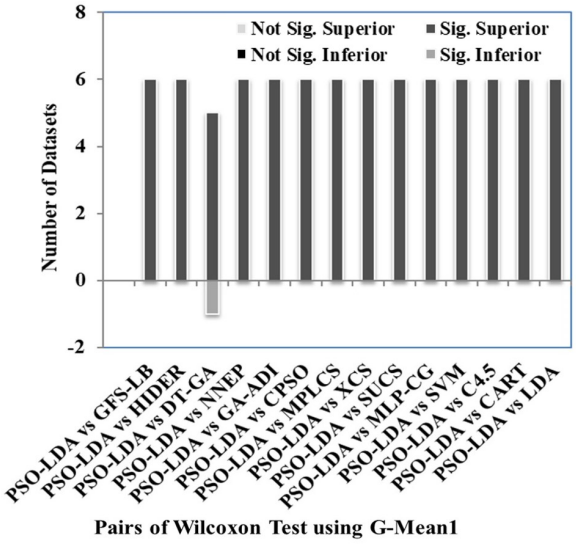


Figure 6.8: Wilcoxon Test Results using G-Mean1 values

cates that the PSO-LDA technique significantly outperforms all the other techniques on all the datasets except in one case, where DT-GA outperforms PSO-LDA. There are no cases with “not significant” results. Thus, we reject the null hypothesis H4.

Figure 6.9 depicts the pairwise comparison results using the G-Mean3 measure.

The PSO-LDA technique significantly outperforms all the other techniques on all the datasets except the Calendar dataset. Thus, the results indicate acceptance of the alternate hypothesis H5.

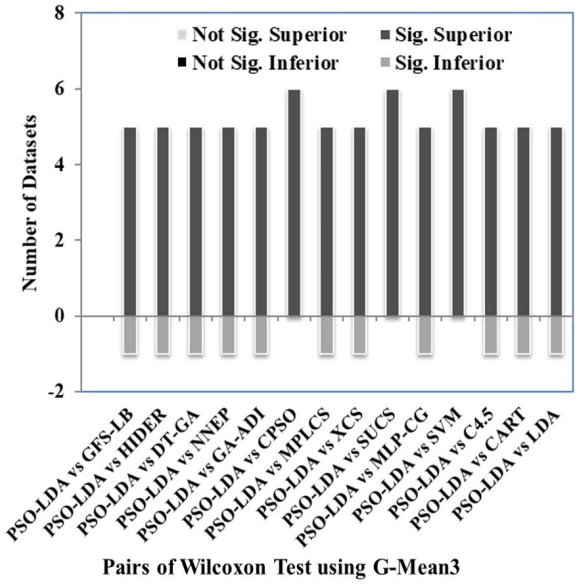


Figure 6.9: Wilcoxon Test Results using G-Mean3 values

***Analysis of RQ2 Results***

As depicted by Friedman tests (based on Balance, G-Mean1 and G-Mean3 values), the PSO-LDA technique clearly outperformed the other investigated techniques. The hybrid nature of PSO-LDA, which combines the PSO technique (a search-based algorithm) for extracting all relevant and significant features, along with the LDA (a statistical technique), yields a successful and effective classifier. Previous studies by Lin et al. [258] and Huang and Dun [259] also advocate the use of PSO as an effective feature selector. The PSO technique chooses an effective feature set by representing a particle in the population of a D-dimensional space, where D represents the number of features. The PSO technique, thereafter, computes the fitness (classification accuracy) of each particle to evaluate whether the set of features is appropriate

for model development using LDA. It may be noted that in PSO-LDA, the LDA develops models using only the features selected by the PSO. As the use of PSO as a feature selector augments the predictive capability of the LDA technique [260], the combination of PSO and LDA is favourable. In addition, the results of Wilcoxon signed rank test also support the use of the PSO-LDA technique as it outperformed the models developed using all the other investigated techniques, in a majority of the cases (using Balance, G-Mean1 and G-Mean3 values).

***Answer to RQ2***

According to the results of Friedman tests, the change prediction models developed using the PSO-LDA technique, a hybridized technique obtained the best results. Also, the change prediction models developed by the NNEP technique, another hybridized technique obtained the next best results. The results of Wilcoxon tests ascertained that the change prediction models developed using the PSO-LDA technique were significantly improved as compared to the ones developed using the other investigated techniques. The PSO-LDA technique yields an effective combination of PSO, a search-based algorithm for feature selection and the statistical technique, LDA for model development.

### 6.3.5 Results specific to RQ3

As discussed in Section 6.1, it is important to evaluate the CPU time taken by the investigated techniques. Therefore, we consider the minimum (Min), maximum (Max) and the mean CPU time (in seconds) taken by the investigated techniques to develop change prediction models on all the six datasets of the chapter (Figure 6.10).

The results depicted in Figure 6.10 indicate that the NNEP technique is the slowest technique for developing change prediction models in terms of CPU time. However, the fastest techniques were the C4.5 and the LDA, with a mean CPU time of 13

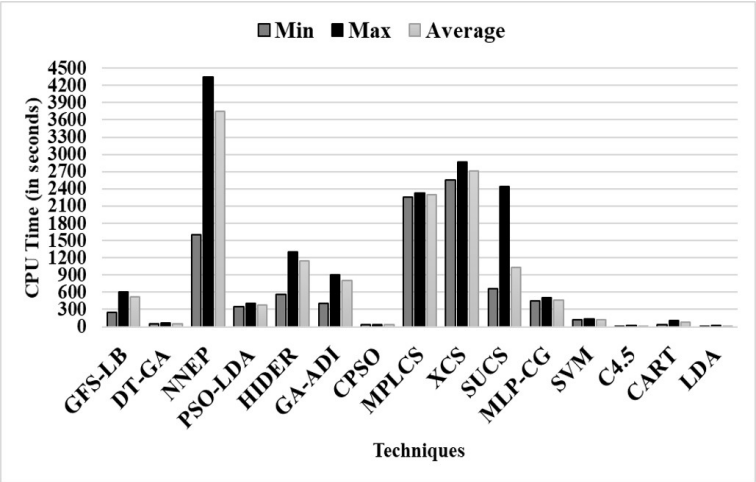


Figure 6.10: CPU time statistics of different techniques

seconds each. The CPSO technique, a search-based algorithm depicted the least CPU time (34 seconds) amongst SBA. It should be noted that the actual CPU time taken by a technique is approximately 30 times the mean value depicted in Figure 6.10. For instance, the NNEP technique took a mean time of 3,746 seconds for developing change prediction on all the six investigated datasets. However, this time needs to be multiplied by 30 (3,746 * 30 = 37,650 seconds) to get the actual mean time taken by the NNEP technique in 30 runs.

We also analyzed the mean CPU time taken by a specific technique while developing change prediction models on a single dataset (Figure 6.11). Again, we note that the slowest technique was NNEP (624 seconds). The C4.5 and the LDA techniques were the fastest with a mean CPU time of approximately 2 seconds each. Other fast techniques were CPSO (6 seconds) and DT-GA (10 seconds). The trend signifies that the ML/statistical techniques are faster than SBA and HBT. The time taken by the PSO-LDA hybridized technique (60 seconds) was comparable to that of the time taken by certain ML techniques such as MLP-CG.

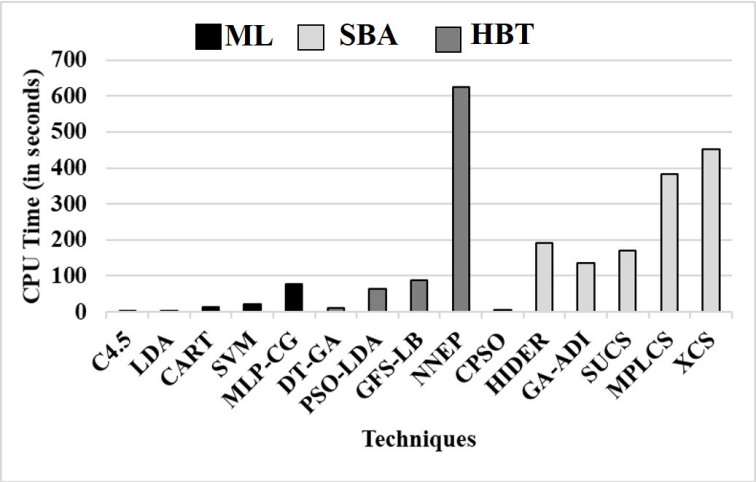


Figure 6.11: CPU time taken by different techniques

***Analysis of RQ3 Results***

After analysis of the mean CPU time for model development, we state the following observations:

- The ML techniques were faster as compared to the SBA and HBT for developing change prediction models. Researchers should explore alternatives like parallel architecture, increasing the efficiency of hardware technology, cloud computing etc. to overcome the issue of SBA and HBT, which take higher CPU times for model development.

- Techniques which are hybrids of a fast ML/statistical technique generally take lower mean CPU time than other HBT. For instance, PSO-LDA and DT-GA are hybridized versions of fast techniques like LDA and C4.5, taking smaller CPU times. On the contrary, the NNEP technique, which is a hybrid of a slower ML technique, similar to MLP-CG, depicts a very poor CPU time performance. Thus, hybridization of an ML technique which is itself time consuming (MLP-CG with 78 seconds) with a search-based algorithm (evolutionary programming) results in a hybridized technique i.e. NNEP, which takes very large CPU

time (624 seconds).

- In general, as SBA perform a global search, they consume large CPU times. However, an exception was the CPSO technique as it consumed low CPU time (6 seconds). This could be because of the use of constriction factor in CPSO, which restricts the search-space explosion leading to effective convergence in CPU time [95].

In terms of CPU time, the ML /statistical techniques outperform the HBT and SBA. Also, the SBA were on an average slower in terms of CPU time than HBT.

***Answer to RQ3***

The analysis of the CPU time performance of the various investigated techniques indicates that most of the ML/statistical techniques are efficient and fast. Certain HBT, such as GFS-LB, PSO-LDA and DT-GA also show effective CPU times for developing software change prediction models. This could be due to hybridization of a fast ML technique with a search-based algorithm. The SBA were found slowest in terms of CPU time as they take high CPU times. The only exception to this trend was the CPSO technique, which takes low CPU time because of constriction coefficient.

### 6.3.6 Results specific to RQ4

The trade-off between CPU time and predictive performance was evaluated by analyzing the mean CPU time and the mean performance measures values (Balance G-Mean1 and G-Mean3) for HBT, SBA and ML/statistical techniques, which are depicted in Table 6.4. We consider a technique to be favorable if it exhibits low CPU time and high predictive performance in terms of Balance, G-Mean1 and G-Mean3 measures.

Table 6.4: CPU Time and Mean Performance Measures of Different Techniques

| Technique | CPU Time | Average Balance | Average G-Mean1 | Average G-Mean3 |
|---|---|---|---|---|
| **HBT** | | | | |
| PSO-LDA | 62.67 | 67.21 | 0.70 | 0.74 |
| NNEP | 624.33 | 55.09 | 0.56 | 0.73 |
| GFS-LB | 87.16 | 55.65 | 0.56 | 0.67 |
| DT-GA | 9.50 | 57.67 | 0.58 | 0.67 |
| **SBA** | | | | |
| CPSO | 5.67 | 59.19 | 0.61 | 0.62 |
| XCS | 451.00 | 49.92 | 0.50 | 0.56 |
| SUCS | 170.83 | 55.09 | 0.58 | 0.55 |
| MPLCS | 383.66 | 53.10 | 0.54 | 0.65 |
| GA-ADI | 134.66 | 54.43 | 0.55 | 0.67 |
| HIDER | 190.83 | 54.62 | 0.55 | 0.66 |
| **ML/ Statistical** | | | | |
| C4.5 | 2.16 | 58.99 | 0.55 | 0.70 |
| CART | 13.83 | 56.51 | 0.52 | 0.61 |
| LDA | 2.16 | 52.44 | 0.49 | 0.67 |
| MLP-CG | 77.50 | 55.29 | 0.56 | 0.63 |
| SVM | 20.00 | 46.38 | 0.41 | 0.54 |

The results in Table 6.4 illustrate that all ML/statistical techniques except MLP-CG are efficient with respect to CPU time. However, their predictive performance in terms of Balance and G-Mean1 values was found poor than most of the investigated HBT and certain SBA. The PSO-LDA and DT-GA techniques are HBT, which exhibited effective Balance and G-Mean1 values and tolerable CPU times, when compared with the majority of other investigated techniques. However, it was noted that though the performance of the NNEP technique (in terms of Balance and G-Mean1 values) is better or equivalent to most of the other techniques, it was found the slowest in terms of CPU time. The mean Balance values of HBT range from 55%-67% and G-Mean1 values from 0.56-0.70, indicating their predictive capability. It was also observed that though SBA (except XCS) exhibit good performance in terms of Balance and G-Mean1 values, they consume large CPU times. Only the CPSO technique, a search-based algorithm was fast in terms of CPU time.

The mean G-Mean3 values observed from Table 6.4 indicate that the HBT exhibit the best G-mean3 values. Moreover, their CPU time consumption is competitive, ex-

cept for the NNEP technique. The SBA depicted better G-Mean3 values than certain ML/statistical techniques but lower G-Mean3 values than HBT. The C4.5 technique, an ML technique showed effective results with a good G-Mean3 value of 59% and an average CPU time of 2 seconds. It was noted that though the CPSO technique was a fast technique, it gave poor G-Mean3 values. SBA such as XCS, SUCS and the MPLCS techniques were found to consume higher CPU times and exhibit low G-Mean3 values.

***Analysis of RQ4 Results***

We summarize few interesting observations after analyzing the trade-off between CPU time and the predictive performance of the investigated techniques:

- As observed from Table 6.4, the HBT gave better predictive performance (in terms of Balance, G-Mean1 and G-Mean3 values) than the other investigated techniques in most of the cases. Change prediction models developed with the aid of DT-GA and the PSO-LDA techniques gave effective values in terms of both (predictive capability and CPU time).

- Effective change prediction models developed using various techniques are beneficial in the efficient allocation of maintenance and testing resources. Thus, techniques which give a very good predictive performance with moderate CPU time such as PSO-LDA may also be used effectively as the average CPU time is compensated by the cost and time saved by application of the change prediction models developed by these techniques. However, techniques which take large CPU time such as NNEP may be carefully selected, taking into account the large CPU time required by them. We performed cost-benefit analysis in order to evaluate the effectiveness of change prediction models developed using PSO-LDA, NNEP and C4.5 models on all the datasets. The effort gain was

calculated according to the following formula [261] [54]:

$$Effort\ Gain = \frac{SLRT - LCPM}{LRT} * 100 \tag{6.1}$$

SLRT signifies the SLOC one needs to test if we perform Random Testing (RT). It is computed as the product of recall and total SLOC of a dataset. LCPM signifies the SLOC, one would test after application of Change Prediction Model (CPM). The effort gain is representative of the effort reduced while testing. A higher effort gain, indicates a better model. We found that the percentage effort gain was approximately between 40%-60% when we used HBT (PSO-LDA and NNEP). This effort gain was a 20% increase from the scenario when we used models developed using the ML technique (C4.5). Thus, we note that though there is a trade-off between predictive performance and the average CPU time taken for model development by a technique. We can still use a specific technique if it balances out the project's costs saved by it on application of the change prediction model developed using the technique.

Thus, after analyzing the trade-off between CPU time and the performance measures (Balance, G-Mean1 and G-Mean3), we can conclude that the PSO-LDA, DT-GA, C4.5 and even NNEP are effective techniques for developing change prediction models. The DT-GA technique is comparable to the C4.5 technique in terms of CPU time as well as predictive performance. Though, PSO-LDA technique takes longer CPU time for model development, it is compensated by effective predictive performance of the developed change prediction model. Furthermore, we note that the NNEP technique may be carefully selected because though it gives an effective percentage gain in the effort but takes a large amount of CPU time for model development.

HBT can be used if they are capable of balancing out the CPU time performance

and exhibit an effective effort gain ratio. SBA were found incapable of producing models which could effectively compensate the time factor. The trade-off indicates that though ML techniques are efficient in terms of CPU time, we should select an ML technique which demonstrates high predictive capability in terms of various performance measures.

***Answer to RQ4***

The analysis of the obtained results indicates the existence of a trade-off between the CPU time needed for model development and the predictive performance of the developed model. The ML/statistical techniques were found quite effective in terms of CPU time, but they exhibited lower predictive capability and effort gain ratio as compared to certain SBA and HBT. HBT such as PSO-LDA are capable of balancing the trade-off between CPU time and predictive performance for developing change prediction models. Even if PSO-LDA takes moderate CPU times, this time can be compensated by the costs and time saved by effective application of the results of change prediction models developed by it.

## 6.4 Comparison of Various Studies

In this section, we compare our results with literature studies and previous chapters in terms of various performance measures (Recall, AUC, G-Mean1, G-Mean3 and Balance). The values of different performance measures from various chapters and studies are listed in Table 6.5. It may be noted that we report the median values of each performance measure, obtained over all the investigated datasets in a chapter or a study. Also, the values for only that technique or scenario is reported, which is advocated as the best in a chapter or a study and where the model is developed using source-code metrics. All the values, which could not be extracted are depicted by

"–".

Table 6.5: Comparison Results

| Performance Measure | Chapter 4 (RF) | Chapter 5 (MPLCS) | Chapter 6 (PSO-LDA) | [4] (NB) | [36] (PSO-LDA) | [137] (LR) | [262] (ELM-PLY) | [144] (CLAMI+) |
|---|---|---|---|---|---|---|---|---|
| Recall | – | 67.95 | 60.26 | 47.08 | – | 49.00 | – | – |
| AUC | 0.781 | – | – | 0.747 | – | 0.555 | 0.645 | 0.645 |
| G-Mean1 | 0.715 | 0.700 | 0.698 | – | – | – | – | – |
| G-Mean3 | – | – | 0.770 | – | 0.735 | – | – | – |
| Balance | – | 69.06 | 66.01 | – | – | – | – | – |
| "–" means could not be extracted; ELM-PLY: Extreme Machine Learning with Polynomial Kernel CLAMI+: Clustering, Labeling, Metric selection and Instance Selection proposed by [144] | | | | | | | | |

As depicted in Table 6.5, the RF technique (a ML technique) was advocated as the best in Chapter 4, the MPLCS technique, a search-based algorithm was the best in Chapter 5 and the HBT, PSO-LDA showed optimum performance in the current chapter. Amongst SVM, MLP and NB, the study by Romano and Pinzger [4] gave the best values for prediction of change-prone Java interfaces with NB technique. However, we extracted the values for models developed using only the OO metrics because in all our previous chapters, we have explored OO metrics as independent variables for determining change-prone nature of a class. The study by Bansal [36] advocated the PSO-LDA technique. However, their results were only validated on two open-source datasets and we report the median results on these two datasets. The study by Catolino et al. [137] evaluated a number of different set of predictors (entropy of changes, number of developers, structural and semantic scattering of developers, evolution-based metrics and OO metrics) using the LR technique for developing change prediction models. We report the median results of the LR technique on ten datasets investigated by Catolino et al. [137], when OO metrics were used as predictors. Kumar et al. [262] evaluated ten different subsets of source code metrics for determining the change-prone nature of a class using ten classification techniques. We report the median values of the models created using these ten subsets of source-

code metrics with Extreme Machine Learning with Polynomial Kernel (ELM-PLY), as it was advocated as the best classification technique. Yan et al. [144] proposed the use of CLAMI+ method for predicting change-prone classes on 14 open-source datasets. We report the median values of the method on the investigated datasets.

According to the results shown in Table 6.5, the MPLCS technique, depicted the best recall results, which were closely followed by the PSO-LDA technique. We have already discussed that HBT are better off than SBA as they are faster and the trade-off between the CPU time and predictive performance favors them. Thus, we advocate the PSO-LDA technique. In terms of AUC performance measure, the results of RF technique in Chapter 4 are superior to the ones investigated in other literature studies [4, 137, 144, 262]. Though, G-Mean1 performance measure has not been reported in literature studies pertaining to software change prediction, the G-Mean1 values for each of the three chapters (Chapter 4-6) are comparable, depicting effective results by each of the advocated technique in corresponding chapters. The G-Mean3 values obtained in this chapter, are more generalizable as we investigated six datasets as compared to only two datasets by Bansal [36]. Also, they are better than those reported by Bansal [36]. However, both studies (our current chapter results and the results of Bansal[36])), advocate the effectiveness of the PSO-LDA technique in the domain of software change prediction. Finally, though the results of Balance performance measure are slightly better for the MPLCS technique, it takes much longer CPU times as compared to the PSO-LDA technique. As shown in Table 6.4 (Section 6.4.4), the CPU time taken by the PSO-LDA technique was 62.67 seconds as compared to the CPU time of 383.66 seconds taken by the MPLCS technique. Thus, we favor the use of the PSO-LDA technique.

## 6.5 Discussion

The chapter assessed the predictive capability of the HBT, SBA and the ML/statistical techniques for developing efficient software change prediction models. In order to do so, we selected four HBT, six SBA, four ML techniques and one statistical technique to develop change prediction models on six Android datasets. The chapter used an effective experimental set up which took into account the non-deterministic nature of SBA and HBT, evaluated the statistical significance of results and provided ease of replication (by stating complete parameter settings and fitness functions). Moreover, we also evaluated the CPU time taken by the investigated techniques for model development.

The results of the chapter advocate the use of HBT over the other investigated SBA and ML/statistical techniques for developing software change prediction models. The PSO-LDA technique, a hybridized technique exhibited the best performance amongst all the investigated techniques. In terms of predictive capability, the SBA were found superior to the investigated ML/statistical techniques. However, with respect to the CPU time used for model development, we found the ML/statistical techniques to be the fastest. These techniques were followed by some effective HBT (DT-GA, GFS-LB). SBA and a few HBT (NNEP) were found to take high CPU time for model development.

After analyzing the trade-off between CPU time and predictive capability, the chapter recommends the use of HBT. The moderate CPU times taken by HBT for model development can be compensated by the cost and time saved by the efficient use of results of the change prediction models developed using them. As the chapter investigates a wide array of techniques, it provides an insight to researchers and practitioners for efficiently selecting modeling techniques for software change prediction.

# Chapter 7

# Ensemble Learners using Particle Swarm Optimization

## 7.1 Introduction

Various researchers in literature have successfully established the association between OO metrics and change-prone nature of a class. The results of the previous chapters also confirm this relationship. However, there is still an active need to explore effective classifiers for developing efficient change prediction models. Few researchers have evaluated the capabilities of SBA, such as GA and PSO in order to develop prediction models for software change [34–36]. Chapter 5 and Chapter 6 also ascertain the effectiveness of these algorithms in the domain of software change prediction. Although SBA have been found effective in this domain, researchers have been actively evaluating approaches which improve their performance.

Recent developments have ascertained that ensemble methodology can be used to improve the prediction performance of individual classifiers. The methodology involves the aggregation of outputs of a number of classifiers. The performance of

individual classifiers is improved, if the constituent classifiers of the ensemble are accurate and diverse [42–44, 263]. The accuracy of a constituent classifier may be determined by its ability to correctly categorize the class of a software as change-prone or not change-prone. However, the diversity of a group of classifiers is determined by their errors. A set of classifiers is termed as diverse, if they have different predictions for the same data point, i.e. their errors are not correlated. For instance, let us assume a new data point which is actually change-prone in nature. A set of four classifiers which make similar mistakes may result in a "not change-prone" prediction i.e. incorrect prediction for the data point. Since, there is no diversity in the predictions of the individual classifiers, an incorrect result will be output by the ensemble of these classifiers as well. However, if the classifiers were diverse in nature, and there is a scenario where three of the classifiers make correct predictions for a new data point and only one predicts incorrectly, there is a chance that the result of their ensemble may output the correct prediction, making it more accurate [42]. Thus, the diversity of constituent classifiers is essential so that an improved resultant classifier can be obtained by appropriately learning from diverse classifiers [43, 44].

Studies in literature have evaluated the use of ensemble methodology for ML techniques. However, the use of SBA such as PSO using ensemble methodology has not been explored. Thus, this chapter evaluates the use of ensemble learning using various PSO variants as constituent learners to develop efficient software change prediction models. Though, we could have used GA, a widely used SBA for our study, we selected PSO as it has several advantages over GA. The advantages include a) faster convergence of PSO as "crossover" and "mutation" operators are not present in PSO and updates in PSO are done by the particle's internal velocity, b) the capability of the PSO to memorize the current best particle which is then passed on to particles in the next generation c) a lesser number of internal parameters which are easily adjusted as compared to GA and d) lower influence in PSO due to modification

in the problem dimensionality [210, 264–266].

The chapter proposes four different ensemble classifiers namely, Majority Voting Ensemble Classifier (MVEC), Weighted Voting Ensemble Classifier (WVEC), Hard Instance Ensemble Classifier (HIEC) and Weighted Voting Hard Instance Ensemble Classifier (WVHIEC). The proposed classifiers were used for developing software change prediction models. Each ensemble classifier consists of seven individual PSO classifiers. The output of these classifiers is aggregated using weighted voting. The weights allocated to a classifier are based on their predictive capability assessed by stable performance measures and their ability to correctly identify the nature of classes which are commonly misclassified (hard instances). Each individual classifier learns from the same training data, but uses a different fitness function. A fitness function of a search-based algorithm is used to guide the search for an optimum solution amongst a multitude of candidate solutions [6]. Previous research has ascertained that variation in the fitness function also leads to variation in the result of a prediction model [46–49]. Thus, use of different fitness functions for PSO, results in diversity of individual classifiers. Also, the PSO technique has been successfully used by previous researchers for developing prediction models [37, 230, 267]. Certain other studies have used hybridized PSO along with other algorithms to yield effective prediction models [183, 198, 268]. Thus, the successful use of PSO in classification tasks aids the accuracy of the individual PSO classifiers. Furthermore, it should also be noted that this chapter uses the CPSO technique as a base algorithm. It is an improved PSO technique as it uses constriction coefficients to restrict the possible explosion in the search space and provides faster convergence of PSO [95, 267].

This chapter evaluates and compares the performance of the four proposed ensemble classifiers with a) individual fitness-based classifiers and b) four well-known ML ensemble classifiers (RF, AB, BG and LB) for developing prediction models which determine the change-prone classes in a software. The chapter empirically

validates the results on datasets extracted from ten open-source software projects. Six of these software projects are Android application packages and the other four are widely used Apache software. The individual CPSO classifiers are based on seven different fitness functions, which are widely used performance measures (Accuracy, G-Mean1, G-Mean3, Balance, G-measure, F-measure and Precision) in the literature for evaluating prediction models [17]. The use of performance measures as fitness functions for SBA has been advocated by Harman and Clark [160].

Thus, the chapter investigates the following RQs:

*RQ1. Are the CPSO fitness-based classifiers diverse and accurate?*

This question ascertains whether the seven individual CPSO classifiers which are based on seven different fitness functions would be effective as constituents for an ensemble classifier.

*RQ1a) What is the accuracy of CPSO classifiers when different fitness variants are used? Does the accuracy vary with different fitness variants?*

The accuracy of CPSO fitness-based classifiers is evaluated by developing software change prediction models using ten-fold cross validation on ten open-source datasets investigated in the chapter. We use G-Mean1 and Balance measures as performance evaluators.

*RQ1b) Are the investigated individual CPSO classifiers based on different fitness variants diverse in nature?*

The diversity of CPSO fitness-based classifiers is evaluated pairwise. For each pair of CPSO fitness-based variants, we compute the percentage of correctly predicted change-prone classes by both the variants and by only a specific variant. These metrics depict the complementarity amongst different fitness-based classifiers.

*RQ2. What is the effectiveness of software change prediction models developed using the proposed ensemble classifiers (MVEC, WVEC, HIEC and WVHIEC) when compared with individual CPSO fitness-based classifiers and well-known ensemble*

*classifiers?*

This question evaluates the performance of the proposed voting ensemble classifiers for developing change prediction models. Furthermore, their performance is compared with different individual fitness-based classifiers as well as with ML ensemble classifiers using G-Mean1 and Balance performance measures.

*RQ2a) What is the predictive performance of proposed ensemble classifiers vs those developed using the seven individual fitness-based variant classifiers for developing software change prediction models?*

The results of change prediction models developed using individual CPSO fitness-based classifiers were evaluated in RQ1a. We compare their results with those of proposed ensemble classifiers. Furthermore, we use Friedman and Wilcoxon test to statistically evaluate the comparison results.

*RQ2b) What is the predictive performance of proposed ensemble classifiers vs those developed using well-known ML ensemble classifiers (RF, AB, BG and LB) for developing prediction models which determine change-prone classes?*

This question statistically compares the performance of proposed voting ensemble classifiers with four well-known ML classifiers for developing effective software change prediction models.

The chapter is organized in the following manner: Section 7.2 states the empirical research framework, while Section 7.3 explains the proposed ensemble classifiers along with their pseudocode. Section 7.4 states the experimental framework which includes the datasets used, feature selection techniques and other design considerations of the chapter. Section 7.5 states the results of the investigated RQs and analyzes them. Finally, Section 7.6 summarizes the findings of the chapter. The initial results of the chapter were published in [269], which were further expanded and published in [270].

## 7.2 Empirical Research Framework

This section states the predictors and the predicted variables used in the chapter. The CPSO technique and the various fitness functions used in the chapter are also mentioned.

### 7.2.1 Independent and Dependent Variables

This chapter uses OO metrics as predictors for determining software change. The metrics examined in the chapter belong to the CK metrics suite [16]. The details of the metrics suite can be referred from Chapter 2. Apart from the CK metrics suite, we also use SLOC metric as an independent variable. The change-proneness attribute of an OO class is the dependent variable investigated in the chapter.

### 7.2.2 CPSO Technique

The PSO technique simulates the "bird flocking" behaviour. The CPSO technique is a variant of PSO, which uses proper constriction coefficients efficiently to avoid the explosion of search space while searching for an optimum solution. The details of the CPSO technique and the parameter settings can be referred from section 2.6.8 (Chapter 2).

The various fitness variants of the CPSO technique were implemented using the Java language in the KEEL tool. We use the default parameter settings of the tool for the CPSO technique in this chapter, which can be referred from Chapter 2. As discussed in Chapter 5, Arcuri and Fraser [249] state that though, parameter settings have a strong influence on the performance of an algorithm, it is an expensive process, which may not always lead to significant improvement in results. Thus, the use of "default" parameter settings is reasonable. Furthermore, it may be noted that the

aim of the chapter was to propose and evaluate the effectiveness of fitness-based voting ensemble classifiers. We have used uniform parameter settings for ensemble classifiers as well as individual CPSO fitness-based classifiers. We do not intend to investigate the best parameter settings of different CPSO variants corresponding to specific datasets, which may lead to overfitting or over-optimistic results. Therefore, the use of default parameter settings is a practical choice for experiments conducted in this chapter.

### 7.2.3 Performance Measures as Fitness Functions

Literature studies have advocated that various performance measures are ideal to be employed as fitness functions [160]. This chapter investigates the use of seven different performance measures (Accuracy, G-Mean1, G-Mean3, Balance, G-measure, F-measure and Precision) as fitness functions for the CPSO technique. A detailed description of these performance measures is given in Chapter 2 (Section 2.10).

### 7.2.4 Validation Method used in Individual Classifiers

The proposed ensemble classifiers aggregate the outputs of the classification models developed using the individual CPSO fitness-based classifiers. Each individual classifier uses ten-fold cross validation method [113] for model development. Furthermore, since CPSO is stochastic in nature, we perform 30 runs for each of the individual CPSO fitness-based classifier and report the median values. Furthermore, we also report the mean values obtained by all the classifiers on each of the ten investigated datasets used in the chapter.

## 7.3 Proposed Ensemble Classifiers

The chapter proposes four voting ensemble classifiers of seven different CPSO fitness variants. The output of the individual CPSO fitness variants is aggregated using votes. The description along with pseudocode of all the four ensemble classifiers is presented in the following sections. Figure 7.1 shows the diagrammatic representation of the basic functioning of the proposed ensemble classifiers.

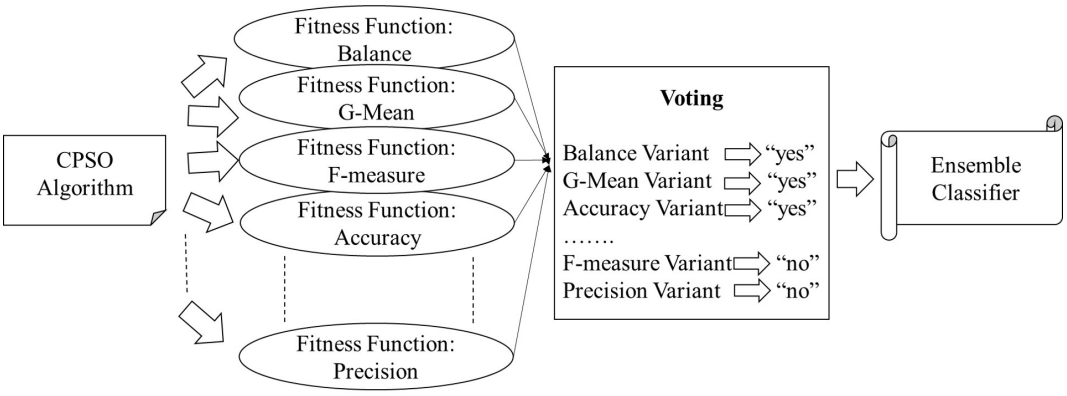


Figure 7.1: Basic Framework of the Proposed Ensemble Classifier

Here, each individual CPSO fitness variant either outputs a "yes" (change-prone) vote or a "no" (not change-prone) vote for a corresponding instance (data point). The ensemble classifier aggregates these "yes" and "no" votes for a data point and outputs the results. The aggregation done by the ensemble classifier could be a simple majority vote, i.e. output the result ("yes" or "no"), which is output by the majority of constituent fitness variants or other proposed weighted voting as explained in the following sections (7.3.1-7.3.4). For instance, in case of majority voting, if five of the constituent classifiers output a "yes" for a data point, then the ensemble's output would be "yes" for the corresponding data point. The nomenclature used in the pseudocodes is presented in Figure 7.2.

***D1, D2, D3, D4, D5, D6, D7, D8, D9, D10*** → Set of 10 Datasets.
***DPk$_i$*** → i$^{th}$ data point of ***Dk,*** which consists of CK metric values and dependent variable named ALTER.
***ALTER (DPk$_i$)*** → The ALTER value i.e. "yes" or "no" for a corresponding class of the data set.
***F1, F2, F3, F4, F5, F6, F7*** → Set of 7 CPSO fitness variants.
***Output (Fj, DPk$_i$)*** →Output prediction of data point ***DPk$_i$,*** *where k =1 to 10, i = 1 to n (number of data points in Dk)* by ***Fj,*** *where j =1 to 7.*
***G-Mean1 (Fj, Dk)*** → G-Mean1 value obtained by predictions of ***Fj,*** *where j =1 to 7* on dataset ***Dk,*** *where k =1 to 10.*
***Balance (Fj, Dk)*** → Balance value obtained by predictions of ***Fj,*** *where j =1 to 7* on dataset ***Dk,*** *where k =1 to 10.*

Figure 7.2: Nomenclature of Pseudocodes

### 7.3.1 Majority Voting Ensemble Classifier

MVEC aggregates the output votes of each individual fitness variant corresponding to a specific data point. It outputs the majority vote obtained for a corresponding data point as the ensemble's output. A pseudocode of the MVEC is shown in Figure 7.3.

```
MVEC (DPk_i) → Output prediction of data point DPk_i by MVEC.
For each Dk ∈ D, where k= 1 to 10
  For each Fj ∈ F, where j = 1 to 7
      Compute 10-fold cross-validation results of Fj on Dk
For each DPk_i ∈ Dk , where k= 1 to 10 and i= 1 to number of data points in Dk
    count-yes = count-no =0
    For each Output (Fj, DPk_i)
       If Output (Fj, DPk_i) == "yes"
          count-yes++
       Else
         count-no++
    If count-yes > count-no
       MVEC (DPk_i) = "yes"
    Else
      MVEC (DPk_i) ="no"
```

Figure 7.3: MVEC Pseudocode

```
WVEC (DPk_i) → Output prediction of data point DPk_i by WVEC.
For each Dk, where k= 1 to 10
    For each Fj, where j = 1 to 7
        Compute 10-fold cross-validation results of Fj on Dk
        Compute G-Mean1 (Fj, Dk)
        Compute Balance (Fj, Dk)
For each Dk, where k=1 to 10
    For each Fj, where j = 1 to 7
     Allocate Performance-Rank (Fj, Dk), according to G-Mean1 (Fj, Dh) & Balance (Fj,
Dh) on Dh, where h= {1 to 10} – {k}
(i.e. assign rank 7 to the Fj with highest G-Mean1 and Balance values on the other nine
datasets, assign rank 6 to Fj with second highest G-Mean1 and Balance values on the other
nine datasets and so on)
For each DPk_i ∈ Dk , where k= 1 to 10 and i= 1 to number of data points in Dk
     count-yes = count-no =0
    For each Output (Fj, DPk_i)
         If Output (Fj, DPk_i) == "yes"
            count-yes = count-yes * Performance-Rank (Fj, Dk)
         Else
            count-no = count-no * Performance-Rank (Fj, Dk)
    If count-yes > count-no
       WVEC (DPk_i) = "yes"
    Else
       WVEC (DPk_i) ="no"
```

Figure 7.4: WVEC Pseudocode

### 7.3.2 Weighted Voting Ensemble Classifier

WVEC first provides a weight to each fitness variant on a specific dataset, according to the predictive capability of a fitness variant on all other datasets of the chapter. The predictive capability is assessed by evaluating its performance in terms of G-Mean1 and Balance values on all the other datasets. Thereafter, the vote of a specific fitness variant is multiplied by its performance-rank and the majority vote obtained by cumulating the votes of all the fitness variants is output as the WVEC output. It may be noted that a better performing fitness variant is allocated a higher performance rank. The pseudocode of the WVEC is shown in Figure 7.4.

### 7.3.3 Hard Instance Ensemble Classifier

HIEC, like WVEC also provides a weight to each fitness variant on a specific dataset, according to its ability to classify hard instances on all the other datasets. An instance is termed as "hard to classify" if it was incorrectly classified by a majority of the fitness variants and correctly classified by only one or two of the fitness variants. We consider those instances as hard which are correctly predicted by very few, much lesser than half of the individual fitness variants. Since, we are considering seven fitness variants, instances which are correctly predicted by only one or two fitness variants are considered "hard to classify".

First, all "hard to classify" instances are identified in a dataset, then a Hard-ID value is computed for each individual fitness variant on each dataset. It is calculated by dividing the number of correctly predicted "hard to classify" instances of a specific dataset by the total number of data points (instances) in the specific dataset. For instance, in dataset A, there are 7 "hard to classify" instances, which are correctly predicted by the Precision fitness variant. The total number of instances in this dataset is 112. Thus, Hard-ID for Precision variant on dataset A is 7/112 = 0.062.

For a specific dataset, a classify rank is allocated to each fitness variant according to the Hard-ID values on all the other datasets except the corresponding dataset. The higher the values of Hard-ID a fitness variant obtains, the higher classify-rank is given to it. Finally, the vote of a fitness variant is multiplied by its obtained classify-rank, and the majority vote obtained by cumulating the votes of all the fitness variants is output as the HIEC output for a data point in a specific dataset. Figure 7.5 shows the pseudocode of HIEC.

```
HIEC (DPk_i) → Output prediction of data point DPk_i by HIEC.
Status (DPk_i) → "Hard" or "Easy" status of DPk_i.
Count-hard (Fj, Dk) → Number of "hard" instances correctly classified by Fj on Dk, where j
=1 to 7 & k =1 to 10.
Hard-ID (Fj, Dk) → Count-hard (Fj, Dk) / No. of Data Points in Dk, where j =1 to 7 & k =1
to 10.
For each Dk, where k= 1 to 10
    For each Fj, where j = 1 to 7
        Compute 10-fold cross-validation results of Fj on Dk
For each DPk_i ∈ Dk , where k= 1 to 10 and i= 1 to number of data points in Dk
        count-correct = count-incorrect =0
        For each Fj, where j = 1 to 7
                If Output (Fj, DPk_i) == ALTER (DPk_i)
                    count-correct ++
                Else
                    count-incorrect++
        If (count-correct ==1 or count-correct==2)
                Status (DPk_i) = "Hard"
        Else
                Status (DPk_i) = "Easy"
For each Dk, where k= 1 to 10
    For each Fj, where j = 1 to 7
        If Status (DPk_i) == "hard"
                Count-hard (Fj,Dk)++
For each Dk, where k= 1 to 10
    For each Fj, where j = 1 to 7
     Hard-ID (Fj, Dk) = Count-hard (Fj, Dk)/ No. of Data Points in Dk
For each Dk, where k= 1 to 10
    Allocate Classify-Rank (Fj, Dk), according to Hard-ID (Fj, Dh), where h= {1 to 10} – {k}
      (i.e. assign rank 7 to F_j with most number of highest Hard-ID's (Fj, Dk) values on all
      other data sets and so on.)
    For each DPk_i ∈ Dk , where k= 1 to 10 and i= 1 to number of data points in Dk
            count-yes = count-no =0
            For each Output (Fj, DPk_i)
                    If Output (Fj, DPk_i) == "yes"
                        count-yes = count-yes * Classify-Rank (Fj, Dk)
                    Else
                        count-no =  count-no * Classify-Rank (Fj, Dk)
            If count-yes > count-no
                        HIEC (DPk_i) = "yes"
            Else
                        HIEC (DPk_i) ="no"
```

Figure 7.5: HIEC Pseudocode

***WVHIEC*** $(DPk_i)$ → *Output prediction of data point* $DPk_i$ *by* ***WVHIEC.***
***Status*** $(DPk_i)$ → *"Hard" or "Easy" status of* $DPk_i$.
***Count-hard*** $(Fj, Dk)$ → *Number of "hard" instances correctly classified by* ***Fj*** *on* ***Dk****, where j =1 to 7 & k =1 to 10.*
***Hard-ID*** $(Fj, Dk)$ → ***Count-hard*** $(Fj, Dk)$ / ***No. of Data Points in Dk*** *, where j =1 to 7 & k =1 to 10.*
*For each* ***Dk****, where k= 1 to 10*
  *For each* ***Fj****, where j = 1 to 7*
    *Compute 10-fold cross-validation results of* ***Fj*** *on* ***Dk***
    *Compute* ***G-Mean1*** $(Fj, Dk)$
    *Compute* ***Balance*** $(Fj, Dk)$
*For each* ***Dk****, where k=1 to 10*
  *For each* ***Fj****, where j = 1 to 7*
    *Allocate* ***Performance-Rank*** $(Fj, Dk)$*, according to* ***G-Mean1*** $(Fj, Dh)$ & ***Balance*** $(Fj, Dh)$ *on* ***Dh****, where h= {1 to 10} – {k}*
    *(i.e. assign rank 7 to the* ***Fj*** *with highest G-Mean1 and Balance values on the other nine datasets, assign rank 6 to* ***Fj*** *with second highest G-Mean1 and Balance values on the other nine datasets and so on)*
*For each* ***Dk****, where k= 1 to 10*
  *For each* ***Fj****, where j = 1 to 7*
    *If* ***Status*** $(DPk_i)$ *== "hard"*
      ***Count-hard*** $(Fj, Dk)$++
*For each* ***Dk****, where k= 1 to 10*
  *For each* ***Fj****, where j = 1 to 7*
    ***Hard-ID*** $(Fj, Dk)$ = ***Count-hard*** $(Fj, Dk)$/ ***No. of Data Points in Dk***
*For each* ***Dk****, where k= 1 to 10*
  *Allocate* ***Classify-Rank*** $(Fj, Dk)$*, according to* ***Hard-ID*** $(Fj, Dh)$*, where h= {1 to 10} – {k}*
    *(i.e. assign rank 7 to* $F_j$ *with most number of highest Hard-ID's (Fj, Dk) values on all other data sets and so on.)*
  *For each* $DPk_i \in Dk$ *, where k= 1 to 10 and i= 1 to number of data points in* ***Dk***
    *count-yes = count-no =0*
    *For each* ***Output*** $(Fj, DPk_i)$
      *If* ***Output*** $(Fj, DPk_i)$ *== "yes"*
        *count-yes = count-yes* * ***Performance-Rank*** $(Fj, Dk)$ * ***Classify-Rank*** $(Fj, Dk)$
      *Else*
        *count-no = count-no* * ***Performance-Rank*** $(Fj, Dk)$ * ***Classify-Rank*** $(Fj, Dk)$
    *If count-yes > count-no*
      ***WVHIEC*** $(DPk_i)$ *= "yes"*
    *Else*
      ***WVHIEC*** $(DPk_i)$ *="no"*

Figure 7.6: WVHIEC Pseudocode

### 7.3.4 Weighted Voting Hard Instance Classifier

WVHIEC provides weights to the CPSO fitness variants on two grounds, i.e. both on the basis of performance (G-Mean1 and Balance values) and on the basis of ability to classify "hard instances". Thereafter, the vote of a fitness variant is multiplied by its performance-rank as well as classify-rank, and the majority vote obtained by cumulating the votes of all the fitness variants is output as the WVHIEC output. The pseudocode of the WVHIEC is shown in Figure 7.6.

## 7.4 Experimental Framework

The following section states the datasets, the feature selection technique, the performance measures and the statistical tests used in the chapter. The section also provides a background of ML ensemble classifiers which were compared with the proposed ensemble classifiers in the chapter. We also state the conditions, which were verified for selecting an individual fitness variant as a constituent of the proposed ensemble classifiers.

### 7.4.1 Empirical Data Collection

The datasets of the chapter were collected using the DCRS tool [106]. The chapter uses six application packages (Calendar 4.0.4-4.1.2, Contacts 4.0.4-4.1.2, Gallery 4.1.2-4.2.2, Bluetooth 5.0.2-5.1.0, MMS 4.0.4-4.1.2 and Telephony 4.2.2-4.3.1) of Android, a popularly used operating system for mobiles. Furthermore, the chapter also uses data extracted from four popular Apache software (Apache Commons IO 1.3-1.4, Apache Commons Math 3.1.1-3.2, Apache Log4j 1.2.16-1.2.17 and Apache Net 3.0-3.1). The details of data collection procedure can be referred from Chapter

2.

It should be noted that the allocation of weights in WVEC, HIEC and WVHIEC ensemble classifiers in a specific dataset is dependent on all other datasets. However, since we have investigated two categories of datasets, i.e. Android applications and Apache datasets, we divided the datasets into two subsets for weight allocation depending on their company i.e. Android datasets and Apache datasets. Therefore, the individual classifier's weights for a specific dataset depended on the performance of individual classifiers on all other datasets of the same company. For instance, the individual classifier's weights for Android Bluetooth dataset were based on only five other datasets, i.e. Calendar, Contacts, Gallery, MMS and Telephony. Similarly, the individual classifier's weights for Apache Net dataset were based on only three other datasets, i.e. Commons Math, Commons IO and Log4j.

### 7.4.2 Feature Selection Technique

This chapter uses CFS method [109], a widely used method for feature selection in ML applications as a feature selector. Table 7.1 states the OO metrics which resulted after application of the CFS method on each dataset.

Table 7.1: Metrics Selected by CFS

| Dataset | Metrics Selected |
|---|---|
| Android Bluetooth | WMC, SLOC, DIT, LCOM |
| Android Calendar | DIT, SLOC |
| Android Contacts | DIT, SLOC |
| Android Gallery | WMC, SLOC, CBO |
| Android MMS | WMC, SLOC |
| Android Telephony | WMC, CBO, SLOC |
| Apache Commons IO | CBO, LCOM |
| Apache Commons Math | WMC, SLOC |
| Apache Log4j | DIT, RFC, SLOC |
| Apache Net | WMC, SLOC, RFC, NOC |

### 7.4.3 Performance Measures & Statistical Evaluation

This study evaluates G-Mean1 and Balance performance measures for assessing the developed change prediction models. Studies in literature have advocated the use of these metrics as they are robust and stable to handle imbalanced datasets [118, 120, 121]. It may be noted that we use two of our investigated fitness functions (Section 7.2.3) as performance measures. Previous studies in the literature have also used the practice of using the same performance measure both as a fitness function as well as for validating the performance of a classifier [38, 40, 196].

The results of the chapter are statistically evaluated using two non-parametric tests, i.e. Friedman test and Wilcoxon test. RQ2 evaluates the capabilities of all the proposed ensemble classifiers along with individual fitness-variant classifiers as well as ML ensemble classifiers for developing effective software change prediction models. Friedman test is used in RQ2a for allocating mean ranks to each of the investigated classifiers. These ranks are assigned on the basis of G-Mean1 and Balance values obtained by prediction models developed using them and are symbolic of the capability of a classifier. A classifier obtaining a lower mean rank is better than a classifier which attains a higher mean rank. Furthermore, a post-hoc Wilcoxon signed rank test is performed in RQ2a to pairwise compare the capabilities of the proposed ensemble classifiers with each of the seven individual classifiers. A Friedman test was also performed in RQ2b to compare the performance of ML ensemble classifiers with the proposed ensemble classifiers.

### 7.4.4 ML Ensemble Classifiers

The chapter compares the results of the proposed ensemble classifiers with four other ML ensemble classifiers namely RF, AB, BG and LB. These classifiers were imple-

mented in the WEKA tool [88] and we use the default parameter settings of the tool for these classifiers, which are mentioned in Chapter 2 (Section 2.6.5). Though, change in default parameter settings might improve the performance of ML ensemble classifiers, it is difficult to explore all possible options in the parameter space [271]. Moreover, a recent study by Tantithamthavorn et al. [271] found that parameter settings of RF and LB had a relatively negligible impact on the performance of these techniques. Moreover, the WEKA tool is known to use sensible default values for a corresponding technique [272]. Thus, the choice of default parameters is reasonable for our experiments. It may be noted that we have not tuned CPSO algorithm too, as the process is costly, which does not ensure definite improvement in results. The use of default parameter settings for both ML ensemble classifiers and the CPSO algorithm ensures fair comparison amongst these techniques and does not introduce optimistic bias.

### 7.4.5 Candidates for Voting Ensemble

It should be noted that all classifiers might not be good candidates for formulating a voting ensemble. As indicated by Dietterich [42], two necessary conditions for a classifier to be a part of an ensemble is its individual accuracy and its diversity with other ensemble classifiers. In this chapter, we evaluated the following conditions before individual classifiers were used for developing voting ensembles:

- The accuracy of each individual classifier was evaluated on the basis of G-Mean1 and Balance values obtained by it on all the datasets. If in a majority of the datasets (greater than or equal to five), the G-Mean1 values obtained by an individual classifier is greater than or equal to 50% and the Balance values obtained by the classifier is greater than or equal to 50%, we term the classifier as accurate.

A rationale behind using 50% as a threshold for Balance values is as follows:

In practice, it is often found that datasets have uneven distribution, i.e., percentage of change-prone classes and that of not change-prone classes is skewed [118, 121]. For such datasets, it is difficult to develop models which attain high Balance values, since a high Balance value indicates a model which can come close to achieving a recall value of 1 (correctly predict all change-prone classes) and a PF value 0 (correctly determine all not change-prone classes). However, a model is not capable of learning the characteristics of both change-prone as well as not change-prone classes appropriately as there is an imbalance present in the training data. The characteristics of classes (change-prone or not change-prone), which are present in minority cannot be effectively learned by the model. Therefore, we chose a threshold of 50% for Balance values, for designating an individual classifier as accurate. A similar threshold was chosen by He et al. [115], for precision values, as it is difficult for a model to attain higher values due to presence of an imbalance in the datasets. Furthermore, we investigated a number of related literature studies which have used Balance as a performance measure for evaluating change prediction and defect prediction models. The Balance values reported by Tosun et al. [273] were in the range of 41-76%, those reported by Misirh et al. [54] were in the range of 34-86%. We reported Balance values in the range 30-75% in Chapter 6 and Li et al. [122] reported them in the range 49-79%. However, the majority of Balance values reported in these studies were above 50%. Thus, we use 50% as a threshold for Balance measure for designating an individual classifier as accurate. On similar grounds, we select the threshold for G-Mean1 values to be 50%.

We also considered an individual classifier if it was found to have high ca-

pability to identify "hard" instances (refer section 7.3.3). In such cases, the classifiers were accurate in terms of Hard-ID values (refer section 7.3.3).

- The diversity of a fitness variant classifier was adjudged by evaluating its uniqueness to correctly identify change-prone classes which could not be identified by other fitness variants. In cases, when a classifier is uniquely able to correctly predict less than 10% of change-prone classes for all the datasets when compared with all the other fitness variants, we drop the fitness variant indicating low diverse nature.

## 7.5 Results and Analysis

This section states the answers to the RQ's of the chapter and discusses the obtained results.

### 7.5.1 Results specific to RQ1

The accuracy and diversity of all the seven CPSO fitness-based classifiers are assessed by evaluating the performance of change prediction models developed using them. We discuss in detail the results obtained by change prediction models developed using ten-fold cross validation using CPSO fitness-based classifiers.

***Accuracy of CPSO fitness-based classifiers (RQ1a)***

The accuracy in terms of G-Mean1 (GM1) and Balance (Bal.) values of the developed change prediction models on the investigated datasets of the chapter is presented in Table 7.2. The table reports the median values of the 30 executions performed for each of the CPSO fitness variant for each dataset. The mean values obtained by each fitness variant over all the datasets is also stated in the table. We can also deduce

the fitness variant achieving the best value in a corresponding dataset (depicted in bold values). However, it should be noted that we only designate a fitness variant as best on a dataset if it attains good values both for G-Mean1 as well as Balance performance measures. For instance, on Commons IO dataset, both the Accuracy variant as well as the G-Mean1 variant gave the best results. The Accuracy variant obtained the best Balance values (60.24), while the G-Mean1 variant obtained the best G-Mean1 values.

Table 7.2: Validation Results of CPSO Fitness Variants

| **Dataset** | **Accuracy V.** | | **Precision V.** | | **G-Mean1 V.** | | **G-Mean3 V.** | | **F-measure V.** | | **G-measure V.** | | **Balance V.** | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *GM1* | *Bal.* | *GM1* | *Bal.* | *GM1* | *Bal.* | *GM1* | *Bal.* | *GM1* | *Bal.* | *GM1* | *Bal.* | *GM1* | *Bal.* |
| Bluetooth | 0.72 | 70.46 | 0.61 | 59.45 | **0.74** | **71.52** | 0.63 | 60.60 | 0.70 | 69.17 | 0.71 | 69.46 | 0.68 | 64.98 |
| Calendar | 0.48 | 47.40 | 0.47 | 45.61 | 0.57 | 54.61 | 0.53 | 50.31 | **0.59** | **56.66** | 0.57 | 54.35 | 0.55 | 52.49 |
| Contacts | 0.62 | 60.50 | 0.58 | 57.41 | **0.67** | **66.79** | 0.47 | 46.30 | 0.63 | 62.97 | 0.53 | 52.09 | 0.66 | 65.37 |
| Gallery | 0.49 | 48.20 | **0.51** | **51.21** | 0.46 | 45.99 | 0.29 | 35.21 | 0.47 | 46.69 | 0.48 | 47.54 | 0.45 | 45.64 |
| MMS | 0.69 | 67.29 | 0.59 | 57.32 | 0.67 | 62.64 | 0.58 | 54.49 | **0.71** | **70.54** | 0.70 | 69.64 | 0.69 | 67.53 |
| Telephony | 0.55 | 51.62 | **0.58** | **58.44** | 0.58 | 55.75 | 0.31 | 36.24 | 0.53 | 50.92 | 0.47 | 45.46 | 0.56 | 53.45 |
| IO | **0.62** | **60.24** | 0.56 | 54.64 | **0.63** | **59.96** | 0.55 | 52.40 | 0.40 | 41.6 | 0.00 | 29.25 | 0.60 | 57.06 |
| Math | 0.59 | 54.32 | 0.59 | 54.32 | 0.59 | 54.32 | 0.59 | 54.32 | 0.59 | 54.32 | 0.59 | 54.32 | 0.59 | 54.32 |
| Log4j | 0.46 | 45.12 | 0.44 | 43.97 | **0.60** | **59.33** | 0.33 | 37.30 | 0.38 | 39.68 | 0.38 | 39.7 | **0.61** | **59.65** |
| Net | 0.62 | 60.62 | 0.39 | 40.55 | 0.63 | 62.64 | 0.64 | 64.10 | 0.63 | 62.54 | 0.63 | 63.05 | **0.66** | **65.72** |
| Mean | 0.58 | 56.58 | 0.53 | 52.29 | 0.61 | 59.35 | 0.49 | 49.07 | 0.56 | 55.51 | 0.51 | 52.49 | 0.61 | 58.62 |
| V. indicates variant, GM1 indicates G-Mean1 & Bal. indicates Balance | | | | | | | | | | | | | | |

Furthermore, we also made the following observations:

- The mean G-Mean1 values attained by CPSO fitness variants varied from 0.49-0.61. Similarly, the variation in mean Balance values was in the range of 49.07-59.35. These mean values indicate satisfactory performance of the investigated CPSO fitness variants on all the datasets used in the chapter. Thus, the investigated CPSO fitness-based variants are accurate.

- The results obtained by different CPSO fitness variants varied on each of the corresponding datasets. For instance, the G-Mean1 values obtained by all the fitness variants on the Log4j dataset were in the range of 0.33-0.61. Similarly, the Balance values obtained by all the fitness variants, varied from 37.30-59.65.

The results of only one dataset, i.e. Commons Math was an exception to this observation. As observed from values depicted in Table 7.2, there was no change in the results obtained by any of the CPSO fitness variant.

- A specific fitness variant might obtain the best results on a specific dataset. However, on the contrary the same variant might obtain poor results on some other dataset. For instance, the F-measure variant attained the best results on Calendar dataset (G-Mean1: 0.59, Balance: 56.66) and MMS dataset (G-Mean1: 0.71, Balance: 70.54), but it obtained poor results on Commons IO dataset (G-Mean1: 0.40, Balance: 41.60). This indicates that there may not be a universal fitness variant which achieves best results on each dataset. The performance of a specific fitness variant varies from dataset to dataset.

As the G-Mean1 and Balance values obtained by the fitness variants were in the acceptable range (G-Mean1 $\geq 50\%$ and Balance $\geq 50\%$) in the majority of the cases, we designate all classifiers as accurate, which can be used as candidates in voting ensembles. From the above observations, we can also see the impact of a fitness function on the results of the developed change prediction models using the CPSO technique. The reason for such an observation is the role of the fitness function in choosing optimum solution candidates. A fitness function guides the traversal for an optimum solution through the search space of candidate solutions, thereby affecting the results of the developed models. The discussions stated in this section confirm the accuracy of the investigated CPSO fitness variants. The observations also indicate that the accuracy of corresponding CPSO fitness-based classifiers differs due to the choice of fitness functions.

***Diversity of CPSO fitness-based classifiers (RQ1b)***

The diversity amongst the CPSO fitness-based variants is assessed by analyzing the complementarity amongst them. The complementarity is evaluated amongst a

pair of fitness variants. We compute three metrics for all corresponding pairs of fitness variants: an intersection result and two difference results [56]. For a pair of fitness variants ($X_1$ and $X_2$), we analyze the metrics described below. CCP corresponds to correct change-prone classes:

$$X_1 \cap X_2 = \frac{Number\ of\ CCP\ classes\ predicted\ by\ both\ X_1\ \&\ X_2}{Number\ of\ CCP\ classes\ predicted\ by\ either\ or\ both\ X_1\ \&\ X_2}\%$$

$$X_1 - X_2 = \frac{Number\ of\ CCP\ classes\ predicted\ by\ X_1\ but\ not\ by\ X_2}{Number\ of\ CCP\ classes\ predicted\ by\ either\ or\ both\ X_1\ \&\ X_2}\%$$

$$X_2 - X_1 = \frac{Number\ of\ CCP\ classes\ predicted\ by\ X_2\ but\ not\ by\ X_1}{Number\ of\ CCP\ classes\ predicted\ by\ either\ or\ both\ X_1\ \&\ X_2}\%$$

$X_1 \cap X_2$ represent the percentage of classes which are change-prone in nature and are correctly identified by both the evaluated CPSO fitness variants. $X_1 - X_2$ and $X_2 - X_1$ represent the percentage of classes which are change-prone in nature, but are correctly identified by either of the evaluated CPSO fitness variants. Table 7.3 depicts the complementarity percentages amongst all pairs of the evaluated CPSO fitness variants. However, because of space limitations, Table 7.3 presents the aggregated results over all the ten datasets of the chapter. The notation used in the Table 7.3 is as follows: Acc. indicates Accuracy, Pr. indicates Precision; GM1 indicates G-Mean1; GM3 indicates G-Mean3, Fmes. indicates F-measure; Gmes. indicates G-measure and Bal. indicates Balance.

According to Table 7.3, the results of the metric $X_1 \cap X_2$ are in the range of 64%-85%. This indicates that 64%-85% of change-prone classes are overlapping and can be correctly predicted by an arbitrary CPSO fitness variant. The cumulative results of $X_1 - X_2$ and $X_2 - X_1$ metrics are in the range of 8%-36%, which represent the per-

centage of non-overlapping classes which are correctly predicted by only a specific fitness variant. This indicates that the investigated CPSO fitness variants are diverse in nature as we require the capabilities of a specific fitness variant for correctly determining these non-overlapping change-prone classes. An ensemble classifier which aggregates the results of these CPSO fitness variant classifiers may yield improved results if proper weights are allocated to the votes of individual constituent CPSO fitness variants.

Table 7.3: Complementarity Results of CPSO Fitness Variants

| | **Acc. V.** | | | **Pr. V.** | | | **GM1 V.** | | | **GM3 V.** | | | **Fmes. V.** | | | **Gmes. V.** | | | **Balance V.** | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *I* | *D1* | *D2* | *I* | *D1* | *D2* | *I* | *D1* | *D2* | *I* | *D1* | *D2* | *I* | *D1* | *D2* | *I* | *D1* | *D2* | *I* | *D1* | *D2* |
| **Acc. V.** | X | X | X | | | | | | | | | | | | | | | | | | |
| **Pr. V.** | 67 | 15 | 18 | X | X | X | | | | | | | | | | | | | | | |
| **GM1 V.** | 82 | 9 | 9 | 68 | 17 | 15 | X | X | X | | | | | | | | | | | | |
| **GM3 V.** | 70 | 15 | 15 | 71 | 16 | 13 | 64 | 18 | 18 | X | X | X | | | | | | | | | |
| **FMes. V.** | 89 | 5 | 6 | 69 | 16 | 15 | 81 | 9 | 10 | 76 | 12 | 12 | X | X | X | | | | | | |
| **GMes. V.** | 80 | 9 | 11 | 67 | 17 | 16 | 75 | 12 | 13 | 81 | 9 | 10 | 89 | 5 | 6 | X | X | X | | | |
| **Bal. V.** | 80 | 8 | 12 | 68 | 16 | 16 | 88 | 4 | 8 | 66 | 16 | 18 | 78 | 10 | 12 | 72 | 13 | 15 | X | X | X |
| I: $X_1 \cap X_2$; D1: $X_1 - X_2$ ; D2: $X_2 - X_1$; X: No comparison possible; V.: Variant | | | | | | | | | | | | | | | | | | | | | |

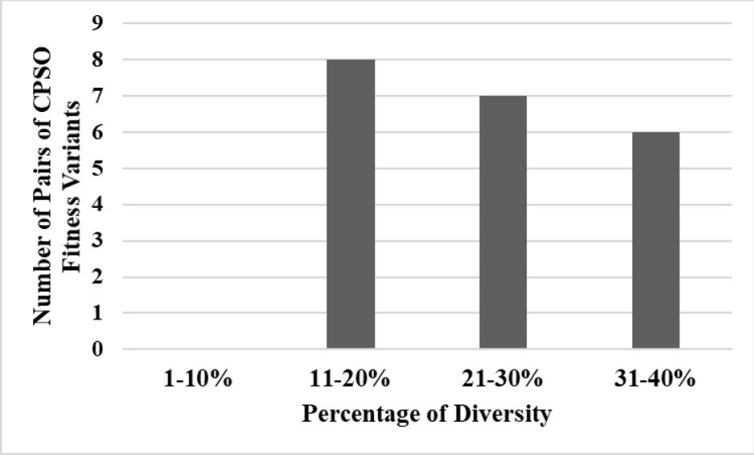


Figure 7.7: Number of pairs of fitness variants with varying diversities

Figure 7.7 depicts the number of pairs which shows the cumulative results of $X_1 - X_2$ and $X_2 - X_1$ in a particular range. It may be noted from the figure, that no pair of fitness variants showed a cumulative non-overlapping percentage of classes

less than 10%. All the pairs, ranged in the categories of 11-40%. This indicates that all pairs of fitness variants are diverse from each other. Thus, it is important to include them as voting candidates in an ensemble. Moreover, the non-overlapping classes cannot be ignored as an incorrect prediction of such a large number of classes would lead to poor classification models. Therefore, using these diverse fitness variants would improve the performance of the ensemble classifier.

***Answer to RQ1***

The investigated CPSO fitness variant classifiers are accurate for determining change-prone classes with majority of G-Mean1 values in the range of 0.50-0.74 and Balance values in the range of 50-70% over all the ten evaluated datasets in the chapter. The change prediction results of the CPSO fitness variants are also diverse as 8-36% of change-prone classes are correctly predicted by only a specific fitness variant. Thus, aggregation of these accurate and diverse CPSO fitness variants using ensemble methodology should yield improved results for software change prediction.

### 7.5.2 Results specific to RQ2

We first evaluate the performance of change prediction models developed using the proposed ensemble classifiers. The models developed using the proposed classifiers were formulated by aggregating the votes of individual classifiers and providing them with certain weights. These models are then compared to the individual CPSO fitness-based classifiers and four well-known ensemble classifiers.

***Proposed ensemble classifiers vs individual fitness-based variant classifiers (RQ2a)***

The G-Mean1 (GM1) and Balance (Bal.) values obtained from the models developed using ensemble classifiers are depicted in Table 7.4. The following observations can be made from Table 7.4:

- The majority of G-Mean1 and Balance values of the ensemble classifiers ranged from 0.54-0.77 and 51.09-76.50% respectively on the investigated datasets. This suggests the effectiveness of ensemble classifiers in developing software change prediction models.

- Amongst the ensemble classifiers, in a majority of the cases, the highest G-Mean1 values on a corresponding dataset was obtained by either the HIEC or the WVHIEC models. Likewise, in the majority of the datasets, the highest Balance values were obtained by the WVHIEC models and the HIEC models.

- In a majority of the datasets, the performance of the ensemble classifiers amongst themselves differed by 0.0-0.8 in the case of G-Mean1 values and 0.0-7.52 in the case of Balance values. However, it was observed that there was no difference in the results of the ensemble classifiers on Commons Math dataset. This is because each of its constituent fitness variants obtained similar results. Thus, providing different weights to similar performing constituent does not change the results achieved by the ensemble classifiers.

Table 7.4: Validation Results of Ensemble Classifiers using G-Mean1 and Balance Values

| **Dataset** | **MVEC** | | **WVEC** | | **HIEC** | | **WVHIEC** | |
|---|---|---|---|---|---|---|---|---|
| | *GM1* | *Bal.* | *GM1* | *Bal.* | *GM1* | *Bal.* | *GM1* | *Bal.* |
| Bluetooth | 0.75 | 74.23 | 0.76 | 75.98 | 0.75 | 74.71 | 0.77 | 76.50 |
| Calendar | 0.55 | 52.29 | 0.57 | 54.35 | 0.57 | 54.61 | 0.57 | 54.35 |
| Contacts | 0.67 | 66.70 | 0.65 | 64.75 | 0.67 | 66.71 | 0.67 | 66.70 |
| Gallery | 0.48 | 47.81 | 0.48 | 47.81 | 0.50 | 49.64 | 0.50 | 49.64 |
| MMS | 0.72 | 71.93 | 0.71 | 70.40 | 0.75 | 74.37 | 0.71 | 70.84 |
| Telephony | 0.56 | 52.39 | 0.55 | 51.62 | 0.52 | 49.33 | 0.54 | 50.92 |
| Commons IO | 0.54 | 51.18 | 0.54 | 51.09 | 0.60 | 57.06 | 0.62 | 58.61 |
| Commons Math | 0.59 | 54.32 | 0.59 | 54.32 | 0.59 | 54.32 | 0.59 | 54.32 |
| Log4j | 0.38 | 39.71 | 0.36 | 38.87 | 0.36 | 38.88 | 0.36 | 38.87 |
| Net | 0.66 | 65.88 | 0.69 | 67.77 | 0.68 | 67.46 | 0.68 | 67.45 |
| Mean Values | 0.59 | 57.64 | 0.59 | 57.70 | 0.60 | 58.71 | 0.60 | 58.82 |

Furthermore, we compared the performance of the ensemble classifier models

amongst each other and with the individual fitness variants using the Friedman test on G-Mean1 and Balance values. The Friedman test using G-Mean1 values obtained a p-value of 0.006 and a chi-square value of 24.49 with 10 degrees of freedom, thus indicating significant results with 95% confidence. The Friedman test results on Balance values were also significant with a p-value of 0.006, chi-square value of 24.75 and degrees of freedom as 10. The ranks obtained by the models developed using various classifiers are depicted in Table 7.5.

Table 7.5: Friedman Ranks obtained by various Classifiers

| Classifier | Ranks using G-Mean1 | Ranks using Balance |
|---|---|---|
| WVHIEC | 4.10 | 4.20 |
| HIEC | 4.45 | 4.30 |
| G-Mean1 Variant | 4.80 | 4.65 |
| MVEC | 5.00 | 4.95 |
| WVEC | 5.20 | 5.40 |
| Balance Variant | 5.85 | 5.75 |
| Accuracy Variant | 6.25 | 6.15 |
| F-measure Variant | 6.60 | 6.80 |
| Precision Variant | 7.05 | 7.00 |
| G-measure Variant | 7.40 | 7.50 |
| G-Mean3 Variant | 9.30 | 9.30 |

The results presented in Table 7.5 show that the WVHIEC ensemble classifier obtained the best Friedman rank using both G-Mean1 values and Balance values for determining change-prone classes amongst all the ensemble classifiers and their constituent fitness variants. The next best rank was allocated to HIEC in terms of both G-Mean1 and Balance performance measures. The MVEC and WVEC classifier obtained lower ranks than the G-Mean1 variant using both G-Mean1 and Balance values.

It was observed that the majority of models developed using different fitness variants (except G-Mean1 fitness variant) were ranked lower than the proposed ensemble classifiers in terms of both G-Mean1 and Balance performance measures. The HIEC and the WVHIEC models obtained improved results in the majority of the

cases when compared with the constituent fitness variants. The G-Mean1 results of the constituent classifiers were improved by 2-70% and the Balance results were improved by 2-66% with the use of WVHIEC models in a majority of the cases. Similar improvements were analyzed with the use of HIEC models. This observation supports the use of the proposed ensemble classifiers as they are effective in developing change prediction models which are better than those developed by individual constituent fitness variants. Thus, the ensemble methodology was effective in terms of software change prediction using different CPSO fitness variants.

Table 7.6: Wilcoxon Test Results using G-Mean1 and Balance Values

| **Fitness Variant** | **G-Mean1 Values** | | | | **Balance Values** | | | |
|---|---|---|---|---|---|---|---|---|
| | **MVEC** | **WVEC** | **HIEC** | **WVHIEC** | **MVEC** | **WVEC** | **HIEC** | **WVHIEC** |
| Accuracy V. | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ |
| Precision V. | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ |
| G-Mean1 V. | ↓ | ↓ | = | ↑ | ↓ | ↓ | ↑ | ↑ |
| G-Mean3 V. | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ |
| F-measure V. | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ |
| G-measure V. | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ |
| Balance V. | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ | ↑ |
| V. represents Variant | | | | | | | | |

Moreover, since the Friedman results were significant, we performed a Wilcoxon post-hoc test to pairwise compare the performance of the models developed using ensemble classifiers with each of its constituent fitness variant. The test was conducted on the basis of G-Mean1 and Balance values obtained by the classifiers on all the datasets used in the study at a cut-off of 0.05. Table 7.6 depicts the Wilcoxon signed rank test results using G-Mean1 and Balance performance measures. Three possible symbols are used to denote the results. "↑" indicates that the model developed using the ensemble classifier is superior to the model developed using the compared constituent fitness variant but the result is not significant. "↓" indicates that the model developed using the ensemble classifier is inferior to the model developed using the compared constituent fitness variant but the result is not significant. "=" indicates

that the model developed using the ensemble classifier is equivalent to the model developed using the compared constituent fitness variant.

According to the Wilcoxon test result conducted on G-Mean1 values (Table 7.6), the models developed using the ensemble classifiers were better than the compared constituent fitness variant in a majority of the cases. Similar results were seen using Balance values. However, the results were not significant. It may be noted that the G-Mean1 fitness variant gave better or equivalent results when compared with the MVEC, WVEC and HIEC models but poor results than the WVHIEC models. However, these results were again not significant. This is because the G-Mean1 variant is specifically effective in optimizing the performance measure i.e. G-Mean1, thus it produces effective results in terms of G-Mean1.

The reason for better results depicted by the WVHIEC classifier is the criteria for allocating weights to individual constituent classifiers. The WVHIEC classifier provides weights to both attributes of a classifier i.e. its performance, which is examined on the basis of G-Mean1 and Balance values as well as its competence in correctly determining the label of "hard" instances. Since, hard instances are incorrectly predicted by the majority of individual classifiers, their correct prediction leads to a considerable improvement in the performance of the developed model. Moreover, the correct prediction of these "hard instances" is the reason for successful performance of the HIEC classifier as it allocates weights to individual classifiers only on the basis of their "hard" instance prediction capability. An instance, which is not "hard" will be correctly predicted by even the MVEC classifier, which takes into account the majority votes. However, the MVEC classifier will completely ignore the "hard" instances. A similar reason is attributed to the slightly poor performance of the WVEC classifier. Since, the classifier allocates weights in order to optimize only the performance (G-Mean1 and Balance values), it ignores the ability of individual classifiers to correctly identify certain instances well, which were incorrectly

predicted by most of the constituent variants.

We also performed a Wilcoxon test to compare the pairwise performance of change prediction models developed by HIEC and WVHIEC with those developed by other ensemble classifiers (MVEC and WVEC). According to the results obtained by Wilcoxon test on G-Mean1 and Balance values, in a majority of the cases the HIEC and WVHIEC classifiers were better than the MVEC and WVEC classifiers. However, this superiority was not found to be significant. Moreover, the WVHIEC classifier was found to be not significantly better than the HIEC classifier.

***Proposed ensemble classifiers vs ML ensemble classifiers (RQ2b)***

We first evaluate the performance of the change prediction models developed using ML ensemble classifiers RF, AB, BG and LB on the investigated datasets by assessing their G-Mean1 (GM1) and Balance (Bal.) values. The models were developed using the ten-fold cross-validation method and the performance measure values are depicted in Table 7.7.

Table 7.7: Validation Results of ML Ensemble Classifiers using G-Mean1 and Balance Values

| **Dataset** | **RF** | | **AB** | | **BG** | | **LB** | |
|---|---|---|---|---|---|---|---|---|
| | *GM1* | *Bal.* | *GM1* | *Bal.* | *GM1* | *Bal.* | *GM1* | *Bal.* |
| Bluetooth | 0.75 | 73.71 | 0.67 | 65.95 | 0.66 | 65.17 | 0.63 | 60.19 |
| Calendar | 0.61 | 60.29 | 0.71 | 70.18 | 0.66 | 65.84 | 0.68 | 67.77 |
| Contacts | 0.57 | 55.74 | 0.55 | 53.02 | 0.62 | 61.02 | 0.59 | 56.89 |
| Gallery | 0.70 | 69.70 | 0.71 | 70.19 | 0.71 | 71.46 | 0.73 | 72.19 |
| MMS | 0.71 | 69.68 | 0.71 | 70.33 | 0.69 | 67.40 | 0.70 | 69.18 |
| Telephony | 0.51 | 51.07 | 0.59 | 57.39 | 0.54 | 52.84 | 0.56 | 54.09 |
| Commons IO | 0.69 | 67.33 | 0.68 | 67.97 | 0.68 | 66.30 | 0.70 | 69.13 |
| Commons Math | 0.59 | 54.72 | 0.59 | 54.32 | 0.59 | 54.32 | 0.59 | 54.32 |
| Log4j | 0.57 | 54.23 | 0.55 | 52.22 | 0.44 | 43.52 | 0.57 | 53.74 |
| Net | 0.73 | 72.40 | 0.76 | 76.20 | 0.76 | 75.20 | 0.72 | 71.15 |
| Mean Values | 0.64 | 62.89 | 0.65 | 63.78 | 0.64 | 62.31 | 0.65 | 62.87 |

The mean of G-Mean1 and Balance values over all the investigated datasets of the study is reported. According to the table, the mean G-Mean1 values obtained by RF, AB, BG and LB techniques are 0.64, 0.65, 0.64 and 0.65 respectively. Similarly,

the mean Balance values obtained by these ensemble classifiers were in the range of 62.31-63.78. Thus, the ML ensemble classifiers were efficient in determining change-prone classes as investigated by previous studies [5, 27, 30, 36].

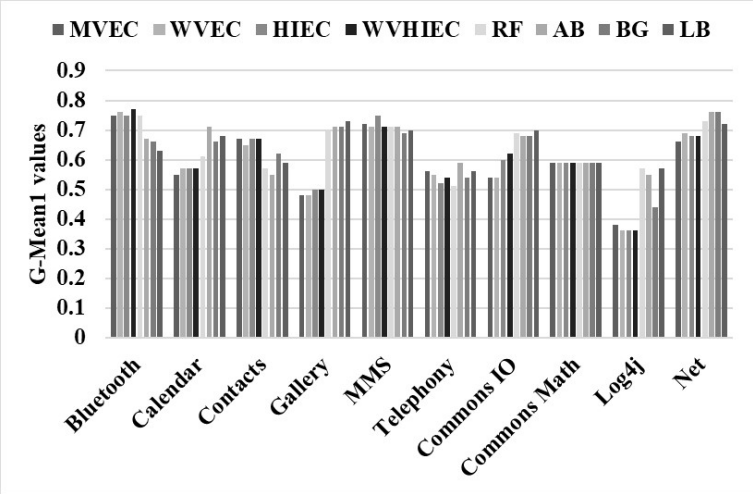


Figure 7.8: Comparative Results of Proposed and ML Ensemble Classifiers using G-Mean1 values

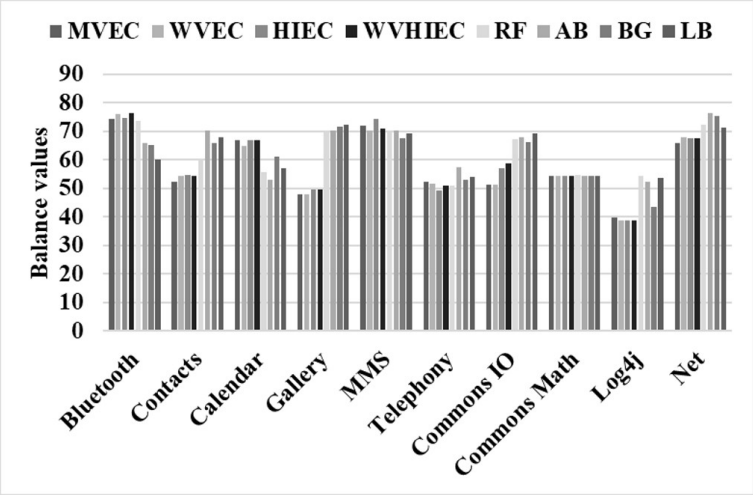


Figure 7.9: Comparative Results of Proposed and ML Ensemble Classifiers using Balance values

We compared the G-Mean1 and Balance values obtained by the proposed en-

semble classifiers and the ML ensemble classifiers in Figures 7.8-7.9. According, to Figure 7.8, the G-Mean1 values obtained by the proposed ensemble classifiers were comparable to that of the ML ensemble classifiers and were in fact better than the ML ensemble classifiers in few cases. However, the results of the proposed ensembles were poorer than the results of ML ensemble classifiers in certain cases. According to Figure 7.9, there was variation in the results of Balance values using the proposed ensemble classifiers and the ML ensemble classifiers. In 3 out of 10 datasets, the performance of the proposed ensemble classifiers was superior to ML ensemble classifiers. However, in 6 out of 10 datasets, the performance of ML ensemble classifiers was better. We performed the Friedman test to evaluate whether this superiority is statistically significant. The results of Friedman test using G-Mean1 and Balance values were found to be not significant at the 0.05 cut-off with p-value of 0.174 and 0.643 respectively. This indicates that there is no statistical difference in the performance of the proposed ensemble classifiers and ML ensemble classifiers. Thus, we can deduce that the performance of the proposed ensemble classifiers were comparable to that of the ML ensemble classifiers for prediction of change-prone classes.

***Answer to RQ2***

Software change prediction models developed using the proposed ensemble classifiers were effective as the models obtained acceptable G-Mean1 and Balance values (Mean G-mean values: 0.59-0.60, Mean Balance values: 57.64-58.82). With the help of statistical tests, it was ascertained that the results of individual fitness classifiers were improved by the proposed ensemble classifiers. Also, the proposed ensemble classifiers were found to be competent with the ML ensemble classifiers for determining change-prone classes.

### 7.5.3 Analysis of Results

The results of the study indicate the effectiveness of the proposed ensemble classifiers in predicting software change. Furthermore, their performance was found superior to individual CPSO fitness-based classifiers. The main cause of such a result is the accuracy and diversity of individual classifiers which were used as constituents of voting ensembles. Though, each fitness-based classifier gave accurate results, in order to improve their accuracy it was necessary to correctly predict non-overlapping change-prone classes. These classes were correctly predicted by only specific fitness variants. This could be done only by allocating proper weights to predictions of individual classifiers so that they can output correct results for these non-overlapping classes. This phenomenon enhanced the efficiency of the classification model as, more number of classes were predicted correctly. Also, it should be noted that a specific fitness variant did not perform well for all the datasets. Thus, it was necessary to use the ensemble of multiple fitness variants to attain improved results. The proposed ensemble classifiers were also found competent with the ML ensemble classifiers for predicting software change.

Amongst the proposed ensemble classifiers, the WVHIEC and HIEC classifier models gave better results. This result can be attributed to the fact that the WVHIEC model allocates weights to individual classifiers on the basis of both i.e. their accuracy (adjudged by G-Mean1 and Balance values) and their ability to correctly predict "hard" instances. It allocates weights to individual classifiers in such a manner so that both the accuracy as well as ability to predict "hard instance" capability is optimized. In WVHIEC, a classifier which has a moderate accuracy and a moderate capability to predict "hard" instance, will be given higher weights than the classifier which only has high accuracy but is incapable of predicting "hard instances". As "hard" instances are rarely predicted correctly, their correct prediction results in

substantial improvement of the developed prediction model resulting in its superior performance. As discussed in section 7.5.1, 64-85% of classes can be correctly predicted by any arbitrary fitness variant classifier. It is the correct prediction of the remaining non-overlapping classes, i.e. hard instances, which contributes to the superiority of the developed model. This is the reason for the HIEC model also to do well. The MVEC model primarily focused on the majority vote, thus missing a number of non-overlapping "hard" instances as they were incorrectly predicted by a majority of fitness-based classifiers. The results of the WVEC classifier were not that effective as the ensemble classifier allocated weights only on the basis of better G-Mean1 and Balance values, ignoring the capability to predict "hard" instances.

However, the results of this chapter differ from our previous work [269], which found the WVEC classifier as the best one. The reason for this difference in results is the change in procedure for weight allocation to votes of different individual classifiers. In the previous study, the weights of WVEC, HIEC and WVHIEC were allocated only on the basis of the performance of the individual classifiers on the corresponding dataset. For instance, it was already known that fitness variant 'X' obtains the best G-Mean1 and Balance value on dataset 'A'. Since, the WVEC classifier was allocated weights by already having the knowledge of the best fitness variant on a specific dataset, it showed the best results. However, this was not proper as the weights were allocated after the actual nature (change-prone or not change-prone) of the classes was known. In a real life scenario, this is not practical. The weights should be allocated without knowing the actual performance of the fitness variant classifier on a specific dataset. Thus, in this study, we allocated weights on the basis of the performance of a fitness variant on other datasets of the same company. For instance, in order to allocate weights to fitness variants for developing a model on a dataset A, we analyze the performance of all the fitness variants on all other datasets of the same company except dataset A. Thus, here the allocated weights are a true

representation of the actual capability of an individual fitness variant. Though the WVEC model performed well in the current analysis too, its results were inferior to than those obtained by the HIEC and WVHIEC models.

## 7.6 Discussion

The chapter proposes four ensemble classifiers namely MVEC, WVEC, HIEC and WVHIEC to determine change-prone classes in an OO software. Each of the proposed ensemble classifier uses seven fitness variants of CPSO. The individual fitness variants of CPSO are each based on a different fitness function and learn together from the same training set. The output of the fitness variants is aggregated by a weighted voting method, where votes of each classifier are allocated weights on different parameters such as its accurate performance and ability to identify hard instances. In order to empirically validate the performance of the proposed ensemble classifiers, the chapter uses ten open-source datasets (six Android application packages and four Apache applications). The results of the chapter are statistically assessed on the basis of two stable performance measures, i.e., G-Mean1 and Balance.

1. The seven investigated fitness variants were found to be accurate and diverse. The constituent CPSO fitness variants were accurate as the mean G-Mean1 and the mean Balance values obtained by them over all the investigated datasets in the chapter were in the range of 0.51-0.61 and 52.29-59.35 respectively. The diversity of the constituent CPSO fitness variants was adjudged by evaluating the pairwise complementarity amongst them. It was found that 64-85% of the correctly predicted change-prone classes were overlapping and could be predicted correctly by any arbitrary constituent CPSO fitness variant. However, in order to correctly predict the non-overlapping change-prone classes, we require

the capabilities of specific fitness variants, which are diverse in nature.

2. We evaluated whether the use of ensemble methodology obtains effective results for predicting change-prone classes using different fitness-based variants of the CPSO technique. The results indicate improved performance of the proposed ensemble classifiers over the individual fitness-based variant classifiers. It was observed that there was an improvement of up to 70% and 66% respectively in the G-Mean1 and Balance values obtained by the HIEC and WVHIEC models. The improvement in the results was statistically evaluated using Friedman test and was found to be significant. The reason for favourable results of the proposed ensemble classifiers was the accuracy and diversity of the constituent CPSO fitness-based variants. Their diversity would result in an improved classifier by predicting higher number of correct change-prone classes. The Wilcoxon test results depicted an improvement in G-Mean1 and Balance performance measures using the HIEC and WVHIEC models, though these results were not significant.

3. It was found that the HIEC and WVHIEC ensemble classifiers performed better than the other proposed ensemble classifiers. The chapter also compared the performance of MVEC, WVEC, HIEC and WVHIEC with four ML ensemble classifiers (RF, AB, BG and LB). The results indicated that the performance of the proposed ensemble classifiers were found comparable to that of the four ML ensemble classifiers.

# Chapter 8

# Dynamic Selection of Fitness Function using Particle Swarm Optimization

## 8.1 Introduction

The effective application of SBA in the domain of software change prediction has urged researchers to continuously examine new approaches, which enhance the predictive capabilities of these techniques. One such approach i.e. ensemble of various fitness variants has been explored in Chapter 7. We have investigated that the effectiveness of SBA is highly dependent on the selection of an optimum fitness function as the search for an effective solution in a search-based algorithm is guided by a fitness function. Though researchers in the past have used SBA for developing change prediction models [34–36], they have used only a single fitness function for the entire dataset. However, this scenario does not take into consideration the possibility that few instances of a dataset could be best predicted by a certain fitness function while few others could be correctly predicted by a different fitness function. This phenomenon could be true due to varied structural characteristics (design metrics) of

the data instances, which may be effectively learned and predicted by different fitness variants of the same technique (same technique using different fitness criteria). Using a varied fitness function for subsets of a dataset could lead to a considerable improvement in the performance of the developed prediction model as it would combine the correct classifications of different fitness variants resulting in better models. Thus, this chapter proposes an adaptive framework, namely Adaptive Selection of Optimum Fitness (ASOF). The framework predicts a dynamic fitness function for each specific instance of a dataset on the basis of its structural characteristics.

The motivation behind ASOF is the nature of SBA, as they search for an optimal solution in the candidate solution space with the aid of a fitness function. The choice of fitness function is critical as it ascertains the suitability of a candidate solution, determining whether it is better or worse than the current solution [6]. Researchers in the past have validated that the selection of a fitness function influences the results of a prediction model [47, 49]. Similar observations were yielded by Chapter 7. Therefore, the decision of selecting a specific fitness function is crucial. In accordance with the above discussion, the aim of this chapter is to choose the best fitness function for each data instance rather than the entire dataset. Therefore, a novel framework for the adaptive selection of a dynamic optimum fitness function for each instance of the dataset is proposed, while developing software change prediction models. The fitness function is selected on the basis of structural characteristics of the corresponding data sample.

The predictive models in this chapter are developed using the CPSO technique, with seven different fitness functions to create individual fitness variants. The framework predicts the best amongst seven fitness variants to be used for a corresponding data sample. As discussed in Chapter 7, the choice of CPSO as a base technique is motivated due to numerous advantages of PSO [210, 264, 265] and forbiddance of search-space explosion [267]. Also, similar to Chapter 7, seven different fitness

variants of CPSO were coded using seven performance measures namely Accuracy, Balance, F-measure, G-Mean1, G-Mean2, G-measure and Precision. The coded fitness variants were implemented in KEEL software tool. The chapter evaluates the results on fifteen datasets collected from popular open-source software.

We proposed four ensembles of multiple fitness variants of CPSO in Chapter 7, which were aggregated using weighted votes [270]. However, it should be noted that the framework proposed in this chapter is distinct from the earlier one. In this chapter, a specific individual fitness variant is output as the "best one" for each instance of the dataset. Furthermore, instances which are correctly output by the same fitness variant i.e. instances whose selected fitness function is same are then combined to obtain their actual predictions by using the corresponding fitness variant. There is no weighted voting involved as was done in the previous chapter.

We also perform an extensive comparison of the models developed using the ASOF framework with those developed by individual CPSO fitness variants. The chapter compares the results of ASOF with nine other baseline techniques. Eight of the baseline techniques are ensemble classifiers as the ASOF technique is based on ensemble methodology. Apart from ensemble classifiers, we also compared the results with the LR technique as it has been used as an effective classifier for determining change-prone classes in literature [1, 5, 137].

Previous studies in literature have used various approaches to combine multiple classifiers for correctly predicting defect-prone classes in a software [50–53, 55]. The closest research to our work is a study performed by Di Nucci et al. [56]. They proposed a unique classifier method for predicting the best ML technique amongst a set of five ML techniques for a specific class (instance) of a dataset. The classifier bases its decision according to structural characteristics of a class. All classes for which a specific ML technique is output are aggregated and thereby, the model developed by the specific ML technique is used for these corresponding classes. However, our

work is different from the work done by Di Nucci et al. [56] as it is a first in proposing a dynamic selection of an optimum fitness function of a search-based algorithm for a specific class instance. The proposal is based on the premise that a dataset may be partitioned with respect to the fitness variant classifiers which predict specific instances well.

The chapter investigates the following RQs:

*RQ1: What is the capability of software change prediction models developed using different CPSO fitness variants?*

The chapter develops change prediction models using datasets extracted from 15 popular open-source software with each of the seven CPSO fitness variants. The results of the developed models were evaluated using G-Mean1 and Balance values.

*RQ1a. Do the results of the developed software change prediction models using the CPSO technique vary with change in fitness function? If yes, which is the best and the worst CPSO fitness variant for developing software change prediction models?*

Similar to Chapter 7, we first evaluate the results of the developed change prediction models using different CPSO fitness variants using Friedman statistical test. We ascertain the fitness variants, which give the best and the worst performing change prediction models.

*RQ1b. Are the results of software change prediction models developed using different CPSO fitness variants complementary to each other?*

This question evaluates the complementarity of the predictive capability of the CPSO technique using seven different fitness variants. In order to do so, we ascertain the percentage of classes that were correctly predicted by only a specific fitness variant and the ones, which are correctly predicted by using an arbitrary fitness variant. This complementarity is evaluated pairwise amongst all the investigated CPSO fitness variants.

*RQ2: What is the predictive capability of software change prediction models*

*developed using the ASOF framework?*

We assess the models developed using the ASOF framework on the 15 datasets investigated in the chapter. This question ascertains whether the developed change prediction models yield acceptable G-Mean1 and Balance values when within project validation (ten-fold cross validation) is performed. Furthermore, the chapter evaluates the rules of a developed model using the ASOF framework of a specific dataset (training) and validates these rules on another dataset (test) to confirm its applicability.

*RQ3: Will the use of ASOF framework, yield better change prediction models than those developed using a) individual CPSO classifiers using different fitness variants b) other fitness-based voting ensemble classifiers c) ML ensemble classifiers and the LR technique?*

This question statistically compares the performance of the models developed using the ASOF framework with each of the seven independent fitness variants of CPSO (using Friedman and post-hoc Wilcoxon test). Furthermore, the comparison of fitness-based voting ensemble classifiers (MVEC, WVEC, HIEC, WVHIEC), four ML ensemble classifiers (RF, BG, AB, LB) and the LR technique with ASOF models was performed using Wilcoxon signed rank tests with Bonferroni correction.

The organization of the chapter includes the empirical framework (Section 8.2), the proposed ASOF framework (Section 8.3), the design of the experiment (Section 8.4) and the answers to the RQ's (Section 8.5). Section 8.6 reports the primary findings of the chapter. The results of the chapter are communicated as [274].

## 8.2 Empirical Research Framework

This section includes a description of the chapter's variables, the description of the CPSO technique and the various fitness functions.

*Independent and Dependent Variables*: This chapter employs the use of the CK metrics suite [16], along with SLOC metric as predictor variables for software change. The definition and acronyms of the metrics are presented in Chapter 2 (Section 2.5.1).

*CPSO Technique*: The chapter uses CPSO as a base technique, whose description and parameter settings can be referred from section 2.6.8 (Chapter 2). In line with the discussion in Chapter 7 (Section 7.2.2), we use "default parameter settings" of KEEL tool for CPSO. It may also be noted that the aim of the chapter was to assess an optimum fitness function of CPSO corresponding to a class, based on its structural characteristics. We do not intend to examine all possible parameters and tune them to indicate the best results of CPSO algorithm corresponding to a dataset or to a specific problem domain. Therefore, we use default parameters as a practical choice in all the evaluated fitness variants of CPSO.

*Fitness Functions*: A fitness function is used as an approximator for "goodness" of a solution, while searching for an optimum solution in a large search space of candidate solutions. The chapter analyzes the use of seven different performance measures (Accuracy, Precision, G-Mean1, G-Mean2, F-measure, Balance and G-measure) as fitness functions for CPSO, which are described in Chapter 2 (Section 2.10). These fitness functions are same as the ones used in Chapter 7. However, we replace the G-Mean3 variant of Chapter 7 with the G-Mean2 variant in this chapter as the G-Mean3 variant provided the worst results using both G-Mean1 and Balance performance measures in Chapter 7 (Section 7.5.2). We intend to choose better fitness variants in order to provide effective prediction results.

It may be noted that a CPSO fitness variant uses a specific performance measure as a fitness function i.e. the CPSO will look for candidate solutions in the search space, which give optimal values for the specific fitness function. For instance, a CPSO accuracy fitness variant will look for candidate solutions which have optimal accuracy values. However, the G-Mean1 and Balance performance measures are

computed for analyzing the performance of a model developed by the CPSO accuracy fitness variant by taking into account the number of TP, TN, FP and FN obtained by the model on a particular dataset. These values (TP, TN, FP and FN) are then used for evaluating the G-Mean1 and Balance values by the formulas given in Chapter 2 (Section 2.10).

## 8.3 ASOF Framework

The ASOF framework involves developing a predictor for outputting the optimum fitness function based on the structural characteristics of a class. The structural characteristics are quantified by OO metrics of the corresponding dataset. However, firstly we need to prepare appropriate training data for the ASOF framework. In order to so, we need to first evaluate the performance of individual CPSO fitness variants models on a specific dataset. Thus, we create change prediction models on a corresponding dataset using each of the seven CPSO fitness variants. The technique used to develop these individual models was ten-fold cross validation method [113]. Furthermore, since SBA i.e. fitness variants of the CPSO algorithm are stochastic in nature, we perform 30 runs of each algorithm for developing change prediction models [37, 257]. It may be noted that the ten-fold cross validation method and the execution of 30 runs is a part of our approach as these steps are necessary to evaluate the performance of CPSO fitness variant models and thereafter prepare the training data of ASOF for a specific dataset.

Next, for each corresponding class of the training data of a specific dataset, we will choose the best fitness function for the class. The "best" is indicated in terms of performance, which is evaluated in terms of G-Mean1 and Balance values. The rules for selecting the fitness variant are diagrammatically represented in Figure 8.1. The first scenario (Figure 8.1: Case a) represents a case when there is no fitness

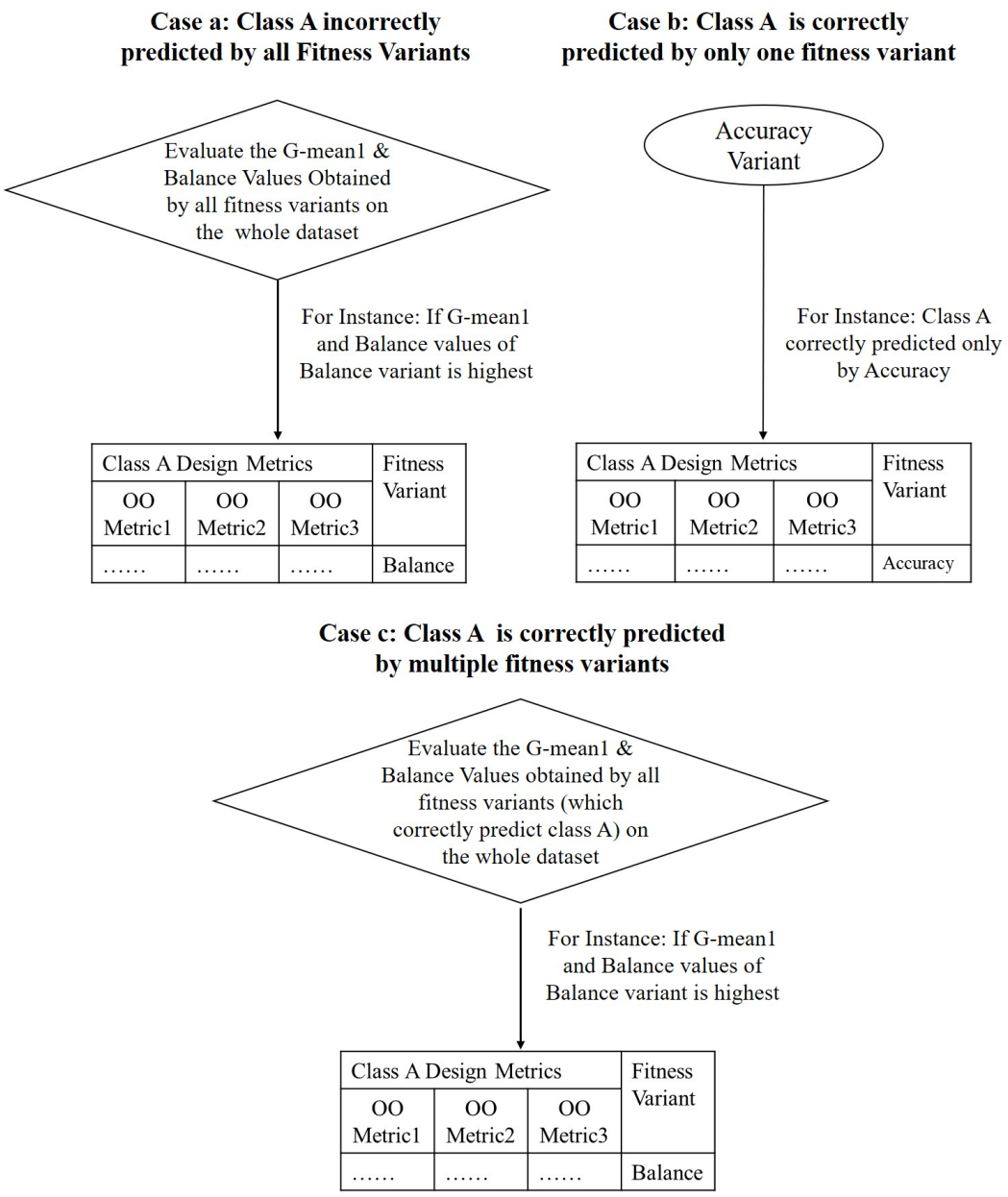


Figure 8.1: Rules for creating training data of ASOF Framework

variant which correctly identifies the true nature i.e. change-prone or not change-prone nature of a class. In such a scenario, we select the fitness variant obtaining highest values of G-Mean1 and Balance on the entire dataset. The rationale behind

choosing such a variant is that if we cannot choose a variant which correctly classifies a specific class, we need to choose a fitness variant which performs minimum number of such erroneous predictions. However, if there is only one fitness variant which correctly determines the change-prone nature of a class (Figure 8.1: Case b), we select that fitness variant for the corresponding instance. If there are multiple fitness variants which correctly identify the change-prone nature of a class, we need to select a fitness variant which obtains the best G-Mean1 and Balance values amongst these fitness variants. All fitness variants which incorrectly predict the nature of a class do not participate in this selection. This is formally stated in terms of Figure 8.1: Case c. The training data for ASOF will consist of OO metrics for the corresponding class and the selected fitness variant for that class.

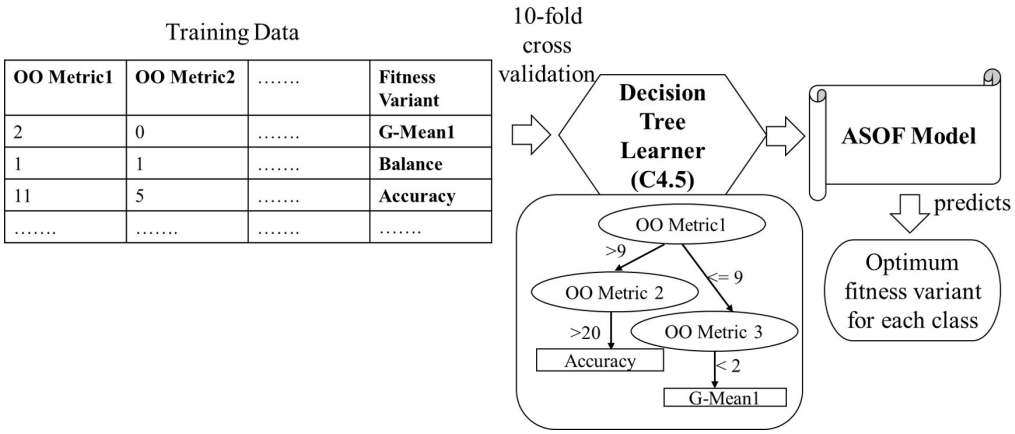


Figure 8.2: Diagrammatic Representation of ASOF Framework

The next step after preparation of training data is the development of the ASOF prediction model. We use the DT (C4.5) algorithm and ten-fold cross validation technique for doing so. The C4.5 technique is chosen as it is capable of formulating rules according to the values of OO design metrics. It can successfully analyze the values of different design metrics and formulate generalizations in the form of rules, i.e., which fitness variant should be used for a corresponding class with specific design

metric values. Figure 8.2 depicts the process of the ASOF framework. Thereafter, when we need to ascertain the change-prone nature of a new data point, we ascertain it with the model developed using the optimum fitness variant, which was outputted by the ASOF prediction model.

***D*** → Set of 15 Datasets.; $DP_k$ → $k^{th}$ data point of training set of $D_i$, which consists of OO metric values and change prone nature a corresponding class of the dataset; ***F*** → Set of 7 CPSO fitness variants; ***Output*** $(F_j)$ →Output prediction of $DP_k$ by $F_j$ ; ***G-Mean1*** $(F_j)$ → G-Mean1 value obtained by predictions of $F_j$ on dataset $D_i$; ***Balance*** $(F_j)$ → Balance value obtained by predictions of $F_j$ on dataset $D_i$ ;***Fitness -Op***$(DP_k)$ → *Optimum Fitness Function of data point* $DP_k$ ; ***Status*** $(F_j, DP_k)$ → *"Correct" or "Incorrect" depending on whether the prediction of* $F_j$ *is correct or not for* $DP_k$ ; ***Name*** $(F_j)$ → *Name of the* $j^{th}$ *fitness variant;* ***ASOF-Training***$(DP_k)$ → Prepare $k^{th}$ data point of $D_i$ for training of ***ASOF***, which consists of OO metric values and ***Fitness -Op***$(DP_k)$ for a corresponding data point of the dataset.

*For each* $D_i \in D$
  *For each* $F_j \in F$
    *Compute 10-fold cross-validation results of* $F_j$ *on* $D_i$
    *Compute* ***G-Mean1***$(F_j)$ *on* $D_i$
    *Compute* ***Balance***$(F_j)$ *on* $D_i$

*For each* $D_i \in D$
  *For each* $DP_k \in$ ***Training Set*** $(D_i)$
    *count-correct = count-incorrect =0*
    *For each* $F_j \in F$
      *If* ***Output***$(F_j)$ *matches the* ***Change prone nature of*** $DP_k$
        *count-correct ++ &&* ***Status*** $(F_j, DP_k)$ *= "correct"*
      *Else*
        *count-incorrect++ &&* ***Status*** $(F_j, DP_k)$ *= "incorrect"*
    *If (count-correct ==0)*
      ***Fitness-Op*** $(DP_k)$ **=** ***Name***$(F_j)$ *where* $F_j$ *is the fitness variant with the highest* ***G-Mean1***$(F_j)$ *and* ***Balance***$(F_j)$ *on* $D_i$
    *Else If (count-correct ==1)*
      ***Fitness-Op*** $(DP_k)$ **=** ***Name***$(F_j)$ *where* $F_j$ *is the fitness variant with* ***Status*** $(F_j, DP_k)$ *= "correct"*
    *Else If (count-correct > 1)*
      ***Fitness-Op*** $(DP_k)$ **=** ***Name***$(F_j)$ *where* $F_j$ *is the fitness variant with the highest* ***G-Mean1***$(F_j)$ *and* ***Balance***$(F_j)$ *on* $D_i$ *&&* ***Status*** $(F_j, DP_k)$ **=** *"correct".*

*For each* $D_i \in D$
  *For each* $DP_k \in$ ***Training Set*** $(D_i)$
    *Create* ***ASOF-Training*** $(DP_k)$
*For each* $D_i \in D$
  *Create an ASOF Prediction model using Decision Tree Algorithm on* ***ASOF-Training -Set*** $(D_i)$
  *For each data point with unknown label (Change prone / Not Change prone)* $\in D_i$
    *Input the OO metrics of the data point into ASOF Prediction model*
    *Evaluate the change Proneness of the new data point by using the model of the fitness variant* $F_k$ *output by the* ***ASOF*** *prediction model.*

Figure 8.3: Pseudocode of ASOF Framework

Figure 8.3 shows the pseudocode of the ASOF framework. According to the figure, for each dataset, ten-fold cross validation models are developed using each of the investigated fitness variants. Thereafter, the values of G-Mean1 and Balance performance measures are evaluated for each of these models. The next step is to allocate the optimum fitness variant (Fitness-Op) to each training instance (data point) according to rules depicted in Figure 8.1. In order to implement the rules, the pseudocode evaluates the count-correct, count-incorrect and status of each instance of a dataset i.e. we need to ascertain the number of fitness variants which correctly identify the change-prone nature of an instance (count-correct) and the count of fitness variants, which misclassify the change-prone nature of a class (count-incorrect). Furthermore, we also need to determine whether a specific fitness variant correctly determines an instance or not. This is determined by ascertaining the value of "status" of a fitness variant with respect to a data point. A status value of "correct" indicates that the change-prone nature of the data point was correctly predicted by the corresponding fitness variant and a status value of "incorrect" indicates vice versa.

The Fitness-Op of a data point is allocated according to the rules specified in Figure 8.1. If the value of count-correct is zero, it represents Fig 8.1: case a. However, if the value of count-correct is "1" Figure 8.1:case b is invoked. For all non-zero values of count-correct greater than 1, Fig 8.1: case c is invoked. Therefore, by correctly determining the Fitness-Op for a data point, we construct the training set for ASOF (ASOF-Training Set). The ASOF-Training Set includes the OO metric values and the corresponding Fitness-Op for each data point of a corresponding dataset. Thereafter, the DT algorithm (C4.5) is used for creating the ASOF prediction model using ten-fold cross validation.

In order to predict the change-prone nature of a new data point, we first use the developed ASOF model of the corresponding datasets for predicting the optimum fitness variant of the new data point according to its structural characteristics (OO

metrics). Finally, the model of the predicted fitness variant is used for ascertaining the change-prone nature of the data point.

## 8.4 Experimental Framework

This section discusses the datasets used, feature selection, validation framework, performance measures and the statistical tests used in this chapter. We also discuss the baseline techniques, with which we compare the results of the models developed using the ASOF framework.

### 8.4.1 Data Collection & Validation Framework

This chapter examines 15 popularly used software datasets, which belong to different domains. Two versions of each dataset were analyzed and common classes between both the versions were extracted. The OO metrics were computed for the initial version of each of the dataset using Understand tool (`http://scitools.com`). The details of data collection can be referred from section 2.7 (Chapter 2).

Table 8.1: Dataset Details

| Dataset Name | Versions | Metrics Selected by CFS |
|---|---|---|
| Art Of Illusion (AOI) | 2.7-2.9.2 | CBO, SLOC, LCOM, WMC |
| Click | 2.2.0-2.3.0 | CBO, WMC, LCOM, SLOC |
| DrJava | r4668-r5686 | CBO, NOC, RFC, LOC, LCOM, WMC |
| Giraph | 1.0-1.1 | CBO, WMC, RFC, SLOC |
| Gora | 0.4-0.5 | CBO, WMC, SLOC, DIT, LCOM |
| Hama | 0.5-0.6 | CBO, WMC, SLOC |
| HyperSQL DB | 2.3.3-2.3.4 | CBO, NOC, SLOC, RFC, WMC |
| Jabref | 3.1-3.2 | CBO, SLOC, LCOM, DIT |
| Jmeter | 1.4.1-2.9 | CBO, NOC, SLOC |
| Jedit | 5.0.0-5.1.0 | CBO, WMC, SLOC |
| LogicalDoc | 7.5-7.6 | CBO, WMC, SLOC |
| Maven | 3.3.3-3.3.9 | CBO, SLOC, DIT |
| Phoenix | 4.2-4.3 | CBO, RFC, SLOC, LCOM, DIT |
| Subsonic | 5.2-5.3 | CBO, SLOC |
| Zookeeper | 3.4.8-3.5.1 | CBO, SLOC, LCOM, RFC, NOC |

The names of the datasets and the specific versions that were investigated are mentioned in Table 8.1. The characteristics of each dataset can be referred from Appendix A. We performed feature selection using CFS method, whose results are mentioned in Table 8.1.

The change prediction models using individual fitness variants, voting ensemble classifiers and ML ensembles were developed using ten-fold cross validation technique, for reducing validation bias [37, 257]. However, ASOF was validated both using ten-fold cross validation and cross-project validation.

### 8.4.2 Performance Measures & Statistical Evaluation

The chapter uses G-Mean1 and Balance as performance measures for evaluating the results of the developed change prediction models. The reasons for choosing these performance measures are similar to the ones discussed in Chapter 7 (Section 7.4.3).

The results of the study are statistically analyzed by using Friedman test [125] and Wilcoxon signed rank test. Wilcoxon test was conducted to ascertain the comparative performance of the models developed using the ASOF framework with other baseline models for each of the 15 datasets. The effect size was reported as Vargha and Delaney's statistic [275] for all the significant cases. The statistic ($A_{XY}$) for correlated samples was computed according to the following formula [275, 276]:

$$A_{XY} = \frac{Number\ of\ cases\ where\ X > Y + 0.5 * (Number\ of\ cases\ where\ X = Y)}{Total\ Number\ of\ cases} \tag{8.1}$$

Here, the total number of cases represent the 15 datasets on which the comparisons are performed. While comparing model X and Y, we ascertain the number of datasets on which model X performs better than model Y and the number of datasets on which they perform equivalently ($X = Y$). As indicated by Vargha and Delaney

[275], an effect size value $(A_{XY})$ of 0.56 is considered small, 0.64 is considered medium and 0.71 is assumed to be large.

### 8.4.3 Description of Baseline Techniques

The models developed using the ASOF technique were compared with the following baseline techniques:

*Fitness-based Voting Ensemble Classifiers*: The voting ensemble classifiers use the approach similar to that of "validation and voting" [185]. We compare the results with MVEC, WVEC, HIEC and WVHIEC techniques. The details of these fitness-based voting ensemble classifiers can be referred from Chapter 7.

*ML Ensemble Classifiers and LR Technique*: Similar to chapter 7, RF, BG, AB and LB are used for baseline comparison as they are ML ensemble classifiers. The change prediction models are also compared with the ones developed using the LR technique. The details of ML ensemble classifiers and the LR technique can be referred from Chapter 2. They were simulated in WEKA tool with default parameters settings [88]. The reasons for choosing default parameters are similar to the ones discussed in Section 7.4.4 (Chapter 7).

## 8.5 Results and Analysis

This section presents in detail the results of the chapter and also answers the RQs of the chapter.

### 8.5.1 Results specific to RQ1

As discussed in Section 8.4.2, the change prediction models were assessed according to their G-Mean1 (GM1) and Balance (Bal.) values, which are presented in Table 8.2.

The values shown in the table are median values of 30 runs of the CPSO technique using ten-fold cross validation. For each dataset, change prediction models were developed using seven CPSO fitness variants. The fitness variant attaining the best results on a corresponding dataset (in terms of both G-Mean1 and Balance) is shown in bold. The table also states the mean G-Mean1 and Balance values obtained by a specific fitness variant over all the datasets. The mean G-Mean1 and Balance values obtained by all the fitness variants were in the range of 0.46-0.58 and 47.21-57.51 respectively.

Table 8.2: Validation Results of CPSO Fitness Variants

| Dataset | Accuracy V. | | Balance V. | | F-measure V. | | G-Mean1 V. | | G-mean2 V. | | G-mes. V. | | Precision V. | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *GM1* | *Bal.* | *GM1* | *Bal.* | *GM1* | *Bal.* | *GM1* | *Bal.* | *GM1* | *Bal.* | *GM1* | *Bal.* | *GM1* | *Bal.* |
| **AOI** | 0.63 | 62.07 | **0.65** | **65.44** | 0.64 | 63.07 | 0.60 | 59.92 | 0.62 | 61.48 | 0.63 | 62.08 | 0.51 | 49.52 |
| **Click** | 0.55 | 53.40 | 0.55 | 53.25 | 0.36 | 38.84 | **0.57** | **54.89** | 0.34 | 37.78 | 0.33 | 37.35 | 0.55 | 53.09 |
| **DrJava** | **0.66** | **65.91** | 0.66 | 65.67 | 0.62 | 61.62 | 0.66 | 65.82 | 0.61 | 60.87 | 0.60 | 60.20 | 0.63 | 62.83 |
| **Giraph** | 0.56 | 54.13 | **0.56** | **54.16** | 0.48 | 46.65 | 0.55 | 53.42 | 0.31 | 36.15 | 0.29 | 35.45 | 0.50 | 48.95 |
| **Gora** | 0.37 | 39.72 | **0.60** | **57.36** | 0.40 | 41.42 | 0.54 | 52.23 | 0.04 | 41.42 | 0.40 | 41.42 | 0.27 | 34.70 |
| **Hama** | **0.54** | **53.87** | 0.54 | 53.72 | 0.47 | 46.61 | 0.53 | 53.15 | 0.41 | 42.39 | 0.46 | 45.60 | 0.41 | 42.17 |
| **HyperSQL** | 0.68 | 67.23 | 0.68 | 67.85 | 0.67 | 66.67 | **0.69** | **68.24** | 0.65 | 63.14 | 0.68 | 67.19 | 0.50 | 49.02 |
| **JabRef** | 0.58 | 55.70 | *0.65* | *64.13* | 0.64 | 62.06 | 0.65 | 64.06 | 0.60 | 57.08 | 0.60 | 57.09 | 0.50 | 48.96 |
| **Jmeter** | 0.53 | 50.78 | 0.48 | 47.71 | 0.53 | 50.78 | 0.54 | 52.70 | 0.19 | 31.88 | 0.19 | 31.88 | **0.59** | *57.88* |
| **Jedit** | **0.58** | **55.35** | 0.51 | 50.53 | 0.56 | 53.74 | 0.37 | 39.60 | 0.56 | 53.71 | 0.56 | 53.74 | 0.47 | 46.72 |
| **Logicaldoc** | **0.57** | **55.40** | 0.52 | 50.86 | 0.53 | 51.24 | 0.54 | 53.02 | 0.45 | 44.27 | 0.48 | 46.50 | 0.24 | 33.60 |
| **Maven** | 0.69 | 68.42 | **0.70** | **70.15** | 0.68 | 67.51 | 0.65 | 63.51 | 0.68 | 67.83 | 0.68 | 67.13 | 0.56 | 54.22 |
| **Phoenix** | 0.59 | 58.36 | **0.62** | **61.22** | 0.47 | 46.70 | 0.58 | 56.87 | 0.26 | 34.10 | 0.35 | 38.4 | 0.40 | 41.74 |
| **Subsonic** | 0.64 | 60.90 | 0.48 | 47.47 | 0.62 | 59.03 | 0.55 | 53.08 | 0.62 | 59.24 | **0.65** | **61.9** | 0.55 | 51.95 |
| **Zookeeper** | **0.56** | **56.01** | 0.54 | 53.15 | 0.29 | 35.51 | 0.55 | 54.28 | 0.28 | 34.99 | 0.30 | 35.74 | 0.22 | 32.77 |
| **Mean** | 0.58 | 57.15 | **0.58** | **57.51** | 0.53 | 52.76 | 0.57 | 56.32 | 0.47 | 48.42 | 0.48 | 49.44 | 0.46 | 47.21 |
| G-mes. indicates G-Measure, V. indicates variant, GM1 indicates G-Mean1 & Bal. indicates Balance | | | | | | | | | | | | | | |
| *The bold values depict the fitness variant which is best in terms of both G-Mean1 and Balance measures. | | | | | | | | | | | | | | |

***Variation in Results of CPSO Fitness Variants (RQ1a)***

According to the results depicted in Table 8.2, we see that each fitness variant gave different results on corresponding datasets. For instance, the G-Mean1 results on Gora dataset varied from 0.27 to 0.60. Likewise, the Balance results on Maven dataset varied from 54.12 to 70.15. It was also noted that though, a fitness variant might obtain the most effective results on a specific dataset, it might obtain the worst

results on some other dataset. For instance, the Precision variant obtained the best results on Jmeter dataset, however, it obtained the worst results on AOI, Gora, Hama, HyperSQL, Logicaldoc, Maven and Zookeeper datasets.

Thus, similar to the observations in Chapter 7 (Section 7.5.1), we found that a variation in fitness function of a technique leads to diversification of results on a corresponding dataset, when evaluated using G-Mean1 and Balance values. This phenomenon is observed because the fitness function dictates the selection of optimum candidates in the solution space.

Table 8.3: Friedman Test Results

| Ranks | Results using G-Mean1 Values | Results using Balance Values |
|---|---|---|
| Rank 1 | Balance Variant (2.67) | Accuracy Variant (2.50) |
| Rank 2 | Accuracy Variant (2.77) | Balance Variant (2.67) |
| Rank 3 | G-Mean1 Variant (3.00) | G-Mean1 Variant (3.13) |
| Rank 4 | F-measure Variant (3.93) | F-measure Variant (3.93) |
| Rank 5 | G-measure Variant (4.60) | G-measure Variant (4.73) |
| Rank 6 | G-Mean2 Variant (5.30) | G-Mean2 Variant (5.30) |
| Rank 7 | Precision Variant (5.73) | Precision Variant (5.73) |

Further, in order to assess the best and the worst fitness variants, we used Friedman test on G-Mean1 and Balance values at 0.05 significance level. The results of the tests were found to be significant with a p-value of less than 0.001, a chi-square value of 30.33 and 32.46 using G-Mean1 and Balance values respectively and with six degrees of freedom. The ranks obtained by all the fitness variants are shown in Table 8.3. The mean ranks allocated by the Friedman test is shown in parenthesis. The fitness variants achieving lower ranks are better performing fitness variants than the others. According to the results shown in the table, the best Friedman rank according to G-Mean1 values was obtained by the Balance variant followed by the Accuracy variant. However, the best Friedman rank using Balance values was obtained by the Accuracy variant followed by the Balance variant. The ranks obtained by other fitness variants were consistent when evaluated using either G-Mean1 and

Balance values. The worst fitness variant was Precision variant as it obtained the lowest ranks both using G-Mean1 and Balance values.

It may be noted that we analyzed the performance of individual fitness variants using G-Mean1 and Balance performance measures. Thus, it may seem a priori that the best results should be obtained by fitness variants that optimize these performance measures i.e. the G-Mean1 variant and the Balance variant. One might think that including just these two fitness variants would be sufficient to build the ASOF model as the performance of ASOF is only adjudged on the basis of G-Mean1 and Balance. But this is not the case. We note that another fitness variant i.e. the Accuracy variant achieved good results. The Accuracy variant achieved better Friedman rank than Balance variant, when the performance was evaluated using Balance measure. Similarly, the Accuracy variant and the Balance variant obtained better Friedman ranks than the G-Mean1 variant when the results were evaluated using G-Mean1 variant. These results indicate that we could not have ignored fitness variants which optimize other performance measures i.e. Accuracy variant, F-measure variant, G-measure variant, G-Mean2 variant and Precision variant as it is possible that they could obtain comparable or even better G-Mean1 and Balance metrics. The individual G-Mean1 fitness variant and the Balance fitness variant might converge to mid-optimum points in a pre-mature fashion while optimizing the corresponding performance measures. Thus, only choosing both G-Mean1 and Balance variants as constituents of the ASOF and for validating the ASOF would lead to biased judgments. Although, there are studies [38, 40, 196] in literature which have used the same performance measure, both for validating the results as well as a fitness function. However, we use several other constituent fitness variants to get reliable results. Furthermore, as we want to obtain unbiased results we evaluate the performance using two performance measures (G-Mean1 and Balance), rather than just any one measure.

***Answer to RQ1a***

The results of the developed software change prediction models using CPSO are influenced by the selection of fitness function, as it is crucial in choosing optimum solution candidates. The Precision variant is the worst fitness variant and the Balance and Accuracy variants are the top two variants as they obtained good values for both G-Mean1 and Balance on all the investigated datasets.

***Complementary Results of CPSO fitness variants (RQ1b)***

We observe the pairwise complementarity amongst different fitness variants [56]. For each set of fitness variants $(X_1 \& X_2)$, we compute the percentage of correctly predicted change-prone classes (CCP) by both the fitness variants $(X_1 \cap X_2)$ and by only one of the fitness variant $(X_1 - X_2$ and $X_2 - X_1)$ by the formulas discussed in section 7.5.1 (RQ1b). Table 8.4 states the corresponding complementarity percentages. Due to space limitations, the percentages are obtained by aggregating the statistics of all the datasets. The notations in table 8.4 are as follows: Acc. indicates Accuracy, Pr. indicates Precision; GM1 indicates G-Mean1; GM2 indicates G-Mean2, Fmes. indicates F-measure; Gmes. indicates G-measure and Bal. indicates Balance.

Table 8.4: Complementarity of CPSO Fitness Variants

| | **Acc. V.** | | | **Bal. V.** | | | **Fmes. V.** | | | **GM1 V.** | | | **GM2 V.** | | | **Gmes. V.** | | | **Pr. V.** | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *I* | *D1* | *D2* | *I* | *D1* | *D2* | *I* | *D1* | *D2* | *I* | *D1* | *D2* | *I* | *D1* | *D2* | *I* | *D1* | *D2* | *I* | *D1* | *D2* |
| **Acc. V.** | X | X | X | 69 | 20 | 11 | 80 | 10 | 10 | 75 | 12 | 13 | 75 | 12 | 13 | 77 | 11 | 12 | 70 | 16 | 15 |
| **Bal. V.** | | | | X | X | X | 72 | 14 | 14 | 79 | 10 | 11 | 70 | 15 | 15 | 70 | 15 | 15 | 68 | 16 | 16 |
| **Fmes.V.** | | | | | | | X | X | X | 74 | 14 | 12 | 89 | 6 | 5 | 91 | 5 | 4 | 74 | 14 | 13 |
| **GM1 V.** | | | | | | | | | | X | X | X | 73 | 13 | 14 | 72 | 15 | 13 | 72 | 15 | 13 |
| **GM2 V.** | | | | | | | | | | | | | X | X | X | 95 | 3 | 2 | 76 | 13 | 11 |
| **Gmes. V.** | | | | | | | | | | | | | | | | X | X | X | 76 | 12 | 12 |
| **Pr. V.** | | | | | | | | | | | | | | | | | | | X | X | X |
| I: $X_1 \cap X_2$; D1: $X_1 - X_2$ ; D2: $X_2 - X_1$; X: No comparison possible; V.: Variant | | | | | | | | | | | | | | | | | | | | | |

The results in Table 8.4 depict that the overlap of correctly classified change-prone classes amongst any two variants ranges from 68% to 95% $(X_1 \cap X_2)$, in a majority of the cases. Thus, any fitness variant is suitable for correct change prediction

of these overlapping (those which are correctly predicted by an arbitrary fitness variant) classes. However, in order to correctly predict the remaining non-overlapping change-prone classes (those which are correctly predicted by a specific fitness variant), it is advisable to use the ability of specific fitness variants. These remaining classes approximately account to 50-70% of actual change-prone classes. As noted in Chapter 7, correct prediction of these non-overlapping change-prone classes can lead to considerable improvement in the ability of change prediction model. It should be noted that if even one of the fitness variant correctly classifies a class, it may be correctly predicted by the model developed using the ASOF framework. Thus, the use of multiple fitness variants would result in better predictive capability of the developed change prediction models.

We aim to optimize not just the performance measures but also want to find the correct predictions for data points, which are generally misclassified by most of the fitness variants. As discussed in Chapter 7, we call such instances as "hard instances". It is possible that an instance with certain structural characteristics might be wrongly predicted by a majority of fitness variants but may be correctly predicted by only one or two fitness variants. We want the ASOF model to learn correct fitness variants for such instances too. Thus, in line with our discussion of RQ1a, we reiterate that it is important to incorporate several fitness variants rather than just one or two.

***Answer to RQ1b***

The results of software change prediction models developed using multiple fitness variants are complementary to each other. The models developed using the ASOF framework can be improved by using complementary constituent fitness variants as the ASOF framework might correctly predict a non-overlapping class even if it was correctly predicted by only one of its constituent classifier.

### 8.5.2 Results specific to RQ2

The G-Mean1 and Balance values of the software change prediction models developed using the ASOF framework are illustrated in Figure 8.4. These results are obtained using ten-fold cross validation. It may be noted from the figure that the G-Mean1 values on all the datasets ranged from 0.55-0.69. The Balance values obtained were in the range 53%-68%. These ranges are acceptable and confirm the predictive capability of the models developed using the ASOF framework in successfully determining the change-prone classes.

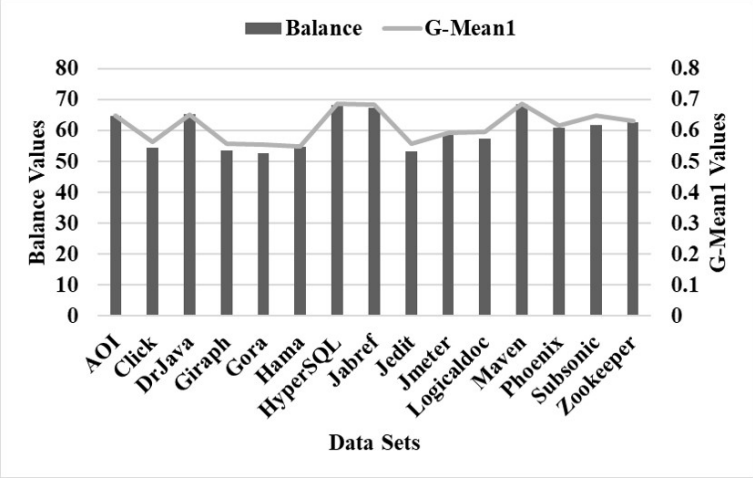


Figure 8.4: Validation Results of Classifiers with ASOF Framework

It may be noted that in 12 out of 15 datasets, the Balance values of the models developed using the ASOF framework were better than those achieved by the models developed using the Accuracy variant (best fitness variant according to Balance values as discussed in section 8.5.1). Also, there was an improvement of 1%-32% in the Balance values of these twelve datasets, when change prediction models were developed using the ASOF framework rather than the accuracy fitness variant. Similarly, the G-Mean1 values of the models developed using the ASOF framework were better than those obtained by Balance fitness variant (best fitness variant according to

G-mean1 values as discussed in section 8.5.1) in 10 out of 15 datasets. The improvement in G-Mean1 values was found in the range of 0.2%-34% in these ten datasets.

The reason for an improvement in the results of the ASOF models was the dynamic selection of fitness variant for each instance of the dataset. We demonstrate this with an example of Zookeeper dataset. The Zookeeper dataset has 591 instances. According to Table 8.2, the best fitness variant on the Zookeeper dataset was the accuracy fitness variant. If we use the predictions of only the accuracy fitness variant for the entire dataset, we obtain a TP value of 130, TN value of 214, an FN value of 139 and an FP value of 108. This would result in a G-Mean1 value of 0.56 and a Balance value of 56.01 (Table 8.2). On the contrary, when we use the ASOF model, it predicts the accuracy fitness variant to be used for 511 instances, G-Mean1 fitness variant to be used for 48 instances, Balance fitness variant to be used for 6 instances, G-measure fitness variant to be used for 25 of its instances and the precision fitness variant to be used for one instance. By using the predictions of the fitness variants suggested by the ASOF model, we get a TP value of 192, a TN value of 240, an FN value of 167 and an FP value of 82. Therefore, the G-Mean1 value is computed as 0.63 and the Balance value is computed as 62.50% by using the predictions of the fitness variants as suggested by the ASOF model. This depicts an improvement of 12.5% and 11.6% respectively in G-Mean1 and Balance values. Therefore, by using appropriate fitness functions for different instances of the dataset as compared to just a single fitness function, the performance of the change prediction model improves.

Apart from evaluating the ten-fold cross validation results, we also assessed whether the prediction of the ASOF models is effective when applied to another software dataset (cross-project validation). This was done by extracting the rules generated by C4.5 when an ASOF model was built on DrJava software dataset. A partial set of these extracted rules is represented in the DT shown in Figure 8.5. Some example rules for allocating optimum Fitness variant (Fitness-Op) to a data point $DP_k$, as

represented in Figure 8.5 are as follows:

- IF LCOM $>$ 26 AND RFC $>$ 19 AND LCOM $>$ 43 for $DP_k$ THEN Fitness-Op ($DP_k$) = Accuracy.
- IF LCOM $>$ 26 AND RFC $\leq$ 19 AND CBO $>$ 18 for $DP_k$ THEN Fitness-Op ($DP_k$) = Balance.

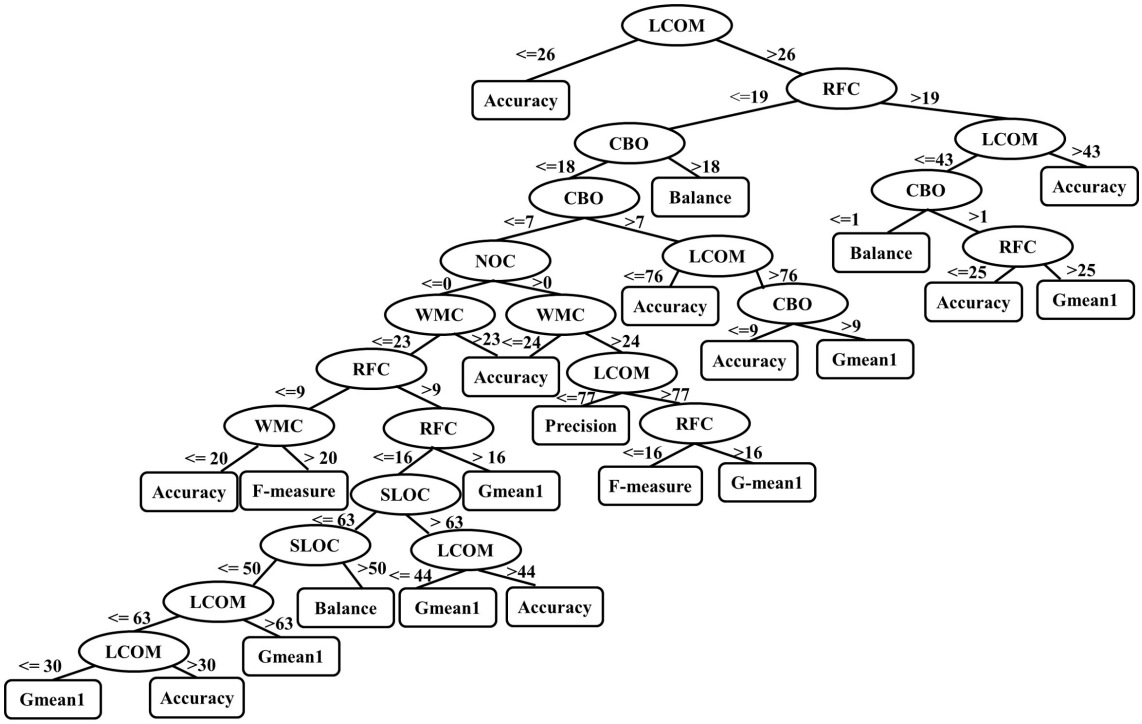


Figure 8.5: Rules of ASOF Framework on DrJava Dataset

We employed these generated rules on three other test datasets (Jedit, Logicaldoc and Maven). As a result, according to the rules we partitioned the data points of Jedit, LogicalDoc and Maven datasets into various subsets which should be evaluated by different specific fitness variants. The change-prone nature of the data points in a subset was ascertained using the models developed using the predicted fitness variants. The validation results obtained on the Jedit dataset exhibited a G-Mean1

value of 0.39 and a Balance value of 40.78. It should be noted that these values were better than models developed using only the G-Mean1 variant using ten-fold cross validation on Jedit dataset. However, they were lower than the values obtained from the models developed using all the other fitness variants (Accuracy, Balance, G-Mean2, Precision, F-measure and G-measure variants) on Jedit dataset (Refer Table 8.2). Similarly, the validation results (G-Mean1: 0.59, Balance: 57.12) on Maven dataset were better than those obtained by only the Precision variant using ten-fold cross validation. However, they were poorer than those obtained by all other investigated variants. These results advocate that the best approach would be to use the training data of the same software dataset for developing change prediction models using the ASOF framework. However, when Logicaldoc dataset was evaluated using rules extracted from DrJava dataset, we found a G-Mean1 value of 0.54 and a Balance value of 52.64. It may be noted that these values were worse than only the Accuracy variant and the G-Mean1 variant using ten-fold cross validation on Logicaldoc dataset. These values were better than all investigated fitness variants models viz. Balance variant, G-Mean2 variant, F-measure variant, G-measure variant and Precision variant. We therefore observe that the results using cross-project validation are acceptable in certain cases.

A possible reason for poor results of ASOF model using cross-project validation is that the structural characteristics of instances vary from one dataset to another. Also, as discussed in section 8.5.1, the performance of different fitness variants varies from dataset to dataset. A fitness variant which works best on a specific dataset may be quite poor for another dataset. For a particular dataset, the ASOF model is trained with specific structural characteristics and training data, which may change and may not be applicable to another dataset. For instance, for the Jedit dataset, one obtains a rule "IF CBO $\leq$ 12 for $DP_k$ THEN Fitness-Op ($DP_k$) = Accuracy ". However, for Logicaldoc dataset, one obtains a rule "IF CBO $\leq$ 10 for $DP_k$ THEN Fitness-Op

($DP_k$) = Accuracy". The rule obtained for AOI dataset is "IF CBO $\leq$ 7 for $DP_k$ THEN Fitness-Op ($DP_k$) =Balance ". Therefore, the threshold value of the CBO metric for allocating the fitness variant Accuracy is different in Jedit and Logicaldoc datasets. Moreover, the rule for AOI dataset designates a Balance fitness variant rather than an Accuracy fitness variant for all the data points with CBO values less than 7. Such contradictory rules may lead to incorrect predictions when the ASOF model trained on one dataset, may be applied to another dataset. Hence, resulting in poor performance of the ASOF model.

Therefore, we conclude that the approach using ASOF framework works well for within-project validation, but might not give the best results for a new project where sufficient training data is not previously available.

***Answer to RQ2***

The results of the software change prediction models developed using the ASOF framework are acceptable for effective prediction of change-prone classes (G-Mean1: 0.55-0.69, Balance: 53-68%). The ASOF models exhibit an improvement in G-Mean1 and Balance results when evaluated using ten-fold cross validation on the same project. However, the ASOF models are not the best choice for cross-project validation.

### 8.5.3 Results specific to RQ3

In order to answer the RQ, we have divided the comparison results of models developed using ASOF into three sections as below.

***Comparison of ASOF framework with individual CPSO fitness-based classifiers***

We first performed Friedman tests for comparing the predictive capability of change prediction models developed using the ASOF framework with individual

CPSO fitness variants using G-Mean1 and Balance values over all the datasets. The test was conducted at $\alpha = 0.05$. The Friedman test results were found significant using both G-Mean1 and Balance values, with a p-value of less than 0.001 in each case. A Friedman statistic (chi-square) of 46.06 and 47.24 was obtained using G-Mean1 and Balance values respectively with seven degrees of freedom. The models developed using the ASOF framework achieved the best rank according to both G-Mean1 and Balance values. This indicates the superiority of the models developed using the ASOF framework. Furthermore, we performed Wilcoxon post-hoc test using Bonferroni correction at $\alpha = 0.05$ on G-Mean1 and Balance values. The p-value should be less than 0.007 for the Wilcoxon test to be significant after Bonferroni correction. The results of the Wilcoxon test are presented in Table 8.5, with p-values in parenthesis. According to the table, in all the cases, the models developed using the ASOF framework outperformed the fitness variant models. Moreover, in ten out of fourteen cases, the superiority was significant. The primary reason for their improved performance is the dynamic allocation of fitness function according to structural properties of each class.

Table 8.5: Wilcoxon test results for ASOF vs CPSO Fitness Variants

| ASOF vs | Using G-Mean1 values | Using Balance Values |
|---|---|---|
| Accuracy Variant | ↑ (0.012) | ↑ (0.009) |
| Balance Variant | ↑ (0.088) | ↑ (0.125) |
| F-measure Variant | ↑* (0.001) [0.93] | ↑* (0.001) [0.93] |
| G-Mean1 Variant | ↑* (0.003) [0.87] | ↑* (0.005) [0.80] |
| G-Mean2 Variant | ↑* (0.001) [0.93] | ↑* (0.001) [0.93] |
| G-measure Variant | ↑* (0.001) [0.90] | ↑* (0.001) [0.90] |
| Precision Variant | ↑* (0.001) [1.00] | ↑* (0.001) [1.00] |
| ↑: Non-significant superiority of the ASOF model; ↑*: Significant superiority of the ASOF model) | | |

We also computed the Vargha and Delaney's statistic to estimate the effect size of the Wilcoxon test as discussed in Section 8.4.2. The computed Vargha and Delaney's statistic is mentioned in square brackets for all the significant cases. As the statistic ranged from 0.87-1.00 for all significant results in Table 8.5, we assume a large effect

size according to guidelines mentioned in Section 8.4.2.

One might notice that the comparison of models developed using the ASOF framework are significantly different from the models developed using the G-Mean1 CPSO variant, especially using the G-Mean1 performance measure (Table 8.5). As already discussed in Section 8.2, a G-Mean1 CPSO fitness variant will select a candidate solution, which would provide the "optimal" G-Mean1 value on the entire dataset, i.e. a classification rule which when used gives the values of TP, TN, FP and FN, which provide optimal G-Mean1 performance measure. However, as discussed in section 8.5.1, an individual G-Mean1 fitness variant might converge with mid-optimal values, which may not be the "best" as a termination condition might reach before getting a solution with the "best" value or the constriction coefficient might limit the "exploration", which may result in a sub-optimal but not the "best" solution. We may achieve a better "optimal" solution by the use of ASOF framework. Since, the ASOF framework applies a combination of different CPSO fitness variants for prediction, the number of TP, TN, FP and FN would be different from the "G-Mean1 fitness variant". For instance, there may still be some instances which are classified as FP and FN by the "G-Mean1 fitness variant", but may be correctly predicted by some other fitness variant. Therefore, using other fitness variants might increase the G-Mean1 value of the model developed using the ASOF framework leading to significantly better results.

***Comparison of ASOF framework with Fitness-based voting ensemble classifiers***

We first ascertain the performance of the four fitness-based voting ensemble classifiers viz. MVEC, WVEC, HIEC and WVHIEC on the fifteen datasets investigated in the study. The G-Mean1 and Balance values of the change prediction models developed using the mentioned fitness-based voting ensemble classifiers are depicted in Table 8.6. We also restate the results of change prediction models developed us-

ing the ASOF framework for easy comparison. The values of the best model on a corresponding dataset, with respect to G-Mean1 and Balance values individually are depicted in bold. According to the table, the models developed using the ASOF framework depicted the best Balance values in nine out of fifteen datasets. Furthermore, the ASOF models achieved the best G-Mean1 values in ten datasets. The mean G-Mean1 and Balance values of fitness-based voting ensemble classifiers on all the datasets ranged from 0.53-0.57 and 52.71-56.06 respectively. The mean G-Mean1 and Balance values of ASOF models were 0.62 and 60.24 respectively.

Table 8.6: Validation Results of Fitness-based Voting Ensemble Classifiers

| Dataset | G-Mean1 Values | | | | | Balance Values | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | **MVEC** | **WVEC** | **HIEC** | **WVHIEC** | **ASOF** | **MVEC** | **WVEC** | **HIEC** | **WVHIEC** | **ASOF** |
| **AOI** | 0.63 | 0.64 | 0.61 | **0.65** | **0.65** | 62.55 | 62.93 | 59.28 | 63.73 | **64.73** |
| **Click** | 0.52 | 0.55 | **0.56** | 0.55 | **0.56** | 49.44 | 52.77 | 53.56 | 53.29 | **54.48** |
| **DrJava** | 0.65 | 0.66 | **0.70** | 0.66 | 0.65 | 64.87 | 66.02 | **69.66** | 65.92 | 65.17 |
| **Giraph** | 0.49 | 0.55 | 0.56 | **0.59** | 0.56 | 47.46 | 52.57 | 53.78 | **56.12** | 53.45 |
| **Gora** | 0.37 | 0.37 | 0.27 | 0.34 | **0.55** | 39.81 | 39.72 | 34.66 | 37.96 | **52.55** |
| **Hama** | 0.45 | 0.45 | 0.50 | **0.56** | 0.55 | 45.31 | 45.46 | 49.68 | **55.86** | 54.63 |
| **HyperSQL** | 0.68 | 0.68 | 0.67 | **0.70** | 0.69 | 66.21 | 67.42 | 65.30 | **69.34** | 68.16 |
| **JabRef** | 0.61 | 0.64 | 0.64 | 0.66 | **0.68** | 58.30 | 61.64 | 61.99 | 65.03 | **67.32** |
| **Jmeter** | 0.51 | 0.52 | 0.53 | 0.54 | **0.59** | 49.22 | 49.74 | 51.13 | 51.65 | **58.81** |
| **Jedit** | **0.56** | **0.56** | 0.48 | 0.45 | **0.56** | **53.76** | **53.76** | 46.82 | 44.61 | 53.29 |
| **Logicaldoc** | 0.45 | 0.51 | 0.47 | 0.53 | **0.60** | 44.63 | 48.79 | 45.65 | 51.13 | **57.34** |
| **Maven** | 0.68 | **0.70** | 0.69 | 0.69 | 0.69 | 67.42 | **69.64** | 67.24 | 68.47 | 68.52 |
| **Phoenix** | 0.43 | 0.44 | 0.54 | 0.56 | **0.62** | 42.90 | 44.07 | 51.87 | 53.58 | **60.80** |
| **Subsonic** | 0.63 | 0.63 | 0.59 | 0.55 | **0.65** | 60.18 | 59.91 | 55.74 | 51.88 | **61.90** |
| **Zookeeper** | 0.35 | 0.50 | 0.50 | 0.53 | **0.63** | 38.58 | 48.57 | 48.96 | 52.35 | **62.50** |
| **Mean** | 0.53 | 0.56 | 0.55 | 0.57 | **0.62** | 52.71 | 54.87 | 54.35 | 56.06 | **60.24** |
| *The bold values depict the best values for a corresponding dataset. | | | | | | | | | | |

We also performed a Wilcoxon signed rank test at $\alpha = 0.05$ to compare the predictive capability of the developed ASOF Framework models with the fitness-based voting ensemble classifiers. The test was performed with Bonferroni correction. After Bonferroni correction, the p-value should be less than 0.013. The results of the test are stated in Table 8.7 with p-values depicted in parenthesis.

Table 8.7: Wilcoxon test results for ASOF vs Fitness-based Voting Ensemble Classifiers

| ASOF vs | Using G-Mean1 values | Using Balance Values |
|---|---|---|
| MVEC | ↑* (0.001) [0.93] | ↑* (0.001) [0.93] |
| WVEC | ↑* (0.006) [0.80] | ↑* (0.004) [0.80] |
| HIEC | ↑* (0.005) [0.87] | ↑* (0.002) [0.87] |
| WVHIEC | ↑ (0.031) | ↑ (0.023) |
| ↑: Non-significant superiority of the ASOF model; ↑*: Significant superiority of the ASOF model) | | |

According to the results, the prediction models developed using the ASOF framework were better than all the other fitness-based voting ensemble classifiers using G-Mean1 and Balance values, more so significantly in three cases (when compared with MVEC, WVEC and HIEC). The effect size of all significant cases was computed using Vargha and Delaney's statistic and was depicted in square brackets. According to the guidelines discussed in section 8.4.2, the effect size was found to be large (0.80-0.93). These observations advocate competency of the ASOF framework prediction models with the compared fitness-based voting ensemble models. The reason for the better performance of ASOF models could be due to use of dynamic fitness function, which is varied on the basis of the structural characteristics of a class. It may be noted that the WVHIEC models, which allocate weights to constituent fitness variants both on the basis of "performance" and the "ability to predict hard instances" are comparable to the ASOF models. However, each of the four fitness-based ensemble classifiers take into account the prediction models of all the seven constituent fitness variant classifiers (Accuracy-variant, Balance variant, G-Mean1 variant, G-Mean2 variant, G-measure variant, F-measure variant and Precision variant), while predicting an unknown instance. On the other hand, the ASOF model only refers to the prediction of one fitness variant model, which is predicted by it for classifying an unknown instance as change-prone or not change-prone. Therefore, the predictions of ASOF models are faster than the WVHIEC models and the other three fitness-based ensemble classifiers (MVEC, WVEC, HIEC), indicating its superiority.

***Comparison of ASOF framework with ML ensemble classifiers & the LR technique***

In order to compare the results of models developed using the ASOF framework with the ML ensemble classifiers (RF, BG, AB and LB) and the LR technique, we report the G-Mean1 and Balance values of the developed models in Figure 8.6 and 8.7 respectively. We repeat the results of ASOF models for ease of comparison. According to Figure 8.6, the ASOF model shows comparable results with the best ML ensemble classifier or the LR technique in six datasets (DrJava, Giraph, Hama, HyperSQL, Jmeter and Maven). In a majority of datasets (except Jedit), the difference in G-Mean1 values of the ASOF models and the other compared models was no more than 20%. In certain cases, the G-Mean1 values improved by 2-25% with the use of ASOF models as compared to the other models. According to Figure 8.7, the ASOF models depicted comparable Balance values with the best ML ensemble classifier or the LR technique in four datasets (DrJava, Hama, Jmeter and Maven). In more than 50% of the cases, there was an improvement in the Balance values obtained using the ASOF models, which ranged from 1-30%.

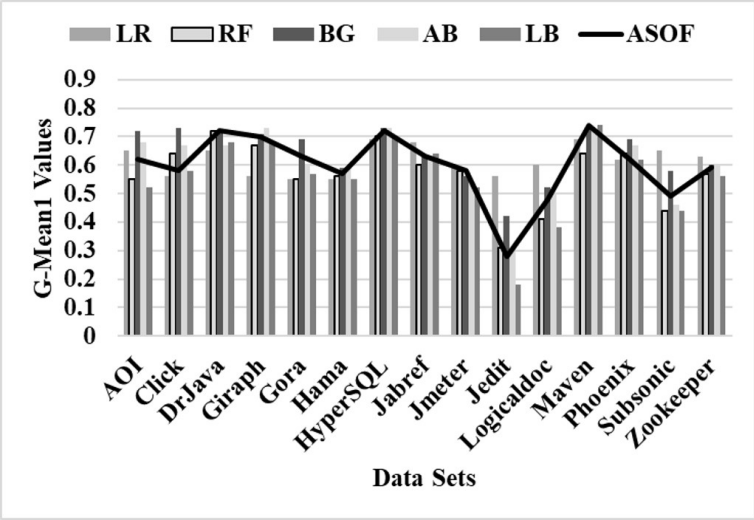


Figure 8.6: G-Mean1 Values of ML Ensemble Classifiers, LR Technique and ASOF Models

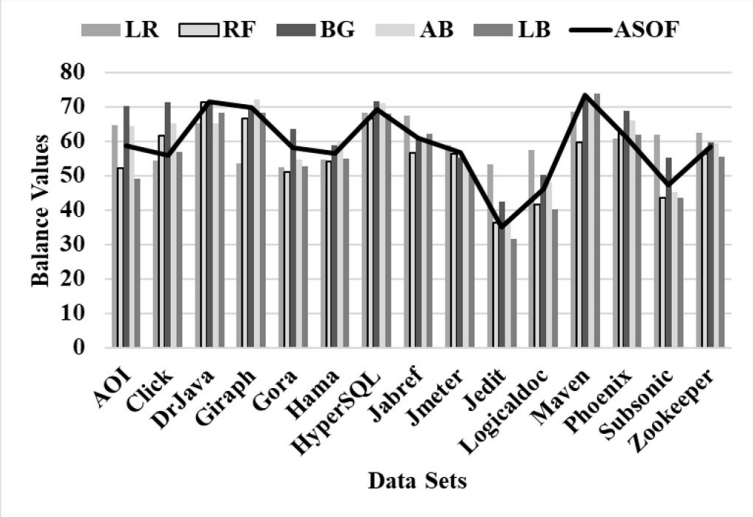


Figure 8.7: Balance Values of ML Ensemble Classifiers, LR Technique and ASOF Models

We further compared the results of ASOF models with ML ensemble classifiers and the LR technique statistically using the Wilcoxon signed rank test at $\alpha = 0.05$. The test was performed with Bonferroni correction. The G-Mean1 and Balance values obtained by all the models (LR, RF, BG, AB and LB) on all the fifteen datasets were compared pairwise with those obtained using ASOF models. The test results are reported in Table 8.8. The obtained p-values are depicted in parenthesis. According to Table 8.8, the G-Mean1 results of ASOF models were superior to LR, BG, AB and LB models and inferior to the RF model, but not significantly. Similarly, the Balance results of ASOF models were better than four other models (LR, BG, AB and LB) and only worse than RF. But these results were not significant. Thus, the performance of change prediction models developed by the ASOF framework were competent to that of models developed using ML ensemble classifiers and the LR technique.

Table 8.8: Wilcoxon test results for ASOF vs ML Ensemble Classifiers and LR Technique

| ASOF vs | Using G-Mean1 values | Using Balance Values |
|---|---|---|
| LR | ↑ (0.256) | ↑ (0.078) |
| RF | ↓ (0.256) | ↓ (0.281) |
| BG | ↑ (0.865) | ↑ (0.776) |
| AB | ↑ (0.256) | ↑ (0.233) |
| LB | ↑ (0.820) | ↑ (0.551) |
| ↑: Non-significant superiority of the ASOF model; ↓: Non-Significant inferiority of the ASOF model) | | |

The ML ensembles work by creating multiple models from the same training data. The given training data is altered to create multiple training samples either by giving weights to specific training instances or by creating bootstrap samples or by other methods [89, 91]. Each ML ensemble technique aggregates multiple models. We analyzed the mean G-Mean1 and Balance values obtained by the ML ensemble classifiers along with the LR technique on all the 15 investigated datasets. The mean G-Mean1 values of LR, RF, BG, AB and LB was 0.57, 0.64, 0.61, 0.56 and 0.60 respectively. The mean Balance values were found to be 55.73 (LR), 62.87 (RF), 59.44 (BG), 55.82 (AB) and 58.60 (LB). We see that except the RF technique, the ASOF technique obtained better mean G-Mean1 and mean Balance values (G-Mean1: 0.62, Balance: 60.24) than all the other compared ML ensemble techniques and the LR technique indicating the effectiveness of ASOF models. With respect to the RF technique, it may be noted that RF is one of the best ML technique advocated in literature for developing defect prediction models (a related area of change prediction) [17]. However, there was only a difference of 3-4% in mean G-Mean1 and mean Balance values of the ASOF and the RF models. Thus, the ASOF models depicted similar results.

As discussed in section 8.3, in order to develop the ASOF prediction model, we use the C4.5 technique, whose run time is comparable to that of ML ensemble classifiers and the LR technique. However, the ASOF outputs only the best fitness

variant. We need to refer to the model of the predicted fitness variant to determine the "change-prone" or "not change-prone" nature of a data point. Thus, evaluation of the actual nature of a data point might take a little longer than of the ML ensemble classifiers. However, the difference in runtime is not significant.

***Answer to RQ3***

The results of change prediction models developed using the ASOF framework are superior to the results of individual fitness variants models as the ASOF framework allocates dynamic fitness function based on the structural characteristics of each class. The ASOF framework models yielded superior G-Mean1 and Balance results when compared to that of change prediction models developed using fitness-based voting ensemble classifiers. They were found competent with change prediction models developed using the investigated ML ensemble classifiers and the LR technique.

## 8.6 Discussion

This chapter proposes a novel framework, namely ASOF, which predicts the optimum fitness function for each instance of the training set based on its structural characteristics (OO metrics). The output of the ASOF model is one of the seven possible fitness functions (Accuracy, Balance, G-Mean1, G-Mean2, G-measure, F-measure and Precision) explored in the chapter. Thereafter, software change prediction models which are developed using the CPSO technique by employing the fitness function predicted by the ASOF model are used for predicting the change-prone nature of a class. In order to perform empirical validation, the results are evaluated on 15 popular open source datasets using two stable performance measures i.e. G-Mean1 and Balance. The key results of the chapter are as follows:

1. The fitness function used by the CPSO technique influences the result of the developed software change prediction models. This is because a fitness function dictates the selection of optimum solution candidates. Amongst the explored fitness variants in the chapter, the Accuracy fitness variant and the Balance fitness variants perform well. However, the Precision variant obtained the worst results. The results of the chapter confirm the complementarity amongst different fitness variants. It was found that though the majority (68-95%) of change-prone classes are correctly predicted by an arbitrary fitness variant, the use of specific fitness variants would improve the predictive capability of the ASOF model by correctly identifying the non-overlapping change-prone classes.

2. The use of ASOF framework was successful in developing effective models for determining change-prone classes. The developed models exhibited G-Mean1 and Balance values in the range of 0.55-0.69 and 53-66% respectively. The ASOF framework is favorable in scenarios when the project is mature enough and appropriate training data is available within the project as compared to a cross-project training.

3. The use of ASOF framework yielded change prediction models which were better than those developed using individual fitness variant's models and the fitness-based voting ensemble classifiers. The results of ASOF framework models were found competent to those developed by ML ensemble classifiers.

Practitioners in the software industry and researchers can effectively use the results of the study for selecting an optimum fitness function while using a search-based algorithm.

# Chapter 9

# Software Bug Categorization using Change Impact and Maintenance Effort

## 9.1 Introduction

Software practitioners often strive to achieve a "bug-free" software, but, this is a myth. Stringent deadlines, intense complexity and deficient testing are prime reasons for the presence of bugs in a software. The bugs generally get introduced during the development or maintenance phase of the software development lifecycle and lead to poor quality software products with unsatisfied customers. Therefore, software maintenance activities, which remove these bugs and ensure the smooth functioning of a software are mandatory [277]. However, it should be noted that maintenance resources are always at constraint [5]. The software community requires to manage these resources appropriately in order to deliver timely upgrades of the software.

An effective method for managing maintenance resources, while removing software bugs is Software Bug Categorization (SBC).

SBC aids in software maintenance activities, which focus on management of bugs and thereby, their elimination. It involves cataloging of software bugs into different levels on the basis of various bug attributes such as their criticality, priority etc. [278–280]. With the aid of SBC, a software developer can take informed decisions while handling a bug and planning its correction. This chapter proposes development of SBC models, which assign levels (viz. "low", "moderate" or "high") to a software bug. The levels are allocated on the basis of three aspects i.e., maintenance effort required to correct a bug, its change impact and the combined effect of both of these. The maintenance effort required to correct a software bug, estimates the SLOC which are required to be added or deleted during bug correction. The change impact of a bug is computed by the number of classes, which are modified while removing the corresponding bug. It is important to categorize software bugs on the basis of maintenance effort and change impact as a bug fixing regime is highly dependent on them. A software developer who is aware that a specific bug belongs to "high" level corresponding to its maintenance effort, will allocate more resources for correction of such a bug as compared to those belonging to "low" and "moderate" levels. Similarly, if a software professional is aware that a bug belongs to "high" level with respect to its change impact, he will be more careful with regression testing after bug correction and will execute a larger number of test cases as a higher number of classes are affected while correcting the bug. Bugs which are allocated "high" level on the basis of combined effect of the maintenance effort and change impact, need both, more maintenance resources and stringent regression testing for their effective resolution.

It may be noted that literature studies have successfully categorized software bugs on the basis of their criticality and priority [278–280]. Two recent studies by Jindal et al. [73, 74], attempted to categorize bugs into different levels on the basis of their

maintenance effort. However, to the best of our knowledge, no study till date has evaluated bug categorization on the basis of its change impact. The existence of this research gap was ascertained by the review conducted in Chapter 3. Thus, this chapter is a pioneer in conducting experiments, which categorize bugs according to their change impact values and the combined effect of maintenance effort and change impact values into three corresponding levels (low, moderate and high).

In order to develop SBC models and correctly predict levels of a software bug, we extract features (words) from the bug description which are present in the bug report. These features are extracted using a text mining approach. This study is a first in successfully identifying the change impact levels of a bug on the basis of its description. The SBC models are developed using six classification techniques namely RF, LR, LB, BG, MLP and NB. The experiments were conducted on five application packages of the Android software (Calendar, Bluetooth, Browser, Camera and MMS). We investigate the following RQs in this chapter:

*RQ1: What is the accuracy of the model developed for SBC, which assigns levels to a bug according to the maintenance effort required to correct it?*

SBC models are developed which allocate a level (low, moderate or high) to each bug on the basis of its maintenance effort using ten-fold cross validation. The maintenance effort is computed by summing up the SLOC required to be added or deleted for bug removal. We develop SBC models using Top-10, Top-25, Top-50 and Top-100 words with six classification techniques. The accuracy of the developed models is adjudged using AUC, accuracy and sensitivity.

*RQ2: What is the accuracy of the model developed for SBC, which assigns levels to a bug according to the change impact of a bug?*

SBC models are developed which allocate level to a bug on the basis of the number of classes that require modification to correct the bug (change impact). Similar to RQ1, we develop models using Top-10, Top-25, Top-50 and Top-100 words with

six classification techniques. The performance is evaluated using AUC, accuracy and sensitivity performance measures.

*RQ3: What is the accuracy of the model developed for SBC, which assigns levels to a bug according to the combined effect of its maintenance effort and change impact?*

We develop models, which categorize a bug into any of the three levels (low, moderate or high) on the basis of combined effect (product) of maintenance effort and change impact. The details of the process are similar to RQ1 and RQ2.

*RQ4: What is the comparative performance of the SBC models when levels are allocated using the combined effect of maintenance effort and change impact with a) maintenance effort and b) change impact?*

The SBC models developed in RQ3 are compared with ones developed in RQ1 and RQ2 using Wilcoxon signed rank test on the basis of AUC values.

The developed SBC models can be successively used by software managers in organizing maintenance and testing resources and improving software quality by delivering good quality products within the defined timeline .

The chapter is organized into five sections. Section 9.2 discusses in detail the proposed SBC framework and the various steps involved in it. Section 9.3 discusses the research methodology, while Section 9.4 states the results of the chapter with respect to each RQ. Section 9.5 states the key findings of the experiments conducted in the chapter. The results of the chapter are communicated as [281].

## 9.2 Software Bug Categorization Framework

This section discusses in detail the framework used for SBC. We first give an overview of the framework and then discuss the steps performed in the text mining module.

### 9.2.1 Overview of the Framework

The diagrammatic representation of the framework is provided in Figure 9.1. We first extract the change logs available in Google's GIT software repository `https://android.googlesource.com` for five application packages of Android operating system so that their corresponding bug data may be analyzed. It must be noted that two versions of each application package were analyzed to extract the changes that have been performed in them. Data present in the change logs between two versions will help in identifying the bugs, which have been corrected and their characteristics. The five application packages and their corresponding versions were: Android Bluetooth (4.1-4.4), Android Browser (2.3-4.0), Android Calendar (4.1-4.4), Android Camera (2.3-4.0) and Android MMS (2.3-4.0).

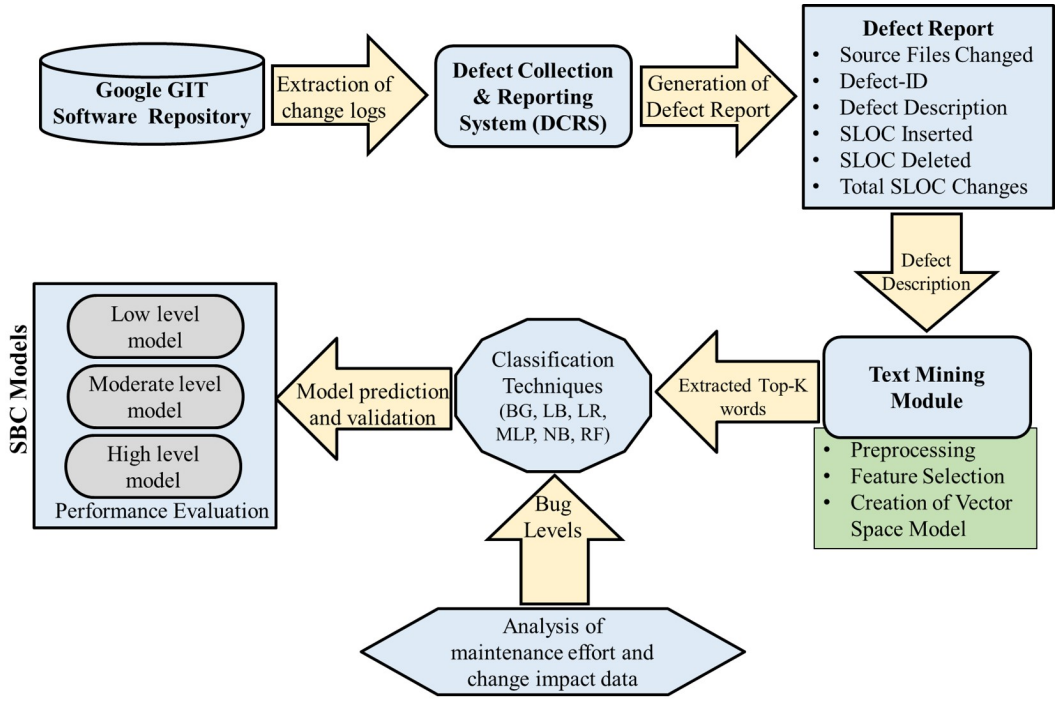


Figure 9.1: Software Bug Categorization Framework

Next, we used the DCRS tool to generate defect reports from the extracted change

logs [106]. Each defect report contains six fields: a) the name of the source code class (Source File Changed), which is affected by bug removal; b) the ID of the bug that has been removed (Defect-ID), c) the textual description of the bug (Defect description); d) the number of SLOC inserted while removing a bug (SLOC Inserted); the number of SLOC deleted during bug removal (SLOC Deleted) and e) the sum of SLOC inserted or deleted for bug removal (Total SLOC changes).

In the next step, we perform text mining steps to extract important words from textual descriptions of the bugs identified in the defect reports. A detailed description of the text mining steps is provided in Section 9.2.2. As an output of these steps we obtain Top-10, Top-25, Top-50 and Top-100 scoring words, which are used as independent variables while creating SBC models. We also analyze the defect reports and allocate levels to bugs on the basis of maintenance effort and change impact characteristics for each software. It should be noted that the maintenance effort, the number of classes impacted and their product (combined) are continuous variables. We convert these continuous variables into ordinal variables by binning the values into three equal sized categories according to their empirical distribution [85]. A similar practice of dividing continuous data into three categories has been followed by previous literature studies, which treat maintenance effort as an ordinal variable [75–77, 262]. The three categories formed were “low”, “moderate” and “high”. Table 9.1 depicts the ranges of continuous values which are allocated to each level for a particular dataset. It also mentions the number of data points allocated to each level in a specific dataset. For instance, according to Table 9.1, all bugs which require [1-6.5] SLOC for correction are allocated “low” level and there are 26 such data points in Bluetooth dataset. Similarly, there are 54 bugs whose modification would affect [1-1.5] classes and are designated “low” level in Bluetooth dataset. A data point consists of the independent variables (Top-10/ Top-25/ Top-50/ Top-100 words) and the allocated level (low/ moderate/ high) for a corresponding bug description.

Table 9.1: Dataset Level Details

| **Dataset Name** | **Level** | **Maintenance Effort** | | **Change Impact** | | **Combined** | |
|---|---|---|---|---|---|---|---|
| | | *Range* | *Data Points* | *Range* | *Data Points* | *Range* | *Data Points* |
| Bluetooth | Low | [1-6.5] | 26 | [1-1.5] | 54 | [1-6.5] | 25 |
| | Moderate | (6.5-39] | 27 | (1.5-3.5] | 15 | (6.5-60.5] | 27 |
| | High | >39 | 26 | >3.5 | 10 | >60.5 | 27 |
| Browser | Low | [1-9.5] | 199 | [1-1.5] | 328 | [1-10.5] | 193 |
| | Moderate | (9.5-38.5] | 196 | (1.5-2.5] | 110 | (10.5-66.5] | 200 |
| | High | >38.5 | 191 | >2.5 | 148 | >66.5 | 193 |
| Calendar | Low | [1-7.5] | 56 | [1-1.5] | 100 | [1-8.5] | 55 |
| | Moderate | (7.5-37.5] | 55 | (1.5-2.5] | 35 | (8.5-45] | 55 |
| | High | >37.5 | 54 | >2.5 | 30 | >45 | 55 |
| Camera | Low | [1-14.5] | 118 | [1-1.5] | 156 | [1-19.5] | 116 |
| | Moderate | (14.5-54.5] | 118 | (1.5-2.5] | 82 | (19.5-117.5] | 119 |
| | High | >54.5 | 117 | >2.5 | 115 | >117.5 | 118 |
| MMS | Low | [1-7.5] | 54 | [1-1.5] | 95 | [1-9.5] | 56 |
| | Moderate | (7.5-30.5] | 57 | (1.5-2.5] | 42 | (9.5-53] | 56 |
| | High | >30.5 | 57 | >2.5 | 31 | >53 | 56 |

We next develop SBC models with six classification techniques using ten-fold cross validation [113]. For each dataset, three SBC models (low level, moderate level and high level) are developed using each classification technique. It should be noted that each SBC model predicts a binary outcome ascertaining whether a bug belongs to a specific level or not. For instance, all "low" level models ascertain that whether a bug belongs to "low" level or "not low" level. Similarly, "moderate" level and "high" level models determine whether a bug belongs to the corresponding categories according to its textual description. Thereafter, the performance of the developed SBC models is evaluated.

### 9.2.2 Text Mining Module

There are a series of steps performed in this module which can be primarily divided into preprocessing tasks, feature selection and the creation of vector space model [9]. We discuss each of these tasks in detail.

*Preprocessing*: The primary aim of preprocessing is to decrease the size of a fea-

ture space represented by the bug descriptions. This step involves tokenization, stop-word removal and stemming [9, 282]. Tokenization includes conversion of individual characters into well-defined tokens. In order to do so, all punctuation characters are replaced with blanks, all non-printable escape characters are removed and all upper-case alphabets are converted to lowercase. All the stop-words such as articles, verbs, prepositions, conjunctions, nouns, pronouns, adjectives and adverbs are discarded in stop-word removal step. Thereafter, stemming is performed. It involves removal of all the words that have the same stem, while retaining the stem. For instance, words like "display", "displayed", "displaying" and "displays" are removed while only the word "display" is retained. As a result of preprocessing, a reduced set of words is obtained.

*Feature Selection*: Though, the initial set of words is reduced after preprocessing, we still need to choose an effective set of words for classification. We use infogain measure for feature selection, which ranks all the words extracted from the preprocessing step [9]. Thereafter, we extract the top K words based on their infogain ranks. The concept of infogain is illustrated with the help of an example. For instance, a dataset may have 35% of "low" level bugs, 25% of "moderate" level bugs and 40% of "high" level bugs. The bug distribution $D_0$ of the dataset would be with bugs, b(1) = "low", b(2) = "moderate" and b(3) = "high". The corresponding frequencies are f(1) = 0.35, f(2) = 0.25 and f(3)= 0.40. In order to encode bug distribution $D_0$ i.e. H($D_0$) can be defined below [283]:

$$p(b) = \frac{f(b)}{\sum_{b \in D} f(b)} \tag{9.1}$$

$$H(D) = -\sum_{b \in D} p(b) log_2 p(b) \tag{9.2}$$

If S denotes the set of features, then the bits required for encoding a bug level are:

$$H(D|S) = -\sum_{s \in S} p(s) \sum_{b \in D} p(b|s) log_2 p(b|s) \tag{9.3}$$

A feature $S_i$ is ranked highest with respect to infogain, if it minimizes the encoding required for the data, after the feature has been used. Therefore,

$$Infogain(S_i) = H(D) - H(D|S_i)$$

It should be noted that we assume that the most informative features are sufficient to describe the corresponding bug reports for their categorization into different levels.

*Creation of Vector Space Model*: After features have been selected, we create a vector space model, which includes all the defect reports in the form of a vector. Here, each defect report is represented in the form of the top-k selected features (terms). Each of the features in a defect report (vector) is weighted using Term Frequency- Inverse Document Frequency (TFIDF) approach. Term frequency refers to the occurrence of a feature in a specific defect report, while document frequency depicts the occurrence of a feature across all defect reports. A higher term frequency indicates higher usage of a term in a specific defect report. On the contrary, if the same term is present in many documents, it is less important due to its decreased discriminative power. A feature is weighted in accordance with high term frequency and the inverse of its document frequency. The TFIDF is defined as follows [9]:

$$TFIDF = T_{pq} * log_2(\frac{ND}{ND_q}) \tag{9.4}$$

Here, $T_{pq}$ represents the frequency of $q^{th}$ term in $p^{th}$ defect report; $ND$ is the total number of defect reports and $ND_q$ is the total number of defect reports containing $q^{th}$ term. Thereafter, we normalize the feature vectors, before they are passed on to the classification techniques.

## 9.3 Research Methodology

This section briefly states the classification techniques used for developing SBC models. We also state the performance measures and the statistical test used to assess and compare the developed SBC models.

*Classification Techniques*: The chapter uses six classification techniques (RF, LR, LB, BG, MLP and NB), which have been simulated in the WEKA tool [88]. The investigated techniques have been previously explored by researchers for developing efficient prediction models [17, 123]. For each classification technique, we use default parameter settings of WEKA tool. The description of each classification technique and the parameter settings can be referred from Chapter 2 (Section 2.6).

*Performance Measures*: The chapter uses AUC for evaluating the SBC models as it is considered as a stable performance measure [118, 284]. However, in order to understand the meaning of AUC in terms of SBC models, we first need to define sensitivity and specificity. Similar to the specificity and sensitivity defined for change-prone and not change-prone classes in Chapter 2 (Section 2.10), the sensitivity of a "high" level model is defined as the number of data points which are correctly predicted to be "high", amongst the total number of points which have been predicted to belong to "high" level. The specificity of a "high" level model is defined as the total number of data points which are correctly predicted to be of "not high" level, amongst the total number of points which have been predicted to belong to "not high" level. The sensitivity and specificity of "low" level and "moderate" level models are defined similarly. AUC is a plot between (1-Specificity) on the horizontal axis versus the sensitivity on the vertical axis [117]. A higher value of AUC indicates a preferred prediction model.

We also analyze the accuracy for a developed SBC model. The accuracy of a

"high" level model is the ratio of correctly predicted "high" level and "not high" level bugs amongst the total number of data points. A good prediction model will have higher accuracy values than others. However, as already discussed accuracy is not termed as a stable performance measure as it does not take into account the class distribution of a dataset [118, 284].

*Statistical Test*: In order to answer RQ4 and compare the performance of the developed SBC models using the combined approach with the other two approaches we use Wilcoxon signed rank test with Bonferroni correction. The test performs pairwise comparisons amongst SBC models. Furthermore, we also computed the effect size of significant cases of Wilcoxon test as mentioned in Chapter 2 (Section 2.11.2).

## 9.4 Analysis and Results

This section presents the results of the chapter and answers each of the investigated RQ.

### 9.4.1 Results specific to RQ1

Table 9.2 states the AUC values obtained by the developed SBC models for each of the investigated classification technique (Tech.) on the five datasets used in the chapter. All the AUC values which are greater than 0.7 are depicted in bold. The AUC values for "low level" SBC models ranged from 0.614-0.853, 0.583-0.698, 0.616-0.828, 0.584-0.747 and 0.540-0.750, while the AUC values of "moderate level" SBC models were found to be in the range of 0.594-0.767, 0.542-0.671, 0.612-0.728, 0.577-0.724 and 0.549-0.680 in a majority of the cases in Bluetooth, Browser, Calendar, Camera and MMS datasets respectively. The maximum AUC values for "high level" mod-

els were as high as 0.786, 0.727, 0.856, 0.799 and 0.715 respectively in Bluetooth, Browser, Calendar, Camera and MMS datasets. These AUC values indicate that the categorization of bugs into different levels according to the corresponding maintenance effort required to correct them is acceptable and accurate.

We also assessed the average AUC values (AVG) obtained by all the classification techniques, for a specific level and a dataset (depicted in italics). The best model (Top-10, Top-25, Top-50 or Top-100) in a particular scenario with respect to average AUC values is denoted in bold with gray cell color. It may be noted that in general, the SBC models developed using Top-10 words gave lower average AUC values than other models (Top-25, Top-50 and Top-100). This could be due to the fact that just 10 words may not be enough to identify the level of a bug report as the number of predictors are very few. Moreover, it may be seen that in seven out of 15 cases, the best average values were obtained by the Top-100 word models as they efficiently learn the keywords required to appropriately distinguish a bug report. These models are efficient in allocating a suitable level to a bug report based on the maintenance effort required to correct it. In the other cases, Top-25 and Top-50 word models were found more suitable.

Table 9.3 presents the average accuracy values obtained by all the SBC models developed using the six classification techniques used in the chapter. The model depicting the best average accuracy value is depicted in bold. It may be seen that the majority of average accuracy values are greater than 60%. Furthermore, it may be noted that the majority of Top 100 models depicted the best average accuracy values at each level (low, moderate and high).

Figure 9.2 depicts the boxplots of sensitivity values obtained by Top-10, Top-25, Top-50 and Top-100 SBC models for each level, for each of the investigated classification technique. The average sensitivity values on all the investigated datasets of Top-10, Top-25, Top-50 and Top-100 models were 0.566, 0.601, 0.628 and 0.641

Table 9.2: AUC Values of SBC Models based on Maintenance Effort

| Tech. | | Bluetooth | | | | Browser | | | | Calendar | | | | Camera | | | | MMS | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | **T10** | **T25** | **T50** | **T100** | **T10** | **T25** | **T50** | **T100** | **T10** | **T25** | **T50** | **T100** | **T10** | **T25** | **T50** | **T100** | **T10** | **T25** | **T50** | **T100** |
| **Low Level** | BG | **0.702** | 0.634 | 0.696 | 0.676 | 0.591 | 0.633 | 0.668 | 0.681 | 0.597 | **0.730** | 0.656 | 0.679 | 0.597 | 0.494 | 0.689 | 0.688 | 0.649 | **0.723** | 0.692 | **0.711** |
| | LB | **0.722** | **0.725** | **0.760** | 0.614 | 0.595 | 0.621 | 0.621 | 0.603 | 0.589 | **0.760** | **0.701** | 0.691 | 0.593 | 0.626 | 0.647 | 0.611 | 0.618 | 0.689 | **0.706** | 0.652 |
| | LR | **0.739** | 0.661 | **0.757** | **0.761** | 0.583 | 0.627 | 0.667 | 0.672 | 0.641 | **0.799** | **0.796** | **0.797** | 0.608 | 0.656 | **0.737** | 0.663 | 0.599 | 0.688 | **0.702** | 0.613 |
| | MLP | **0.771** | **0.741** | **0.796** | **0.724** | 0.588 | 0.642 | 0.676 | 0.693 | 0.627 | **0.811** | **0.777** | **0.781** | 0.625 | 0.666 | **0.747** | **0.730** | 0.645 | **0.706** | 0.689 | **0.750** |
| | NB | **0.746** | **0.780** | **0.853** | **0.761** | 0.579 | 0.591 | 0.636 | 0.660 | 0.599 | **0.780** | **0.777** | **0.828** | 0.568 | 0.629 | 0.691 | 0.679 | 0.540 | **0.704** | **0.722** | **0.744** |
| | RF | **0.708** | **0.775** | **0.801** | **0.783** | 0.584 | 0.606 | 0.658 | 0.698 | 0.616 | **0.784** | 0.676 | **0.764** | 0.584 | 0.633 | **0.728** | **0.741** | 0.660 | **0.725** | **0.721** | **0.729** |
| | AVG | *0.731* | *0.719* | ***0.777*** | *0.720* | *0.587* | *0.620* | *0.654* | ***0.668*** | *0.612* | ***0.777*** | *0.731* | *0.757* | *0.596* | *0.617* | ***0.707*** | *0.685* | *0.619* | ***0.706*** | *0.705* | *0.700* |
| **Moderate Level** | BG | 0.594 | 0.545 | 0.489 | 0.429 | 0.556 | 0.581 | 0.594 | 0.603 | 0.629 | 0.674 | 0.533 | 0.579 | 0.577 | 0.516 | 0.629 | 0.643 | 0.627 | 0.583 | 0.620 | 0.584 |
| | LB | 0.605 | 0.676 | 0.636 | 0.498 | 0.544 | 0.558 | 0.582 | 0.549 | 0.615 | 0.669 | 0.603 | 0.559 | 0.595 | 0.621 | 0.660 | 0.604 | 0.587 | 0.587 | 0.611 | 0.547 |
| | LR | 0.697 | **0.720** | 0.654 | **0.762** | 0.530 | 0.560 | 0.631 | 0.671 | 0.634 | 0.690 | 0.680 | **0.728** | 0.595 | 0.626 | 0.695 | 0.683 | 0.568 | 0.549 | 0.580 | 0.530 |
| | MLP | 0.657 | **0.702** | 0.656 | **0.767** | 0.542 | 0.583 | 0.635 | 0.642 | 0.612 | **0.722** | 0.694 | **0.730** | 0.635 | 0.617 | 0.689 | **0.707** | 0.574 | 0.561 | 0.642 | 0.633 |
| | NB | 0.682 | **0.709** | **0.742** | **0.764** | 0.533 | 0.543 | 0.560 | 0.585 | 0.627 | **0.718** | **0.723** | **0.702** | 0.604 | 0.629 | 0.672 | 0.687 | 0.573 | 0.554 | 0.566 | 0.615 |
| | RF | 0.636 | **0.735** | 0.667 | **0.704** | 0.548 | 0.570 | 0.596 | 0.609 | 0.639 | 0.668 | 0.579 | 0.640 | 0.586 | 0.577 | 0.679 | **0.724** | 0.577 | 0.596 | 0.680 | 0.670 |
| | AVG | *0.645* | ***0.681*** | *0.641* | *0.654* | *0.542* | *0.566* | *0.600* | ***0.610*** | *0.626* | ***0.690*** | *0.635* | *0.656* | *0.599* | *0.598* | *0.671* | ***0.675*** | *0.584* | *0.572* | ***0.617*** | *0.597* |
| **High Level** | BG | 0.537 | 0.554 | 0.543 | 0.549 | 0.635 | 0.684 | 0.693 | 0.697 | 0.653 | **0.767** | **0.735** | **0.711** | 0.669 | 0.501 | 0.694 | **0.715** | 0.625 | 0.662 | 0.688 | 0.673 |
| | LB | 0.512 | 0.620 | 0.623 | 0.580 | 0.640 | 0.670 | 0.647 | 0.628 | 0.649 | **0.769** | **0.758** | 0.641 | 0.695 | **0.727** | 0.685 | 0.662 | 0.606 | 0.678 | 0.634 | 0.632 |
| | LR | 0.537 | 0.602 | 0.642 | **0.733** | 0.643 | 0.665 | 0.697 | 0.649 | 0.646 | **0.753** | **0.722** | **0.754** | 0.697 | **0.752** | **0.756** | **0.718** | 0.561 | 0.562 | 0.579 | 0.534 |
| | MLP | 0.539 | **0.705** | 0.672 | **0.786** | 0.622 | 0.660 | 0.688 | **0.727** | 0.670 | **0.774** | **0.827** | **0.804** | **0.709** | **0.746** | **0.754** | **0.799** | 0.664 | 0.632 | 0.620 | 0.680 |
| | NB | 0.593 | **0.755** | **0.764** | **0.713** | 0.638 | 0.662 | 0.690 | 0.698 | 0.688 | **0.846** | **0.856** | **0.846** | 0.683 | **0.754** | **0.768** | **0.752** | 0.574 | 0.678 | 0.660 | **0.715** |
| | RF | 0.539 | 0.672 | 0.660 | 0.685 | 0.626 | 0.667 | 0.688 | **0.705** | **0.705** | **0.812** | **0.787** | **0.790** | 0.671 | **0.701** | **0.726** | **0.765** | 0.602 | 0.676 | **0.719** | **0.715** |
| | AVG | *0.543* | *0.651* | *0.651* | ***0.674*** | *0.634* | *0.668* | ***0.684*** | ***0.684*** | *0.669* | ***0.787*** | *0.781* | *0.758* | *0.687* | *0.697* | *0.731* | ***0.735*** | *0.605* | *0.648* | *0.650* | ***0.658*** |

T10: Top 10; T25: Top 25; T50: Top 50; T100: Top 100; AVG: Average AUC values of all models; Gray Cells: Best Top-K model for a specific level and for a specific dataset

Table 9.3: Average Accuracy Values of SBC Models based on Maintenance Effort

| Level | Bluetooth | | | | Browser | | | | Calendar | | | | Camera | | | | MMS | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **T10** | **T25** | **T50** | **T100** | **T10** | **T25** | **T50** | **T100** | **T10** | **T25** | **T50** | **T100** | **T10** | **T25** | **T50** | **T100** | **T10** | **T25** | **T50** | **T100** |
| **Low** | 62.24 | 63.50 | **70.46** | 68.14 | 54.10 | 56.94 | 59.61 | **61.92** | 55.66 | 66.16 | 66.36 | **66.77** | 55.33 | 56.75 | **65.16** | 63.88 | 55.95 | 62.90 | 66.17 | **66.37** |
| **Moderate** | 56.75 | 60.34 | 58.65 | **62.66** | 51.88 | 53.01 | 55.09 | **56.00** | 56.97 | **61.92** | 59.19 | 60.10 | 55.85 | 55.90 | 60.44 | **61.71** | 54.96 | 53.27 | **56.25** | 55.36 |
| **High** | 48.52 | 58.65 | 59.07 | **64.77** | 56.68 | 60.61 | **65.02** | 62.40 | 57.58 | 68.28 | **71.11** | 68.28 | 63.41 | 63.60 | 66.62 | **68.04** | 56.05 | 61.71 | 63.29 | **64.39** |

T10: Top 10; T25: Top 25; T50: Top 50; T100: Top 100; Bold contents: Best Top-K model for a specific level and for a specific dataset

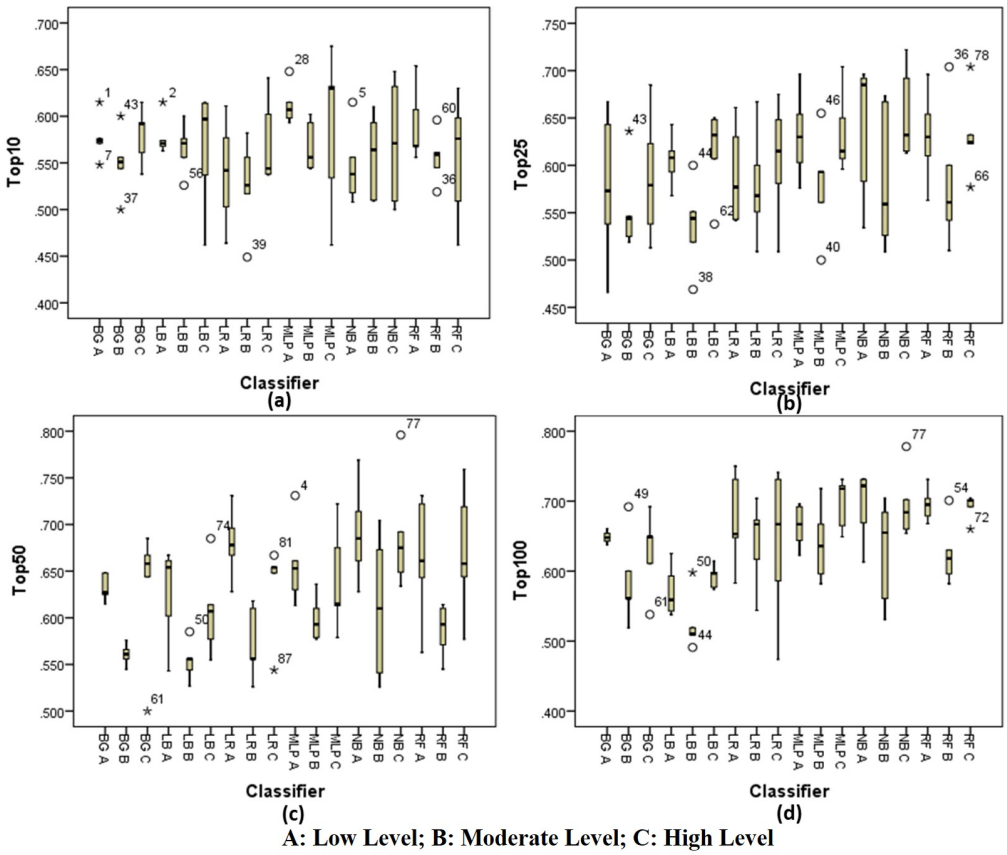


Figure 9.2: Sensitivity Values of (a) Top-10 (b) Top 25 (c) Top 50 and (d) Top 100 SBC Models based on Maintenance Effort

respectively. The trends of sensitivity values depicted in the boxplots indicated their increase in general, with the increase in the number of predictor variables. This is possible as more number of predictor variables are able to effectively encapsulate the textual description of bugs and correctly predict the appropriate level.

An analysis of the average values (AVG) of “low” level, “moderate” level and “high” level models indicate that the “high” level SBC models obtained higher average AUC and accuracy values on three of the investigated datasets (Browser, Calendar and Camera). As discussed earlier, it is essential to identify “high” level bugs according to maintenance effort so that software managers can allocate appropriate

resources while correcting these bugs. This will lead to an effective software maintenance regime with optimized resource usage.

### 9.4.2 Results specific to RQ2

The AUC values of SBC models which categorize a bug in accordance with the number of classes, which will be impacted while correcting a corresponding software bug are depicted in Table 9.4. The developed SBC models identify three specific levels allocated to software bugs i.e. “low”, “moderate” or “high”. As mentioned earlier, the SBC model developed for each level predicts a binary outcome, ascertaining whether a bug belongs to a specific level or not on the basis of its bug report. We develop four models at each level using different number of predictor variables (Top-10, Top-25, Top-50 and Top-100) for each of the investigated classification technique. We depict all AUC values with a value greater than 0.7 as bold in Table 9.4. According to the table, the SBC models developed at “high” level obtained AUC values in the range 0.501-0.709, 0.522-0.624, 0.562-0.785, 0.644-0.730 and 0.510-0.835 in a majority of the cases for Bluetooth, Browser, Calendar, Camera and MMS datasets respectively. The SBC models at “low” level obtained maximum AUC values of 0.687, 0.622, 0.754, 0.721 and 0.668, while those at “moderate” level obtained maximum AUC values of 0.651, 0.525, 0.764, 0.642 and 0.599 respectively for Bluetooth, Browser, Calendar, Camera and MMS datasets. It may be noted that though most of these values (0.6-0.7) are accurate and acceptable, however, few SBC models obtained AUC values in the range 0.3-0.4. These poor values were obtained as the models were developed from highly imbalanced training data. For instance, in Browser dataset, there were only 110 bugs with “moderate” level in the training data, while all other 476 bugs belonged to either “low” or “high” levels (Table 9.1). This means that while developing binary models for “moderate” level, only 18% of the training instances

Table 9.4: AUC Values of SBC Models based on Change Impact

| Tech. | | Bluetooth | | | | Browser | | | | Calendar | | | | Camera | | | | MMS | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | **T10** | **T25** | **T50** | **T100** | **T10** | **T25** | **T50** | **T100** | **T10** | **T25** | **T50** | **T100** | **T10** | **T25** | **T50** | **T100** | **T10** | **T25** | **T50** | **T100** |
| **Low Level** | BG | 0.587 | 0.566 | 0.468 | 0.548 | 0.562 | 0.575 | 0.603 | 0.622 | 0.556 | 0.684 | 0.675 | **0.703** | 0.596 | 0.646 | 0.667 | 0.677 | 0.631 | 0.633 | 0.665 | 0.666 |
| | LB | 0.538 | 0.626 | 0.503 | 0.586 | 0.561 | 0.559 | 0.571 | 0.552 | 0.605 | **0.722** | **0.748** | 0.676 | 0.606 | 0.616 | 0.658 | 0.645 | 0.640 | 0.569 | 0.605 | 0.625 |
| | LR | 0.476 | 0.548 | 0.561 | 0.637 | 0.549 | 0.555 | 0.576 | 0.605 | 0.516 | 0.687 | 0.699 | **0.735** | 0.619 | 0.638 | 0.648 | 0.670 | 0.631 | 0.485 | 0.597 | 0.498 |
| | MLP | 0.481 | 0.671 | 0.556 | 0.560 | 0.564 | 0.562 | 0.604 | 0.600 | 0.587 | **0.754** | **0.754** | **0.718** | 0.628 | 0.639 | 0.649 | 0.682 | 0.642 | 0.526 | 0.580 | 0.587 |
| | NB | 0.553 | 0.617 | 0.567 | 0.455 | 0.557 | 0.581 | 0.574 | 0.562 | 0.533 | 0.646 | 0.682 | 0.688 | 0.621 | 0.604 | 0.671 | 0.653 | 0.668 | 0.651 | 0.668 | 0.612 |
| | RF | 0.491 | 0.602 | 0.529 | 0.687 | 0.544 | 0.531 | 0.574 | 0.631 | 0.561 | 0.692 | **0.720** | **0.749** | 0.596 | 0.653 | **0.703** | **0.721** | 0.618 | 0.609 | 0.641 | 0.629 |
| | AVG | *0.521* | ***0.605*** | *0.531* | *0.579* | *0.556* | *0.561* | *0.584* | ***0.595*** | *0.560* | *0.698* | ***0.713*** | *0.712* | *0.611* | *0.633* | *0.666* | ***0.675*** | ***0.638*** | *0.579* | *0.626* | *0.603* |
| **Moderate Level** | BG | 0.547 | 0.611 | 0.563 | 0.591 | 0.483 | 0.470 | 0.516 | 0.534 | 0.560 | 0.625 | 0.669 | 0.662 | 0.526 | 0.559 | 0.573 | 0.585 | 0.536 | 0.549 | 0.531 | 0.566 |
| | LB | 0.314 | 0.563 | 0.604 | 0.487 | 0.460 | 0.479 | 0.524 | 0.462 | 0.614 | 0.641 | 0.663 | 0.574 | 0.529 | 0.572 | 0.575 | 0.590 | 0.513 | 0.546 | 0.552 | 0.599 |
| | LR | 0.433 | 0.499 | 0.640 | 0.584 | 0.498 | 0.490 | 0.534 | 0.500 | 0.574 | 0.624 | 0.649 | 0.633 | 0.527 | 0.581 | 0.601 | 0.642 | 0.531 | 0.522 | 0.508 | 0.540 |
| | MLP | 0.465 | 0.651 | 0.623 | 0.542 | 0.473 | 0.481 | 0.532 | 0.525 | 0.576 | **0.764** | 0.614 | 0.649 | 0.532 | 0.576 | 0.581 | 0.582 | 0.474 | 0.500 | 0.558 | 0.593 |
| | NB | 0.432 | 0.556 | 0.615 | 0.564 | 0.508 | 0.485 | 0.511 | 0.487 | 0.510 | 0.549 | 0.560 | 0.551 | 0.528 | 0.501 | 0.542 | 0.539 | 0.561 | 0.582 | 0.596 | 0.571 |
| | RF | 0.419 | 0.619 | 0.642 | 0.609 | 0.436 | 0.473 | 0.475 | 0.501 | 0.592 | 0.644 | 0.660 | 0.625 | 0.533 | 0.569 | 0.572 | 0.609 | 0.441 | 0.463 | 0.551 | 0.545 |
| | AVG | *0.435* | *0.583* | ***0.615*** | *0.563* | *0.476* | *0.480* | ***0.515*** | *0.502* | *0.571* | ***0.641*** | *0.636* | *0.616* | *0.529* | *0.560* | *0.574* | ***0.591*** | *0.509* | *0.527* | *0.549* | ***0.569*** |
| **High Level** | BG | 0.567 | 0.562 | 0.501 | 0.630 | 0.546 | 0.547 | 0.605 | 0.618 | 0.607 | **0.708** | **0.713** | **0.730** | 0.644 | 0.687 | 0.685 | 0.686 | 0.632 | **0.711** | **0.716** | **0.716** |
| | LB | 0.438 | 0.559 | 0.499 | **0.709** | 0.581 | 0.562 | 0.579 | 0.561 | 0.575 | **0.709** | **0.702** | 0.696 | 0.645 | 0.674 | 0.696 | 0.674 | 0.671 | 0.687 | 0.644 | 0.660 |
| | LR | 0.430 | 0.466 | 0.543 | 0.662 | 0.561 | 0.560 | 0.593 | 0.618 | 0.598 | 0.674 | 0.654 | **0.710** | 0.665 | **0.730** | **0.723** | 0.651 | 0.624 | 0.510 | 0.616 | 0.616 |
| | MLP | 0.447 | 0.503 | 0.592 | 0.609 | 0.593 | 0.583 | 0.634 | 0.624 | 0.584 | **0.734** | **0.738** | **0.741** | 0.656 | **0.717** | 0.698 | **0.702** | 0.677 | 0.659 | 0.667 | 0.650 |
| | NB | 0.424 | 0.523 | 0.646 | 0.508 | 0.574 | 0.569 | 0.600 | 0.630 | 0.615 | **0.792** | **0.784** | **0.784** | 0.668 | **0.717** | **0.709** | 0.692 | 0.617 | **0.721** | **0.792** | **0.740** |
| | RF | 0.548 | 0.587 | 0.550 | **0.738** | 0.562 | 0.522 | 0.569 | 0.618 | 0.562 | **0.725** | **0.760** | **0.785** | 0.637 | **0.713** | 0.699 | **0.720** | 0.588 | **0.757** | **0.811** | **0.835** |
| | AVG | *0.476* | *0.533* | *0.555* | ***0.643*** | *0.570* | *0.557* | *0.597* | ***0.612*** | *0.590* | *0.724* | *0.725* | ***0.741*** | *0.653* | ***0.706*** | *0.702* | *0.688* | *0.635* | *0.674* | ***0.708*** | *0.703* |

T10: Top 10; T25: Top 25; T50: Top 50; T100: Top 100; AVG: Average AUC values of all models; Gray Cells: Best Top-K model for a specific level and for a specific dataset

Table 9.5: Average Accuracy Values of SBC Models based on Change Impact

| Level | Bluetooth | | | | Browser | | | | Calendar | | | | Camera | | | | MMS | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | **T10** | **T25** | **T50** | **T100** | **T10** | **T25** | **T50** | **T100** | **T10** | **T25** | **T50** | **T100** | **T10** | **T25** | **T50** | **T100** | **T10** | **T25** | **T50** | **T100** |
| **Low** | 48.31 | **56.54** | 51.48 | **56.54** | 52.79 | 53.24 | 56.48 | **57.14** | 51.32 | 63.23 | 64.14 | **66.16** | 56.19 | 58.02 | 63.32 | **64.40** | 59.52 | 56.55 | **61.41** | 60.12 |
| **Moderate** | 43.40 | 57.17 | **59.71** | 52.96 | 49.91 | 47.89 | **51.05** | 49.35 | 51.21 | 58.49 | **56.06** | 57.68 | 52.22 | 53.64 | 54.25 | **55.81** | 49.70 | 50.30 | 51.89 | **54.37** |
| **High** | 46.84 | 54.64 | 54.85 | **64.77** | 55.58 | 53.93 | 55.55 | **57.51** | 52.63 | 64.34 | 65.96 | **69.19** | 61.40 | **65.30** | 64.92 | 64.69 | 62.80 | 64.88 | 65.97 | **67.56** |

T10: Top 10; T25: Top 25; T50: Top 50; T100: Top 100; Bold contents: Best Top-K model for a specific level and for a specific dataset

were those of "moderate" category. The classification techniques were not able to effectively learn the characteristics of bugs so that they could categorize them efficiently into "moderate" level. This leads to poor AUC values. Similarly, the Bluetooth dataset had only 13% of "high" level bugs in the training data. This again leads to SBC models with poor AUC performance.

We also analyzed the best model for a specific level (low, medium or high) for all the investigated datasets using the average AUC values (AVG) obtained by the SBC models with all the six classification techniques. The AVG values of the best model are depicted in bold and with gray cell color. In seven cases, the Top-100 models gave superior results than Top-10, Top-25 and Top-50 models. However, in three and four cases each the Top-25 and Top-50 models also obtained the best average AUC models indicating their suitability for identifying levels of a software bug on the basis of change impact. It was interesting to note that in MMS dataset, the Top-10 word model gave the best values for "low" category.

The average accuracy values of all the SBC models developed using the six classification techniques are depicted in Table 9.5. The best average accuracy value for a specific dataset and for a specific level is depicted in bold. It may be noted that the Top-100 models gave the best average accuracy values in ten out of fifteen cases. However, the average accuracy values of a majority of cases were greater than 50%.

We also analyze the sensitivity values obtained by the developed SBC models, which categorize bugs according to these change impact values. The values are depicted as boxplots in Figure 9.3. Similar to the trend observed in RQ1, the sensitivity values of the developed SBC models improved with the increase in the number of predictor variables. The sensitivity values of Top-10 word models ranged from 0.333-0.677, while that of Top-100 models ranged from 0.400-0.774.

We also compared the average AUC and average accuracy values of SBC models developed for identifying "low", "moderate" and "high" levels of a software bug. It

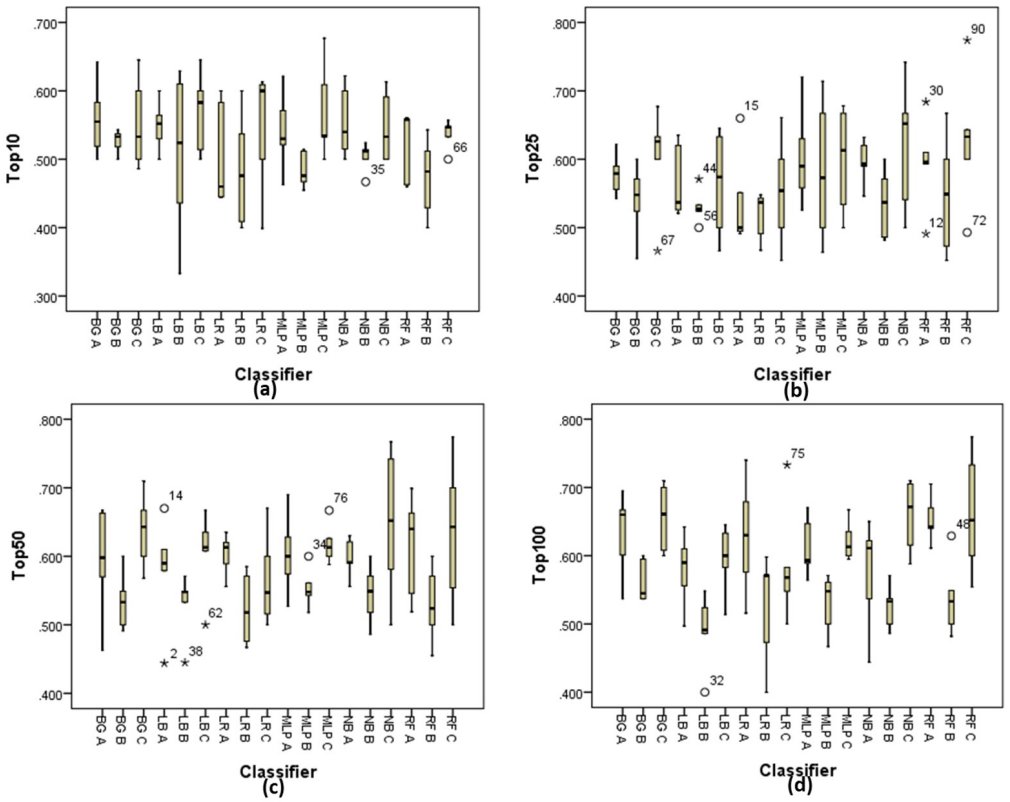


Figure 9.3: Sensitivity Values of (a) Top-10 (b) Top 25 (c) Top 50 and (d) Top 100 SBC Models based on Change Impact

was found that the “high” level bugs were categorized with higher average AUC and average accuracy values as compared to “low” and “moderate” level categories in a majority of the cases. This indicates that the bugs which have high change impact values can be identified effectively. Software managers may allocate more testing resources while conducting regression testing after such bugs are corrected. This is a precautionary measure to identify that no new errors may be introduced while correcting such bugs as these bugs impact a large number of classes during their correction.

### 9.4.3 Results specific to RQ3

SBC models were developed which allocated three possible levels (low, moderate or high) to a bug in accordance with the product of a bug's required maintenance effort and its change impact values. The models used words from the bug description as predictors and determined a binary outcome i.e. whether a bug belonged to a specific level or not. Four different models were developed using each of the six classification techniques of the chapter at each level using a varied number of predictors i.e. 10, 25, 50 and 100. Table 9.6 states the AUC results of the developed models. All AUC values greater than 0.7 are depicted in bold. It can be seen from the table that the AUC values of the "low" level models of Bluetooth, Browser, Calendar, Camera and MMS datasets ranged from 0.620-0.830, 0.586-0.664, 0.577-0.814, 0.519-0.726 and 0.656-0.798 respectively. The ranges of mean AUC values (AVG) obtained by "moderate" level SBC models developed using all the classification techniques were 0.564-0.624 (Bluetooth), 0.541-0.595 (Browser), 0.583-0.680 (Calendar), 0.506-0.635 (Camera) and 0.641-0.734 (MMS). Similarly, the mean AUC value ranges for "high" level SBC models were 0.533-0.606, 0.609-0.667, 0.577-0.752, 0.638-0.735 and 0.625-0.735 for the Bluetooth, Browser, Calendar, Camera and MMS datasets respectively. The majority of obtained AUC values by the SBC models depicted in Table 9.6 were acceptable.

Table 9.7 depicts the average accuracy values of all the SBC models developed using the six classification techniques, with the best model in a specific scenario depicted in bold. The accuracy values of the "low" level models ranged from 54.52-71.03%, while that of the "moderate" level models ranged from 50.97-64.59%. The range of accuracy values of "high" level models was 50.42-70.04%.

Table 9.6: AUC Values of SBC Models based on Combined Effect of Maintenance Effort and Change Impact

| Tech. | | Bluetooth | | | | Browser | | | | Calendar | | | | Camera | | | | MMS | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | T10 | T25 | T50 | T100 | T10 | T25 | T50 | T100 | T10 | T25 | T50 | T100 | T10 | T25 | T50 | T100 | T10 | T25 | T50 | T100 |
| Low Level | BG | 0.684 | 0.620 | 0.620 | 0.549 | 0.598 | 0.624 | 0.655 | 0.661 | 0.579 | **0.717** | 0.668 | 0.690 | 0.623 | 0.639 | 0.658 | 0.645 | 0.686 | **0.717** | **0.742** | **0.740** |
| | LB | 0.677 | **0.742** | **0.799** | 0.676 | 0.601 | 0.635 | 0.623 | 0.593 | 0.613 | **0.761** | **0.814** | **0.751** | 0.519 | 0.617 | 0.673 | 0.639 | 0.672 | **0.730** | **0.762** | 0.672 |
| | LR | **0.709** | **0.706** | **0.805** | **0.758** | 0.592 | 0.631 | 0.652 | 0.640 | 0.621 | **0.773** | **0.776** | **0.720** | 0.649 | 0.669 | **0.709** | 0.653 | 0.656 | **0.717** | 0.694 | **0.733** |
| | MLP | **0.721** | **0.705** | **0.781** | **0.730** | 0.572 | 0.614 | 0.650 | 0.625 | 0.631 | **0.775** | **0.771** | **0.764** | 0.618 | 0.649 | **0.711** | **0.726** | **0.709** | **0.735** | **0.702** | **0.774** |
| | NB | **0.734** | **0.727** | **0.830** | **0.759** | 0.588 | 0.611 | 0.619 | 0.627 | 0.577 | **0.761** | **0.769** | **0.759** | 0.622 | 0.624 | 0.678 | 0.660 | 0.671 | **0.754** | **0.760** | **0.775** |
| | RF | **0.709** | **0.749** | **0.789** | **0.786** | 0.586 | 0.600 | 0.628 | 0.664 | 0.599 | **0.763** | **0.738** | **0.755** | 0.627 | 0.642 | 0.691 | **0.704** | 0.683 | **0.747** | **0.767** | **0.798** |
| | AVG | *0.706* | *0.708* | ***0.771*** | *0.710* | *0.590* | *0.619* | ***0.638*** | *0.635* | *0.603* | ***0.758*** | *0.756* | *0.740* | *0.610* | *0.640* | ***0.687*** | *0.671* | *0.680* | *0.733* | *0.738* | ***0.749*** |
| Moderate Level | BG | 0.626 | 0.509 | 0.510 | 0.478 | 0.539 | 0.585 | 0.602 | 0.601 | 0.598 | 0.625 | 0.600 | 0.595 | 0.510 | 0.555 | 0.608 | 0.626 | 0.634 | 0.671 | 0.665 | 0.666 |
| | LB | 0.581 | 0.648 | 0.609 | 0.562 | 0.553 | 0.568 | 0.581 | 0.556 | 0.569 | 0.689 | **0.752** | **0.707** | 0.473 | 0.562 | 0.633 | 0.589 | 0.650 | **0.728** | **0.735** | 0.649 |
| | LR | 0.637 | 0.649 | 0.543 | 0.571 | 0.557 | 0.624 | 0.622 | 0.610 | 0.625 | **0.717** | **0.733** | 0.601 | 0.524 | 0.602 | 0.624 | 0.652 | 0.618 | 0.653 | **0.742** | 0.692 |
| | MLP | 0.593 | 0.676 | 0.541 | 0.614 | 0.532 | 0.558 | 0.597 | 0.581 | 0.612 | 0.683 | 0.697 | **0.713** | 0.529 | 0.584 | 0.667 | 0.643 | 0.650 | **0.701** | **0.751** | **0.782** |
| | NB | 0.603 | 0.629 | 0.612 | 0.694 | 0.543 | 0.556 | 0.564 | 0.575 | 0.515 | 0.585 | 0.660 | 0.650 | 0.509 | 0.561 | 0.595 | 0.621 | 0.666 | **0.721** | **0.761** | **0.767** |
| | RF | 0.579 | 0.632 | 0.566 | 0.582 | 0.524 | 0.596 | 0.602 | 0.616 | 0.579 | 0.638 | 0.636 | 0.682 | 0.489 | 0.550 | 0.645 | 0.677 | 0.627 | 0.692 | **0.750** | **0.763** |
| | AVG | *0.603* | ***0.624*** | *0.564* | *0.584* | *0.541* | *0.581* | ***0.595*** | *0.590* | *0.583* | *0.656* | ***0.680*** | *0.658* | *0.506* | *0.569* | *0.629* | ***0.635*** | *0.641* | *0.694* | ***0.734*** | *0.720* |
| High Level | BG | 0.580 | 0.605 | 0.549 | 0.514 | 0.600 | 0.638 | 0.688 | **0.700** | 0.536 | 0.631 | 0.609 | 0.589 | 0.668 | **0.718** | **0.708** | **0.713** | 0.637 | **0.710** | **0.751** | **0.739** |
| | LB | 0.504 | 0.611 | 0.543 | 0.484 | 0.613 | 0.648 | 0.649 | 0.622 | 0.563 | **0.761** | **0.798** | **0.724** | 0.503 | **0.716** | **0.713** | 0.681 | 0.669 | 0.327 | **0.735** | 0.690 |
| | LR | 0.519 | 0.516 | 0.473 | 0.601 | 0.615 | 0.673 | 0.677 | 0.634 | 0.571 | **0.755** | **0.782** | 0.657 | 0.675 | **0.755** | **0.741** | 0.698 | 0.620 | 0.559 | 0.535 | 0.646 |
| | MLP | 0.530 | 0.616 | 0.570 | 0.629 | 0.620 | 0.607 | 0.658 | 0.660 | 0.637 | **0.769** | **0.794** | **0.771** | 0.653 | **0.751** | **0.767** | **0.743** | 0.640 | 0.678 | 0.654 | **0.735** |
| | NB | 0.567 | 0.675 | 0.693 | 0.635 | 0.594 | 0.619 | 0.658 | 0.666 | 0.587 | **0.776** | **0.802** | **0.799** | 0.671 | **0.750** | **0.749** | **0.736** | 0.645 | **0.752** | **0.792** | **0.809** |
| | RF | 0.495 | 0.614 | 0.553 | 0.637 | 0.611 | 0.620 | 0.674 | **0.701** | 0.570 | 0.693 | **0.725** | **0.729** | 0.656 | **0.717** | **0.718** | **0.752** | 0.631 | **0.725** | **0.777** | **0.791** |
| | AVG | *0.533* | ***0.606*** | *0.564* | *0.583* | *0.609* | *0.634* | ***0.667*** | *0.664* | *0.577* | *0.731* | ***0.752*** | *0.712* | *0.638* | ***0.735*** | *0.733* | *0.721* | *0.640* | *0.625* | *0.707* | ***0.735*** |

T10: Top 10; T25: Top 25; T50: Top 50; T100: Top 100; AVG: Average AUC values of all models; Gray Cells: Best Top-K model for a specific level and for a specific dataset

Table 9.7: Average Accuracy Values of SBC Models based on Combined Effect of Maintenance Effort and Change Impact

| Level | Bluetooth | | | | Browser | | | | Calendar | | | | Camera | | | | MMS | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | T10 | T25 | T50 | T100 | T10 | T25 | T50 | T100 | T10 | T25 | T50 | T100 | T10 | T25 | T50 | T100 | T10 | T25 | T50 | T100 |
| **Low** | 60.13 | 62.24 | **68.57** | 67.30 | 54.52 | 57.45 | 59.27 | **59.98** | 54.75 | 66.87 | **67.98** | 66.26 | 56.19 | 57.65 | **62.84** | 62.42 | 60.72 | 65.97 | 69.84 | **71.03** |
| **Moderate** | 54.64 | **56.33** | 54.43 | **56.33** | 50.97 | 54.44 | 56.03 | **56.26** | 54.85 | 59.49 | 59.49 | **60.91** | 49.39 | 53.87 | 57.60 | **59.73** | 57.64 | 61.61 | **64.59** | 64.19 |
| **High** | 50.42 | **57.17** | 54.43 | **57.17** | 56.97 | 57.40 | **63.11** | 62.00 | 52.63 | 64.65 | 64.65 | **65.05** | 60.06 | 66.05 | **67.28** | 67.00 | 57.94 | 60.22 | 66.17 | **70.04** |

T10: Top 10; T25: Top 25; T50: Top 50; T100: Top 100; Bold contents: Best Top-K model for a specific level and for a specific dataset

As investigated in RQ1 and RQ2, we analyzed the best model for a specific dataset and a specific level (low, medium, high) using the average AUC values. The best average AUC model for a particular scenario is depicted with gray cell color. According to the analysis, the Top-50 models were found best in eight out of 15 cases, followed by the Top-25 models in four cases. The Top-100 word model was designated as best in only three cases. These results indicate that it is not necessary that the increase in predictor variables leads to an increase in AUC values. A lot of variables may be representing redundant information in Top-100 models. This could be a reason for the good performance of the Top-50 word models. However, in terms of average accuracy values the Top-100 models were best in nine out of 15 cases, followed by the Top-50 models in six cases. As literature studies have criticized accuracy [118, 284] as an effective performance measure, we advocate the results of Top-50 models in the scenario where SBC models are developed to predict the combined effect of change impact and maintenance effort. It may also be noted that Top-10 models were not found best in any case in terms of average AUC or average accuracy values as the number of predictor variables were too few to encapsulate the information found in bug reports for effective SBC.

The sensitivity values of the developed SBC models were also evaluated. They are depicted as boxplots in Figure 9.4. The maximum sensitivity values obtained by Top-10, Top-25, Top-50 and Top-100 word models were 0.679, 0.732, 0.768 and 0.768 respectively. Moreover, it was noted that there was an increase in sensitivity values with the increase in the number of predictor variables. The average sensitivity value of all the Top-10 word models at all levels using all the classification techniques was computed as 0.557, while that of the Top-100 word model was computed as 0.629.

A comparison of average AUC values and average accuracy values (of all the investigated classification techniques) of the three categories of SBC models i.e. “low”,

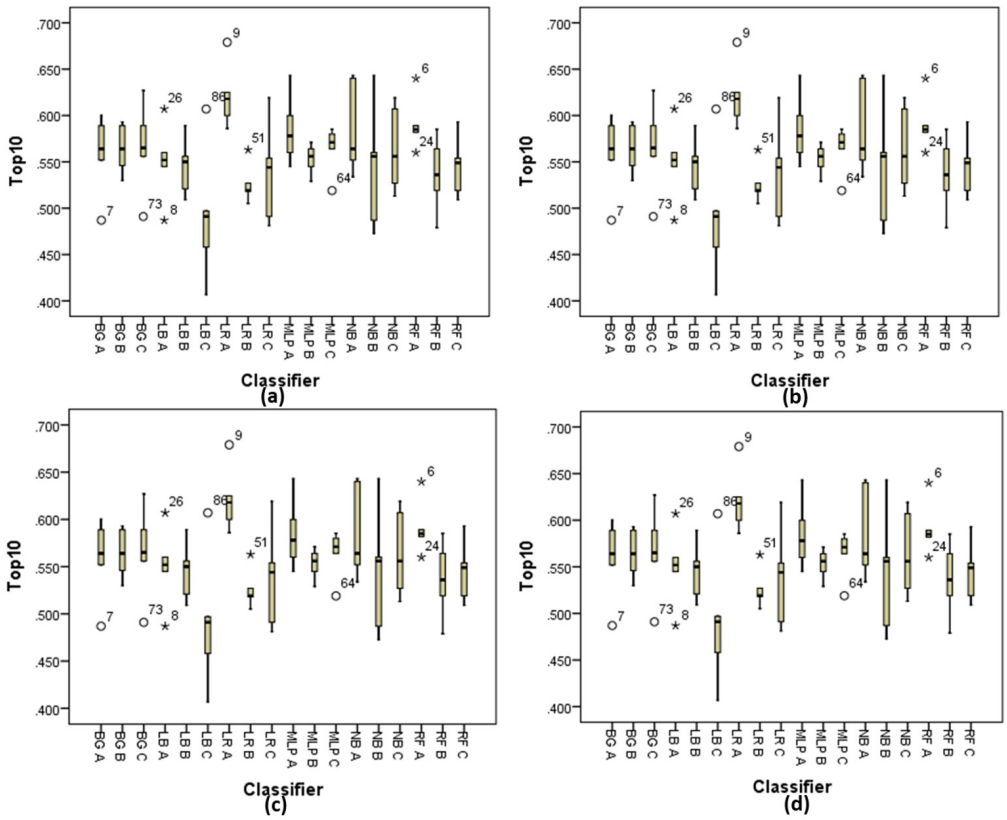


Figure 9.4: Sensitivity Values of (a) Top-10 (b) Top 25 (c) Top 50 and (d) Top 100 SBC Models based on Combined effect of Maintenance Effort and Change Impact

"moderate" and "high" was conducted for Top-10, Top-25, Top-50 and Top-100 models. We found that in Bluetooth, Calendar and MMS datasets, the "low" level models obtained higher average AUC and average accuracy values than "moderate" level and "high" level models. However, in Browser and Camera datasets the "high" level models obtained better average AUC and average accuracy models than the other two. It is important to categorize "high" level bugs efficiently so that software managers can effectively plan software bug regimes. Bugs belonging to "high" levels in accordance with the combined effect of maintenance effort and change impact should be allocated sufficient resources. Also, stringent regression testing should be per-

formed after correction of such bugs. These activities would result in a cost-effective software product with good quality.

### 9.4.4 Results specific to RQ4

In order to compare the performance of the developed SBC models we use Wilcoxon signed rank test with Bonferroni correction. The test was conducted to compare the pairwise performance of the SBC models which categorized bugs in accordance with the combined approach with the other two SBC models (SBC models which categorized according to change impact and SBC models which categorized according to maintenance effort) at all the three levels ("low", "medium" and "high"). We conduct the Wilcoxon test on average AUC values and average accuracy values obtained by all the investigated classification techniques (BG, LB, LR, MLP, NB and RF) for Top-25, Top-50 and Top-100 word models on all the five datasets of the chapter (Bluetooth, Browser, Calendar, Camera and MMS). The hypothesis is evaluated at a significance level of $\alpha = 0.05$. It should be noted that we do not take into account Top-10 word models as they were not very efficient in SBC categorization.

Table 9.8: Wilcoxon test results for Comparing Combined Approach SBC models based on average AUC values

| SBC Level | Combined vs Maintenance Effort | Combined vs Change Impact |
|---|---|---|
| Low | ↑ (0.932) | ↑* (0.001) |
| Medium | ↓ (0.427) | ↑* (0.003) |
| High | ↓ (0.140) | ↑ (0.074) |
| ↑: Not significantly better; ↓: Not significantly poor; ↑*: Significantly better | | |

The results of Wilcoxon test based on average AUC values is depicted in Table 9.8 with p-values depicted in parenthesis. It can be seen that the performance of the combined approach SBC models was better than SBC models based on change impact in all the three levels, more so significantly in two levels (low & moderate). Thus, SBC

models based on change impact were found poor than the combined approach SBC models. The performance of combined approach SBC models was comparable to SBC models based on maintenance effort as the difference was not found significant at any level. The effect size of Wilcoxon test for significant cases was found to be large (0.5-0.6), indicating the superiority of the combined approach as compared to SBC models based on change impact.

Table 9.9: Wilcoxon test results for Comparing Combined Approach SBC models based on average Accuracy values

| SBC Level | Combined vs Maintenance Effort | Combined vs Change Impact |
|---|---|---|
| Low | ↑ (0.842) | ↑* (0.004) |
| Medium | ↓ (0.865) | ↑* (0.006) |
| High | ↓ (0.088) | ↑ (0.460) |
| ↑: Not significantly better; ↓: Not significantly poor; ↑*: Significantly better | | |

Table 9.9 presents the results of Wilcoxon signed rank test based on average accuracy values. Similar to the Wilcoxon test results based on average AUC values, the SBC models based on change impact were found significantly inferior to SBC models based on combined approach for low and moderate levels. The effect size for these significant cases was found to be large (0.5). The high level change impact SBC models were found poorer than combined approach models but not significantly. It may be noted that the results of combined approach SBC models were found comparable to SBC models based on maintenance effort as the results were not significant in either of the three cases.

### 9.4.5 Analysis of Chapter's Results

The cumulative results of the chapter and the statistical analysis indicate that SBC models using combined approach were comparable to SBC models when bugs were allocated levels on the basis of maintenance effort. However, it should be noted that

combined models are more useful as they take into account both the maintenance effort and change impact values. Thus, they aid in the planning of both developer manpower as they are capable of forecasting levels of a bug on the basis of maintenance effort, as well as tester's effort as they are able to forecast bugs which influence change in a large number of classes. A tester may focus his effort on executing test cases for those classes which are affected by a change to ensure that the software is functioning smoothly.

We also observed that the performance of combined SBC models were better than the SBC models which categorized bugs in accordance with change impact values. The primary reason for poor performance of SBC models based on change impact values was the imbalanced nature of the training data. A careful analysis of Table 9.1 indicates that majority of software bugs in the training data of all the five datasets belonged to "low" level (Bluetooth: 68%, Browser: 56%, Calendar: 61%, Camera: 44%, MMS:56%), when they were allocated levels based on change impact. This is because most of the software bugs were resolved by making modifications in just one class. Thus, the classification techniques were not able to learn "moderate" level and "high" level instances appropriately indicating slightly poor performance of these developed SBC models as compared to other (based on combined approach or maintenance effort) developed SBC models. It may be noted from Table 9.1 that the training data for SBC models based on maintenance effort or the combined approach have almost equal number of data points in all the three categories, indicating a more balanced training data.

Another trend found while analyzing all the developed SBC models was that the Top-10 models showed poor performance as compared to Top-25, Top-50 and Top-100 models. In a majority of the cases, the average AUC and average accuracy values of SBC models developed by all the classification techniques using Top-10 words were lesser than Top-25, Top-50 or Top-100 words. As discussed earlier, the

primary reason for such an outcome was the inability of Top-10 words to encapsulate the required information for distinguishing the level of a bug based on its bug report. On the other hand, both Top-50 and Top-100 models were found appropriate with effective AUC values. The reason for the comparable performance of Top-50 and Top-100 models could be redundancy in the predictors. There is a possibility that Top-100 words may be representing redundant information in certain cases, which was effectively encapsulated in just Top-50 words. In such cases, developing models with Top-50 words is more practical.

We also performed a Wilcoxon signed rank test at a cut-off of 0.05 to compare all the developed SBC models "level" wise. We compared the average AUC values and average accuracy values of "high" level models obtained using Top-25, Top-50 and Top-100 words for all the three discussed bug categorizing approaches with "low" and "moderate" level models. This was done in order to ascertain which "level" developed SBC models were most accurate. The Wilcoxon test results obtained are depicted in Table 9.10 with p-values in parenthesis. According to the results in Table 9.10, the performance of "high" level models were better than the "low" level models and significantly better than the "moderate" level models. This indicates that the "high" level bugs are identified with higher accuracy and thus, such bugs should be first determined and allocated appropriate maintenance and testing resources. Such a practice would aid software managers is maintaining good quality software in a cost-effective manner. Thereafter, the remaining resources should be distributed amongst "low" level and "moderate" level bugs.

Table 9.10: Wilcoxon test results for Comparing SBC models Level-wise

| Results | High level vs Low level | High level vs Moderate level |
|---|---|---|
| Based on average AUC values | $\uparrow$ (0.484) | $\uparrow^*$ ($<$0.001) |
| Based on average accuracy values | $\uparrow$ (0.221) | $\uparrow^*$ ($<$0.001) |
| $\uparrow$: Not significantly better; $\uparrow^*$ : Significantly better | | |

## 9.5 Discussion

This chapter proposes a categorization framework for software bugs based on three bug attributes, i.e. its maintenance effort, its change impact and the product of both maintenance effort and change impact. Three binary models using each of the investigated bug attributes are developed which are capable of predicting whether a software bug belongs to "low" level or "not low" level, "moderate" level or "not moderate" level, "high" level or "not high" level. The categorization is done by extracting relevant features from bug reports. Four models are developed in each scenario using Top-10, Top-25, Top-50 and Top-100 relevant features (words) for five application packages of Android software. The SBC models are developed using six classification techniques. Experimental results of the chapter are statistically evaluated using AUC values and average accuracy values of the developed SBC models. The key results of the chapter are as follows:

1. We summarize the results of average AUC values (average of SBC models developed using all the six classification techniques) of "high" level SBC models on all the datasets investigated in the chapter using Top-25, Top-50 and Top-100 words as predictors. The SBC models based on maintenance effort obtained average AUC values in the range 0.651-0.787 for all the datasets. The majority of average AUC values of SBC models based on change impact values were in the range 0.612-0.741. Average AUC values in the range of 0.606-0.752 were obtained by the majority of "high" level SBC models, which categorized bugs on the basis of combined effect of maintenance effort and change impact. These trends indicate an acceptable capability of the developed SBC models. Similar trends were observed by "moderate" level and "low" level SBC models.

2. The accuracy (in terms of AUC values and average accuracy values) of the combined approach SBC models were found to be statistically superior to SBC models which categorized bugs according to change impact and comparable to the SBC models based on maintenance effort.

3. The accuracy of "high" level models were found statistically superior to "moderate" level and "low" level models when compared using average AUC and average accuracy values.

Software managers can use the developed SBC models to predict "high" level bugs in accordance with combined approach as they should be given larger percentage of maintenance resources and adequate testing resources. Software practitioners can also efficiently plan bug fixing regimes as a bug, which is categorized "high" in accordance with maintenance effort and "low" in accordance with change impact values will focus on changes in few classes. Such bugs may require more developer effort but low testing effort.

# Chapter 10

# Software Change Prediction using Imbalanced Data

## 10.1 Introduction

A number of studies in the literature [1–3, 5, 27, 31] and previous chapters have validated the use of various categories of techniques for determination of change-prone classes on several datasets. In order to develop a useful change prediction model, a learning technique requires an efficient training dataset, which has a sufficient number of both change-prone and not change-prone classes so that the model can effectively learn to identify them. However, in the real world a number of datasets are imbalanced in nature i.e. a majority of classes belong to a particular category, with very few instances of the other category, which is generally of more interest (change-prone classes in our case). This fact is ascertained from the observations in Chapter 3, where it was found that 25-100% of datasets in the majority of change-proneness prediction studies were imbalanced in nature. Learning from such imbalanced datasets leads to higher misclassifications for the minority class. This is because of the lack of

information available about the minority class. Such models are rarely of any practical use as the important change-prone classes are neglected and may not be properly identified. This would lead to improper testing and maintenance plans as sufficient resources would not be allotted to the unidentified change-prone classes leading to poor quality software products. Moreover, such products would be costlier to maintain and manage and may not be completed under tight schedule deadlines. This chapter deals with Imbalanced Learning Problem (ILP) while developing software change prediction models. It may be noted that no study in literature has dealt with this problem in the domain of software change prediction. We investigate two specific approaches for efficiently learning from imbalanced datasets: data sampling approaches and cost-sensitive classification.

- *Data Sampling Approach*: The approach modifies the available training data so that the classification technique is provided with adequate number of training instances of both change-prone and not change-prone nature. This may involve oversampling the minority class or undersampling the majority class to provide a nearly uniform distribution. [285, 286].

- *Cost-Sensitive Classification*: The approach allocates weights to different mis-classification errors in a manner that it reduces the cost of total mis-classifications [286, 287].

Apart from these methodologies, this chapter also illustrates as to why stable performance measures should be adopted while evaluating models developed from imbalanced datasets. Thus, the RQ's investigated in the chapter are mentioned below:

- RQ1a: Does the performance of different ML techniques in this chapter significantly improve by using various sampling approaches for ILP in software change prediction?

- RQ1b: Which sampling method performs best amongst different sampling methods analyzed in the chapter?
- RQ2: What is the effectiveness of different MetaCost learners for improving learning through imbalanced data?
- RQ3: What is the comparative performance of the best sampling approach and MetaCost learner for ILP?

In order to evaluate the above RQs, we empirically evaluate six widely used open-source datasets using six ML techniques (AB, RF, BG, LB, NB, MLP). The ILP was addressed by using three data sampling approaches (Resampling with replacement, Synthetic Minority Oversampling Technique (SMOTE), Spread Subsample) and by using MetaCost learners.

Furthermore, the chapter assesses the performance using several stable performance measures like G-Mean1, AUC and Balance and also compares these stable performance measures with traditional measures such as "accuracy", "recall" and "precision". We develop change prediction models using ten-fold cross validation and inter-release validation and statistically evaluate the obtained results.

The organization of the chapter is as follows: Section 10.2 discusses the ILP in detail. Section 10.3 discusses the empirical research framework and Section 10.4 describes the experimental design. Section 10.5 states the research methodology. Section 10.6 states and analyzes the results of the experiments conducted using ten-fold cross validation and Section 10.6 discusses the results specific to each RQ using inter-release validation. Section 10.7 state the key findings of the chapter. The results of the chapter are published as [284].

## 10.2 Imbalanced Learning Problem

ILP has been extensively explored in the literature where a dataset has highly disproportionate number of examples of different categories of classes. We deal with the imbalance problem where the categories of classes are binary i.e. change-prone or not change-prone. In general, we find that there are low number of change-prone classes in a dataset as compared to the number of not change-prone classes. The reason for such an observation is that software datasets adhere to Pareto principle. The principle states that the majority of defects and changes in a software dataset originate from only 20% of classes or modules [3]. Therefore, it is essential to develop models which correctly determine these critical change-prone classes as these classes should be rigorously monitored and designed in the early stages of software development. Such a practice would ensure that minimum number of defects and changes propagate to later stages of software development, leading to a good quality and cost-effective software product.

A classification model should be such that it identifies both change-prone as well as not change-prone classes accurately. However, if the available training data is deficient in the number of change-prone classes (i.e. imbalanced), the model may not be able to accurately learn the characteristics of these classes. Such models may give highly accurate predictions of not change-prone classes, but are unable to identify a majority of the change-prone classes. These models are sometimes incorrectly contemplated as good classification models as they may still exhibit good overall accuracy rates. Such models are highly incompetent and would lead to erroneous judgments and high losses for the software organization.

Let us analyze the discussed scenario with the aid of an example. Assume a software dataset has 1000 data points, which either belong to the change-prone cate-

gory or the not change-prone category. However, the dataset is imbalanced with only 5% of classes belonging to the change-prone category. A classification model which uses the discussed dataset for training would be biased with very low accurate predictions for change-prone classes ranging from 0-10%, but accurate predictions for change-prone classes (close to 100%) [118]. If we assume 10% accuracy for change-prone classes, only 5 out of 50 change-prone classes are accurately predicted. This is unfavorable as change-prone classes require allocation of sufficient resources so that they can be effectively designed, scrutinized, verified and tested. A low accuracy for such classes would mean negligence of these classes which would degrade software quality as higher number of defects get propagated to later stages. Similarly, if we had assumed that the prediction accuracy for not change-prone classes is low, it would mean that a number of not change-prone classes are allocated ample resources. However, these resources are wasted as they are not required for managing such classes. This would lead to overshooting of schedule deadlines and budget overruns leading to a poor reputation of the software organization. Thus, it is crucial for software practitioners to effectively handle ILP where both types of misclassifications are given equal importance so that good quality products are delivered within the allocated time and budget.

## 10.3 Empirical Research Framework

This section states and explains the various design decisions of the conducted experiments. It includes the variables used, data collection, performance measures and statistical tests.

### 10.3.1 Independent and Dependent Variables

In order to develop change prediction models, we use eighteen metrics which represent various OO characteristics. These metrics are used to predict a software quality attribute i.e. change-proneness (dependent variable). The metrics used are six metrics of CK metrics suite [16] (CBO, NOC, DIT, LCOM, WMC, RFC), certain metrics from QMOOD metrics suite [25] (CAM, NPM, MOA, MFA, and DAM) , coupling metrics by Matin [23] (Ca & Ce), and some other commonly used metrics SLOC, LCOM3, CBM, AMC and IC. The details of these metrics are presented in Chapter 2 (Section 2.5.1).

### 10.3.2 Data Collection

The chapter uses three application packages of the Android dataset (Calendar, Bluetooth & MMS) and three common software developed by Apache (IO, Net, Log4j). Both Android and Apache uses GIT as the version control system. The datasets of the chapter are collected using the DCRS tool [106]. A number of studies in literature have analyzed imbalanced data of varying nature i.e. datasets which are majorly imbalanced, moderately imbalanced to datasets which are mildly imbalanced [286, 288, 289]. This chapter uses datasets in which percentage of change-prone classes is in the range of 6%-37%, indicating the use of highly imbalanced to moderately imbalanced datasets for developing change prediction models. The open-source nature of the selected datasets aids the replicability of the experiment. The wide use of Android and Apache datasets enhances the external validity of the experiment as the results of the chapter can be applied effectively in similar scenarios.

It may be noted that the chapter validates change prediction models using both ten-fold cross validation and inter-release validation. The datasets used for develop-

ing ten-fold cross validation models are Android Calendar 4.3.1-4.4.2 and Android Bluetooth 4.3.1-4.4.2 with 19% change each, Android MMS 2.3.7-4.0.2 with 30% change, Apache IO 2.3-2.4 (6% change), Apache Net 3.0-3.1 (37% change) and Apache Log4j 1.2.16-1.2.17 with 25% change. Table 10.1 reports the dataset versions used for training and validation for developing inter-release change prediction models. The table also denotes the percentage of changed classes in each dataset (shown in parenthesis). The details of the number of data points in each dataset can be referred from Appendix A.

Table 10.1: Dataset used for Inter-release Validation

| Dataset Name | Versions for Training | Versions for Validation |
|---|---|---|
| Apache Net | 3.0-3.1 (29%) | 3.1-3.2 (51%) |
| Apache IO | 1.3-1.4 (30%) | 1.4-2.0 (60%) |
| Apache Log4j | 1.2.13-1.2.14 (6%) | 1.2.16-1.2.17 (25%) |
| Android Bluetooth | 4.3.1-4.4.2 (16%) | 5.0.2-5.1.0 (30%) |
| Android Calendar | 4.0.2-4.0.4 (35%) | 4.0.4-4.1.2 (61%) |
| Android MMS | 2.3.7-4.0.2 (27%) | 4.1.2-4.2.2 (52%) |

### 10.3.3 Performance Measures

Appropriate performance measures are required to assess classification models which are developed using imbalanced datasets. Previously, literature studies have criticized the use of traditional performance measures such as precision and accuracy [110, 118, 119] in a scenario, where imbalanced datasets are used for training. On the other hand, the favorability of using robust performance measures which include G-Mean1, Balance and AUC have been discussed by various studies [38, 57, 118, 120, 121, 123]. We use both traditional as well as robust performance measures for evaluating the models developed from imbalanced datasets.

### 10.3.4 Statistical Tests

In order to investigate whether there is any significant difference amongst the performance of different sampling methods (RQ1), we use Friedman test. In case the Friedman test yields significant results, we use Wilcoxon test to determine the pairwise differences of different sampling methods. Similarly, the pairwise differences amongst the MetaCost learners and the best sampling method is ascertained using Wilcoxon test (RQ3).

## 10.4 Experimental Framework

We first discuss the experimental design of the chapter as depicted in Figure 10.1. According to the figure, the experiment involves three phases, which are discussed in detail in the following sections.

### 10.4.1 Data Preprocessing and Feature Selection

In order to evaluate dataset characteristics, we assess the descriptive statistics of all the independent variables of the chapter. Thereafter, we identify outliers from each dataset using the IQR filter implemented in the WEKA tool [88]. Finally, the CFS method is used for identifying significant predictors from each dataset.

### 10.4.2 Approaches for Handling Imbalanced Data

In this phase, we either use sampling methods (Resample with replacement, SMOTE, Spread Subsample) or MetaCost learners in order to provide effective training data for developing classification models.

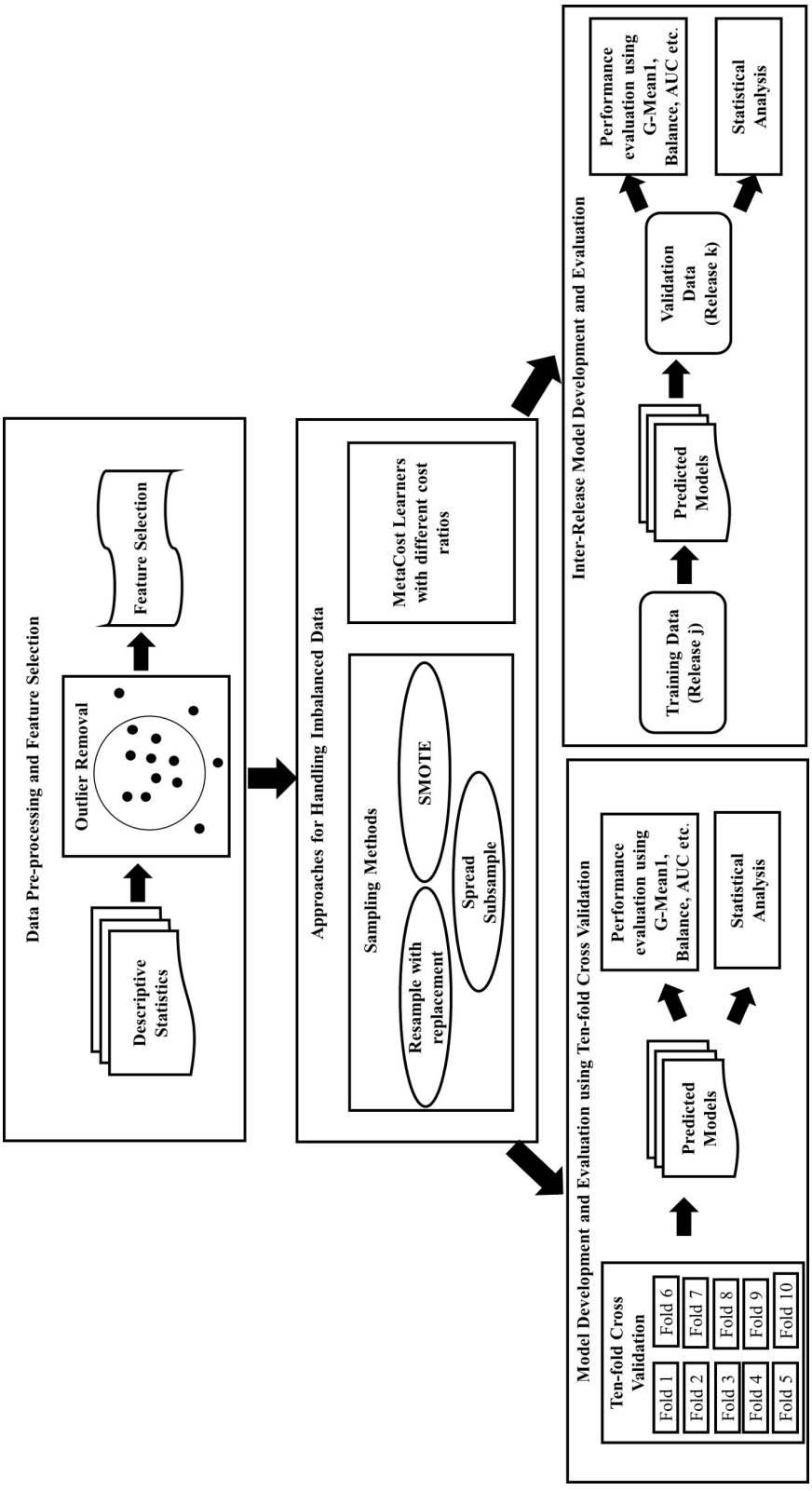


Figure 10.1: Experimental Design for Developing Models using Imbalanced Data

### 10.4.3 Model Development and Evaluation

This phase uses six ML techniques for model development. The developed models are validated either using ten-fold cross validation [113] or by using inter-release validation. In order to develop inter-release validation models, we remove outliers only from the training datasets. The performance of the models are assessed using G-Mean1, Balance and AUC measures (stable measures) as well as accuracy, recall and precision (traditional measures). Furthermore, statistical analysis is also performed to evaluate the performance of different approaches for handling imbalanced data.

### 10.4.4 Hypothesis Evaluation using Statistical Tests

This section states the hypothesis evaluated by using the statistical tests.

*Hypothesis for Friedman Test*: In order to evaluate whether the use of different sampling methods improve the performance of various ML techniques for ILP we use Friedman test to ascertain the following hypothesis (H0, H1, H2) in RQ1. These hypothesis assess the performance of ML techniques based on G-Mean1, Balance and AUC values.

*Alternate Hypothesis H0 /H1 /H2*: Change prediction models developed using various ML techniques, when assessed using G-Mean1 /Balance /AUC performance measure, show significant differences when various sampling methods are used for handling ILP as compared to the scenario when no sampling method is used.

Here, the sampling methods investigated were resampling with replacement method, SMOTE and Spread subsample.

*Hypothesis for Wilcoxon Signed Rank Test*: If the Friedman test yielded significant results, showing X as the best sampling method, we would perform post-hoc Wilcoxon signed rank test with Bonferroni correction in order to evaluate the pair-

wise comparisons amongst the performances of X and other investigated sampling methods in RQ1.

*Alternate Hypothesis H3 /H4 / H5*: Change prediction models developed using various ML techniques (MLP, RF, NB, AB, LB, BG) show significant differences in performance measures (G-Mean1, Balance and AUC) when sampling method X is used instead of other sampling method A for handling imbalanced datasets.

Here, sampling method A corresponds to all other sampling methods used in the chapter except the best ranked sampling method X by the Friedman test. For example, if Resampling with replacement (Sampling Method X) is the best ranked sampling method according to Friedman test, then sampling method A corresponds to SMOTE and Spread Subsample.

RQ3 compares the best sampling method X and MetaCost learners for ILP on the basis of G-Mean1, Balance and AUC performance measures. We investigate the following hypothesis in RQ3 using Wilcoxon test.

*Alternate Hypothesis H6 /H7/ H8*: Change prediction models developed using various ML techniques show significant differences in performance measures (G-Mean1, Balance and AUC) when sampling method X is used instead of MetaCost learners.

## 10.5 Research Methodology

The ML techniques used in the chapter include AB, RF, BG, LB, NB, MLP. The details of these ML techniques can be referred from Chapter 2 (Section 2.6). We use the default parameter settings of WEKA tool [88] for simulation. This section briefly describes the three investigated sampling methods and the MetaCost learners.

### 10.5.1 Resample with Replacement

This method randomly creates a number of samples from the training data. Similar to the bootstrapping approach, the samples are drawn with replacement. The method influences the original distribution of the training data by oversampling the minority class instances. A uniform class distribution is achieved by increasing the number of training instances of change-prone classes, which are exact duplicates of the instances present in the earlier training set. As the training samples are created randomly with replacement, a different training sample may result each time the method is applied. In order to reduce this bias, we perform ten iterations of the method and report the average results. It may be noted that we use a ratio of 1:1 to obtain a uniform distribution of both change-prone and not change-prone classes. Therefore, the dataset obtained after application of this method is balanced.

### 10.5.2 Spread Subsample

It is an undersampling method which eliminates the instances of the majority class. Once appropriate number of majority class instances are removed, a better and more uniform spread of both the categories of classes is obtained [62]. We test several ratios for the method which range from 1:1-10:1 for each dataset. The best ratio for a corresponding dataset may be different as it depends on the percentage of imbalance in the dataset.

### 10.5.3 SMOTE

SMOTE like resampling, is an oversampling method, which adds a number of minority class instances. However, the difference between SMOTE and resampling is the method through which new minority class instances are added. While in resampling

the added instances are exact duplicates of the original instances of the datasets, in SMOTE artificial instances are created. The SMOTE method first obtains k-nearest neighbors of a specific instance of the minority class [285]. We use a k value of 5. Thereafter, we select some of these k-nearest neighbors on the basis of oversampling percentage. An oversampling percentage of 400% would mean four out five neighbors are selected. Next, one synthetic sample is created in the direction of each of the selected neighbor. In order to create a synthetic sample, we first need to compute the Euclidean distance between the original sample and its selected neighbor. Thereafter, we multiply the Euclidean distance with a random number (between 0 and 1). The result is then added to the original sample. Thus, by creating new artificial instances, we generalize the decision boundaries of the minority class instances [285]. We investigate five oversampling ratios for each dataset (100, 200, 300, 400 and 500) and select the ratio which gives the best results.

### 10.5.4 MetaCost Learners

We have already discussed in Section 10.2 that we need to minimize both types of misclassification errors, one that mis-classifies a change-prone class and the other that mis-classifies a not change-prone class. MetaCost learners are used for wrapping a specific classification technique in another procedure which cost-sensitizes it. If a specific class 'm' is classified as 'n', we say the cost of this classification is C(m,n). The mechanism used by MetaCost learners in order to minimize conditional risk ($R(m|x)$) is Bayes optimal prediction. The risk evaluates the cost which is expected if we mis-classify a class x as the one belonging to category 'm'. The $P(m|x)$ depicted in equation 10.1 is the probability that the class m would belong to 'm' category. The space of all the instances are divided into 'r' regions. Each region

signifies that class 'm' is the optimal prediction of that region with respect to cost.

$$R(m|x) = \sum_{n} P(m|x)C(m,n) \tag{10.1}$$

The MetaCost learners are capable of modifying the training data such data they belong to their optimal class. In order to do so, an ensemble of classifiers is used along with voting. Each instance is allocated a label on the basis of probability estimates and the votes received by it. This newly created training data is used by the base classification technique to develop a cost-sensitive prediction model [58, 287].

## 10.6 Data Preprocessing Results

This section describes the results of data preprocessing and analyzes them.

After analyzing the descriptive statistics of the datasets used in the chapter, we found that the inheritance was not much used in the datasets. Also, the datasets exhibit significantly high LCOM values. Each dataset had varying class size in terms of SLOC.

We also determined the outliers in each dataset and removed them. The outliers in Calendar, Bluetooth and MMS datasets were 6, 7 and 22 data points respectively. Apache IO and Apache Net datasets had 11 and 39 data points as outliers respectively. Apache Log4j was detected with 65 data points as outliers.

Table 10.2: Metrics Selected by CFS

| Dataset | Metrics Selected |
|---|---|
| Apache Net | LCOM3, NOC, WMC, SLOC |
| Apache IO | LCOM3, NPM, AMC |
| Apache Log4j | MOA, RFC, SLOC, AMC |
| Android Bluetooth | WMC, RFC, SLOC, CAM |
| Android Calendar | CBO, Ce |
| Android MMS | DAM, MOA, LCOM3, SLOC, CAM, AMC |

The OO metrics selected by the CFS method on each dataset are depicted in Table 10.2. The most commonly selected metric was SLOC, followed by AMC and LCOM3.

## 10.7 Ten-fold Cross Validation Results

This section answers the three investigated RQs of the chapter with respect to ten-fold cross-validation results.

### 10.7.1 Results specific to RQ1

Tables 10.3-10.6 and Appendix D.1 report the values of different performance measures when change prediction models were developed using different sampling methods on each investigated dataset of the chapter. The tables also state the scenario where we do not use any sampling approach. As discussed in Section 10.5, we investigated different ratios and oversampling percentages for spread subsample and SMOTE methods. However, we only state the best values obtained by these methods for a specific dataset.

The results depicted in Table 10.3 show that the accuracy values obtained without the use of any sampling method are higher when compared with the scenario where sampling methods are used in IO and Calendar datasets. A similar trend was observed for a few cases in the Log4j dataset. These trends are an indicator of the ineffectiveness of the accuracy measure. It is a biased indicator, which does not take into account the uneven class distributions [110, 118, 152]. In all other datasets, it was observed that in a majority of the cases, the accuracy measure was improved with the use of sampling methods. On analyzing the recall and precision results stated in Appendix D.1, we noticed an increase in recall and precision values in the majority

Table 10.3: Accuracy Results using Different Sampling Methods

| ML Tech. | Net | | | | ML Tech. | IO | | | | ML Tech. | Log4j | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *Re.S* | *SMOTE* | *Sub.S* | *NSamp.* | | *Re.S* | *SMOTE* | *Sub.S* | *NSamp.* | | *Re.S* | *SMOTE* | *Sub.S* | *NSamp.* |
| **AB** | 78.06 | 81.75 | 76.92 | 77.11 | **AB** | 81.44 | 86.58 | 82.22 | 93.05 | **AB** | 74.60 | 71.27 | 82.46 | 82.46 |
| **RF** | 94.82 | 85.19 | 75.27 | 74.63 | **RF** | 98.18 | 85.71 | 86.67 | 91.44 | **RF** | 88.98 | 81.14 | 79.65 | 78.60 |
| **BG** | 88.76 | 84.92 | 76.92 | 77.61 | **BG** | 94.12 | 84.85 | 88.89 | 94.65 | **BG** | 85.36 | 79.17 | 81.40 | 81.40 |
| **LB** | 87.36 | 83.60 | 75.27 | 78.11 | **LB** | 88.93 | 85.71 | 82.22 | 94.12 | **LB** | 79.96 | 76.32 | 82.46 | 82.81 |
| **NB** | 72.45 | 66.40 | 74.18 | 74.13 | **NB** | 74.65 | 78.79 | 84.44 | 93.58 | **NB** | 67.54 | 59.21 | 78.95 | 79.65 |
| **MLP** | 88.75 | 77.78 | 77.47 | 76.62 | **MLP** | 81.55 | 84.85 | 88.89 | 95.72 | **MLP** | 77.89 | 63.82 | 84.23 | 81.40 |
| **ML Tech.** | **Bluetooth** | | | | **ML Tech.** | **Calendar** | | | | **ML Tech.** | **MMS** | | | |
| | *Re.S* | *SMOTE* | *Sub.S* | *NSamp.* | | *Re.S* | *SMOTE* | *Sub.S* | *NSamp.* | | *Re.S* | *SMOTE* | *Sub.S* | *NSamp.* |
| **AB** | 95.23 | 78.90 | 86.96 | 75.38 | **AB** | 74.45 | 56.22 | 82.56 | 85.00 | **AB** | 79.59 | 85.05 | 73.10 | 69.94 |
| **RF** | 96.62 | 82.57 | 82.61 | 78.46 | **RF** | 80.17 | 76.76 | 79.07 | 83.00 | **RF** | 95.72 | 89.95 | 69.66 | 71.68 |
| **BG** | 91.08 | 80.73 | 82.61 | 83.08 | **BG** | 78.04 | 69.73 | 81.40 | 83.00 | **BG** | 89.88 | 87.75 | 76.55 | 78.61 |
| **LB** | 96.61 | 86.24 | 82.61 | 78.46 | **LB** | 77.86 | 74.59 | 82.56 | 85.00 | **LB** | 84.34 | 85.05 | 73.10 | 73.99 |
| **NB** | 83.85 | 78.90 | 86.96 | 83.08 | **NB** | 54.62 | 55.14 | 79.07 | 81.00 | **NB** | 77.86 | 78.19 | 69.66 | 70.52 |
| **MLP** | 87.09 | 81.65 | 73.91 | 83.08 | **MLP** | 71.03 | 51.89 | 82.56 | 85.00 | **MLP** | 88.43 | 86.03 | 75.17 | 75.72 |
| Re.S indicates Resample with Replacement, SubS. indicates Subsample & NSamp. indicates No Sample | | | | | | | | | | | | | | |

Table 10.4: G-Mean1 Results using Different Sampling Methods

| ML Tech. | Net | | | | ML Tech. | IO | | | | ML Tech. | Log4j | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *Re.S* | *SMOTE* | *Sub.S* | *NSamp.* | | *Re.S* | *SMOTE* | *Sub.S* | *NSamp.* | | *Re.S* | *SMOTE* | *Sub.S* | *NSamp.* |
| **AB** | 0.78 | 0.76 | 0.72 | 0.72 | **AB** | 0.81 | 0.75 | 0.65 | 0.51 | **AB** | 0.73 | 0.70 | 0.47 | 0.49 |
| **RF** | 0.95 | 0.83 | 0.71 | 0.66 | **RF** | 0.98 | 0.75 | 0.77 | 0.51 | **RF** | 0.89 | 0.81 | 0.54 | 0.49 |
| **BG** | 0.89 | 0.82 | 0.70 | 0.70 | **BG** | 0.94 | 0.69 | 0.79 | 0.30 | **BG** | 0.85 | 0.79 | 0.41 | 0.43 |
| **LB** | 0.87 | 0.80 | 0.68 | 0.71 | **LB** | 0.89 | 0.74 | 0.71 | 0.52 | **LB** | 0.79 | 0.76 | 0.47 | 0.51 |
| **NB** | 0.70 | 0.69 | 0.60 | 0.54 | **NB** | 0.73 | 0.52 | 0.66 | 0.52 | **NB** | 0.60 | 0.52 | 0.32 | 0.37 |
| **MLP** | 0.89 | 0.73 | 0.65 | 0.61 | **MLP** | 0.81 | 0.62 | 0.74 | 0.52 | **MLP** | 0.76 | 0.63 | 0.48 | 0.41 |
| **ML Tech.** | **Bluetooth** | | | | **ML Tech.** | **Calendar** | | | | **ML Tech.** | **MMS** | | | |
| | *Re.S* | *SMOTE* | *Sub.S* | *NSamp.* | | *Re.S* | *SMOTE* | *Sub.S* | *NSamp.* | | *Re.S* | *SMOTE* | *Sub.S* | *NSamp.* |
| **AB** | 0.95 | 0.79 | 0.87 | 0.28 | **AB** | 0.73 | 0.48 | 0.42 | 0.42 | **AB** | 0.79 | 0.76 | 0.72 | 0.63 |
| **RF** | 0.97 | 0.83 | 0.83 | 0.49 | **RF** | 0.80 | 0.77 | 0.41 | 0.24 | **RF** | 0.96 | 0.86 | 0.60 | 0.58 |
| **BG** | 0.91 | 0.81 | 0.83 | 0.30 | **BG** | 0.77 | 0.70 | 0.34 | 0.41 | **BG** | 0.90 | 0.81 | 0.69 | 0.61 |
| **LB** | 0.97 | 0.86 | 0.83 | 0.49 | **LB** | 0.77 | 0.75 | 0.42 | 0.42 | **LB** | 0.84 | 0.77 | 0.70 | 0.58 |
| **NB** | 0.80 | 0.79 | 0.87 | 0.58 | **NB** | 0.41 | 0.51 | 0.52 | 0.52 | **NB** | 0.77 | 0.74 | 0.70 | 0.70 |
| **MLP** | 0.86 | 0.81 | 0.74 | 0.51 | **MLP** | 0.68 | 0.52 | 0.42 | 0.42 | **MLP** | 0.88 | 0.80 | 0.67 | 0.62 |
| Re.S indicates Resample with Replacement, SubS. indicates Subsample & NSamp. indicates No Sample | | | | | | | | | | | | | | |

Table 10.5: Balance Results using Different Sampling Methods

| ML Tech. | Net | | | | ML Tech. | IO | | | | ML Tech. | Log4j | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *Re.S* | *SMOTE* | *Sub.S* | *NSamp.* | | *Re.S* | *SMOTE* | *Sub.S* | *NSamp.* | | *Re.S* | *SMOTE* | *Sub.S* | *NSamp.* |
| **AB** | 77.10 | 72.29 | 70.35 | 70.15 | **AB** | 79.49 | 71.49 | 61.21 | 48.53 | **AB** | 70.82 | 45.38 | 68.84 | 46.61 |
| **RF** | 94.45 | 81.83 | 70.50 | 63.56 | **RF** | 97.56 | 71.37 | 73.95 | 48.47 | **RF** | 88.65 | 51.26 | 79.88 | 47.57 |
| **BG** | 88.47 | 80.63 | 67.51 | 67.43 | **BG** | 93.16 | 63.89 | 74.20 | 35.72 | **BG** | 84.61 | 41.67 | 78.75 | 42.89 |
| **LB** | 87.11 | 78.24 | 66.03 | 68.58 | **LB** | 86.96 | 70.16 | 67.26 | 48.56 | **LB** | 77.85 | 45.38 | 75.83 | 47.85 |
| **NB** | 67.67 | 66.43 | 56.39 | 51.69 | **NB** | 70.52 | 49.70 | 61.37 | 48.55 | **NB** | 55.80 | 36.67 | 49.98 | 39.15 |
| **MLP** | 88.54 | 70.49 | 61.30 | 57.63 | **MLP** | 77.62 | 56.29 | 67.86 | 48.57 | **MLP** | 73.45 | 45.42 | 62.38 | 41.67 |
| **ML Tech.** | **Bluetooth** | | | | **ML Tech.** | **Calendar** | | | | **ML Tech.** | **MMS** | | | |
| | *Re.S* | *SMOTE* | *Sub.S* | *NSamp.* | | *Re.S* | *SMOTE* | *Sub.S* | *NSamp.* | | *Re.S* | *SMOTE* | *Sub.S* | *NSamp.* |
| **AB** | 94.85 | 78.86 | 85.86 | 35.24 | **AB** | 71.32 | 47.47 | 41.76 | 41.76 | **AB** | 76.83 | 71.80 | 71.55 | 61.76 |
| **RF** | 96.37 | 82.53 | 82.56 | 47.98 | **RF** | 79.24 | 75.59 | 41.62 | 33.44 | **RF** | 95.32 | 83.45 | 58.78 | 56.26 |
| **BG** | 89.95 | 80.59 | 82.56 | 35.70 | **BG** | 76.28 | 69.83 | 37.60 | 41.71 | BG | 89.42 | 77.94 | 67.04 | 57.57 |
| **LB** | 96.49 | 86.22 | 82.56 | 47.98 | **LB** | 75.67 | 74.20 | 41.76 | 41.76 | **LB** | 83.62 | 72.87 | 68.97 | 55.46 |
| **NB** | 76.23 | 77.48 | 85.86 | 54.70 | **NB** | 43.34 | 49.73 | 49.71 | 49.73 | **NB** | 76.23 | 73.25 | 70.29 | 70.42 |
| **MLP** | 85.16 | 80.11 | 73.84 | 48.42 | **MLP** | 66.55 | 51.88 | 41.76 | 41.76 | **MLP** | 87.43 | 77.65 | 64.15 | 59.87 |
| Re.S indicates Resample with Replacement, SubS. indicates Subsample & NSamp. indicates No Sample | | | | | | | | | | | | | | |

Table 10.6: AUC Results using Different Sampling Methods

| ML Tech. | Net | | | | ML Tech. | IO | | | | ML Tech. | Log4j | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *Re.S* | *SMOTE* | *Sub.S* | *NSamp.* | | *Re.S* | *SMOTE* | *Sub.S* | *NSamp.* | | *Re.S* | *SMOTE* | *Sub.S* | *NSamp.* |
| **AB** | 0.84 | 0.87 | 0.80 | 0.78 | **AB** | 0.93 | 0.87 | 0.84 | 0.66 | **AB** | 0.79 | 0.78 | 0.64 | 0.66 |
| **RF** | 0.98 | 0.92 | 0.83 | 0.80 | **RF** | 1.00 | 0.92 | 0.81 | 0.67 | **RF** | 0.96 | 0.88 | 0.62 | 0.61 |
| **BG** | 0.95 | 0.90 | 0.81 | 0.78 | **BG** | 0.99 | 0.89 | 0.74 | 0.67 | **BG** | 0.93 | 0.85 | 0.65 | 0.68 |
| **LB** | 0.93 | 0.87 | 0.81 | 0.79 | **LB** | 0.98 | 0.88 | 0.86 | 0.70 | **LB** | 0.87 | 0.82 | 0.65 | 0.68 |
| **NB** | 0.81 | 0.81 | 0.78 | 0.78 | **NB** | 0.88 | 0.82 | 0.85 | 0.72 | **NB** | 0.68 | 0.62 | 0.58 | 0.62 |
| **MLP** | 0.89 | 0.85 | 0.84 | 0.77 | **MLP** | 0.88 | 0.85 | 0.84 | 0.74 | **MLP** | 0.80 | 0.69 | 0.67 | 0.67 |
| **ML Tech.** | **Bluetooth** | | | | **ML Tech.** | **Calendar** | | | | **ML Tech.** | **MMS** | | | |
| | *Re.S* | *SMOTE* | *Sub.S* | *NSamp.* | | *Re.S* | *SMOTE* | *Sub.S* | *NSamp.* | | *Re.S* | *SMOTE* | *Sub.S* | *NSamp.* |
| **AB** | 0.96 | 0.83 | 0.83 | 0.66 | **AB** | 0.80 | 0.61 | 0.46 | 0.43 | **AB** | 0.87 | 0.89 | 0.78 | 0.76 |
| **RF** | 0.97 | 0.88 | 0.89 | 0.70 | **RF** | 0.87 | 0.82 | 0.55 | 0.61 | **RF** | 0.98 | 0.92 | 0.78 | 0.61 |
| **BG** | 0.95 | 0.88 | 0.89 | 0.77 | **BG** | 0.84 | 0.79 | 0.54 | 0.54 | **BG** | 0.96 | 0.90 | 0.82 | 0.79 |
| **LB** | 0.96 | 0.91 | 0.83 | 0.67 | **LB** | 0.84 | 0.76 | 0.47 | 0.49 | **LB** | 0.91 | 0.89 | 0.82 | 0.79 |
| **NB** | 0.91 | 0.90 | 0.95 | 0.85 | **NB** | 0.59 | 0.56 | 0.56 | 0.58 | **NB** | 0.82 | 0.82 | 0.79 | 0.79 |
| **MLP** | 0.87 | 0.88 | 0.77 | 0.79 | **MLP** | 0.75 | 0.56 | 0.51 | 0.53 | **MLP** | 0.90 | 0.88 | 0.79 | 0.80 |
| Re.S indicates Resample with Replacement, SubS. indicates Subsample & NSamp. indicates No Sample | | | | | | | | | | | | | | |

of the cases when the datasets were balanced using sampling methods. The no sampling scenario depicted low accuracy, recall and precision values.

An analysis of G-Mean1, Balance and AUC values depicted in Table 10.4-10.6 show improvement when a sampling method was used. The AUC values improved up to 35% with the use of sampling methods, while the G-Mean1 and Balance measures showed an improvement of up to 60% each.

Table 10.7: Friedman Results

| **Using G-Mean1 values** | | | | | |
|---|---|---|---|---|---|
| Dataset | Rank I | Rank II | Rank III | Rank IV | p-value |
| Net | Resample (1.00) | SMOTE (2.00) | Subsample (3.33) | No Sample (3.67) | 0.001 |
| IO | Resample (1.00) | Subsample (2.33) | SMOTE (2.75) | No Sample (3.92) | 0.001 |
| Log4j | Resample (1.00) | SMOTE (2.00) | Subsample (3.33) | No Sample (3.67) | 0.001 |
| Bluetooth | Resample (1.17) | Subsample (2.25) | SMOTE (2.58) | No Sample (4.00) | 0.002 |
| Calendar | Resample (1.50) | SMOTE (2.17) | Subsample (3.17) | No Sample (3.17) | 0.052 |
| MMS | Resample (1.00) | SMOTE (2.00) | Subsample (3.08) | No Sample (3.92) | $< 0.001$ |
| **Using Balance values** | | | | | |
| Dataset | Rank I | Rank II | Rank III | Rank IV | p-value |
| Net | Resample (1.00) | SMOTE (2.00) | Subsample (3.17) | No Sample (4.00) | 0.001 |
| IO | Resample (1.00) | SMOTE (2.00) | Subsample (3.17) | No Sample (4.00) | 0.001 |
| Log4j | Resample (1.00) | SMOTE (2.00) | No Sample (3.83) | Subsample (3.67) | 0.001 |
| Bluetooth | Resample (1.33) | Subsample (2.17) | SMOTE (2.50) | No Sample (4.00) | 0.004 |
| Calendar | Resample (1.50) | SMOTE (1.92) | No Sample (3.17) | Subsample (3.42) | 0.018 |
| MMS | Resample (1.00) | SMOTE (2.00) | Subsample (3.17) | No Sample (3.83) | 0.001 |
| **Using AUC values** | | | | | |
| Dataset | Rank I | Rank II | Rank III | Rank IV | p-value |
| Net | Resample (1.00) | SMOTE (1.67) | Subsample (3.00) | No Sample (4.00) | 0.001 |
| IO | Resample (1.33) | SMOTE (2.00) | Subsample (2.67) | No Sample (4.00) | 0.003 |
| Log4j | Resample (1.00) | SMOTE (2.17) | No Sample (3.00) | Subsample (3.83) | 0.001 |
| Bluetooth | Resample (1.33) | SMOTE (2.33) | Subsample (2.50) | No Sample (3.83) | 0.010 |
| Calendar | Resample (1.00) | SMOTE (2.17) | No Sample (3.00) | Subsample (3.83) | 0.001 |
| MMS | Resample (1.17) | SMOTE (1.83) | Subsample (3.17) | No Sample (3.83) | 0.001 |

Furthermore, we also analyze the improvement in the capabilities of ML techniques for developing change prediction models by testing the hypothesis stated in Section 10.4.4 using Friedman test. The sampling method obtaining the lowest rank in a particular scenario is considered as the best one. The degrees of freedom for Friedman test is 3 as we compare three sampling methods and the scenario where no sampling method is used. The test was conducted at a cutoff of 0.05. It was conducted

on the values of performance measures attained by the change prediction models developed using all the six ML techniques on a specific dataset. The Friedman test was assessed individually on each of the investigated dataset of the chapter.

Table 10.7 states the Friedman test results with the mean ranks stated in parenthesis and the p-values, when assessed using G-Mean1, Balance and AUC values on each dataset of the chapter. The results indicate that the best rank was obtained by the resample with replacement method in all the six datasets. According to Friedman test results, the no sampling scenario obtained the worst ranks in all the six datasets using G-Mean1 values and in four datasets each using AUC and Balance values. The Friedman test results were significant in 17 out of 18 cases. Since, the resample with replacement method significantly outperformed the other scenarios, we accept the alternate hypothesis H0, H1 and H2. The results favor the application of a sampling method for developing change prediction models from imbalanced datasets.

As Friedman test results were found significant, we use post-hoc Wilcoxon test with Bonferroni correction to perform pairwise comparisons amongst resample with replacement method and the other sampling methods. The comparisons were performed using G-Mean1, Balance and AUC performance measures at a cut-off of 0.05. The test was evaluated on the results of all the ML techniques on all the investigated datasets of the chapter.

Table 10.8: Wilcoxon Test Results on Sampling Methods Performance

| Pair | G-Mean1 | Balance | AUC |
|---|---|---|---|
| Resample with replacement vs SMOTE | ↑* (<0.001) | ↑* (<0.001) | ↑* (<0.001) |
| Resample with replacement vs Spread Subsample | ↑* (<0.001) | ↑* (<0.001) | ↑* (<0.001) |
| ↑* represents that the results are significantly better | | | |

The Wilcoxon test results depicted in Table 10.8 indicate that the resample with replacement method is significantly better than all the other investigated methods (based on G-Mean1, Balance, AUC values). The p-values depicted in parenthesis in-

dicate that the results are statistically significant. Thus, we reject null hypothesis H3, H4 and H5 establishing the statistical superiority of the resample with replacement method.

***Answer to RQ1***

The results indicate that the performance of different ML techniques improved with the application of different sampling methods when evaluated using G-Mean1, Balance or AUC values. The results of Friedman and Wilcoxon tests advocate the resample with replacement method as the best sampling method. This is because it increases the number of minority class instances through resampling without increasing the number of total training instances (as in the case of SMOTE), or by removing instances and losing information (as in the case of Spread Subsample).

### 10.7.2 Results specific to RQ2

We assess the performance of different MetaCost learners by evaluating the change prediction models developed using them on the basis of G-Mean1, Balance and AUC values. We also evaluated the scenario when no cost-sensitive approach was used. We assess the total cost of mis-classifications of each model for each Cost Ratio (CR) using the following formula:
*Cost = (Cost of a FN prediction * No. of FN predictions) + (Cost of an FP prediction * No. of FP predictions)*

The chapter analyzes seven different CR's (5, 10, 15, 20, 25, 30 and 50) on each dataset. Table 10.9-10.12 states the values of accuracy, G-Mean1, Balance and AUC measures obtained by change prediction models developed using AB, RF, BG, LB, NB and MLP techniques on each dataset. The models were developed using different CR's and the "original" technique without any CR. The recall values, precision values and the total cost of each ML technique on each dataset using various CR's

are mentioned in Appendix D.2.

Table 10.9 depict the accuracy values of MetaCost learners. According to the table, the accuracy values of change prediction models decreased with the application of MetaCost learners as compared to the "original" technique without any CR. As discussed in Section 10.7.1, this observation was due to the biased nature of accuracy measure, which makes it inappropriate for evaluating models developed using imbalanced datasets [110, 118, 152]. The accuracy measure only accounted for the large number of not change-prone classes and neglected the low number of change-prone classes. This fact can be confirmed from the low recall values depicted in Appendix D.2 (Table D.3).

The G-Mean1 values with different CR's are depicted in Table 10.10. A decrease in G-Mean1 values was observed in two datasets in the majority of the cases (Calendar and Log4j). However, there was an improvement in G-Mean1 values in three datasets (Bluetooth, IO and Net). The G-Mean1 values were comparable in MMS dataset. The decrease in G-Mean1 values can be explained as it is the geometric mean of both specificity and recall. The use of cost-sensitive MetaCost learners results in improvement of recall values (correct prediction of change-prone classes). However, this improvement might also result in reduction of correct predictions of not change-prone classes, which may not be compensated well by the increase in recall values as change-prone classes are fewer in number. This can be confirmed by observing the decrease in precision values (Appendix D.2: Table D.4). This could be a reason for decrease in overall G-Mean1 values in two datasets of the chapter.

The Balance results using different MetaCost learners are mentioned in Table 10.11. The application of MetaCost learners led to an improvement of Balance values. We found an improvement of up to 4% and 15% respectively on Balance values of Net and Log4j datasets. Similarly, Balance values on IO, MMS, Bluetooth and Calendar datasets improved up to 20%, 50%, 31% and 11% respectively.

Table 10.9: Accuracy Results of MetaCost Learners using ML techniques

| ML Tech. | Net | | | | | | | | ML Tech. | Log4j | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *Org.* | *CR5* | *CR10* | *CR15* | CR20 | *CR25* | *CR30* | *CR50* | | *Org.* | *CR5* | *CR10* | *CR15* | CR20 | *CR25* | *CR30* | *CR50* |
| **AB** | 77.11 | 64.68 | 56.22 | 54.73 | 53.73 | 52.74 | 52.74 | 52.74 | **AB** | 82.46 | 56.49 | 37.54 | 18.95 | 20.00 | 20.00 | 20.00 | 20.00 |
| **RF** | 74.63 | 68.66 | 63.68 | 61.69 | 59.70 | 58.71 | 58.71 | 60.20 | **RF** | 78.60 | 62.11 | 50.88 | 45.96 | 42.81 | 41.40 | 39.30 | 32.98 |
| **BG** | 77.61 | 63.18 | 57.21 | 55.22 | 52.74 | 39.80 | 39.80 | 29.35 | **BG** | 81.40 | 64.91 | 32.28 | 20.00 | 20.00 | 20.00 | 20.00 | 20.00 |
| **LB** | 78.11 | 63.18 | 59.70 | 55.72 | 55.22 | 52.24 | 52.24 | 31.34 | **LB** | 82.81 | 65.26 | 36.84 | 22.46 | 20.00 | 20.00 | 20.00 | 20.00 |
| **NB** | 74.13 | 76.12 | 73.63 | 66.67 | 65.17 | 61.19 | 61.19 | 60.70 | **NB** | 79.65 | 64.21 | 31.93 | 23.16 | 22.46 | 22.81 | 20.70 | 21.05 |
| **MLP** | 76.62 | 66.67 | 62.19 | 52.24 | 44.78 | 38.31 | 38.31 | 31.34 | **MLP** | 81.40 | 61.75 | 25.26 | 20.00 | 20.00 | 20.00 | 20.00 | 20.00 |
| **ML Tech.** | **IO** | | | | | | | | **ML Tech.** | **MMS** | | | | | | | |
| | *Org.* | *CR5* | *CR10* | *CR15* | CR20 | *CR25* | *CR30* | *CR50* | | *Org.* | *CR5* | *CR10* | *CR15* | CR20 | *CR25* | *CR30* | *CR50* |
| **AB** | 93.05 | 89.84 | 89.30 | 85.03 | 78.07 | 77.01 | 76.47 | 69.52 | **AB** | 69.94 | 56.49 | 37.54 | 18.95 | 20.00 | 20.00 | 20.00 | 20.00 |
| **RF** | 91.44 | 86.10 | 86.10 | 83.96 | 82.89 | 81.28 | 78.61 | 75.40 | **RF** | 71.68 | 62.11 | 50.88 | 45.96 | 42.81 | 41.40 | 39.30 | 32.98 |
| **BG** | 94.65 | 91.44 | 90.37 | 86.10 | 82.89 | 75.94 | 62.03 | 16.04 | **BG** | 78.61 | 64.91 | 32.28 | 20.00 | 20.00 | 20.00 | 20.00 | 20.00 |
| **LB** | 94.12 | 91.44 | 86.10 | 81.28 | 73.26 | 70.05 | 65.78 | 44.39 | **LB** | 73.99 | 65.26 | 36.84 | 22.46 | 20.00 | 20.00 | 20.00 | 20.00 |
| **NB** | 93.58 | 77.54 | 73.26 | 72.73 | 71.12 | 69.52 | 67.91 | 64.71 | **NB** | 70.52 | 64.21 | 31.93 | 23.16 | 22.46 | 22.81 | 20.70 | 21.05 |
| **MLP** | 95.72 | 92.51 | 80.75 | 71.66 | 66.31 | 64.17 | 56.15 | 47.59 | **MLP** | 75.72 | 61.75 | 25.26 | 20.00 | 20.00 | 20.00 | 20.00 | 20.00 |
| **ML Tech.** | **Bluetooth** | | | | | | | | **ML Tech.** | **Calendar** | | | | | | | |
| | *Org.* | *CR5* | *CR10* | *CR15* | CR20 | *CR25* | *CR30* | *CR50* | | *Org.* | *CR5* | *CR10* | *CR15* | CR20 | *CR25* | *CR30* | *CR50* |
| **AB** | 75.38 | 76.92 | 69.23 | 67.69 | 66.15 | 64.62 | 61.54 | 55.38 | **AB** | 75.00 | 82.00 | 17.00 | 16.00 | 16.00 | 17.00 | 17.00 | 17.00 |
| **RF** | 78.46 | 81.54 | 64.06 | 63.08 | 60.00 | 58.46 | 56.92 | 55.38 | **RF** | 50.00 | 68.00 | 33.00 | 31.00 | 29.00 | 26.00 | 25.00 | 23.00 |
| **BG** | 83.08 | 76.92 | 66.15 | 61.54 | 46.15 | 26.15 | 18.46 | 16.92 | **BG** | 50.00 | 74.00 | 17.00 | 17.00 | 17.00 | 17.00 | 17.00 | 17.00 |
| **LB** | 78.46 | 73.85 | 72.31 | 63.08 | 63.08 | 55.38 | 50.77 | 44.62 | **LB** | 75.00 | 78.00 | 25.00 | 18.00 | 16.00 | 16.00 | 16.00 | 17.00 |
| **NB** | 83.08 | 84.62 | 78.46 | 76.92 | 75.38 | 75.38 | 75.38 | 75.38 | **NB** | 41.67 | 73.00 | 64.00 | 34.00 | 17.00 | 17.00 | 17.00 | 17.00 |
| **MLP** | 83.08 | 75.38 | 64.62 | 56.92 | 50.77 | 47.69 | 44.62 | 33.85 | **MLP** | 75.00 | 73.00 | 16.00 | 17.00 | 17.00 | 17.00 | 17.00 | 17.00 |
| Org. indicates Original; CRXX indicates Cost Ratio XX | | | | | | | | | | | | | | | | | |

Table 10.10: G-Mean1 Results of MetaCost Learners using ML techniques

| ML Tech. | Net | | | | | | | | ML Tech. | Log4j | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *Org.* | *CR5* | *CR10* | *CR15* | CR20 | *CR25* | *CR30* | *CR50* | | *Org.* | *CR5* | *CR10* | *CR15* | CR20 | *CR25* | *CR30* | *CR50* |
| **AB** | 0.72 | 0.69 | 0.62 | 0.60 | 0.59 | 0.58 | 0.58 | 0.58 | **AB** | 0.49 | 0.58 | 0.47 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **RF** | 0.66 | 0.70 | 0.68 | 0.66 | 0.65 | 0.64 | 0.64 | 0.66 | **RF** | 0.49 | 0.58 | 0.55 | 0.52 | 0.49 | 0.48 | 0.47 | 0.41 |
| **BG** | 0.70 | 0.67 | 0.62 | 0.61 | 0.58 | 0.38 | 0.38 | 0.0 | **BG** | 0.43 | 0.55 | 0.40 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **LB** | 0.71 | 0.68 | 0.65 | 0.61 | 0.61 | 0.57 | 0.57 | 0.19 | **LB** | 0.51 | 0.62 | 0.46 | 0.20 | 0.0 | 0.0 | 0.0 | 0.0 |
| **NB** | 0.54 | 0.67 | 0.73 | 0.70 | 0.70 | 0.67 | 0.67 | 0.66 | **NB** | 0.37 | 0.59 | 0.40 | 0.23 | 0.21 | 0.21 | 0.13 | 0.13 |
| **MLP** | 0.61 | 0.70 | 0.68 | 0.57 | 0.48 | 0.36 | 0.36 | 0.17 | **MLP** | 0.41 | 0.62 | 0.26 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **ML Tech.** | **IO** | | | | | | | | **ML Tech.** | **MMS** | | | | | | | |
| | *Org.* | *CR5* | *CR10* | *CR15* | CR20 | *CR25* | *CR30* | *CR50* | | *Org.* | *CR5* | *CR10* | *CR15* | CR20 | *CR25* | *CR30* | *CR50* |
| **AB** | 0.51 | 0.58 | 0.65 | 0.69 | 0.66 | 0.65 | 0.65 | 0.67 | **AB** | 0.63 | 0.63 | 0.91 | 0.66 | 0.65 | 0.65 | 0.64 | 0.59 |
| **RF** | 0.51 | 0.49 | 0.63 | 0.63 | 0.62 | 0.62 | 0.61 | 0.59 | **RF** | 0.58 | 0.58 | 0.69 | 0.67 | 0.93 | 0.66 | 0.66 | 0.64 |
| **BG** | 0.30 | 0.51 | 0.65 | 0.63 | 0.62 | 0.59 | 0.58 | 0.32 | **BG** | 0.61 | 0.61 | 0.68 | 0.64 | 0.63 | 0.65 | 0.64 | 0.48 |
| **LB** | 0.52 | 0.59 | 0.63 | 0.62 | 0.58 | 0.67 | 0.65 | 0.52 | **LB** | 0.58 | 0.58 | 0.68 | 0.62 | 0.59 | 0.58 | 0.57 | 0.52 |
| **NB** | 0.52 | 0.66 | 0.77 | 0.77 | 0.76 | 0.75 | 0.74 | 0.72 | **NB** | 0.70 | 0.64 | 0.70 | 0.70 | 0.69 | 0.70 | 0.70 | 0.72 |
| **MLP** | 0.52 | 0.51 | 0.72 | 0.72 | 0.73 | 0.72 | 0.67 | 0.61 | **MLP** | 0.62 | 0.62 | 0.68 | 0.65 | 0.65 | 0.65 | 0.66 | 0.65 |
| **ML Tech.** | **Bluetooth** | | | | | | | | **ML Tech.** | **Calendar** | | | | | | | |
| | *Org.* | *CR5* | *CR10* | *CR15* | CR20 | *CR25* | *CR30* | *CR50* | | *Org.* | *CR5* | *CR10* | *CR15* | CR20 | *CR25* | *CR30* | *CR50* |
| **AB** | 0.28 | 0.75 | 0.71 | 0.70 | 0.69 | 0.71 | 0.69 | 0.64 | **AB** | 0.42 | 0.61 | 0.11 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **RF** | 0.49 | 0.78 | 0.70 | 0.70 | 0.67 | 0.66 | 0.65 | 0.64 | **RF** | 0.24 | 0.55 | 0.42 | 0.41 | 0.39 | 0.35 | 0.33 | 0.30 |
| **BG** | 0.30 | 0.75 | 0.72 | 0.69 | 0.56 | 0.35 | 0.18 | 0.0 | **BG** | 0.41 | 0.49 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| **LB** | 0.49 | 0.77 | 0.72 | 0.67 | 0.67 | 0.61 | 0.58 | 0.55 | **LB** | 0.42 | 0.59 | 0.33 | 0.15 | 0.0 | 0.0 | 0.0 | 0.0 |
| **NB** | 0.58 | 0.80 | 0.76 | 0.75 | 0.74 | 0.74 | 0.78 | 0.81 | **NB** | 0.52 | 0.49 | 0.56 | 0.43 | 0.0 | 0.0 | 0.0 | 0.0 |
| **MLP** | 0.51 | 0.74 | 0.71 | 0.65 | 0.62 | 0.59 | 0.57 | 0.45 | **MLP** | 0.42 | 0.49 | 0.10 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| Org. indicates Original; CRXX indicates Cost Ratio XX | | | | | | | | | | | | | | | | | |

Table 10.11: Balance Results of MetaCost Learners using ML techniques

| ML Tech. | Net | | | | | | | | ML Tech. | Log4j | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *Org.* | *CR5* | *CR10* | *CR15* | CR20 | *CR25* | *CR30* | *CR50* | | *Org.* | *CR5* | *CR10* | *CR15* | CR20 | *CR25* | *CR30* | *CR50* |
| **AB** | 70.15 | 67.35 | 58.54 | 55.62 | 54.62 | 53.63 | 53.63 | 53.18 | **AB** | 46.61 | 57.63 | 46.26 | 29.19 | 29.29 | 29.29 | 29.29 | 29.29 |
| **RF** | 63.56 | 70.09 | 66.42 | 64.46 | 62.41 | 60.99 | 60.99 | 61.09 | **RF** | 47.57 | 58.13 | 54.83 | 51.95 | 49.18 | 48.40 | 47.00 | 42.33 |
| **BG** | 67.43 | 65.94 | 59.99 | 57.25 | 54.43 | 39.75 | 39.75 | 29.29 | **BG** | 42.89 | 55.05 | 41.53 | 29.29 | 29.29 | 29.29 | 29.29 | 29.29 |
| **LB** | 68.58 | 65.95 | 60.99 | 56.61 | 55.67 | 53.13 | 53.13 | 31.77 | **LB** | 47.85 | 61.42 | 45.74 | 32.04 | 29.29 | 29.29 | 29.29 | 29.29 |
| **NB** | 51.69 | 65.00 | 73.41 | 69.15 | 67.53 | 63.16 | 63.16 | 62.35 | **NB** | 39.15 | 58.37 | 41.63 | 33.14 | 32.52 | 32.35 | 30.49 | 30.52 |
| **MLP** | 57.63 | 69.07 | 63.47 | 53.55 | 46.10 | 38.25 | 38.25 | 31.28 | **MLP** | 41.67 | 62.27 | 34.24 | 29.29 | 29.29 | 29.29 | 29.29 | 29.29 |
| **ML Tech.** | **IO** | | | | | | | | **ML Tech.** | **MMS** | | | | | | | |
| | *Org.* | *CR5* | *CR10* | *CR15* | CR20 | *CR25* | *CR30* | *CR50* | | *Org.* | *CR5* | *CR10* | *CR15* | CR20 | *CR25* | *CR30* | *CR50* |
| **AB** | 48.53 | 54.74 | 61.02 | 66.56 | 64.75 | 64.42 | 64.24 | 66.61 | **AB** | 61.76 | 61.76 | 91.33 | 61.16 | 60.60 | 59.57 | 58.44 | 54.52 |
| **RF** | 48.47 | 48.07 | 60.60 | 60.24 | 60.04 | 59.71 | 59.08 | 58.22 | **RF** | 56.26 | 56.26 | 65.47 | 62.69 | 93.00 | 62.13 | 61.57 | 58.44 |
| **BG** | 35.72 | 48.47 | 61.13 | 60.60 | 60.04 | 58.37 | 58.33 | 37.00 | **BG** | 57.57 | 57.57 | 64.36 | 59.48 | 58.92 | 59.59 | 57.91 | 45.56 |
| **LB** | 48.56 | 54.86 | 60.60 | 59.71 | 57.57 | 66.87 | 64.75 | 52.30 | **LB** | 55.46 | 55.46 | 64.14 | 58.23 | 55.00 | 54.44 | 52.84 | 48.91 |
| **NB** | 48.55 | 64.59 | 76.82 | 76.49 | 75.47 | 74.43 | 73.39 | 71.25 | **NB** | 70.42 | 63.70 | 68.43 | 68.45 | 66.86 | 67.69 | 67.69 | 69.59 |
| **MLP** | 48.57 | 48.52 | 71.25 | 72.15 | 72.32 | 70.89 | 65.38 | 59.34 | **MLP** | 59.87 | 59.87 | 63.40 | 61.02 | 60.60 | 60.60 | 60.69 | 59.57 |
| **ML Tech.** | **Bluetooth** | | | | | | | | **ML Tech.** | **Calendar** | | | | | | | |
| | *Org.* | *CR5* | *CR10* | *CR15* | CR20 | *CR25* | *CR30* | *CR50* | | *Org.* | *CR5* | *CR10* | *CR15* | CR20 | *CR25* | *CR30* | *CR50* |
| **AB** | 35.24 | 75.12 | 70.55 | 69.55 | 68.52 | 69.64 | 67.25 | 62.38 | **AB** | 41.76 | 57.85 | 30.02 | 29.17 | 29.17 | 29.29 | 29.29 | 29.29 |
| **RF** | 47.98 | 77.40 | 69.08 | 68.45 | 66.04 | 64.83 | 63.61 | 62.38 | **RF** | 33.44 | 54.38 | 43.23 | 42.40 | 40.74 | 38.09 | 37.27 | 35.62 |
| **BG** | 35.70 | 75.12 | 70.83 | 67.25 | 54.92 | 38.41 | 31.61 | 29.29 | **BG** | 41.71 | 48.68 | 29.29 | 29.29 | 29.29 | 29.29 | 29.29 | 29.29 |
| **LB** | 47.98 | 76.52 | 72.47 | 66.41 | 66.41 | 60.87 | 57.41 | 53.66 | **LB** | 41.76 | 57.17 | 37.25 | 30.87 | 29.17 | 29.17 | 29.17 | 29.29 |
| **NB** | 54.70 | 78.65 | 75.93 | 75.12 | 74.28 | 74.28 | 77.61 | 79.33 | **NB** | 49.73 | 48.48 | 56.06 | 44.02 | 29.29 | 29.29 | 29.29 | 29.29 |
| **MLP** | 48.42 | 74.28 | 69.64 | 63.61 | 58.90 | 56.31 | 53.72 | 44.63 | **MLP** | 41.76 | 48.48 | 29.65 | 29.29 | 29.29 | 29.29 | 29.29 | 29.29 |

Org. indicates Original; CRXX indicates Cost Ratio XX

Table 10.12: AUC Results of MetaCost Learners using ML techniques

| ML Tech. | Net | | | | | | | | ML Tech. | Log4j | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *Org.* | *CR5* | *CR10* | *CR15* | CR20 | *CR25* | *CR30* | *CR50* | | *Org.* | *CR5* | *CR10* | *CR15* | CR20 | *CR25* | *CR30* | *CR50* |
| **AB** | 0.78 | 0.79 | 0.79 | 0.71 | 0.70 | 0.72 | 0.72 | 0.76 | **AB** | 0.66 | 0.62 | 0.65 | 0.50 | 0.50 | 0.50 | 0.50 | 0.50 |
| **RF** | 0.80 | 0.79 | 0.78 | 0.75 | 0.73 | 0.73 | 0.72 | 0.72 | **RF** | 0.61 | 0.58 | 0.53 | 0.52 | 0.51 | 0.51 | 0.57 | 0.49 |
| **BG** | 0.78 | 0.76 | 0.69 | 0.66 | 0.68 | 0.63 | 0.54 | 0.50 | **BG** | 0.68 | 0.60 | 0.60 | 0.50 | 0.50 | 0.50 | 0.50 | 0.50 |
| **LB** | 0.79 | 0.78 | 0.80 | 0.79 | 0.77 | 0.78 | 0.78 | 0.50 | **LB** | 0.68 | 0.65 | 0.66 | 0.60 | 0.52 | 0.52 | 0.52 | 0.52 |
| **NB** | 0.78 | 0.78 | 0.79 | 0.79 | 0.78 | 0.79 | 0.77 | 0.75 | **NB** | 0.62 | 0.61 | 0.54 | 0.60 | 0.60 | 0.60 | 0.61 | 0.62 |
| **MLP** | 0.77 | 0.76 | 0.74 | 0.75 | 0.75 | 0.53 | 0.51 | 0.30 | **MLP** | 0.67 | 0.70 | 0.52 | 0.36 | 0.31 | 0.30 | 0.29 | 0.29 |
| **ML Tech.** | **IO** | | | | | | | | **ML Tech.** | **MMS** | | | | | | | |
| | *Org.* | *CR5* | *CR10* | *CR15* | CR20 | *CR25* | *CR30* | *CR50* | | *Org.* | *CR5* | *CR10* | *CR15* | CR20 | *CR25* | *CR30* | *CR50* |
| **AB** | 0.66 | 0.63 | 0.68 | 0.71 | 0.75 | 0.75 | 0.77 | 0.76 | **AB** | 0.76 | 0.66 | 0.79 | 0.79 | 0.80 | 0.80 | 0.80 | 0.80 |
| **RF** | 0.67 | 0.64 | 0.68 | 0.70 | 0.68 | 0.67 | 0.68 | 0.69 | **RF** | 0.61 | 0.52 | 0.78 | 0.73 | 0.71 | 0.71 | 0.72 | 0.74 |
| **BG** | 0.67 | 0.71 | 0.71 | 0.66 | 0.70 | 0.70 | 0.58 | 0.51 | **BG** | 0.79 | 0.56 | 0.71 | 0.72 | 0.70 | 0.70 | 0.74 | 0.61 |
| **LB** | 0.70 | 0.68 | 0.74 | 0.73 | 0.75 | 0.73 | 0.74 | 0.70 | **LB** | 0.79 | 0.61 | 0.80 | 0.76 | 0.78 | 0.78 | 0.81 | 0.78 |
| **NB** | 0.72 | 0.76 | 0.77 | 0.77 | 0.77 | 0.77 | 0.76 | 0.74 | **NB** | 0.79 | 0.62 | 0.77 | 0.78 | 0.78 | 0.78 | 0.78 | 0.78 |
| **MLP** | 0.74 | 0.68 | 0.69 | 0.79 | 0.76 | 0.79 | 0.74 | 0.73 | **MLP** | 0.80 | 0.54 | 0.70 | 0.71 | 0.71 | 0.71 | 0.72 | 0.75 |
| **ML Tech.** | **Bluetooth** | | | | | | | | **ML Tech.** | **Calendar** | | | | | | | |
| | *Org.* | *CR5* | *CR10* | *CR15* | CR20 | *CR25* | *CR30* | *CR50* | | *Org.* | *CR5* | *CR10* | *CR15* | CR20 | *CR25* | *CR30* | *CR50* |
| **AB** | 0.66 | 0.76 | 0.70 | 0.74 | 0.72 | 0.73 | 0.72 | 0.69 | **AB** | 0.43 | 0.66 | 0.44 | 0.47 | 0.48 | 0.50 | 0.50 | 0.50 |
| **RF** | 0.70 | 0.78 | 0.76 | 0.80 | 0.79 | 0.77 | 0.77 | 0.76 | **RF** | 0.61 | 0.52 | 0.44 | 0.47 | 0.48 | 0.49 | 0.49 | 0.48 |
| **BG** | 0.77 | 0.77 | 0.78 | 0.73 | 0.63 | 0.48 | 0.50 | 0.50 | **BG** | 0.54 | 0.56 | 0.50 | 0.50 | 0.50 | 0.50 | 0.50 | 0.50 |
| **LB** | 0.67 | 0.78 | 0.73 | 0.76 | 0.75 | 0.66 | 0.69 | 0.69 | **LB** | 0.49 | 0.61 | 0.43 | 0.46 | 0.39 | 0.50 | 0.50 | 0.51 |
| **NB** | 0.85 | 0.87 | 0.86 | 0.87 | 0.86 | 0.86 | 0.86 | 0.87 | **NB** | 0.58 | 0.62 | 0.61 | 0.54 | 0.56 | 0.59 | 0.59 | 0.59 |
| **MLP** | 0.79 | 0.81 | 0.82 | 0.81 | 0.76 | 0.78 | 0.73 | 0.66 | **MLP** | 0.53 | 0.54 | 0.41 | 0.42 | 0.42 | 0.42 | 0.42 | 0.42 |
| Org. indicates Original; CRXX indicates Cost Ratio XX | | | | | | | | | | | | | | | | | |

The AUC results of MetaCost learners depicted in Table 10.12 exhibit that certain CR's showed an improvement of 2%, 5%, 23%, 5% and 5% respectively on each of Log4j, IO, MMS, Bluetooth and Calendar datasets.

The results were statistically assessed using Friedman test on G-Mean1, Balance and AUC values to ascertain the performance of different MetaCost learners as compared to the "original" technique. The Friedman test results using MetaCost learners were found significantly better that the "original" technique in three, five and six datasets each using AUC, Balance and G-Mean1 values respectively. As we found that the results using MetaCost learners were better that the "original" technique, in a majority of the cases, these techniques are favorable for learning through imbalanced data.

It was observed that each dataset gave the best results with a different CR. Also, the cost of the model for each CR (Appendix D.2: Table D.5) was considered while choosing the best CR for a specific dataset. A CR of 5 gave the best results on Net, Log4j, Bluetooth and Calendar datasets. A CR=10 was evaluated as the best one for the MMS dataset, while a CR=15 was the best one on IO dataset. Therefore, similar to the recommendation of Seliya and Khoshgoftaar [58], we advocate that developers should evaluate varied CR's in order to assess the best MetaCost learners on specific datasets.

***Answer to RQ2***

The analysis of the results indicate that MetaCost learners are effective in handling ILP. The statistical analysis signify improved results with the application of MetaCost learners as compared to the original technique. However, one may have to evaluate varied CR's in order to ascertain the best one for a specific dataset. The application of a MetaCost learner with an effective CR leads to improvement in the G-Mean1, Balance and AUC values of the developed change prediction model.

### 10.7.3 Results specific to RQ3

The results of RQ1 indicate the resampling with replacement as the best sampling method. We now assess the pairwise performance of the resample with replacement method with the best MetaCost learner for each dataset. In order to do so, Wilcoxon signed rank test is conducted using G-Mean1, Balance and AUC values of change prediction models at a cut-off of $\alpha = 0.05$. We conduct the test on each dataset individually to test hypothesis H6, H7 and H8.

Wilcoxon test results are reported in Table 10.13 with its p-value (shown in parenthesis). According to the results, the Resample with replacement method significantly outperformed the MetaCost learners in the majority of the cases (p-value $<$ 0.05). Only in MMS and IO datasets for the Balance values and in MMS dataset for the G-Mean1 values, the results are not significant. However, in these cases too, the resample method showed better results than the corresponding MetaCost learner. Thus, we accept the alternate hypothesis H6, H7 and H8. Therefore, the Resample with replacement method is significantly better than the MetaCost learners.

Table 10.13: Wilcoxon Test Results on Resample Method vs MetaCost Learners

| Dataset | G-Mean1 | Balance | AUC |
|---|---|---|---|
| Net | ↑* (0.046) | ↑* (0.028) | ↑* (0.027) |
| IO | ↑* (0.046) | ↑ (0.075) | ↑* (0.027) |
| Log4j | ↑* (0.028) | ↑* (0.046) | ↑* (0.028) |
| Bluetooth | ↑* (0.028) | ↑* (0.046) | ↑* (0.027) |
| Calendar | ↑* (0.046) | ↑* (0.046) | ↑* (0.046) |
| MMS | ↑ (0.075) | ↑ (0.075) | ↑* (0.027) |
| ↑* represents that the results are significantly better; ↑ represents results are better but not significantly | | | |

***Answer to RQ3***

On comparing resample with replacement (the best sampling method) with MetaCost learners, we found that the sampling method develops better change prediction models. These results were advocated by Wilcoxon signed rank test on G-Mean1,

Balance and AUC performance measures. The reason for better results using the resampling method could be the equal ratio of change-prone and not change-prone training instances provided by the method on each dataset. On the contrary, MetaCost learners always need to cost-sensitize the results with different CR's and achieve an optimum balance between recall and correct prediction of not change-prone classes. However, this balance may not be always as good as the equal ratio provided by the resample with replacement method.

## 10.8 Inter-Release Validation Results

This section discusses the results of inter-release validation models which are developed using imbalanced datasets. For each dataset, we first develop models using imbalanced training data of a specific version. Thereafter, these developed models are validated on another version of the same dataset. The details of specific versions of datasets for training and validation can be referred from Section 10.3.2.

In order to develop models, we remove outliers from training datasets. The percentage change reported in Table 10.1 is the one we obtain after conducting the outlier removal step. All the other steps are the same as the ones discussed in Section 10.4. The following subsections discuss the answers to the investigated RQs with respect to the results of inter-release validation models.

### 10.8.1 Results specific to RQ1

The accuracy results of inter-release validation models improved with the application of sampling methods in four of the investigated datasets. Though, the accuracy values declined in Bluetooth and Log4j datasets, this can be attributed to the biased nature of the accuracy measure, which makes it an ineffective evaluator for models

developed using imbalanced data [110, 118, 152]. We found that more number of change-prone classes were correctly predicted when sampling methods were used before model development. This trend was ascertained from the improvement in recall values. An analysis of precision values indicate a decrease in the obtained values with the application of sampling methods. However, when we analyzed both recall and precision collectively, we observed that in a majority of the cases, there was only a decrease of 10% in precision values but there was an increase of 30% in recall values in certain cases. Thus, the decrease in precision values is not much as compared to improvement of recall values. As discussed by Menzies et al. [119], predictive models should attain both high precision and recall values, however, in order to optimize one of these measures, the other may have to be compromised especially for imbalanced datasets with low number of positive (change-prone) instances. Also, a certain decrease in precision values means that few resources might get wasted when allocated to classes which are not change-prone but are predicted as change-prone but critical change-prone classes will not be missed by software practitioners as the models exhibit high recall values.

On the application of Friedman test on G-Mean1 values, we found that the results using inter-release validation showed positive improvement in all the cases, as the "No Sample" scenario obtained poor ranks than the other investigated sampling methods. These results were found significant on all the datasets except the Bluetooth dataset. Similar trends were observed when Friedman test was applied using Balance and AUC values. Though the evaluated Balance results were found better on all the six datasets, they were found statistically significant on only four datasets. An analysis of AUC results also indicated their improvement with the application of sampling methods. However, the results were found statistically significant on only two datasets using Friedman test. Thus, we advocate the use of sampling methods for providing balanced training data in order to develop effective change prediction

models using ML techniques. These models exhibit better G-Mean1, Balance and AUC values.

In order to assess the best sampling method, we analyzed all the eighteen cases of Friedman test (applied on six datasets each using G-Mean1, Balance and AUC values). The resample with replacement method obtained the best rank in seven out of these eighteen cases. Moreover, these ranks were significant in four cases. The inter-release validation results were found comparable to ten-fold cross validation results as Resample with replacement method is advocated as the best sampling method amongst the investigated methods. We also observed that the subsample method was designated as the best sampling method significantly in six cases. Therefore, we also promote its use as a sampling method.

### 10.8.2 Results specific to RQ2

Similar to the case of ten-fold cross validation models, we evaluated the inter-release validation results using seven CR's, which were 5, 10, 15, 20, 25, 30 and 50. Also, these results were compared with the "original" scenario when no cost-sensitive approach was applied. We observed an increase in the accuracy values of inter-release validation models in only three datasets (Net, IO and MMS), with the use of Meta-Cost learners. These trends were acceptable due to incompetency of accuracy values for assessing change prediction models developed using imbalanced datasets. Moreover, there was a drastic improvement in recall values in five datasets. The precision values depicted an improvement or were comparable in three datasets (Net, IO and Calendar), while there was a decrease in precision values in other datasets. As we discussed earlier, precision values may have to be compromised to achieve high recall values [119].

An analysis of G-Mean1, Balance and AUC values indicate an improvement with

the application of MetaCost learners as compared to the "original" scenario. A statistical analysis was conducted using Friedman test at $\alpha = 0.05$ on each dataset by using G-Mean1, Balance and AUC values. It was found that the G-Mean1 and Balance results using a specific CR outperformed the "original" scenario significantly in five datasets each. Similarly the AUC results using MetaCost learners were significantly better in three datasets. Thus, these results support the use of MetaCost learners for ILP.

Similar to ten-fold cross validation results, we found that the best results on a specific dataset was obtained by a different CR (Net: CR=25, Log4j: CR=5, IO: CR=5, Bluetooth: CR=10, MMS: CR=15 and Calendar: CR=5).

### 10.8.3 Results specific to RQ3

We compare the results of the best sampling method (resample with replacement) with the MetaCost learners using inter-release change prediction models. The comparison was done using Wilcoxon test at $\alpha = 0.05$ on the basis of G-Mean1, Balance and AUC values. The Wilcoxon test results using G-Mean1 values indicate significantly better results of the MetaCost learners on four datasets. However, in Calendar dataset the G-Mean1 values of resampling method were better than the MetaCost learner. However, these results were quite contradictory in the case of Balance values. In four out of six datasets, the Resample method performed better than the MetaCost learners, but not significantly.

On analyzing the Wilcoxon test results on AUC values, we found that the resample method outperformed the MetaCost learners on five datasets. However, the results were significant in only one dataset. The results of resample method and MetaCost learner were found equivalent on the Net dataset.

The above discussed results indicate that though the Resample with replacement

method is better than the MetaCost learners in a majority of the cases, the results are not significant. Thus, we conclude that the results of the inter-release change prediction models are comparable to that of the ten-fold cross validation models as the Resample with replacement method could not significantly outperform the MetaCost learners using G-Mean1, Balance and AUC values.

## 10.9 Discussion

The chapter developed and analyzed change prediction models using six imbalanced software datasets. In order to deal with ILP, we used three data sampling methods (Resample with replacement, Spread subsample and SMOTE) and MetaCost learners using seven different CRs. The change prediction models were developed using six ML techniques. The developed models were validated using two approaches: ten-fold cross validation and inter-release validation. The models were assessed using three unbiased and robust performance measures namely G-Mean1, Balance and AUC. Moreover, the trend of traditional measures like accuracy, recall and precision was also observed on models developed by using different techniques for imbalanced learning. The main contribution of the chapter is to advocate the use of different methods for handling imbalanced datasets as the use of such methods leads to vast improvement in the results of the change prediction models. We also conducted statistical analysis of the obtained results to strengthen our findings. The key findings of the chapter are reported as follows:

1. The performance of various ML techniques significantly improved after use of different sampling methods. Moreover, the resample with replacement method gave the best results when evaluated in terms of G-Mean1, Balance and AUC performance measures. However, the other two sampling methods i.e. SMOTE

and spread subsample also showed good results.

2. The use of MetaCost learners is yet another efficient way of handling ILP as they sensitize the ML technique by providing different costs for different kinds of misclassification errors so that the total overall cost of misclassification is decreased and the performance of the developed change prediction model improves. However, one should evaluate different CRs on a specific dataset to identify the best CR that works well on it.

3. A comparative performance of the best sampling method i.e. resample with replacement with the MetaCost learner having the most effective CR on all the datasets reveals that the sampling method significantly outperforms the MetaCost learners. The results were supported by both ten-fold cross validation models and inter-release validation models, when evaluated on the basis of G-Mean1, Balance and AUC performance measures. These results advocate that providing effective training data using resampling is a better approach for handling ILP rather than cost-sensitizing the learners.

Thus, the results of the chapter can be used by software practitioners and researchers to develop effective change prediction models when faced with imbalanced datasets. Moreover, such models can be developed using optimum costs.

# Chapter 11

# Analyzing Evolution-based Metrics Suite & the Evolution Patterns of Object-Oriented Metrics

## 11.1 Introduction

Evolution of a software is essential to keep it useful and functional. However, the quality of an evolving software may degrade due to improper incorporation of changes. In order to keep a track of software quality, one may assess the trends of OO design metrics during evolution as these metrics are effective indicators of the structure of classes, which are more prone to change in an OO software. Previous literature studies [4, 5, 27–29, 142] and previous chapters have proposed successful use of these design metrics to predict change-prone classes. Simply the prediction of change-prone parts does not lead to software quality improvement. The essence of analyzing metrics during the evolution of a software is to outline a systematic plan which strength-

ens its internal structure and prevents its degeneration. Furthermore, apart from OO metrics, which are a category of product metrics, other process metrics should also be evaluated for effective prediction of change-prone classes. This research gap was analyzed from the review conducted in Chapter 3. Thus, we ascertain the capability of evolution-based metrics suite, which quantifies the release by release history of changes in a software by developing software change prediction models. These metrics are representative of evolution characteristics of a class over all its previous releases and are important in order to understand the progression and change-prone nature of a class.

A recent study by Elish and Al-Khiaty [1] suggested the use of evolution-based metrics for prediction of change-prone classes. Analyzing only the previous release of a software, is not sufficient in order to understand how a system would evolve and which are the possible classes which are likely to change in the forthcoming releases of the software product [1]. As evolution-based metrics encompass evolution history of how the class has evolved over all the previous releases of the software, they represent a crucial and significant dimension. This dimension is not encapsulated by OO design metrics. Hence, it is important to analyze both OO metrics along with evolution-based metrics in order to develop competent change prediction models. Also, it should be noted that it is not always necessary that evaluating and analyzing different software dimensions such as change history and structural properties would lead to a vast difference in the performance of developed models. However, even a little improvement in the performance can effectively save a number of software resources by proper planning and allocation.

Therefore, the current chapter has two main objectives:

- Analyze the evolution trends of 7 popular OO metrics, which depict four primary characteristics of a class in an OO software i.e. its reusability, its depen-

dence on other classes, its cohesiveness and size.

- Evaluate the use of evolution-based metrics when used in conjunction with OO metrics for prediction of change-prone classes.

In order to empirically validate the results, the chapter uses two application packages of the Android software namely Contacts and Gallery2. The experiments are conducted over five versions (4.0.2-4.3.1) of the Android software.

With respect to the first objective, we examine the trends of CK metrics suite [16], along with the SLOC metric. The CK metrics suite was chosen as it was found to be the most popular metrics suite in literature for developing effective change prediction models [5, 30, 31, 33, 36, 257, 284]. The seven investigated metrics were categorized into four dimensions: size dimension (WMC and SLOC), cohesion dimension (LCOM), coupling dimension (CBO and RFC) and inheritance dimension (DIT and NOC). In order to examine the trends, we extracted the classes which were common to each of the five investigated versions (4.0.2-4.3.1) of the Android software. These common classes were then divided into two categories i.e. Changed Classes (CC) and Unchanged Classes (UCC) on the basis of whether a class had undergone change in any of the five investigated versions or not. The characteristics of both the categories of classes were examined to ascertain the generalized trends. Furthermore, the actually changed classes in each consecutive version were also observed on the basis of change in its metric values along each dimension. The change in metric values were divided into three categories viz. “Constant”, “Increasing” and “Decreasing”.

With respect to the second objective, the chapter explores the best possible metrics suite for prediction of change-prone classes and evaluates four possibilities: i) Evolution-based metrics used in conjunction with OO metrics, ii) Only evolution-based metrics iii) Only OO metrics and iv) Internal class probability of changes

(ICP) metric in a class, for developing change prediction models. Similar to the investigation of RQ1, the OO metrics used were CK metrics suite along with the SLOC metric. The evolution-based metrics suite consists of sixteen metrics proposed by Elish & Al-Khiaty [1]. We develop change prediction models using six ML techniques (MLP, NB, RF, AB, BG, LB) and the statistical technique LR. Various classification techniques were used so as to ascertain whether an effective metric suite differs with a different classification technique or the performance of the metric suite is consistent with different techniques. The performance of the models were analyzed using accuracy and AUC performance measures. In order to determine the best set of change-prone predictors, the comparative results of the change prediction models developed in the chapter were statistically evaluated using Wilcoxon signed rank test with Bonferroni correction. The RQs addressed in the chapter are:

- RQ1: What are the evolution trends of the investigated OO metrics with respect to (i) size, (ii) cohesion, (iii) coupling and (iv) inheritance dimensions?
- RQ2: Which evolution-based metrics are appropriate predictors of change-prone nature of a class?
- RQ3: What is the capability of the models developed using (i) combined metrics suite (evolution-based and OO) vs only evolution-based metrics, (ii) combined metrics suite vs only OO metrics and (iii) combined metrics suite vs the models developed using the ICP metric of a class?
- RQ4: Is the superiority of the combined metrics suite for determining change-prone nature of a class statistically significant as compared to other metrics suite?

RQ1 investigates the evolution trends of OO metrics with respect to four dimensions (size, cohesion, coupling and inheritance). RQ2 evaluates the evolution-based

metrics, which are significant predictors of change in a class. RQ3 investigates the comparative performance of combined metrics suite with three other scenarios i.e. models developed using only the evolution-based metrics, models developed using only the OO metrics and models developed using the ICP metric. RQ4 statistically compares the performance of the combined metrics suite with the change prediction models developed in the other three scenarios.

The organization of the chapter is as follows: Section 11.2 describes the empirical research framework, while section 11.3 states the experimental design. Section 11.4 states the results and answers of each RQ. A comparison of the chapter's results with previous studies is stated in section 11.5. The key findings of the chapter are included in Section 11.6. The results of the chapter are published as [290] and communicated as [291].

## 11.2 Empirical Research Framework

This section states the dependent and independent variables and the data collection.

### 11.2.1 Dependent and Independent Variables

In order to answer RQ1, we only analyze the trends of CK metrics suite [16] and the SLOC metric. However, in order to answer RQ2-RQ4, we need to develop change prediction models. For developing such models, the dependent variable used is change-proneness. The independent variables used for developing these models can be divided into three categories: i) evolution-based metrics suite; ii) OO metrics (CK metrics suite and the SLOC metric) and the iii) ICP metric. The definition of the CK metrics suite and the SLOC metric can be referred from Section 2.5.1 (Chapter 2). We define the evolution-based metrics suite and the ICP metric in the following

sections.

*Evolution-based Metrics Suite*: A brief explanation of the various evolution-based metrics used in the chapter is given in Table 11.1. They are explained in detail by Elish and Al-Khiaty [1].

Table 11.1: Evolution-based Metrics [1]

| Metric Name | Acronym | Definition |
|---|---|---|
| Birth Of a Class | BOC | This metric reports the release number when a specific class appears for the first time. |
| Total Amount of Changes | TACH | This metric reports the number of added, deleted and twice the number of changed lines between release v-1 and v of a software. |
| First Time Changes in a Class | FCH | This metric reports the release number when a specific class was first time subjected to changes. |
| Last Time Changes in a Class | LCH | This metric reports the release number when a class was subjected to most recent changes. |
| Change Occurred | CHO | This is a binary metric which has a value of 1 if the class was subject to changes from release v-1 to v and 0 otherwise. |
| Frequency of Changes | FRCH | This reports the number of releases in which a class has been subjected to changes. |
| Change Density | CHD | It is computed by dividing the total change size of a class (TACH) by the size of the class (SLOC). |
| Weighted Changes | WCH | It is a weighted metric using TACH, which gives more weight to recent changes. It is calculated as: $\sum_{p+1}^{k} TACH_v * 2^{v-k}$ where p=BOC and less than k; where k is the current version. |
| Weighted Change Density | WCD | It is a weighted metric using CHD. It is calculated as: $\sum_{p+1}^{k} CHD_v * 2^{v-k}$ where p=BOC and less than k; where k is the current version. |
| Weighted Frequency of Changes | WFR | It is a weighted metric using CHO. It is calculated as: $\sum_{2}^{k}(v-1)CHO_v$ where k is the current version. |
| Aggregated Change Size normalized by FRCH | ATAF | In case FRCH=0, ATAF is 0. Otherwise it is calculated as: $\frac{\sum_{2}^{k} TACH_v}{FRCH}$ where k is the current version. |
| Last Change Amount | LCA | It reports the size of changes (TACH) in a class, when last time it was subjected to changes. In case LCH is not equal to k or 0, this metric is equal to $TACH_v$, where r is the last time when the class was changed, otherwise it is $TACH_k$,the change size of the current version(k). |
| Last Change Density | LCD | It reports the change density (CHD) in a class, when last time it was subjected to changes. In case LCH is not equal to k or 0, this metric is equal to $CHD_v$, where r is the last time when the class was changed, otherwise it is $CHD_k$, the change density of the current version. |

| OO Metric | Acronym | Definition |
|---|---|---|
| Changes Since Birth | CSB | This metric reports the difference in size of the class between its current version and between the version when the class first appeared. |
| Changes Since Birth Normalized by Size | CSBS | This metric is computed by dividing CSB by the size of the class when it first appeared. |
| Aggregated Change Density Normalized by FRCH | ACDF | In case FRCH=0, ACDF=0. Otherwise it is calculated as: $\frac{\sum_2^k CHD_v}{FRCH}$ where k is the current version. |

*ICP Metric*: The ICP metric was proposed by Tsantalis et al. [32] and the metric attempts to encapsulate evolution history of a class as it is calculated by the dividing the total number of releases by the count of releases in which the class was present.

The evolution-based and ICP metrics are calculated for each class of the software release by release. The OO metrics suite is calculated with the help of CKJM tool.

### 11.2.2 Data Collection

The datasets used in the chapter were collected using DCRS tool [106]. We investigated five versions of two application packages (Contacts & Gallery2). In order to answer RQ1, the trends of various metrics were observed for only those common classes which were existent in all the versions from version 4.0.2 to version 4.3.1 (Gallery:184, Contacts:156). The number of common classes in each application package and the percentage of changed classes in consecutive versions of each package amongst the common classes are mentioned in Table 11.2. The table also states the actual number of common classes which changed while progressing from one version to another. For instance, 36 classes changed out of the common classes while progressing from version 4.0.2-4.0.4. It should be noted that the "actual changed classes" is a cumulative figure computed by analyzing both the application packages.

Table 11.2: Dataset Details

| Dataset | No. of Common Classes | Percentage Change | | | |
|---|---|---|---|---|---|
| | | 4.0.2-4.0.4 | 4.0.4-4.1.2 | 4.1.2-4.2.2 | 4.2.2-4.3.1 |
| Gallery2 | 184 | 15.2 | 35.3 | 34.8 | 27.7 |
| Contacts | 156 | 5.1 | 42.3 | 10.9 | 8.3 |
| Actual Changed Classes | | 36 | 131 | 81 | 64 |

In order to answer RQ2-RQ4, change prediction models were developed on the investigated datasets. For each class in a dataset, both OO metrics and evolution-based metrics are computed. The change-prone nature of a class is represented by the "ALTER" (Chapter 2: Section 2.7) variable. For a class in version v, the OO metrics are from version v and the evolution-based metrics are computed from all previous versions till version v. A data point consists of OO metrics, evolution-based metrics and the dependent variable "ALTER". For example, in order to prepare data for Contacts 4.2.2, evolution-based metrics are collected from all previous releases of Contacts namely (Contacts 4.0.2, 4.0.4 and 4.1.2), the OO metrics are from Contacts 4.2.2 and the ALTER variable is calculated by computing the change from current release i.e. Contacts 4.2.2 and next release i.e. Contacts 4.3.1 as shown in Figure 11.1.

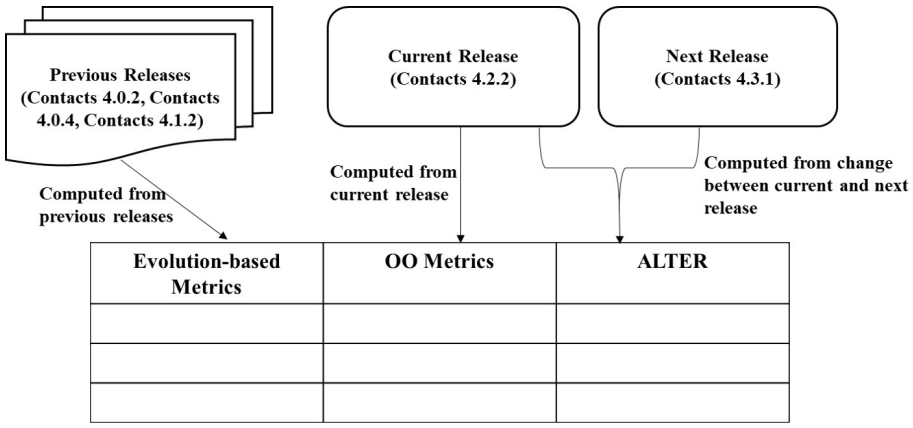


Figure 11.1: Data Collection for Contacts 4.2.2: An Example

The first version of each dataset will not have any evolution-based metrics since

there is no change history before the first version [1]. Thus, change prediction models cannot be developed for Contacts 4.0.2 and Gallery2 4.0.2. The details of the investigated datasets can be referred form Appendix A.1.

## 11.3 Experimental Design

This section states the comparison of experimental design with a study performed by Elish & Al-Khiaty [1]. We then state the hypothesis investigated in the chapter. We also state the feature selection method, performance measures and the classification techniques used for developing change prediction models. In order to answer RQ2-RQ4, we have developed four models for each release of the software: a) Model I which uses both evolution-based and OO metrics as predictors b) Model II which uses only evolution-based metrics as predictors c) Model III which uses only OO metrics as predictors and d) Model IV which uses the ICP metric as the only independent variable.

### 11.3.1 Experimental Design Comparison with Elish & Al-Khiaty

With respect to RQ2-RQ4, the chapter is closely related to a study by Elish and Al-Khiaty [1]. However, it is different and better in the following parameters:

- Elish and Al-Khiaty [1] only developed statistical prediction models, but this chapter explores six other ML techniques.
- They used classification accuracy as a performance measure for change prediction models but studies in literature have criticized the use of accuracy [37, 115, 284]. However, AUC gives a realistic estimate of the developed models, which is used for performance comparison in this chapter.

- Elish and Al-Khiaty [1] used only two open-source software (VSSPlugin and PeerSim). However, this chapter empirically validates two application packages of Android software. Thus, the chapter's results are highly generalizable to similar domains which also strengthen its external validity.

### 11.3.2 Hypothesis Investigated

We investigated the following hypothesis both for RQ1 and RQ4.
*Hypothesis for RQ1*: For each of the four dimensions (size, cohesion, coupling and inheritance), we formulate a set of hypothesis, which are assessed by analyzing the trends of OO metrics. The hypothesis for RQ1 is as follows:

- *H1 (WMC)*: Due to evolution in a software, class size in terms of the number of methods in a class will increase.
- *H2 (SLOC)*: Due to evolution in a software, class size in terms of SLOC will increase.
- *H3 (LCOM)*: Due to evolution in a software, cohesion in terms of LCOM values will decrease. As the software evolves, classes should be designed in a manner to increase their cohesiveness. A lower value of LCOM indicates better cohesiveness.
- *H4 (CBO)*: Due to evolution in a software, coupling attribute in terms of CBO values will decrease as a class with better design will have low export and import coupling.
- *H5 (RFC)*: Due to evolution in a software, coupling attribute in terms of RFC will decrease as a better designed class will have a lower response set in terms of the number of methods.

- *H6 (DIT)*: Due to evolution in a software, the inheritance attribute in terms of DIT value will increase as more classes will be added in the hierarchy.

- *H7 (NOC)*: Due to evolution in a software, the inheritance attribute in terms of NOC value will increase as more classes should be derived from the previously existent ones.

*Hypothesis for RQ4*: In RQ4, we statistically compare the pairwise performance of Model I with the other three (Model II, Model III, Model IV) developed models, using Wilcoxon test with Bonferroni correction.
*Hypothesis based on AUC Values*:

- *H8/ H9/ H10 (AUC)*: The results obtained by change prediction models in terms of AUC values developed using evolution-based metrics in conjunction with OO metrics differs significantly when compared with the performance of models developed using only evolution-based metrics/ only OO metrics/ only the ICP metric.

*Hypothesis based on Accuracy Values*:

- *H11 / H12 / H13 (Accuracy)*: The results obtained by change prediction models in terms of Accuracy values developed using evolution-based metrics in conjunction with OO metrics differs significantly when compared with the performance of models developed using only evolution-based metrics/ only OO metrics/ only the ICP metric.

### 11.3.3 Feature Selection & Performance Measures

CFS is used as a feature selection technique to develop competent change prediction models [109]. The change prediction models were developed using ten-fold cross

validation technique [113] and were evaluated using AUC as well as Accuracy performance measures. The models were developed using six ML techniques and the statistical technique LR. A brief description of the techniques and the parameters employed can be referred from Chapter 2 (Section 2.6).

## 11.4 Analysis and Results

The current section states the results of the conducted experiments and answers the RQs explored in the chapter.

### 11.4.1 Results specific to RQ1

The trends of metrics were analyzed for all the four dimensions for both the investigated application packages of the chapter. We now state the trends corresponding to each metric dimension and evaluate the hypothesis mentioned in section 11.3.2. As discussed in Section 11.1, the common classes were categorized into CC and UCC. Also, those classes which actually changed between two specific consecutive versions were identified as having "Constant", "Increasing" or "Decreasing" trends in consecutive versions. A class is categorized as having a "Constant" trend if the value of a specific metric does not change for the class while progressing from version A to consecutive version B. A class trend is termed as "Increasing" if the value of a specific metric increased for the class while progressing from version A to consecutive version B. Similarly, a "Decreasing" trend represents the decrease in metric value as compared to the metric value observed in the version A than the metric value when it was transferred to consecutive version B.

*Trends of Size Metrics*

Figure 11.2 depicts the mean values of WMC and SLOC metrics (size dimension)

over all the five versions (4.0.2-4.3.1) for CC and UCC. According to the figure, it may be noted that the CC have higher mean values than UCC. For instance, in Gallery2, the mean WMC values of CC (17) is greater than that of UCC (6). The median values were also analyzed, which depicted a similar trend (Appendix E.1: Table E.1). An analysis of SLOC metric values in Figure 11.2 indicates a difference of 56%-63% in the mean SLOC values of CC and UCC.

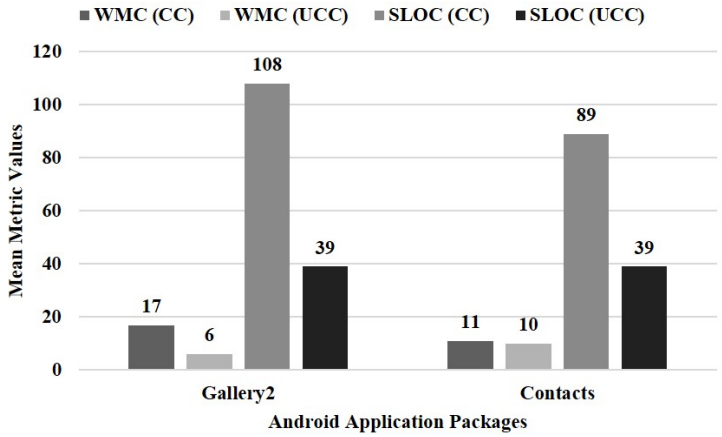


Figure 11.2: Mean values of Size Metrics

We also observe the trend frequency of the two observed size metrics for all the actual changed classes in consecutive versions. The number of classes depicting "constant" (Const.), "increasing" (Inc.) and "decreasing" (Dec.) trends is depicted in Table 11.3. A prominent trend for the SLOC metric was increase in class size as 53%, 48% and 65% of classes respectively in 4.0.2-4.0.4, 4.0.4-4.1.2 and 4.1.2-4.2.2 consecutive versions showed an "increasing" trend. The next popular trend for SLOC metric was "constant" followed by the "decreasing" trend. An increase in class size, may not always lead to increase in number of class methods (WMC). Thus, most classes depicted a "constant" trend for WMC in versions 4.0.4-4.1.2 and 4.2.2-4.3.1, followed by the "increasing trend". However, the other two analyzed consecutive versions depicted an "increasing" trend for WMC metric. Thus, according to the

above discussion, hypothesis H1 and H2 are accepted.

Table 11.3: Version specific Size metric trends

| Size Metrics | 4.0.2-4.0.4 | | | 4.0.4-4.1.2 | | | 4.1.2-4.2.2 | | | 4.2.2-4.3.1 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Const. | Inc. | Dec. | Const. | Inc. | Dec. | Const. | Inc. | Dec. | Const. | Inc. | Dec. |
| WMC | 18 | 16 | 2 | 51 | 49 | 31 | 24 | 47 | 10 | 41 | 15 | 8 |
| SLOC | 13 | 19 | 4 | 33 | 63 | 35 | 18 | 53 | 10 | 31 | 19 | 14 |

*Trends of Cohesion Metrics*

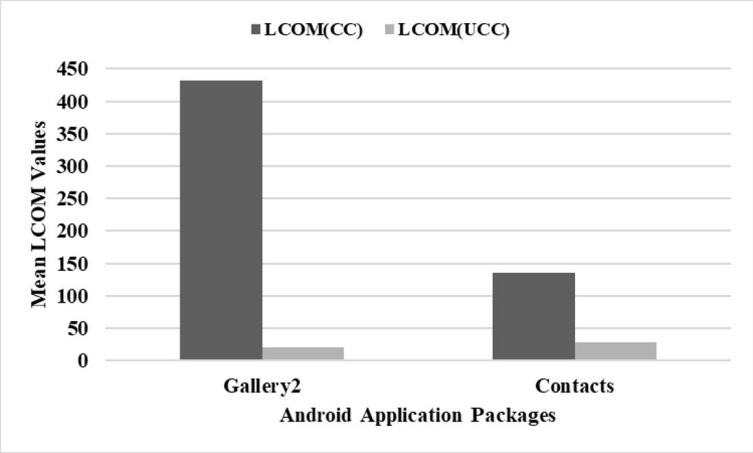


Figure 11.3: Mean values of Cohesion Metrics

In order to analyze the cohesive characteristics of the classes, the mean (Figure 11.3), median, minimum and maximum values of LCOM was examined over all the two versions of the application packages. A lower value for LCOM is desired. It was observed from Figure 11.3 that the UCC exhibited better cohesiveness as compared to the CC using LCOM values. As shown in the figure, for Contacts application package, the mean value of LCOM for UCC was 29. However, the mean LCOM value for CC was found to be 136. The observation of median values also depicted a large difference between LCOM value of CC and UCC (Appendix E.1).

The version specific trends of LCOM cohesion metric is depicted in Table 11.4. It may be noted that a change in class may not always lead to a change in cohesion met-

ric values. Thus, “constant” trend is observed in a large number of classes. However, we analyze the “increasing” and “decreasing” trends to evaluate the hypothesis and observe how cohesion metrics change with evolution. A prominent trend in cohesion metrics was increase in LCOM values (23-58%), in various consecutive versions of the investigated application packages. Thus, hypothesis H3 is rejected.

Table 11.4: Version specific Cohesion Metric trends

| Metric | 4.0.2-4.0.4 | | | 4.0.4-4.1.2 | | | 4.1.2-4.2.2 | | | 4.2.2-4.3.1 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Const. | Inc. | Dec. | Const. | Inc. | Dec. | Const. | Inc. | Dec. | Const. | Inc. | Dec. |
| LCOM | 18 | 16 | 2 | 51 | 49 | 31 | 24 | 47 | 10 | 41 | 15 | 8 |

*Trends of Coupling Metrics*

The coupling dimension was categorized by two metrics viz. CBO and RFC. Figure 11.4 represents the mean values of coupling metrics over all the investigated versions (4.0.2-4.3.1) of the Android software. We also analyzed the median values, the minimum and the maximum values of the coupling metrics obtained by classes over all the five versions for both CC and UCC. After analyzing Figure 11.4, it was observed that the mean values of coupling metrics are higher for CC as compared to UCC in most of the cases. For instance, the mean CBO values of CC in Gallery2 and Contacts was double the mean CBO values of UCC in these application packages. Similarly, the median CBO values of CC were also greater than UCC in both the application packages (Appendix E.1: Table E.3). A similar trend was shown by the mean values of RFC coupling metric. This indicates that a class with higher coupling values is prone to change in future versions.

Table 11.5: Version specific Coupling metric trends

| Metrics | 4.0.2-4.0.4 | | | 4.0.4-4.1.2 | | | 4.1.2-4.2.2 | | | 4.2.2-4.3.1 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Const. | Inc. | Dec. | Const. | Inc. | Dec. | Const. | Inc. | Dec. | Const. | Inc. | Dec. |
| CBO | 33 | 3 | 0 | 98 | 21 | 12 | 53 | 22 | 6 | 47 | 3 | 14 |
| RFC | 18 | 16 | 2 | 51 | 49 | 31 | 24 | 47 | 10 | 41 | 15 | 8 |

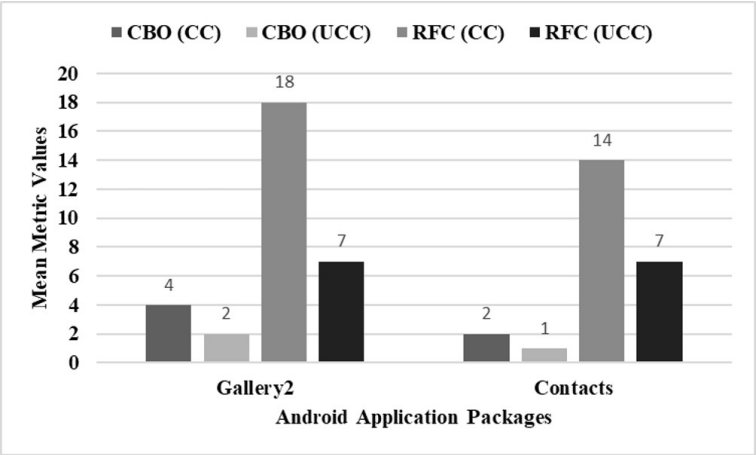


Figure 11.4: Mean values of Coupling Metrics

Table 11.5 states the trends of coupling metrics. According to the table, the majority (65-92%) of CBO metric values exhibited a "constant" trend. However, for the RFC metric apart from the "constant" trend, the "increasing" trend was quite popular (23-58%) in all the version specific trends. It may be noted that changes in a class may not always lead to a change in RFC values. According to the discussed trends, hypothesis H4 and H5 are rejected.

*Trends of Inheritance Metrics*

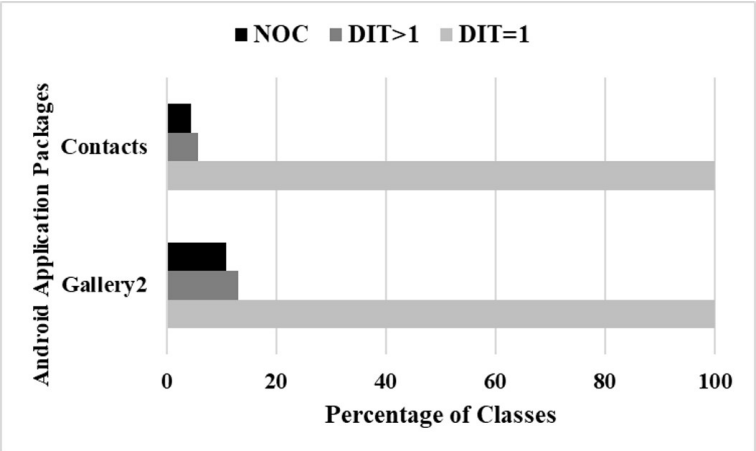


Figure 11.5: Classes exhibiting inheritance attributes

The inheritance attribute of classes was characterized by two metrics DIT and NOC. Figure 11.5 depicts the percentage of classes which used any of the inheritance attribute (i.e. have a non-zero value) in any of the two investigated versions. According to the figure, all the classes used the DIT attribute in corresponding application packages as all classes are descendant of the object class (Java language). Therefore, we analyzed the percentage of classes with DIT value greater than 1, to understand its actual usage. Figure 11.5 shows that 6% and 13% of classes exhibited a DIT value of greater than 1 in Contacts and Gallery2 package respectively. The NOC metric was rarely used by classes as very few classes depicted a non-zero value (Gallery2:4%; Contacts:13%).

We also observed the minimum and maximum values obtained by the DIT and NOC in both the datasets. It was found that the maximum values of inheritance metrics are higher for CC as compared to UCC. The mean and median values of inheritance metrics did not give much information as in the majority of the cases the mean and median values of NOC was 0 and that of DIT was 1.

We analyzed the number of classes which depicted a non-zero NOC value and a DIT value $>1$ for each application package for CC and UCC. The CC exhibited greater NOC and DIT values than UCC. Also, most of the classes were found having either one or two children. There were very few cases with classes having three or more children. Similarly, there were certain number of classes having a DIT value of 1 or 2, but very few classes with a DIT value of three or more.

Table 11.6: Version specific Inheritance metric trends

| Metrics | 4.0.2-4.0.4 | | | 4.0.4-4.1.2 | | | 4.1.2-4.2.2 | | | 4.2.2-4.3.1 | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Const. | Inc. | Dec. | Const. | Inc. | Dec. | Const. | Inc. | Dec. | Const. | Inc. | Dec. |
| DIT | 36 | 0 | 0 | 131 | 0 | 0 | 81 | 0 | 0 | 64 | 0 | 0 |
| NOC | 36 | 0 | 0 | 126 | 3 | 2 | 78 | 3 | 0 | 59 | 1 | 4 |

According to the trends observed by actual changed classes (Table 11.6) in consecutive versions, majority (92-100%) of them did not exhibit any change in inheri-

tance metric values and were “constant”. Only very few classes (2-6%) depicted an “increasing” or “decreasing”. Thus, we reject hypothesis H6 and H7.

### 11.4.2 Results specific to RQ2

In order to ascertain evolution-based metrics, which are significant predictors of change-prone nature of a class, we apply the CFS technique while developing Model I and Model II. However, we also use the CFS technique while developing Model III, whose results also stated in the section.

Table 11.7: CFS Results for Contacts Dataset

| Release Number | Model | Selected Metrics |
|---|---|---|
| Contacts 4.0.4 | Model I | WMC, CBO, SLOC, TACH, CSB, CSBS |
| | Model II | TACH, CSB, CSBS |
| | Model III | SLOC |
| Contacts 4.1.2 | Model I | WMC, CBO, SLOC, TACH, ACDF |
| | Model II | TACH, LCH, CHD, CSB, ACDF |
| | Model III | WMC, CBO, SLOC |
| Contacts 4.2.2 | Model I | SLOC, WCD, ATAF |
| | Model II | WCD, ATAF |
| | Model III | SLOC |
| Contacts 4.3.1 | Model I | DIT, CBO, SLOC, BOC, TACH |
| | Model II | BOC, TACH |
| | Model III | DIT, CBO, SLOC |

Table 11.8: CFS Results for Gallery2 Dataset

| Release Number | Model | Selected Metrics |
|---|---|---|
| Gallery2 4.0.4 | Model I | WMC, CBO, RFC, SLOC, CHD, WCD, CSBS, ACDF |
| | Model II | CHD, WCD, LCD, CSBS |
| | Model III | WMC, CBO, SLOC |
| Gallery2 4.1.2 | Model I | WMC, CBO, SLOC, WCD, CSB, CSBS |
| | Model II | CHD, WCD, CSB, CSBS |
| | Model III | WMC, CBO, SLOC |
| Gallery2 4.2.2 | Model I | WMC, SLOC, TACH, LCA, LCD, CSB, CSBS |
| | Model II | TACH, LCA, LCD, CSB, CSBS |
| | Model III | WMC, SLOC |
| Gallery2 4.3.1 | Model I | WCD |
| | Model II | WCD |
| | Model III | WMC |

Table 11.7 and 11.8 depict the CFS results on Contacts and Gallery2 package for

all the versions and for Model I, Model II and Model III. The results in the tables depict TACH and CSBS as the most commonly used evolution-based metrics since they were selected the most number of times. Other commonly used evolution-based metrics were CSB and WCD. CHD, ATAF, LCA, LCD and ACDF were also used in certain scenarios. The most commonly used OO metrics were SLOC, WMC and CBO.

Thus, in accordance with CFS results TACH and CSBS metrics were useful predictors of change as they are selected in 8 out of 16 times for the development of change prediction models. The CSB and WCD metrics are also used by several models for predicting change. Thus, these four evolution-based metrics are appropriate for determining change in a class.

### 11.4.3 Results specific to RQ3

In order to answer this question, we first develop models using ten-fold cross validation technique. As discussed earlier, the models were evaluated using AUC as well as accuracy (Acc.) performance measures. It may be noted that we do not develop models for Gallery2 4.3.1, as in all the cases (Table 11.8), the CFS results yielded only one metric. Thus, these models are not very realistic as they are all dependent on only one metric.

Tables 11.9-11.11 show the AUC and accuracy (in percentage) values of the developed models on each release of Contacts and Gallery2 respectively using the seven classification techniques. The best model in a particular scenario is highlighted in bold whether it is Model I, Model II, Model III, or Model IV in terms of both AUC and accuracy performance measures. It can be seen from the tables that Model I is better in a majority of the cases in terms of AUC values. However, according to accuracy values, both Model I and Model III show good results. As previously

Table 11.9: AUC & Accuracy Results for Contacts Dataset

| Tech. | Contacts 4.0.4 | | | | | | | | Contacts 4.1.2 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Model I | | Model II | | Model III | | Model IV | | Model I | | Model II | | Model III | | Model IV | |
| | AUC | Acc. | AUC | Acc. | AUC | Acc. | AUC | Acc. | AUC | Acc. | AUC | Acc. | AUC | Acc. | AUC | Acc. |
| LR | **0.75** | **71.6** | 0.62 | 66.9 | 0.74 | 70.0 | 0.54 | 67.5 | 0.79 | **91.4** | 0.73 | **91.4** | **0.80** | 90.7 | 0.70 | 89.8 |
| MLP | 0.71 | 69.4 | 0.63 | 68.8 | **0.72** | **69.1** | 0.55 | 67.5 | 0.74 | **91.7** | 0.75 | **91.7** | **0.77** | 90.1 | 0.73 | 90.7 |
| NB | **0.71** | 69.1 | 0.61 | 65.6 | **0.71** | **69.7** | 0.54 | 67.5 | 0.75 | 86.4 | 0.72 | 88.6 | 0.77 | 87.7 | 0.70 | **90.7** |
| RF | **0.67** | 67.5 | 0.62 | **70.7** | 0.63 | 62.5 | 0.54 | 67.5 | **0.71** | 90.1 | 0.64 | 88.9 | 0.68 | 89.8 | 0.70 | **90.7** |
| AB | **0.74** | 70.3 | 0.61 | **71.0** | 0.71 | 68.1 | 0.54 | 67.5 | **0.77** | 89.8 | 0.66 | 90.7 | 0.74 | 90.4 | 0.69 | **90.7** |
| BG | **0.74** | 70.7 | 0.63 | **71.9** | 0.71 | 66.6 | 0.55 | 67.5 | 0.75 | **91.0** | 0.71 | 70.7 | **0.76** | 90.4 | 0.66 | **90.7** |
| LB | **0.72** | 70.3 | 0.61 | **71.0** | 0.70 | 66.2 | 0.54 | 67.5 | **0.80** | 89.2 | 0.69 | 90.1 | 0.75 | **91.0** | 0.70 | 90.7 |
| **Tech.** | **Contacts 4.2.2** | | | | | | | | **Contacts 4.3.1** | | | | | | | |
| LR | 0.75 | 95.6 | **0.77** | 95.9 | 0.73 | 96.0 | 0.73 | **96.3** | **0.81** | **74.3** | 0.61 | 64.9 | 0.79 | 73.8 | 0.54 | 57.4 |
| MLP | 0.73 | 95.9 | 0.73 | 95.9 | 0.71 | **96.3** | **0.77** | **96.3** | **0.82** | 71.3 | 0.64 | 65.3 | 0.80 | **73.8** | 0.59 | 55.4 |
| NB | 0.72 | 90.3 | 0.70 | 93.3 | **0.73** | 94.7 | 0.72 | **95.2** | **0.77** | **72.8** | 0.65 | 67.8 | 0.74 | 71.8 | 0.55 | 55.0 |
| RF | 0.70 | 95.0 | 0.62 | 96.2 | 0.57 | 93.9 | **0.72** | **96.3** | **0.83** | **75.7** | 0.61 | 68.3 | 0.78 | 74.8 | 0.51 | 58.4 |
| AB | **0.73** | 95.9 | 0.67 | 95.9 | 0.62 | **96.3** | 0.72 | **96.3** | **0.80** | **75.2** | 0.64 | 68.3 | 0.75 | 72.3 | 0.57 | 58.4 |
| BG | **0.71** | 95.9 | 0.70 | 95.9 | 0.49 | **96.3** | 0.49 | **96.3** | **0.82** | **76.2** | 0.65 | 67.3 | 0.80 | 74.3 | 0.53 | 55.9 |
| LB | **0.77** | 95.9 | 0.70 | 95.3 | 0.64 | 96.0 | 0.71 | **96.3** | **0.82** | **95.2** | 0.60 | 68.3 | 0.81 | 75.7 | 0.51 | 58.4 |

Table 11.10: AUC & Accuracy Results for Gallery2 Dataset (4.0.4 & 4.1.2)

| Tech. | Gallery2 4.0.4 | | | | | | | | Gallery2 4.1.2 | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Model I | | Model II | | Model III | | Model IV | | Model I | | Model II | | Model III | | Model IV | |
| | AUC | Acc. | AUC | Acc. | AUC | Acc. | AUC | Acc. | AUC | Acc. | AUC | Acc. | AUC | Acc. | AUC | Acc. |
| LR | **0.76** | 77.6 | 0.61 | 77.6 | 0.74 | 76.0 | 0.57 | **78.3** | **0.75** | **72.1** | 0.73 | 71.8 | 0.51 | 49.8 | 0.57 | 57.8 |
| MLP | **0.74** | 77.0 | 0.65 | 76.6 | 0.74 | **78.3** | 0.63 | **78.3** | **0.73** | 71.1 | **0.73** | **71.4** | 0.52 | 53.0 | 0.59 | 57.8 |
| NB | **0.73** | 77.0 | 0.62 | 78.0 | 0.69 | 75.0 | 0.57 | **78.3** | 0.70 | 64.5 | **0.76** | **65.5** | 0.38 | 47.4 | 0.57 | 57.8 |
| RF | **0.75** | 73.7 | 0.62 | 73.4 | **0.75** | 76.3 | 0.58 | **78.3** | **0.79** | 71.1 | 0.78 | **75.3** | 0.78 | 72.5 | 0.57 | 70.0 |
| AB | **0.74** | **79.9** | 0.64 | 77.6 | 0.72 | 78.0 | 0.57 | 78.3 | **0.81** | **72.8** | 0.74 | 72.5 | 0.76 | 71.4 | 0.56 | 57.8 |
| BG | **0.71** | 78.6 | 0.63 | 74.0 | 0.71 | **78.9** | 0.59 | 76.6 | 0.77 | 72.5 | 0.78 | **73.9** | **0.80** | 73.5 | 0.58 | 56.8 |
| LB | **0.77** | **80.9** | 0.60 | 77.6 | 0.72 | 78.9 | 0.57 | 78.3 | **0.81** | 71.4 | 0.77 | **75.3** | 0.77 | 72.8 | 0.56 | 57.8 |

Table 11.11: AUC & Accuracy Results for Gallery2 Dataset (4.2.2)

| Tech. | Gallery2 4.2.2 | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | Model I | | Model II | | Model III | | Model IV | |
| | AUC | Acc. | AUC | Acc. | AUC | Acc. | AUC | Acc. |
| LR | **0.70** | 68.1 | 0.70 | 70.2 | 0.67 | 70.2 | 0.69 | **73.2** |
| MLP | 0.65 | 70.2 | 0.66 | 71.5 | **0.71** | 70.2 | 0.69 | **72.3** |
| NB | 0.63 | 69.8 | 0.59 | 69.8 | 0.62 | 67.7 | **0.69** | **72.3** |
| RF | **0.73** | 71.9 | 0.73 | 71.9 | 0.63 | 65.1 | 0.67 | **73.2** |
| AB | **0.75** | 71.1 | 0.74 | 70.6 | 0.66 | 68.5 | 0.69 | **73.2** |
| BG | 0.74 | **74.9** | **0.73** | 73.6 | 0.68 | 71.1 | 0.69 | 71.9 |
| LB | **0.76** | 71.1 | 0.72 | **75.3** | 0.66 | 66.4 | 0.67 | 73.2 |

discussed, accuracy is criticized as a performance measure [37, 118, 284], thus we base our results primarily on AUC values.

According to AUC values shown in Tables 11.9-11.11, Model I, which is developed using evolution-based metrics in conjunction with OO metrics is better than models developed using only OO metrics (Model III) in most of the cases. However, models developed using only evolution-based metrics (Model II) are not that efficient as they are not able to incorporate OO characteristics of a software. Also, the models developed using the ICP metric only incorporate the change probability of a class and do not quantify OO characteristics of the software. Majority of Model I's developed on different releases of Contacts and Gallery2 dataset have obtained AUC values greater than 0.7 and accuracy values in the range of 70%-95%, which shows that these models are practical and can be successfully used by the industry.

We also analyzed the average ten-fold cross validation results for different models (Model I, Model II, Model III & Model IV) developed in the chapter on all the releases of each dataset.

Figures 11.6(a) and 11.6(b) depict the average AUC values obtained by models developed using different classification techniques on all releases of Contacts and Gallery2 datasets respectively. According to the figures, it can be observed that in most of the cases the models developed using only OO metrics (Model III) shows better results than Model II (only evolution-based metrics) and Model IV (only ICP

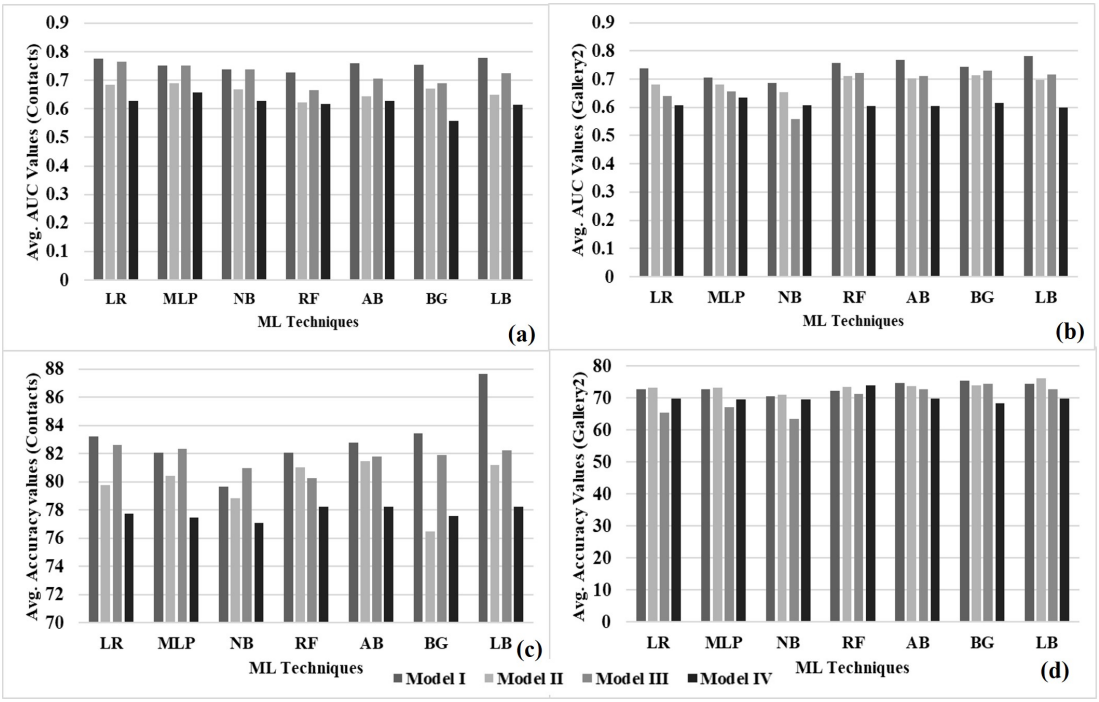


Figure 11.6: Average AUC values on (a) Contacts Dataset (b) Gallery2 Dataset for all Techniques and Average Accuracy values on (c)Contacts Dataset (d) Gallery2 Dataset for all Techniques

metric). Models based only on OO metrics have been successfully used in literature and have been recommended by various researchers for predicting change-prone classes [4, 5, 28–31, 36, 147]. However, it can be clearly seen that the average AUC values obtained by Model I is higher than Model II, Model III and Model IV respectively. On an average, there was an improvement of 6% and 3% respectively on Gallery2 and Contacts dataset in the AUC values of models developed using the combined metrics suite (Model I) as compared to the traditional OO metrics model (Model III), which have been advocated in literature. This observation establishes the superiority of the models developed using conjunction of evolution-based and OO metrics from all other scenarios. This is because such a model incorporates both the OO characteristics of a software as well as evolution history of a software and is

therefore highly capable in predicting change.

Figures 11.6(c) and 11.6(d) show the average accuracy values attained by change prediction models developed using different classification techniques. It is observed from Figure 11.6(c) that the average accuracy values of Model I are much higher than the average accuracy values of Model II, Model III and Model IV on Contacts dataset. Also, the average accuracy of Model III is higher than Model II and Model IV. However, the average accuracy values of Model I were comparable to that of Model II on Gallery2 dataset (Figure 11.6(d)). Thus, on Gallery2 dataset, models developed using only evolution-based metrics (Model II) also showed promising results. This observation points out the capability of evolution-based metrics in predicting software change.

Thus, overall we can deduce that Model I i.e. the models developed using both OO metrics and evolution-based metrics depict better average AUC and accuracy values when used for ascertaining software change. These models, which use both evolution-based and OO metrics suite are powerful indicators and increase the capability of models which are only based on OO metrics or only based on evolution-based metrics. Thus, these models are useful and can be used for efficient determination of change-prone classes by software practitioners.

### 11.4.4 Results specific to RQ4

This question statistically compares the pairwise performance of Model I with the other three (Model II, Model III, Model IV) developed models, using Wilcoxon signed rank test with Bonferroni correction at a significance value of 0.05. It was validated using average AUC values and average accuracy values obtained on all the releases by the models developed using each specific technique for the two investigated datasets. The test evaluates the hypothesis stated in Section 11.3.2.

Table 11.12 shows the Wilcoxon test results on average AUC values and average accuracy values with p-value denoted in parenthesis. According to the results, Model I is found significantly superior to the other developed models. Moreover, Model I is significantly better than Models II, III, IV. Thus, we reject Null Hypothesis (H8, H9 and H10).

Similarly, we also evaluate hypothesis H11, H12 and H13 stated in section 11.3.2 on the basis of average accuracy values. According to the results in Table 11.12, Model I is superior to the Model III and Model IV significantly. Moreover, Model I is superior to Model II too, but the results are not significant. Therefore, we accept alternate hypothesis H12 and H13 but reject alternate hypothesis H11.

Table 11.12: Wilcoxon Test Results

| | Based on Average AUC Values | Based on Average Accuracy Values |
|---|---|---|
| Model I vs Model II | ↑* (0.001) | ↑ (0.064) |
| Model I vs Model III | ↑* (0.002) | ↑* (0.004) |
| Model I vs Model IV | ↑* (0.001) | ↑* (0.002) |
| ↑*: Significantly superior; ↑: Superior but not significantly | | |

According to Table 11.12, the results of the Wilcoxon signed rank test were significant in five out of six cases, which indicates that Model I which is developed by using evolution-based metrics in conjunction with OO metrics performs significantly better than all the other possible scenarios (Model II, Model III & Model IV) explored in the chapter. Thus, evolution-based metrics when used in conjunction with OO metrics are effective metrics suite for change-proneness prediction.

### 11.4.5 Analysis of Chapter's Results

The observations with respect to evolution trends of the OO metrics are stated below:
*Observations with respect to Size Dimension*

- The mean class size in terms of SLOC and WMC is much higher for CC as

compared to UCC. This indicates that classes tend to increase in size in terms of SLOC and number of total methods.

- It was observed that during evolution, the most common trend was increase in class size. Thus, it is rare that the addition of new functionality leads to a decrease in class size.

*Observations with respect to Cohesion Dimension*

- The investigation of mean and median values of the LCOM metric indicates poor cohesiveness of the CC when cohesion was evaluated using the LCOM metric.

- Majority of the classes which evolved during the investigated versions of the Android software did not exhibit any change in the values of their cohesion metrics. However, changes made to a class during evolution may lead to increase in LCOM values in the Android software.

*Observation with respect to Coupling Dimension*

- The analysis of mean and median of coupling metrics indicates that a class with higher coupling values is prone to change in future versions as CC exhibited higher coupling metric values.

- Majority of the classes did not exhibit any change in the values of their coupling metrics. However, if there was a change in a class, there are higher chances of increasing the dependency of a class on other classes during evolution.

*Observations with respect to Inheritance Dimension*

- As the investigated application packages were developed in Java language, all the classes exhibit a DIT value of greater than one as all classes are derived

from the Java object class. However, the number of classes exhibiting a DIT value of greater than one and a non-zero NOC value were low. This indicates that inheritance is rarely used in the investigated datasets.

- An observation of DIT metric values indicates that there were few classes with DIT values of 2 and 3 and hardly any classes with DIT value of 4 and 5. This indicates that inheritance levels don't go too deep in the investigated datasets. Similarly, an observation of NOC values indicate few classes with three or more children. Thus, classes in the investigated datasets rarely grow to large breadths. Generally, they only have one or two children.

- An analysis of maximum and minimum values of inheritance metrics indicate higher values exhibited by CC. This implies classes exhibiting higher inheritance characteristics are more prone to changes during evolution of the software.

- The trends observed by inheritance metrics indicate that 92-100% of classes exhibited no change in their DIT and NOC values. Thus, it is rare that a change due to evolution might affect inheritance attribute of a class.

With respect to the various change prediction models developed in the chapter, it may be noted that the models developed using only evolution-based metrics confirm the predictive capability of these metrics for predicting software change. The AUC values obtained by Model III were generally in the range of 0.60-0.78. Similarly, the accuracy values of such models ranged from 65%-95% in the majority of cases. These observations support the capability of evolution-based metrics for software change prediction. Moreover, TACH and CSBS evolution-based metrics are more effective than their other evolution-based counter-parts for prediction of change-prone classes. This was indicated by the application of CFS method which eliminated noisy

and superfluous attributes.

However, several literature studies have supported the use of OO metrics (Model III) for developing change prediction models [2, 3, 5, 27, 30, 31, 36, 257, 284]. The results of this chapter too, support the use of such models. The models developed using OO metrics were found competent and effective in predicting software change. Though, both Model II and Model III gave effective change prediction models, this chapter statistically proves the superiority of the combined models (using both evolution-based and OO metrics) as compared to using only evolution-based or only OO metrics for change prediction. The prime reason for its superiority is the encapsulation of diverse characteristics by both sets of metrics. As both metric sets are representative of different characteristics i.e version history and structural features, the developed change prediction models are effective. Thus, researchers and software practitioners in the future should use both these metric suites for developing superior change prediction models. Such effective change prediction models would help in better optimization of limited software project resources and good quality software products.

It was also observed that Model IV gave the worst results. This is because model IV was developed using only one metric (ICP), which encapsulated just one aspect of evolution history. Using such a uniform and non-diverse predictor variable would lead to poor prediction models as the model is unable to learn various software characteristics which are necessary for predicting change.

## 11.5 Comparison with Previous Studies

This section states the comparison of the chapter's results with previous studies. We first report the comparison of evolution patterns of OO metrics and then those of change prediction models developed in RQ2-RQ4.

### 11.5.1 Comparison of Evolution Patterns of OO Metrics

This section compares the evolution patterns of OO metrics as observed by us and the ones which are reported by previous studies.

- A study by Lee et al. [24] rigorously analyzed the evolution of an open-source system, JFreeChart. The results of the study report that the number of classes i.e. the overall size of the software increases gradually with each release. Since, this chapter only evaluates the trends of common classes, we cannot comment on the increase in size of the software with respect to the number of classes. However, it may be noted that the common CC tend to increase in size over various releases of the software, rather than decrease.

- A study by Alenezi and Zarour [292] evaluated the "modularity" evolution of two open-source software systems by analyzing OO metrics corresponding to coupling, cohesion and complexity. They reported increasing LCOM metric values in two open-source software datasets while analyzing cohesion metrics. Our analysis also indicated the "increasing" trend to be popular for LCOM metric.

- Nasseri et al. [293] investigated seven open-source software systems to assess the patterns depicted by four inheritance metrics. Their study concluded that software systems tend to increase in size by adding more number of new classes i.e. "breadth-wise" rather than adding classes somewhere in the inheritance hierarchy i.e. "depth-wise". Similar to their results, we analyzed that most of the classes exhibited either a DIT value of 1 or 2 indicating shallow inheritance levels.

### 11.5.2 Comparison of Change prediction Models Developed by Combined Metric Suite

As discussed in section 11.3.1, with respect to RQ2-RQ4, our experimental design is an improvisation of the one followed by Elish and Al-Khiaty [1]. They proposed the use of both evolution-based and OO metrics for determination of change-prone classes. However, they used only a statistical method, LR. We performed an extension of their experiments by using six other ML techniques. Our results with LB technique have shown improvement over Elish and Al-Khiaty [1] results (for Model I, Model II and Model III). The results are compared using accuracy performance measure as it was used by Elish and Al-Khiaty [1] for evaluating the models. Table 11.13 depicts the comparison of our results (developed using the LB technique) and the results obtained by Elish and Al-Khiaty [1]. The best results in each case are depicted in bold.

Table 11.13: Comparison Results using Accuracy

| Prediction Models | Elish & Al-Khiaty [1] | | Our Results | |
|---|---|---|---|---|
| | VSSPlugin | PeerSim | Contacts | Gallery2 |
| Model I | 80.5 | 83.8 | **87.7** | 74.5 |
| Model II | 80.0 | 80.2 | **81.2** | 76.1 |
| Model III | 77.2 | 81.2 | **82.2** | 72.7 |
| Model IV | 76.4 | **78.9** | 78.2 | 69.8 |

With respect to statistical significance also, our results slightly vary than those which were obtained by Elish and Al-Khiaty [1]. As they assessed whether the models developed using the combined metrics suite are significantly better in performance than other metrics suite (Model II, Model III & Model IV) on two open-source datasets namely VSSPLUGIN and Peersim using Wilcoxon test on accuracy values, the results of their tests were not found significant. This could be due to the choice of an inadequate performance measure i.e. accuracy. They concluded that the inclusion

of evolution-based metrics increases the capability of OO metrics and gives an effective set of change predictors. However, these results were not found significantly better than the usage of only the OO metrics. The results of this chapter significantly prove the superiority of the combined metric suite over other possible combinations. Our results were evaluated using AUC, a stable performance measure.

## 11.6 Discussion

This chapter analyzed the evolution patterns of seven OO metrics on two application packages of the Android software. The OO metrics were categorized into four dimensions corresponding to size, coupling, cohesion and inheritance. The metrics for common classes in five versions (4.0.2, 4.0.4, 4.1.2, 4.2.2 and 4.3.1) of each application package were evaluated and classes in each application package were categorized as CC or UCC. The trends of actual changed classes between consecutive versions were categorized as "Constant", "Increasing" or "Decreasing". We observed that the CC exhibited higher chances of increase in class size, higher coupling values and higher inheritance characteristics as compared to UCC. Moreover, the most common trend of actual changed classes between consecutive versions was increase in size metrics (WMC and SLOC), RFC coupling metric and LCOM cohesion metric and no change in inheritance and other coupling metrics (CBO). We observed that very few classes exhibited the inheritance attribute and classes which used inheritance, did not exhibit deep inheritance levels or a large number of subclasses.

Another key objective of the chapter was to explore the capability of evolution-based metrics when used in conjunction with OO metrics for ascertaining change-prone nature of classes. Numerous prediction models were created release by release for two application packages of the Android software. Four possible scenarios were evaluated for predicting change-prone classes (i) use of combined evolution-based

and OO metrics, (ii) use of solely evolution-based metrics (ii) use of solely OO metrics and (iv) use of the ICP metric. Also, the chapter evaluated the use of seven classification techniques in order to determine change-prone nature of classes in an OO software. Wilcoxon test was performed for statistical comparison. We found that:

- TACH and CSBS are primary evolution-based metrics which are good indicators of change-prone nature of a class as they were selected for construction of most of the change prediction models after application of the CFS method.
- After analyzing the capability of the combined metrics suite, solely evolution-based metrics, solely OO metrics and just the ICP metric, we concluded that the performance of the models using both evolution-based metrics along with OO metrics is better for developing change prediction models than all the other scenarios. This is because the combined metric suite efficiently represents both the OO properties of a class such as inheritance, cohesion etc. along with its change history. Thus, it incorporates two separate dimensions (structural properties as well as evolution history) which help in the effective determination of change-prone classes.
- The superiority of the combined metric model (both evolution-based and OO metrics) is statistically confirmed with the help of Wilcoxon test on AUC values and accuracy values. Hence, the chapter advocates the use of this combined set of indicators for ascertaining change-prone classes in future studies.

Thus, this chapter recommends the use of combined metrics suite as predictors for change-prone classes as it increases the capability of traditional models which have been used in literature and use only OO metrics for change prediction.

# Chapter 12

# Conclusion

## 12.1 Summary of the Work

Software evolution is one of the most critical phases of a software lifecycle. An existing software may no longer be useful if it is not capable of adapting itself to modifying user requirements or changes in environmental conditions. These modifications may not just be superficial and may lead to drastic changes in the design and structure of a software system. Thus, one has to ensure that the software quality does not degenerate with the introduction of modifications. The primary aim of the work conducted in this thesis is to construct models which aid in the easy allocation of constraint software resources during its evolution. This would ensure that a cost-effective and maintainable software product is produced. Such good quality products are reliable and would achieve high customer satisfaction. The thesis proposes and analyzes several techniques to develop optimum models, which determine the change-prone nature of a class in an OO software. These techniques range from the well-known ML techniques, which are widely used in the domain of predictive modeling to the recently explored SBA. As SBA were found effective in the domain

of predictive modeling, we also proposed ensembles of SBA for improved results. The thesis also evaluates the use of different predictors apart from OO metrics to determine change-prone classes. Moreover, the work conducted in the thesis examines certain other aspects of software evolution, for instance evolution patterns of OO metrics and categorization of software bugs on the basis of its change impact. With the aid of organized empirical experiments, which have been conducted rigorously, the work conducted in the thesis is proven to be highly useful for software practitioners and managers.

We first discussed the various steps performed in order to conduct methodical and proper empirical experiments. A description of the various variables (both dependent and independent) involved in the studies along with a specification of all the classification techniques used in the thesis is presented. The characterization, basic functioning and parameter settings of each investigated classification technique is mentioned. Thereafter, we also summarized the empirical data collection process and the various datasets used in the thesis. We focus on open-source datasets for performing the experiments conducted in the thesis as they are easily available, aid in easy replication and generalization of results. Moreover, the open-source paradigm is widely popular, aids better innovation, boasts of large community support, and develops cost-effective products. Thereafter, we categorize the datasets used in the work on the basis of the number of data points into three categories (“small”, “medium” and “large”). The descriptive statistics of each category of datasets is mentioned. We also state other data pre-processing procedures (outlier analysis and feature selection) used in the work. We also provide a description of the validation methods (internal and external) and performance measures, which are used to assess the efficiency of developed models. In order to strengthen the conclusion validity of the obtained results, we also conduct statistical analysis of the results obtained in each chapter. The tests used in the thesis for this purpose are also explained.

To evaluate and assess the current state of literature in the domain of software evolution, we performed a systematic literature review. We evaluated studies, which either developed models to determine the change-prone nature of an OO class or the studies which assess the change impact of a software change request. We found 34 primary studies in the period from January 2000 to December 2017. These studies were analyzed to answer various RQs with respect to the type of predictors, experimental settings, categories of various data analysis techniques used for developing models, predictive performance of ML techniques (most popular category of data analysis techniques found), the statistical tests used for results verification and the threats encountered and addressed while setting up experiments. Apart from answering the RQs, we state a brief overview of the key parameters of all the 34 primary studies in tabular form (Appendix B.1). It was found that a majority of change prediction studies used product metrics for predicting change-prone nature of an OO class. The ML techniques, especially their subcategory, SBA, were found effective in developing change prediction models. However, we found a lack of studies (only eight studies) which assessed the change impact of a change request on the basis of its textual description.

We also identified several future directions in the domain. The use of process metrics and the combination of process and product metrics should be explored for determining software change. As the majority of datasets used in the literature studies were imbalanced in nature, methods should be evaluated for developing reliable models from such datasets. Furthermore, more studies are required to empirically verify, assess and compare the performance of various categories of data analysis techniques. Moreover, the comparison results should be statistically assessed as 51% of primary studies did not use any statistical test for proper verification of obtained results.

While developing effective software change prediction models, we first need

to evaluate the relationship between the OO metrics and the dependent variable "change-proneness". Furthermore, we need to adjudge significant predictors of change amongst the investigated OO metrics. In order to fulfill the above stated objectives, we develop change prediction models using eleven ML techniques on six open-source datasets. As investigated in the literature review, ML techniques are popular and effective in the domain of predictive modeling. These techniques are adaptive in nature, do not need to be specifically programmed and easily learn from previous historical data. The results evaluated using AUC, Balance and G-Mean1 performance measures indicate the effectiveness of ML techniques (Mean AUC values: 0.68-0.78, Mean G-Mean1 values: 0.51-0.67, Mean Balance value: 55-66), specifically the ones based on ensemble methodology (RF, LB and BG). Moreover, the results of models developed using the ML techniques were found comparable and in certain cases better than the statistical technique, LR, though not significantly. As effective change prediction models were constructed, the chapter confirms the predictive capability of OO metrics for determining change-proneness attribute of a class. Also, the CBO, WMC and SLOC metrics were evaluated as significant predictors of change.

As the results of models developed using ML techniques were found suitable, we also explored the use of SBA, a sub-category of ML techniques for predicting change-prone classes. However, we first analyzed the use of SBA in SEPM literature, specific to four attributes (effort estimation, defect-proneness prediction, maintainability prediction and change-proneness prediction). The review was conducted in order to outline empirical evidence of their effectiveness and to assess the best practices and experimental settings with regard to the use of SBA in SEPM literature. Though, SBA were effectively used in SEPM literature, there were very few studies which analyzed their use in the domain of maintainability prediction and software change prediction. Thus, in order to bridge this gap, we performed an empirical study with effective experimental setup (conduct multiple runs, use stable performance mea-

sures, completely specify parameter settings and fitness functions, perform rigorous statistical evaluation) to develop change prediction models using eight SBA on fourteen datasets. These developed models were compared with those of four ML techniques and the statistical technique, LDA. The models developed using the MPLCS technique, a search-based algorithm, obtained superior results than all the other investigated techniques, when assessed using Friedman test on G-Mean1 and Balance values. The average G-Mean1 and Balance values of change prediction models developed using the MPLCS technique on all the fourteen investigated datasets were 0.67 and 65.41 respectively. The successful results of the models developed using the MPLCS technique were attributed to the selection of its fitness function. The fitness function of the MPLCS technique analyzed both the accuracy and the complexity of the rule set. The pairwise comparisons of the models developed using the MPLCS technique and the other investigated techniques was evaluated using the Wilcoxon test. It was found that the models developed using the MPLCS technique were better than most of the other compared techniques, though not significantly.

As both ML and SBA were found effective, we also explored HBT. These techniques combined an ML/statistical technique with a search-based algorithm. The idea was to investigate a preferred approach which combines the characteristics and strengths of both its constituent techniques. HBT can be classified as a sub-category of SBA. We assess the effectiveness of four HBT with eleven other techniques. These eleven techniques included representatives of SBA, ML techniques and the statistical techniques. The evaluation was done in terms of both predictive performance as well as CPU time taken to develop change prediction models. The hybridized technique, PSO-LDA was a clear winner in terms of predictive performance. The average Balance and G-Mean1 values of the models developed using the PSO-LDA technique was 67.21 and 0.70 respectively. These results were also statistically supported by the use of Wilcoxon test, which conducted pairwise comparisons amongst

the models developed using the PSO-LDA technique and the other investigated techniques. The results of the Wilcoxon test were found significant. Moreover, most of the investigated HBT (DT-GA, PSO-LDA and GFS-LB) also fared well in terms of CPU time consumption as they were only next to ML techniques. We also investigated the trade-off between CPU time and predictive performance of the models developed using the various techniques and advocate the use of HBT for developing change-proneness prediction models. Even if HBT take slightly longer CPU time, as compared to the ML techniques, the time and costs saved by the application of the change prediction models developed using these HBT compensated for it. We found a 20% increase in the effort gain, with the application of models developed using the HBT as compared to ML techniques.

The literature review conducted on software evolution also pointed out that few recent studies have explored the use of an ensemble of techniques for predicting the change-prone nature of a class. We also evaluated the use of ensembles of techniques as they are proven to give improved results than individual classifiers [42–44]. However, we pioneered in creating ensembles of SBA, by changing their fitness function. We proposed four fitness-based ensemble classifiers namely MVEC, WVEC, HIEC and WVHIEC by combining seven fitness variants of a search-based algorithm, CPSO. We first evaluated whether the individual fitness variants can be effective constituents of an ensemble. In order to do so, we adjudged the accuracy and complementarity of the models developed using individual fitness variant classifiers. We further assessed the effectiveness of the proposed ensemble classifiers and found the models developed using the HIEC and the WVHIEC techniques as the ones which depicted the best results. An improvement of up to 70% and 66% was found in the G-Mean1 and Balance values with the use of proposed HIEC and WVHIEC models. Furthermore, these models were found comparable to traditional ML ensemble classifiers.

The investigation of fitness-based ensembles revealed that the result of a search-based algorithm is highly dependent on the selection of its fitness function. However, a fitness function is generally selected on the basis of the performance of an algorithm using the specific fitness function on the entire dataset. In this scenario, we ignore an important aspect, which is that the structural characteristics of each data point may be different. Therefore, a specific data point of the same dataset may give better results with a different fitness variant, while other data points of the dataset may give best results with other fitness variants. Thus, rather than selecting a single fitness variant for an entire dataset, we explored a framework for selecting an optimum fitness function for each data point. We named this framework ASOF. The framework predicts the best fitness variant on the basis of structural characteristics (OO metrics) of a data point. Thereafter, the model of the fitness variant is used for determining the change-prone nature of the specific data point. We assessed the performance of the change prediction models developed using the proposed framework and found them to be effective (G-Mean1: 0.55-0.69, Balance: 53-68%), when evaluated on fifteen popular open-source datasets. These models were further compared with the individual fitness-variant classifiers and the four proposed ensemble classifiers (MVEC, WVWC, HIEC, WVHIEC) and were found statistically superior in a majority of cases. Furthermore, the models developed using the ASOF framework were found competent to those developed using the LR technique and four traditional ML ensemble classifiers (RF, BG, LB and AB).

We also develop SBC models to categorize software bugs into three categories i.e. "low", "moderate" and "high". These categories are predicted on the basis of bug description. The bugs are categorized on the basis of three bug attributes i.e. maintenance effort (number of SLOC required to correct a specific software bug), change impact (number of classes requiring change in order to correct the bug) and the combined effect of both (product of maintenance effort and change impact). The

allocation of a level to software bugs would help managers and software testers in deciding effective bug regimes and allocation of resources. The developed SBC models are binary in nature i.e. they predict whether a bug belongs to "low" level or "not low" level and so on. The SBC models were developed using six data analysis techniques on five application packages of the Android software. For each category, four models were developed using just Top-10 words, using Top-25 words, using Top-50 words and by using Top-100 words. Though, models developed using the Top-10 words were not that efficient, the other models depicted acceptable performance. Moreover, the SBC models developed using the combined approach were found statistically better than SBC models based only on change impact and comparable to SBC models based on maintenance effort. It was also found that the models which categorized bugs into "high" and "not high" level were superior to models which categorized bugs into "moderate" and "not moderate" level or those which categorized bugs into "low" and "not low" level.

As investigated in the review, methods to develop effective and reliable models from imbalanced datasets should be explored. Such methods are important as models developed from imbalanced data may not be able to learn to distinguish minority class instances well (as such instances are lower in number and the model may be incapable of learning from just a few instances). This will lead to higher misclassification errors and poor models. We explored the use of three data sampling methods (Resample with replacement, SMOTE and Spread subsample) and a cost-sensitive method (MetaCost learners) in order to address the ILP. The results were empirically validated using six datasets which were highly or moderately imbalanced. It was found that the models developed after application of the Resample with replacement method were superior to the other explored methods for addressing ILP. There was an improvement of up to 60% each in G-Mean1 and Balance values and of up to 35% in AUC values with the application of Resample with replacement method

while developing software change prediction models using imbalanced datasets. The results were consistent using both ten-fold cross validation and inter-release validation. The Resample with replacement method achieved a uniform ratio of both change-prone and not change-prone classes, attributing to its success. Moreover, the pairwise comparison of the Resample method with the other investigated imbalanced learning methods using Wilcoxon test was found statistically significant in the scenario of ten-fold cross-validation and comparable in the scenario of inter-release validation. We also analyzed that the traditional performance measures such as accuracy, precision and sensitivity are not very good indicators for evaluating models developed using imbalanced datasets. Stable performance measures such as AUC, G-Mean1 and Balance should be used for evaluating such models.

We also assessed the evolution patterns of OO metrics (CK metrics & SLOC) in five consecutive versions (4.0.2-4.3.1) of Contacts and Gallery2 Android application packages. The investigated metrics were divided into four dimensions namely size, cohesion, coupling and inheritance. It was observed that the classes which changed in the consecutive versions of the software exhibited trends of increase in class size and coupling values. The LCOM values increased indicating poor cohesion. Also, the increase in metric values of coupling values needs to be controlled as higher coupling leads to higher complexity and ripple effects. However, there were very few classes which exhibited non-zero values for inheritance metrics (NOC and DIT), indicating that inheritance is less used in the systems.

In lieu of the investigated research gap to address process metrics and the combination of process and product metrics, we investigated the use of evolution-based metrics (process metrics) for determining change. These metrics were evaluated independently and in conjunction with CK metrics (product metrics) to determine the change-prone nature of a class. The evolution-based metrics encapsulate the evolution history of a class, while the CK metrics are representative of a class's design. It

was found that the models developed using the conjunction of evolution-based metrics and the CK metrics were superior to models developed using only the evolution-based metrics or only the CK metrics on the two investigated application packages of Android software (Contacts & Gallery2). We found a percentage improvement of 3-6% in the AUC values of the models developed using the combined metrics suite over the ones developed using only the OO metrics. Moreover, this superiority was confirmed using statistical tests. The prime reason for the effectiveness of the combined metrics suite is that both sets of metric suites are representatives of different dimensions i.e. structural (CK metrics) and evolution history (evolution-based). It was also found that the TACH and CSBS evolution-based metrics were significant predictors of change-prone classes.

## 12.2 Applications of the Work

The work conducted in the thesis can aid software practitioners and researchers in the following ways:

***Aid to Researchers***

- Researchers can track and assess the metrics found significant (both OO and evolution-based) in the work to efficiently predict change-prone classes.
- As multiple classification techniques are evaluated in the research work, it provides guidelines to researchers for selecting an appropriate modeling technique for software prediction tasks.
- The work provides a well-defined & systematic approach for the development of effective software change prediction models, which can be used for further experiments or replication of existing experiments.

***Aid to Software Practitioners***

- Managers can plan cost-effective resource allocation for assessment & improvement of existing products and practices.
- Developers can pay focused attention to problematic change-prone classes and re-design them suitably.
- Testers can perform rigorous testing, prioritize testing effort & design effective bug solving regimes.

## 12.3 Future Directions

Although the work conducted in the thesis evaluates a wide category of classification techniques and datasets, however, future studies may plan to replicate the experiments on datasets belonging to varied application domains and developed using other programming languages such as C# or Python. This would increase the generalizability of the obtained results.

In future, researchers may also replicate the proposed fitness-based ensemble classifiers and the ASOF framework using other SBA, such as GA or GP. The methods investigated for learning from imbalanced data may be evaluated further by using SBA for classification. Such experiments can be conducted in order to generalize the effectiveness of the investigated imbalanced learning methods.

Replication is important as it aids in addition of evidence, based on which researchers and practitioners might make choices and plan their experiments. It also helps in determining the applicability of results to other real-world scenarios or industrial settings. Therefore, future studies may replicate our experiments to yield generalized conclusions.

# Appendices

# Appendix A

# Details of Datasets used in the Work

This appendix states the details of the datasets used in the thesis along with their descriptive statistics.

## A.1 Dataset Details

We list the datasets used in the work along with their characteristics, which includes the name, releases, number of data points, percentage of change and brief description of the dataset (Table A.1).

Table A.1: Details of Datasets

| Dataset Name | Previous Release | Recent Release | Number of Classes | Changed Classes(%) | Description |
|---|---|---|---|---|---|
| Art of Illusion (AOI) | 2.7 | 2.9.2 | 434 | 30% | 3D modelling software. |
| Android Bluetooth | 4.3.1 | 4.4.2 | 72 | 19% | Application package for Bluetooth services in Android. |
| | 5.0.2 | 5.1.0 | 112 | 29% | |
| Android Calendar | 4.0.2 | 4.0.4 | 77 | 29% | Application package for Calendar in Android. |
| | 4.0.4 | 4.1.2 | 78 | 62% | |
| | 4.3.1 | 4.4.2 | 106 | 19% | |

| Dataset Name | Previous Release | Recent Release | Number of Classes | Changed Classes(%) | Description |
|---|---|---|---|---|---|
| Android Contacts | 4.0.2 | 4.0.4 | 324 | 6% | Application package for contacts handling in Android. |
| | 4.0.4 | 4.1.2 | 331 | 36% | |
| | 4.1.2 | 4.2.2 | 357 | 10% | |
| | 4.2.2 | 4.3.1 | 375 | 4% | |
| | 4.3.1 | 4.4.2 | 210 | 48% | |
| Android Telephony | 4.2.2 | 4.3.1 | 249 | 63% | Application package for telephonic conversation in Android. |
| Android Gallery2 | 4.0.2 | 4.0.4 | 305 | 10% | Application package for handling Gallery services in Android. |
| | 4.0.4 | 4.1.2 | 310 | 26% | |
| | 4.1.2 | 4.2.2 | 330 | 52% | |
| | 4.2.2 | 4.3.1 | 374 | 41% | |
| Android MMS | 2.3.7 | 4.0.2 | 195 | 30% | Application package for MMS services in Android. |
| | 4.0.4 | 4.1.2 | 206 | 33% | |
| | 4.1.2 | 4.2.2 | 223 | 52% | |
| Apache Click | 2.2.0 | 2.3.0 | 436 | 62% | A web-based application framework for Java Execution Environment. |
| Apache IO | 1.3 | 1.4 | 109 | 41% | Library of utilities to assist with developing IO functionality. |
| | 1.4 | 2.0 | 266 | 89% | |
| | 2.3 | 2.4 | 198 | 6% | |
| Apache Giraph | 1.0 | 1.1 | 484 | 60% | A graph processing software. |
| Apache Gora | 0.4 | 0.5 | 222 | 18% | A framework for in-memory data model. |
| Apache Hama | 0.5 | 0.6 | 270 | 44% | A framework for distributed computing. |
| Apache Jmeter | 2.8 | 2.9 | 335 | 60% | A load testing tool by Apache. |
| Apache LogicalDoc | 7.5 | 7.6 | 929 | 23% | A document management system. |
| Apache Maven | 3.3.3 | 3.3.9 | 831 | 34% | A build automation tool for Java. |
| Apache Log4j | 1.2.13 | 1.2.14 | 296 | 9% | Utilities for asynchronous logging services. |
| | 1.2.16 | 1.2.17 | 350 | 25% | |
| Apache Math | 3.1.1 | 3.2 | 1404 | 88% | A library for mathematical and statistical components. |
| Apache Net | 3.0 | 3.1 | 240 | 36% | Implements client side of many protocols. |
| | 3.1 | 3.2 | 331 | 95% | |

| Dataset Name | Previous Release | Recent Release | Number of Classes | Changed Classes(%) | Description |
|---|---|---|---|---|---|
| Apache Phoenix | 4.2 | 4.3 | 1101 | 42% | A relational database engine. |
| Apache Zookeeper | 3.4.8 | 3.5.1 | 591 | 46% | Software for enabling distributed connections. |
| Apollo | 0.1 | 0.2 | 252 | 27% | Editor and compiler for Java migration software. |
| AviSync | 1.1 | 1.2 | 73 | 40% | It is used for adjusting synchronization issues for DivX AVI format audio/ video files. |
| Celestia | 1.4.1 | 1.6.1 | 254 | 58% | Software for 3D visualization of space. |
| DrJava | 1.6.0 | 1.8.1 | 278 | 60% | Software for managing digital assets. |
| DSpace | r4668 | r5686 | 402 | 50% | A programming environment for Java which includes a program editor, debugger, unit testing tool and an interactions pane for program evaluation. |
| Eclipse | 2.0 | 2.1 | 4830 | 60% | A toolset for software management. |
| Frinika | 0.2.0 | 0.6.0 | 248 | 51% | A complete music workstation software. |
| Glest | 1.0.10 | 3.2.2 | 108 | 66% | A real time 3D strategy game. |
| HyperSQL | 2.3.0 | 2.3.1 | 510 | 41% | A multi-threaded relational database engine. |
| | 2.3.3 | 2.3.4 | 609 | 38% | |
| JabRef | 3.1 | 3.2 | 1062 | 34% | A software for reference management. |
| Jedit | 5.0.0 | 5.1.0 | 980 | 16% | A text editor software. |
| PMD | 3.9 | 4.3 | 431 | 60% | A source-code analyzer. |
| Robocode | 1.7.2.2 | 1.7.4.4 | 266 | 26% | Programming game where one can develop a robot battle tank. |
| Simutrans | 111.3 | 112.3 | 510 | 45% | A transport simulation game. |
| SubSonic | 2.8 | 4.6 | 123 | 76% | A web-based media streamer. |
| | 5.2 | 5.3 | 332 | 22% | |

## A.2 Descriptive Statistics

This section lists the descriptive statistics of the datasets. Table A.2 states the range of descriptive statistics for “small-sized” datasets. There were 11 such datasets used in the thesis. In each cell of the table we state the range of the statistic, i.e. the min-

imum value and the maximum value obtained while computing the statistic on each of the 11 datasets. Similarly, Table A.3 and Table A.4 states the range of descriptive statistics for thirty "medium-sized" and eleven "large-sized" datasets respectively.

Table A.2: Descriptive statistics range for Small-sized datasets

| Metric Name | Min. | | Max | | Mean | | Median | | S.D. | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Min. | Max. | Min. | Max. | Min. | Max. | Min. | Max. | Min. | Max. |
| CBO | 0.00 | 0.00 | 10.00 | 45.00 | 1.14 | 6.00 | 1.00 | 3.00 | 1.98 | 7.96 |
| DIT | 0.00 | 1.00 | 1.00 | 5.00 | 0.72 | 2.26 | 0.00 | 2.00 | 0.00 | 1.33 |
| NOC | 0.00 | 0.00 | 0.00 | 19.00 | 0.00 | 0.56 | 0.00 | 0.00 | 0.00 | 2.32 |
| RFC | 0.00 | 3.00 | 44.00 | 154.00 | 8.20 | 18.83 | 6.00 | 14.00 | 8.12 | 22.96 |
| SLOC | 2.00 | 10.00 | 359.00 | 1,972.00 | 61.23 | 137.23 | 35.00 | 87.00 | 67.47 | 242.74 |
| LCOM | 0.00 | 1.00 | 95.00 | 11,628.00 | 41.95 | 407.65 | 15.00 | 85.00 | 30.29 | 1,394.47 |
| WMC | 0.00 | 2.00 | 60.00 | 153.00 | 7.65 | 17.63 | 5.00 | 11.00 | 8.2 | 22.99 |

Table A.3: Descriptive statistics range for Medium-sized datasets

| Metric Name | Min. | | Max | | Mean | | Median | | S.D. | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Min. | Max. | Min. | Max. | Min. | Max. | Min. | Max. | Min. | Max. |
| CBO | 0.00 | 0.00 | 10.00 | 157.00 | 1.04 | 10.79 | 1.00 | 8.00 | 1.39 | 10.11 |
| DIT | 0.00 | 1.00 | 2.00 | 24.00 | 0.41 | 2.55 | 0.00 | 2.00 | 0.37 | 1.93 |
| NOC | 0.00 | 1.00 | 4.00 | 101.00 | 0.05 | 6.07 | 0.00 | 4.00 | 0.33 | 7.75 |
| RFC | 0.00 | 13.00 | 61.00 | 690.00 | 8.42 | 65.60 | 4.00 | 58.50 | 8.66 | 70.90 |
| SLOC | 1.00 | 8.00 | 100.00 | 8,858.00 | 47.23 | 867.00 | 30.00 | 107.00 | 39.59 | 483.20 |
| LCOM | 0.00 | 2.00 | 5.00 | 31,375.00 | 1.08 | 709.05 | 1.00 | 68.00 | 1.24 | 2,569.42 |
| WMC | 0.00 | 2.00 | 37.00 | 748.00 | 0.40 | 35.26 | 0.50 | 33.00 | 2.2 | 54.29 |

Table A.4: Descriptive statistics range for Large-sized datasets

| Metric Name | Min. | | Max | | Mean | | Median | | S.D. | |
|---|---|---|---|---|---|---|---|---|---|---|
| | Min. | Max. | Min. | Max. | Min. | Max. | Min. | Max. | Min. | Max. |
| CBO | 0.00 | 0.00 | 24.00 | 166.00 | 1.43 | 14.94 | 1.00 | 11.00 | 2.41 | 15.11 |
| DIT | 0.00 | 1.00 | 3.00 | 34.00 | 0.22 | 2.14 | 0.00 | 2.00 | 0.49 | 1.96 |
| NOC | 0.00 | 0.00 | 12.00 | 155.00 | 0.18 | 5.19 | 0.00 | 3.00 | 0.87 | 7.14 |
| RFC | 0.00 | 13.00 | 129.00 | 2,240.00 | 10.20 | 255.20 | 6.00 | 94.00 | 13.37 | 332.80 |
| SLOC | 1.00 | 6.00 | 766.00 | 6,764.00 | 61.14 | 272.78 | 33.00 | 116.00 | 81.12 | 498.83 |
| LCOM | 0.00 | 0.00 | 100.00 | 8,128.00 | 29.47 | 174.20 | 0.00 | 69.00 | 30.97 | 2,615.40 |
| WMC | 0.00 | 1.00 | 2.00 | 699.00 | 0.17 | 36.89 | 0.00 | 32.00 | 0.39 | 57.04 |

# Appendix B

# Key Parameters of Primary Studies in Review on Software Change Prediction

This is an appendix to Chapter 3. It reports the key parameters investigated in the software change prediction review conducted in Chapter 3.

## B.1 Key Parameters of primary Studies

Table B.1 reports the key parameters of the review for each of the selected primary study in Chapter 3. The table includes the study number allocated to a specific primary study, the set of predictors, the name and number of datasets, the classification or modeling techniques used, the performance measures and the statistical tests. These parameters are listed for each of the 34 primary studies, which were selected in Chapter 3.

Table B.1: Key Parameters of Primary Studies

| Study No. | Predictors | Datasets | Data Analysis Techniques | Performance Measures | Validation Method | Statistical Test |
|---|---|---|---|---|---|---|
| CP1 | NOCI, 4 Lines of Code measures | Windows based Software Application (VLWA Dataset) | LR, GP | Type I error, Type II error, Overall misclassification rate | Hold-out validation | — |
| CP2 | NOCI, 4 Lines of Code measures | Windows based Software Application (VLWA Dataset) | GP, Decision tree based GP | Type I error, Type II error, Overall misclassification rate | Hold-out | — |
| CP3 | CK, NOM, Probability values related to history, Probability of Change | JFlex, JMol | LR | Accuracy, Recall, False Positive Ratio, False Negative Ratio, Goodness of Fit | — | — |
| CP4 | AID, ALD, CC, SLOC, NOP, MNOB, MPC, NOLV, NIC | JFlex | Non-linear system of equations, Linear system of equations, Depth first search graphs, Binary dependencies | Accuracy, False Positive Ratio, False Negative Ratio | — | — |
| CP5 | 22 OO metrics: 4 cohesion metrics, 4 coupling metrics, 7 inheritance metrics, 7 size metrics | Bean Browser, Ejbvoyager, Free, Javamapper, Jchempaint, Jedit, Jetty, Jigsaw, Jlex, Lmjs, Voji, 4 versions of JDK | C4.5 Decision Tree, GA | Accuracy, J-Index | Ten-fold | — |
| CP6 | CK, Lorenz & Kidd [19], MOOD metrics, BDM | Jflex (13 versions) | Stepwise Multiple Regression | Goodness of Fit | — | ANOVA |

| Study No. | Predictors | Datasets | Data Analysis Techniques | Performance Measures | Validation Method | Statistical Test |
|---|---|---|---|---|---|---|
| CP7 | 22 OO metrics: 4 cohesion metrics, 4 coupling metrics, 7 inheritance metrics, 7 size metrics | Bean Browser, Ejbvoyager, Free, Javamapper, Jchempaint, Jedit, Jetty, Jigsaw, Jlex, Voji | C4.5 Decision Tree, ACO | Accuracy | Ten-fold | Wilcoxon Signed Rank |
| CP8 | CK, QMOOD | Yari (3 versions), UCdetector (4 versions), JFreeChart (4 versions) | Combined Rank List Mechanism | Hit-Ratio (Recall), Change Cost, Cost ratio | — | — |
| CP9 | 62 OO metrics: 18 cohesion metrics, 20 coupling metrics, 17 inheritance metrics, 7 size metrics | 102 Java Systems | Random effect meta-analysis | AUC | — | — |
| CP10 | CK, 3 usage metrics, 3 complexity metrics, Interface Usage Cohesion metric | 8 Eclipse plug-in projects and 2 Hibernate Systems | NB, SVM, MLP | Recall, Precision, AUC | Ten-Fold | Wilcoxon Signed Rank |
| CP11 | CK, 7 Network centrality measures from social network analysis | 19 Eclipse plug-in projects, Azureus | MLP, BN | Recall, Precision, AUC | Ten-Fold | Friedman, Wilcoxon Signed rank |
| CP12 | CK, 16 Evolution-based metrics | PeerSim (9 versions), VSS-Plugin (13 versions) | LR | Accuracy | Ten-Fold | Wilcoxon Signed Rank |
| CP13 | CK, 16 other class-level metrics | Frinika, FreeMind, OrDrumBox | LR, RF, MLP, BG | Recall, Specificity, AUC | Ten-Fold | — |
| CP14 | CK, SLOC | Apache Abdera (4 versions), Apache POI (4 versions), Apache Rave (4 versions) | LB | Precision, AUC | Ten-Fold, Cross-project | — |

| Study No. | Predictors | Datasets | Data Analysis Techniques | Performance Measures | Validation Method | Statistical Test |
|---|---|---|---|---|---|---|
| CP15 | CK, SLOC | AOI, HyberSQL, Jmeter, Abra, Celestia, Glest | BG, AB, RF, MLP, AIRS, Immunos99, Clonal Selection | Accuracy, AUC | Ten-Fold | T-test |
| CP16 | NOM, DIT, RFC, NOC, CBO, TCC, SLOC | ArgoUmL, Findbugs, FOP, FreeCol | GP | Recall, Precision | —- | Proportion test |
| CP17 | CK | PeerSim, VSSPlugin | LR, MLP, RBF, SVM, DT, GEP, K-means, Ensemble of Models (Best in training, Bagging, Boosting, Majority Voting, Non-linear Decision tree Forest) | Accuracy, AUC | Hold-out, leave-one out | —- |
| CP18 | CK, SLOC, NOM, NIV, NPM, NIM, NOA | Simutrans, Glest, Celestia | LR, RF, BG, MLP, AB, CPSO, HIDER, MPLCS, SUCS, GFS-SP, NNEP | Accuracy, AUC, Precision, Specificity, Recall, F-measure, G-measure | Ten-fold, Cross-project | Friedman |
| CP19 | CK, SLOC | Apache Rave, Apache Math | NB, BN, LB, AB, GFS-AB, GFS-LB, GFS-MaxLB, HIDER, NNEP, PSO-LDA, GFS-GP, GFS-SP, SLAVE | Accuracy, G-mean | Ten-fold | Wilcoxon Signed Rank |
| CP20 | Entropy of changes, Number of developers, Structural and semantic scattering of developers, Evolution-based metrics, OO metrics | ArgoUML, Apache Ant, Apache Cassandra, Apache Xerces, atunes, Freemind, Jedit, Jfreechart, Jhotdraw, Jvlt | LR | Accuracy, AUC, Precision, Recall, F-measure | Dataset divided into 3 month sliding window for training and testing | Wilcoxon test, Cliff's test |

| Study No. | Predictors | Datasets | Data Analysis Techniques | Performance Measures | Validation Method | Statistical Test |
|---|---|---|---|---|---|---|
| CP21 | CK, 16 Evolution-based metrics | VSSPlugin (13 versions) | GMDH | Accuracy, AUC, Precision, Recall, F-measure | Hold-out | — |
| CP22 | 62 OO metrics: 19 cohesion metrics, 19 coupling metrics, 17 inheritance metrics, 7 size metrics | Eclipse | LR, NB, Extreme Machine Learning (Linear, Polynomial & RBF kernels), SVM (Linear, Polynomial & Sigmoid kernels), Ensembles of Techniques (Best in Training, Majority Voting) | Accuracy, AUC | Ten-fold | — |
| CP23 | 21 OO metrics including CK metrics | Ebay Services (5 versions) | Least Square SVM (Linear, Polynomial & RBF kernels) | Accuracy, F-measure | Twenty-fold | T-test |
| CP24 | 61 OO metrics | 10 Eclipse plug-ins | LR, MLP, RBF, DT, RF, Ensembles of Techniques (Best in Training, Majority Voting, Non-Linear Decision Tree Forest) | Accuracy, Precision, Recall, F-measure | Ten-fold, Cross-project | Wilcoxon Signed Rank |
| CP25 | 13 OO metrics including CK | AOI, SweetHome 3D | LR, RF, AB, BG, MLP, NB, BN, J48, NNGE | Recall, Specificity, AUC | Ten-fold, Cross-project | T-test |
| CP26 | 10 OO metrics including CK, Li & Henry, SIZE1 | Ant, Antlr, Argouml, Azureus, Freecol, Freemind, Hibernate, Jgraph, Jmeter, Jstock, Jung, Junit, Lucene, Weka | LB, MLP, RBF, SVM, K-means, CLAMI, CLAMI+ | Accuracy, AUC, F-measure | Within project, Cross-project | Friedman, Nemenyi |

| Study No. | Predictors | Datasets | Data Analysis Techniques | Performance Measures | Validation Method | Statistical Test |
|---|---|---|---|---|---|---|
| CI1 | Change request, Source code components | Library of Efficient Data Types and Algorithms (LEDA) | — | Recall, Precision | Proposed approach validated | — |
| CI2 | Change request, Free text in software repository | Kcalc, Kpdf, Kspread, Firefox | — | Top 1 Recall, Top 30 Recall, Top 1 Precision | Proposed approach is validated | — |
| CI3 | Change request, Source Files | Kcalc, Kpdf, Kspread, Firefox | — | Precision-Recall Curves | Proposed approach is validated | — |
| CI4 | Change request, Free text in software repository | Gedit, Firefox, ArgoUML | — | Recall, Precision | Proposed approach is validated | — |
| CI5 | Change request, Source Code, Historical Information, Dynamic execution trace | ArgoUML, Jabref, Jedit, Mucommander | — | Recall, Precision | Proposed approach is validated | Wilcoxon Signed Rank |
| CI6 | Change request, Source Code,Change Type, Requirements and Design artefacts | On-board Automobile Project | Breadth First Search | Relative Error, Absolute Relative Error, Mean magnitude of Relative Error | Proposed approach is validated | — |
| CI7 | Textually similar change request, Execution trace, Historical data of co-changed entities | ArgoUML | Association Rules | — | Proposed approach is validated | — |

| Study No. | Predictors | Datasets | Data Analysis Techniques | Performance Measures | Validation Method | Statistical Test |
|---|---|---|---|---|---|---|
| CI8 | Change request, Developer interactions and commit histories of source code | Mylyn | K-NN | Recall@k, Precision@k | Training: 90%, Test: 10% | ANOVA |
| Note: "—-" indicates the corresponding information was not found in the study. | | | | | | |

AID: Access of Imported Data; ALD: Access of Local Data; BDM: Behavioral Dependency Measurement; CC: Cyclomatic Complexity; MNOB: Maximum Number Of Branches; NIC: Number of Imported Classes, NIM: Number of Instance Methods; NIV: Number of Instance Variables; NOCI: Number of Times Source File was Inspected; NOLV: Number Of Local Variables, NOP: Number Of Parameters.

# Appendix C

# Review of SBA for developing SEPM

This is an appendix to the review of SBA conducted in Chapter 5.

## C.1 Inclusion & Exclusion Criteria

The inclusion and exclusion criteria of the SBA review conducted is as follows:

- *Inclusion Criteria*: We included studies which used SBA for estimating the four software attributes (effort estimation, defect prediction, maintainability prediction and change prediction). Studies which evaluated and compared various SBA amongst themselves and with other ML techniques for developing SPM were also included.

- *Exclusion Criteria*: We excluded studies which were based on predicting other dependent variables such as reliability or defect count. Also review studies, poster papers, PhD dissertations were excluded. In case a conference paper was extended in a journal, only the journal version of the paper was included.

## C.2 Quality Questions

Table C.1 lists the quality questions for assessing the 112 studies extracted after inclusion and exclusion criteria. It also states the percentage of primary studies acquiring a specific quality grade for each of the 16 quality questions.

Table C.1: Quality Questions

| Q# | Quality Questions | Yes | Partly | No |
|---|---|---|---|---|
| Q1 | Are the aims of the study clearly stated? | 98% | 2% | 0% |
| Q2 | Are the independent variables of the study properly reported? | 79% | 14% | 7% |
| Q3 | Are the size and number of datasets used by the study appropriate? | 82% | 18% | 0% |
| Q4 | Are the data sources and data collection procedure clearly described? | 78% | 18% | 4% |
| Q5 | Is the use of search-based algorithm and its selection justified in the study? | 49% | 40% | 11% |
| Q6 | Does the study use appropriate validation technique for model evaluation? | 93% | 1% | 6% |
| Q7 | Are the parameter settings and fitness functions of the SBA clearly reported? | 81% | 11% | 8% |
| Q8 | Does the study perform appropriate number of runs to account for the stochastic nature of SBA? | 56% | 7% | 37% |
| Q9 | Are the performance measures used by the study clearly reported? | 89% | 9% | 2% |
| Q10 | Are the performance measures appropriate? | 35% | 19% | 46% |
| Q11 | Does the study perform a comparative analysis of the models predicted using different techniques (statistical vs search-based algorithm, ML vs search-based algorithm or amongst SBA)? | 87% | 2% | 11% |
| Q12 | Does the study perform any comparison of SBA with baseline benchmarks? | 58% | 14% | 28% |
| Q13 | Does the study demonstrate the statistical comparison amongst two techniques with the help of statistical tests? | 51% | 0% | 49% |
| Q14 | Are the limitations or threats to validity of the obtained results clearly reported? | 31% | 0% | 69% |
| Q15 | Is the research methodology used by the study repeatable? | 58% | 33% | 9% |
| Q16 | Does the study clearly report its findings? Do the findings relate to the aims of the research? | 91% | 9% | 0% |

## C.3 Data Collection from Different Sources

Figure C.1 depicts the details of the data collection procedure of the review. We categorize our sources into six parts A-F, which are depicted in the figure. The number of studies collected from each source in each phase is represented in parenthesis.

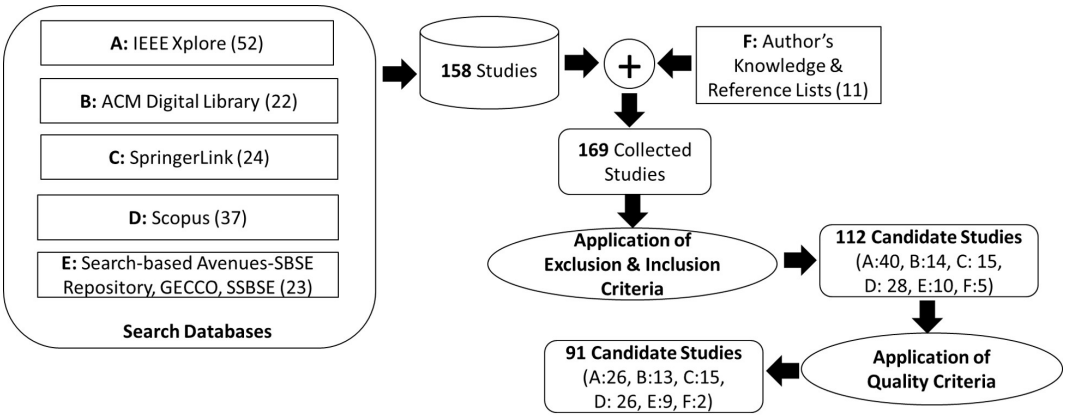


Figure C.1: Data collection of Review Studies

## C.4 Year-wise Distribution of Primary Studies

A year-wise distribution of the 91 primary studies is given in Figure C.2. It may be noted that most number of studies were published in 2010, but there are very few studies before 2001. A possible cause for such a trend could be that the formal term "Search Based Software Engineering" was formulated by Harman and Jones in 2001 [157], thus, leading to rise in the number of studies after 2001.

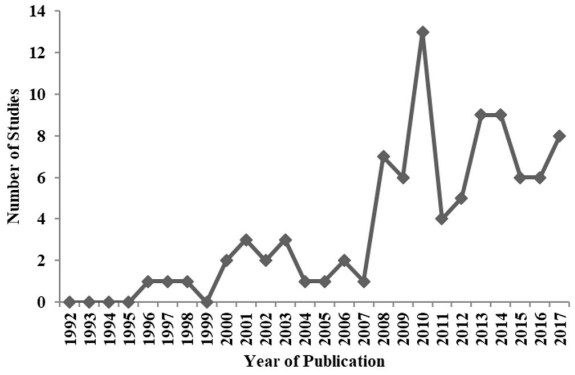


Figure C.2: Year-wise Distribution of Primary Studies using SBA for SEPM

## C.5 Categories of SBA

Table C.2 states the various SBA used by the primary studies, along with the corresponding study numbers according to four categories (Local, Evolutionary, Swarm Intelligence and Hybrid).

Table C.2: SBA used for SEPM

| Category | SBA |
|---|---|
| Local | Hill Climbing [ES19, ES6], Tabu Search [ES18, ES22, ES24]. |
| Evolutionary | GP [ES1, ES2, ES4, ES5, ES7, ES8, ES9, ES10, ES17, ES21, ES23, ES27, ES29, ES31, DS3, DS4, DS5, DS8, DS12, DS15, DS37, CS5], GA [ES11, ES13, ES12, ES14, ES20, ES30, ES42, DS35, CS3, CS10, MS4], Differential Evolution [ES28, ES41], MO Evolutionary Algorithm [ES36], MOGA [ES40, DS23], MOGP [ES31, DS27], GEP [CS8, DS11, DS27], Harmonic Distance MO Evolutionary Algorithm [ES35], Grammatical Evolution [ES33], MPLCS [CS7], SUCS [CS7], Parallel GA [DS35]. |
| Swarm Intelligence | PSO [ES27, ES26, ES32, ES38, ES43, DS14, DS17], MOPSO [ES39, DS7, DS13], Adaptive Dynamic Mean PSO [DS14], CPSO [CS7], Distance Sorting based MOPSO [DS30], ACO [ES17, DS9, CS4], AIRS1 [DS10, DS12, DS24, DS28, CS6], AIRS2 [DS10, DS24], AIRS2P [DS10, DS24], CLG [DS10, DS24], IM1 [DS10, DS24], IM2 [DS10, DS24], IM99 [DS24, DS28, CS6], Clonal Selection Classifier Algorithm [DS24]. |
| Hybrid | GA- ANN [ES3, ES25, DS1, DS2, MS1, MS2, MS3], GA-SVM [ES15, ES25, ES30, DS18, DS21, DS22, DS25, DS32], GP-DT [DS4], Exhaustive Search- Probabilistic Neural Network [DS16], Simulated Annealing- Probabilistic Neural Network [DS16], Evolutionary Programming with Least Square SVM [DS19], Evolutionary Decision Rules for Subgroup Discovery [DS20], PSO-SVM [DS22], PSO-ANN [ES37, DS29], GA-Fuzzy Logic [DS26], GA- Radial Basis Function [ES25, DS32], GA- Model Trees [ES25], Artificial Bee Colony-ANN [DS28], Quantum PSO-ANN [DS29],Quantum PSO-SVM [DS29], HS-NB [DS33], HS-LR [DS33], HS-DT [DS33], GP-NB [DS31], GP-C4.5 [DS31], GA-Grey Relational Analysis [ES16], Tabu-Search-SVM [ES34], NNEP [CS7, CS9, MS2], Evolutionary Fuzzy Rule Learning [MS2], Evolutionary Symbolic Regression Technique [MS2], HIDER [CS7, CS9], GFS-LB [CS9], GFS-MLB [CS9], PSO-LDA [CS9], GFS-AB [CS9], GA-Ensemble Learning [DS34], GA with LR [DS32], GA- Extreme Learning [DS34], GFS-GP [CS9], GFS-SP [CS9], Nearest-Neighbour-GA [DS36]. |
| MO: Multi-Objective; HS: Harmony Search, IM: Immunos; AIRS2P: AIRS2Parallel | |

## C.6 Dataset-wise Outliers for Effort Estimation & Defect Prediction

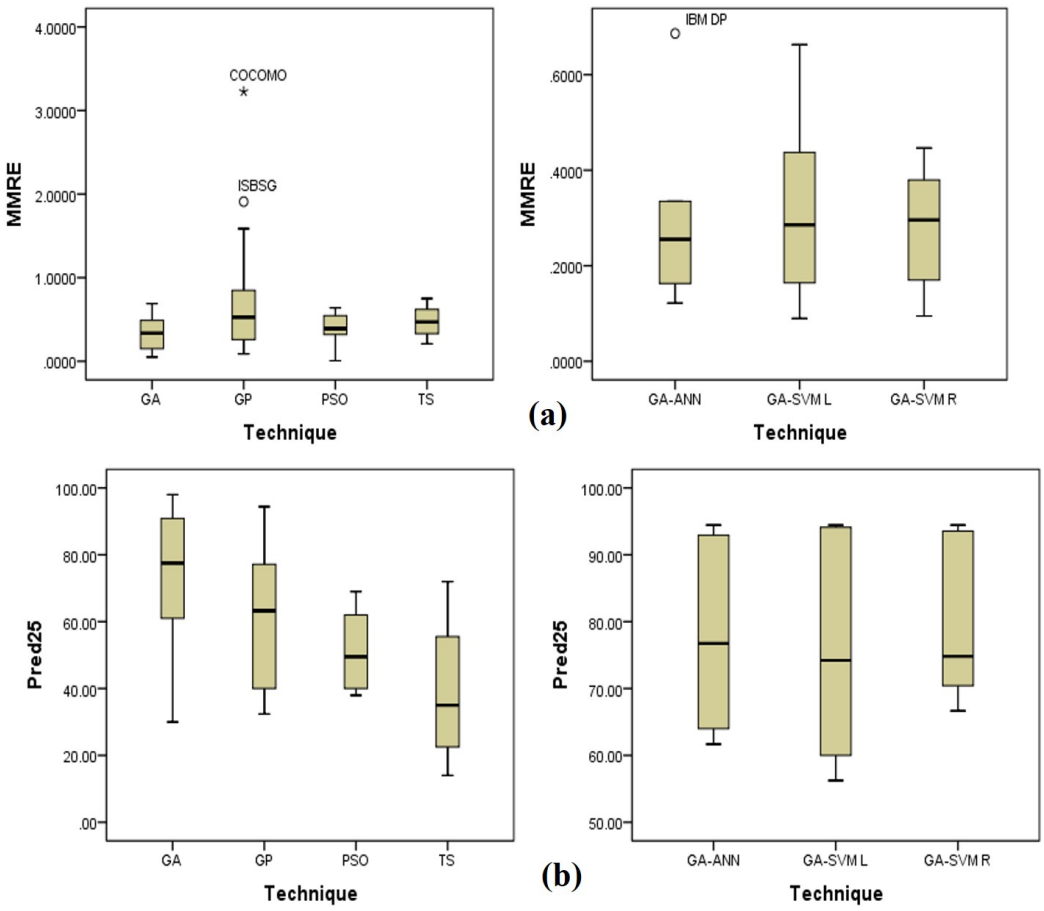


Figure C.3: Outliers for Effort Estimation Models according to Datasets (a) MMRE (b) Pred (25)

Figure C.3 depicts the dataset-wise outliers for MMRE and Pred(25) performance measures corresponding to the effort estimation studies. According to the figure, the GP technique obtained very high MMRE values on the ISBSG and COCOMO datasets and GA-ANN technique obtained a very high MMRE value on the IBM DP

dataset.

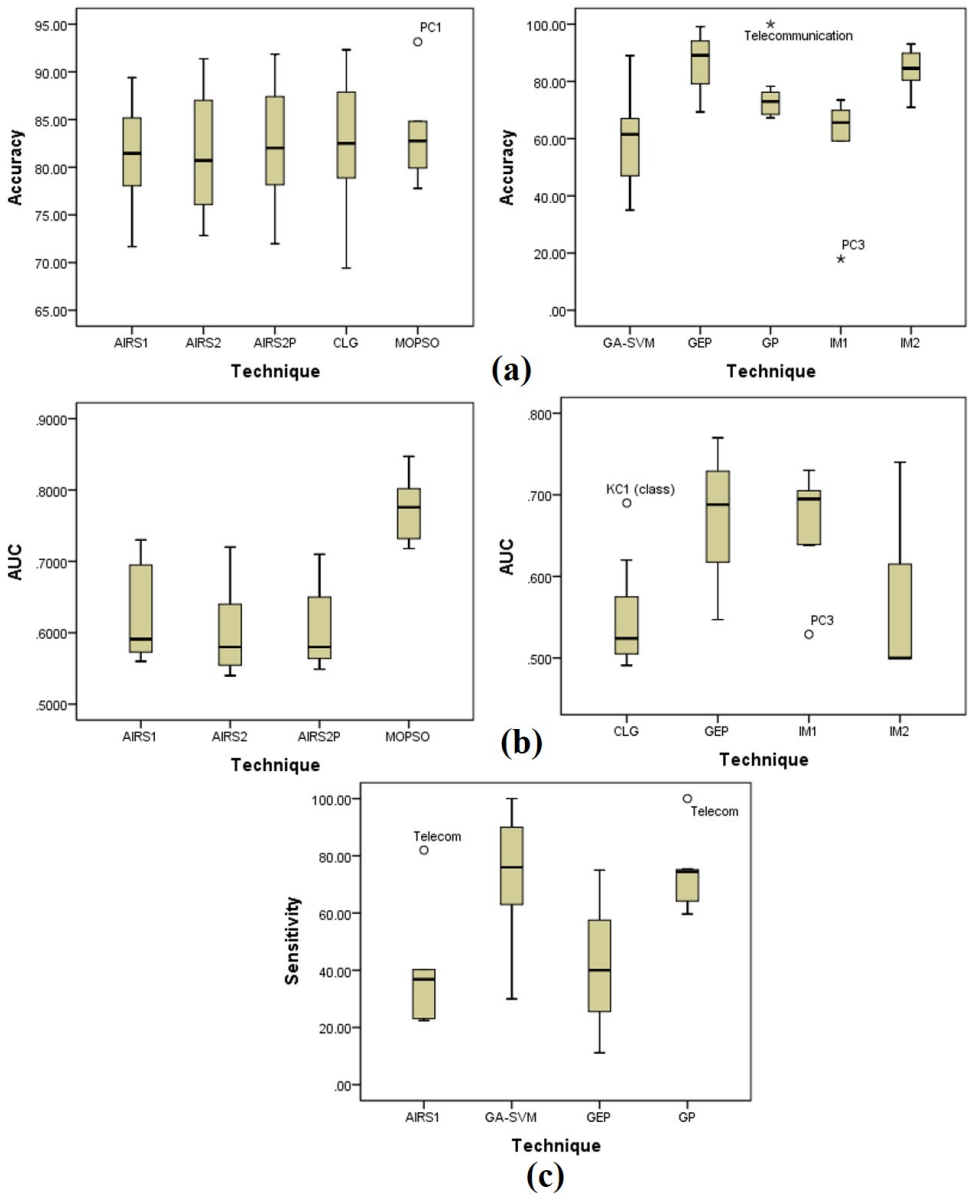


Figure C.4: Outliers for Defect Prediction Models according to DataSets (a) Accuracy (b) AUC (c) Sensitivity

Figure C.4 depicts the dataset-wise outliers for Accuracy, AUC and Sensitivity performance measures corresponding to defect prediction studies. As seen in the fig-

ure, the MOPSO, IM1 and GP had one outlier each according to accuracy values, and the CLG and IM1 techniques also had one outlier each according to AUC values. The Telecommunication dataset was an outlier for AIRS1 and GP technique according to sensitivity values.

## C.7 Threats in Application of SBA to SEPM

Table C.3 lists the conclusion validity threats extracted from the primary studies, along with their supporting studies (SS). It states the possible cause of a threat in the form of a question. If the answer to a corresponding question is "no", the specific threat exists in a study. Each identified conclusion validity threat is allocated a specific number. The table also states the mitigation of the identified conclusion validity threats, which are extracted from the primary studies. Similarly, Tables C.4, C.5 and C.6 states the internal validity threats, construct validity threats and the external validity threats respectively extracted from the primary studies of the review, with their supporting studies (SS) and mitigation.

Table C.3: Conclusion Validity Threats in Primary Studies

| Description & Supporting Studies |
|---|
| **C1: Is there any statistical verification of the results?**<br>***Remedy***: It is important for a study to use an effective statistical test to formally validate the hypothesis of the study.<br>***SS***: ES18, ES22, ES31, ES34, ES36, ES40, ES42, DS12, DS13, DS23, DS27, DS31, DS33, DS34, DS36, CS5. |
| **C2: Does the study verify all the assumptions of the chosen statistical test?**<br>***Remedy***: In case the study uses a parametric test, all the assumptions which are pre-requisites before application of the test should be verified. However, a study may use non-parametric statistical test, which does not require any pre-requisite assumptions on the underlying data.<br>***SS***: ES18, ES22, ES30, ES31, ES34, ES40, DS12, DS23, DS31, DS33, DS34, CS5. |
| **C3: Does the study account for randomness of SBA?**<br>***Remedy***: In order to account for randomness of SBA, they are required to be executed using multiple (at least 10) runs [238]. The study should report either the average or the median of these multiple runs.<br>***SS***: DS21, DS23, DS33, DS35, DS36. |

| Description & Supporting Studies |
|---|
| **C4: Do the results of the study take into account validation bias if any?**<br>***Remedy***: The results of the study could be improper if validation bias is not accounted for. Thus, an appropriate validation technique, such as ten-fold cross validation which takes into account multiple combinations of training and testing data should be used. Other validation approaches such as cross-project scenario [40] are also effective.<br>***SS***: ES22, DS10, DS12, DS13, DS33. |
| **C5: Does the study compare its result with an appropriate baseline benchmark or are the results evaluated by experts?**<br>***Remedy***: The results of the proposed search-based algorithm should either be evaluated by an expert or should be compared with a baseline technique. The increased predictive performance of the proposed techniques is adjudged in terms of added complexity with respect to the baseline technique.<br>***SS***: ES40, DS13, DS34. |

Table C.4: Internal Validity Threats in Primary Studies

| Description & Supporting Studies |
|---|
| **I1: Does the study perform appropriate steps for tuning the parameters of SBA?**<br>***Remedy***: The study should perform appropriate steps to tune the internal parameters of SBA so that optimum results can be obtained. Parameter configurations which have been considered effective by previous studies may be used.<br>***SS***: ES10, ES36, ES42, DS15, DS20, DS35, DS36, DS37. |
| **I2: Does the study take steps to effectively use inconsistent or noisy data?**<br>***Remedy***: The study should use data cleansing techniques such as filters in order to effectively use inconsistent and noisy data.<br>***SS***: ES42, DS10, DS15, DS31, DS34, DS36, DS37. |
| **I3: Does the study use attribute selection methods to eliminate noisy and redundant attributes?**<br>***Remedy***: The study should use attribute selection methods in order to reduce the number of independent attributes and form an effective set of predictors. They help in elimination of noisy and redundant attributes.<br>***SS***: ES38, DS8, DS31, MS3. |
| **I4: Does the study ensure that there is no confounding effect on the relationship between the independent and the dependent variables due to extraneous attributes?**<br>***Remedy***: In order to ensure that the confounding nature of an extraneous variable does not lead to misjudgment of the relationship between the dependent and the independent variables, the extraneous variable should be controlled and the effect of this variable's presence should be studied on the relationship between the independent and the dependent variables. However, such controlled experiments are difficult to perform in practice.<br>***SS***: DS12, DS27, DS30, DS37, CS8, CS10. |

Table C.5: Construct Validity Threats in Primary Studies

| Description & Supporting Studies |
|---|
| **CT1: Does the study use effective representatives of the concepts which represent independent variables?**<br>***Remedy***: The study should use well-established metrics in literature for representing independent variables as they have been previously validated by literature studies. The use of well-known public datasets reduces this threat as the metrics (independent variables) used by the datasets are already assessed for their effectiveness by previous studies.<br>***SS***: ES18, ES22, ES30, ES31, ES34, ES40, ES42, DS10, DS15, DS20, DS21, DS23, DS31, DS36, DS37, CS5, CS6, CS9, MS3. |
| **CT2: Does the study use performance measures which are effective representatives of the capability of the developed models?**<br>***Remedy***: The study should use efficient performance measures for assessing the developed models. For instance, performance measures such as G-Mean1, Balance and AUC are efficient performance evaluators for defect prediction and change prediction models. They give a realistic estimate of the developed model even if the class distribution is imbalanced [117, 118]. Other metrics (such as accuracy, precision, recall) generally provide a biased optimistic estimate which leads to misleading results.<br>***SS***: ES36, ES42, DS10, DS12, DS23, DS31, DS33, DS34, DS35, DS36, DS37. |
| **CT3: Does the study take necessary steps in order to overcome human errors while data collection?**<br>***Remedy***: The study should use well-known automated or semi-automated tools for collecting data. Moreover, the use of public datasets which have already been verified by previous studies also reduces this risk.<br>***SS***: ES10, DS31, DS34, CS10, MS1. |

Table C.6: External Validity Threats in Primary Studies

| Description & Supporting Studies |
|---|
| **ET1: Are the datasets used by the study real industrial datasets?**<br>***Remedy***: The study should use real industrial datasets so that the practical relevance of the results can be assessed. In case, industrial datasets could not be collected, datasets that resemble industrial projects or properties may be used.<br>***SS***: ES10, ES18, ES22, ES34, DS10, DS12, DS13, DS15, DS20, DS21, DS31, DS34, DS36, DS37. |
| **ET2: Are the datasets used in the study appropriate in number?**<br>***Remedy***: The study should use multiple datasets for empirical validation so that the results are easily generalized.<br>***SS***: ES18, ES31, ES36, ES38, DS12, DS20, DS23, DS33, DS34, DS35, DS37, DS36, CS5, CS9, MS1. |
| **ET3: Does the study use appropriately sized or varied sized datasets?**<br>***Remedy***: The study should use medium or large sized datasets so that the results are practical and relevant. Also, datasets with varying sizes should be evaluated to generalize the results.<br>***SS***: ES18, ES22, ES30, ES31, ES34, ES40, DS10, DS13, DS15, DS31, DS35. |

| Description & Supporting Studies |
| --- |
| **ET4: Are the datasets used in the study developed using different programming languages?**<br>***Remedy***: The use of mono-language datasets for empirical validation decreases the generalizability of the results. The effectiveness of the results can be increased by performing experiments on datasets developed using different programming languages such as Java, C, C++ etc.<br>***SS***: DS15, DS21, DS36, CS5, CS6, MS3. |
| **ET5: Do the datasets used in the study belong to different companies/domains or have different characteristics?**<br>***Remedy***: In order to validate the effectiveness of the results on different companies and domains and increase the external validity of the results, it is important for the study to use datasets belonging to different companies/domains or with datasets collected from projects exhibiting different characteristics.<br>***SS***: ES18, ES22, ES30, ES31, ES36, ES40, ES42, DS12, DS20, DS21, DS31, DS35, DS37, CS9. |
| **ET6: Does the study provide proper details for replicability?**<br>***Remedy***: If a study uses publicly available datasets, the replicability of the study increases. Also, the study should specify the parameter settings and fitness functions of the employed techniques so that the results can be replicated easily by the research community.<br>***SS***: ES36, DS31, DS37, CS5. |

# Appendix D

# Imbalanced Learning

This is an appendix to the experiments conducted in Chapter 10.

## D.1 Ten-fold Cross Validation Results using Sampling Approaches

Tables D.1 and D.2 report the recall and precision values exhibited by the six ML techniques on the six datasets of the chapter after application of three sampling methods (Resample with Replacement, SMOTE and Spread Subsample) and when no sampling approach is used. An analysis of Table D.1 indicates extremely low recall values (5%-45%) in majority of the cases on all the datasets when no sampling method is used. A similar trend was shown by precision values as indicated in Table D.2. The table shows low precision values (14%-65%) in majority of the cases when no sampling approach was used. These trends are a result of imbalanced data. As the datasets have very few instances of change-prone classes, the models may not be able to learn to distinguish them efficiently, this would result in low recall

Table D.1: Recall Results using Different Sampling Methods

| ML Tech. | Net | | | | ML Tech. | IO | | | | ML Tech. | Log4j | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *Re.S* | *SMOTE* | *Sub.S* | *NSamp.* | | *Re.S* | *SMOTE* | *Sub.S* | *NSamp.* | | *Re.S* | *SMOTE* | *Sub.S* | *NSamp.* |
| **AB** | 80.53 | 94.07 | 61.02 | 61.02 | **AB** | 88.79 | 60.00 | 45.45 | 27.27 | **AB** | 61.91 | 59.21 | 22.81 | 24.56 |
| **RF** | 94.10 | 90.68 | 62.71 | 50.85 | **RF** | 99.89 | 60.00 | 63.64 | 27.27 | **RF** | 87.21 | 74.12 | 31.58 | 26.32 |
| **BG** | 87.89 | 91.53 | 55.93 | 55.93 | **BG** | 97.58 | 49.09 | 63.64 | 9.09 | **BG** | 81.91 | 75.00 | 17.54 | 19.30 |
| **LB** | 87.68 | 91.53 | 54.24 | 57.63 | **LB** | 94.62 | 58.18 | 54.55 | 27.27 | **LB** | 71.91 | 71.49 | 22.81 | 26.32 |
| **NB** | 56.42 | 54.66 | 38.98 | 32.20 | **NB** | 71.54 | 29.09 | 45.45 | 27.27 | **NB** | 37.79 | 30.26 | 10.53 | 14.04 |
| **MLP** | 89.05 | 88.56 | 45.76 | 40.68 | **MLP** | 94.73 | 38.18 | 54.55 | 27.27 | **MLP** | 64.41 | 53.51 | 22.81 | 17.54 |
| **ML Tech.** | **Bluetooth** | | | | **ML Tech.** | **Calendar** | | | | **ML Tech.** | **MMS** | | | |
| | *Re.S* | *SMOTE* | *Sub.S* | *NSamp.* | | *Re.S* | *SMOTE* | *Sub.S* | *NSamp.* | | *Re.S* | *SMOTE* | *Sub.S* | *NSamp.* |
| **AB** | 94.48 | 80.00 | 81.82 | 9.09 | **AB** | 67.46 | 78.43 | 17.65 | 17.65 | **AB** | 90.69 | 96.10 | 68.09 | 51.06 |
| **RF** | 98.28 | 83.64 | 81.82 | 27.27 | **RF** | 77.27 | 67.65 | 17.65 | 5.88 | **RF** | 97.21 | 95.74 | 44.68 | 40.43 |
| **BG** | 87.93 | 78.18 | 81.82 | 9.09 | **BG** | 74.55 | 68.63 | 11.76 | 17.65 | **BG** | 92.21 | 96.10 | 55.32 | 40.43 |
| **LB** | 97.59 | 85.45 | 81.82 | 27.27 | **LB** | 78.73 | 67.65 | 17.65 | 17.65 | **LB** | 88.60 | 95.39 | 61.70 | 38.30 |
| **NB** | 66.55 | 70.91 | 81.82 | 36.36 | **NB** | 22.36 | 30.39 | 29.41 | 29.41 | **NB** | 86.51 | 83.69 | 72.34 | 70.21 |
| **MLP** | 81.15 | 89.09 | 72.73 | 27.27 | **MLP** | 56.73 | 51.96 | 17.65 | 17.65 | **MLP** | 93.23 | 93.62 | 51.06 | 44.68 |
| Re.S indicates Resample with Replacement, SubS. indicates Subsample & NSamp. indicates No Sample | | | | | | | | | | | | | | |

Table D.2: Precision Results using Different Sampling Methods

| ML Tech. | Net | | | | ML Tech. | IO | | | | ML Tech. | Log4j | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *Re.S* | *SMOTE* | *Sub.S* | *NSamp.* | | *Re.S* | *SMOTE* | *Sub.S* | *NSamp.* | | *Re.S* | *SMOTE* | *Sub.S* | *NSamp.* |
| **AB** | 75.63 | 80.14 | 65.45 | 61.02 | **AB** | 76.97 | 78.57 | 71.43 | 37.50 | **AB** | 81.50 | 78.03 | 68.42 | 66.67 |
| **RF** | 95.03 | 86.29 | 61.67 | 57.69 | **RF** | 96.53 | 75.00 | 77.78 | 27.27 | **RF** | 89.58 | 86.22 | 48.65 | 44.12 |
| **BG** | 88.46 | 85.38 | 67.35 | 63.46 | **BG** | 91.02 | 79.41 | 87.50 | 100.00 | **BG** | 86.72 | 81.82 | 62.50 | 61.11 |
| **LB** | 85.95 | 83.72 | 64.00 | 64.15 | **LB** | 84.82 | 76.19 | 66.67 | 50.00 | **LB** | 84.02 | 79.13 | 68.42 | 68.18 |
| **NB** | 79.76 | 86.58 | 67.65 | 61.29 | **NB** | 76.35 | 61.54 | 83.33 | 42.86 | **NB** | 87.23 | 71.88 | 40.00 | 47.06 |
| **MLP** | 87.46 | 78.57 | 75.00 | 66.67 | **MLP** | 74.50 | 95.45 | 100.00 | 100.00 | **MLP** | 86.11 | 67.40 | 100.00 | 62.50 |
| **ML Tech.** | **Bluetooth** | | | | **ML Tech.** | **Calendar** | | | | **ML Tech.** | **MMS** | | | |
| | *Re.S* | *SMOTE* | *Sub.S* | *NSamp.* | | *Re.S* | *SMOTE* | *Sub.S* | *NSamp.* | | *Re.S* | *SMOTE* | *Sub.S* | *NSamp.* |
| **AB** | 94.83 | 78.57 | 90.00 | 14.29 | **AB** | 76.38 | 57.55 | 75.00 | 75.00 | **AB** | 81.57 | 84.42 | 57.14 | 45.28 |
| **RF** | 94.59 | 82.14 | 81.82 | 33.33 | **RF** | 80.16 | 87.34 | 42.86 | 50.00 | **RF** | 95.79 | 90.30 | 53.85 | 47.50 |
| **BG** | 92.03 | 82.69 | 81.82 | 50.00 | **BG** | 78.21 | 74.47 | 66.67 | 50.00 | **BG** | 90.13 | 87.42 | 66.67 | 67.86 |
| **LB** | 95.04 | 87.04 | 81.82 | 33.33 | **LB** | 76.22 | 83.13 | 75.00 | 75.00 | **LB** | 84.98 | 84.86 | 58.00 | 52.94 |
| **NB** | 95.94 | 84.78 | 90.00 | 50.00 | **NB** | 52.70 | 72.09 | 45.45 | 41.67 | **NB** | 79.54 | 84.59 | 52.31 | 47.14 |
| **MLP** | 89.56 | 77.78 | 72.73 | 50.00 | **MLP** | 76.39 | 56.99 | 75.00 | 75.00 | **MLP** | 85.37 | 87.13 | 64.86 | 56.76 |
| Re.S indicates Resample with Replacement, SubS. indicates Subsample & NSamp. indicates No Sample | | | | | | | | | | | | | | |

and precision values. However, in majority of cases when any sampling approach was used, the recall values ranged from 55%-98% (Table D.1), indicating a positive improvement in the correct prediction of change-prone classes. Similarly, the precision values too showed a positive improvement as they ranged from 70%-96% in the majority of the cases (Table D.2).

## D.2 Ten-fold Cross Validation Results using MetaCost Learners

Table D.3 and D.4 report the recall and precision values obtained on application of all the six ML techniques on each dataset of the chapter, with the use of different CR's and without any cost-sensitive learning ("original"). There was a sharp increase in recall values in all the datasets with the use of MetaCost learners. This is because the MetaCost learners were used in such a way so as to penalize the classifier for FN predictions. With low FN predictions, the recall increases. This was done as the datasets were highly imbalanced with few change-prone classes. Thus, it was important to increase recall rates to achieve a balanced classifier. However, one should be careful that apart from recall, the precision values should also be considered. As shown in Table D.4, there is decrease in precision values in majority of the cases. This is a concerning factor as increase in recall should not mean that a number of not change-prone classes are incorrectly classified as change-prone. One should achieve an optimum balance between recall and precision. It should be a researcher's objective that both change-prone and not change-prone classes should be correctly identified. For example, result of CR=5 in Bluetooth and MMS datasets show comparable precision values. Table D.5 reports the cost of the model for each CR ratio, which was considered while choosing the best CR ratio for a specific dataset.

Table D.3: Recall Results of MetaCost Learners using ML techniques

| ML Tech. | Net | | | | | | | | ML Tech. | Log4j | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *Org.* | *CR5* | *CR10* | *CR15* | CR20 | *CR25* | *CR30* | *CR50* | | *Org.* | *CR5* | *CR10* | *CR15* | CR20 | *CR25* | *CR30* | *CR50* |
| **AB** | 61.02 | 83.05 | 89.83 | 96.61 | 96.61 | 96.61 | 96.61 | 98.31 | **AB** | 24.56 | 59.65 | 87.72 | 94.74 | 100.00 | 100.00 | 100.00 | 100.00 |
| **RF** | 50.85 | 74.58 | 83.05 | 84.75 | 86.44 | 89.83 | 89.83 | 96.61 | **RF** | 26.32 | 52.63 | 63.16 | 66.67 | 64.91 | 66.67 | 68.42 | 73.68 |
| **BG** | 55.93 | 79.66 | 86.44 | 91.53 | 93.22 | 100.00 | 100.00 | 100.00 | **BG** | 19.30 | 43.86 | 89.47 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 |
| **LB** | 57.63 | 83.05 | 94.92 | 96.61 | 98.31 | 96.61 | 96.61 | 98.31 | **LB** | 26.32 | 56.14 | 85.96 | 96.49 | 100.00 | 100.00 | 100.00 | 100.00 |
| **NB** | 32.20 | 52.54 | 72.88 | 81.36 | 88.14 | 91.53 | 91.53 | 93.22 | **NB** | 14.04 | 50.88 | 78.95 | 92.98 | 92.98 | 96.49 | 96.49 | 98.25 |
| **MLP** | 40.68 | 79.66 | 94.92 | 94.92 | 94.92 | 100.00 | 100.00 | 100.00 | **MLP** | 17.54 | 63.16 | 98.25 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 |
| **ML Tech.** | **IO** | | | | | | | | **ML Tech.** | **MMS** | | | | | | | |
| | *Org.* | *CR5* | *CR10* | *CR15* | CR20 | *CR25* | *CR30* | *CR50* | | *Org.* | *CR5* | *CR10* | *CR15* | CR20 | *CR25* | *CR30* | *CR50* |
| **AB** | 27.27 | 36.36 | 45.45 | 54.55 | 54.55 | 54.55 | 54.55 | 63.64 | **AB** | 51.06 | 59.65 | 87.72 | 94.74 | 100.00 | 100.00 | 100.00 | 100.00 |
| **RF** | 27.27 | 27.27 | 45.45 | 45.45 | 45.45 | 45.45 | 45.45 | 45.45 | **RF** | 40.43 | 52.63 | 63.16 | 66.67 | 64.91 | 66.67 | 68.42 | 73.68 |
| **BG** | 9.09 | 27.27 | 45.45 | 45.45 | 45.45 | 45.45 | 54.55 | 90.91 | **BG** | 40.43 | 43.86 | 89.47 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 |
| **LB** | 27.27 | 36.36 | 45.45 | 45.45 | 45.45 | 63.64 | 63.64 | 63.64 | **LB** | 38.30 | 56.14 | 85.96 | 96.49 | 100.00 | 100.00 | 100.00 | 100.00 |
| **NB** | 27.27 | 54.55 | 81.82 | 81.82 | 81.82 | 81.82 | 81.82 | 81.82 | **NB** | 70.21 | 50.88 | 78.95 | 92.98 | 92.98 | 96.49 | 96.49 | 98.25 |
| **MLP** | 27.27 | 27.27 | 63.64 | 72.73 | 81.82 | 81.82 | 81.82 | 81.82 | **MLP** | 44.68 | 63.16 | 98.25 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 |
| **ML Tech.** | **Bluetooth** | | | | | | | | **ML Tech.** | **Calendar** | | | | | | | |
| | *Org.* | *CR5* | *CR10* | *CR15* | CR20 | *CR25* | *CR30* | *CR50* | | *Org.* | *CR5* | *CR10* | *CR15* | CR20 | *CR25* | *CR30* | *CR50* |
| **AB** | 9.09 | 72.73 | 72.73 | 72.73 | 72.73 | 81.82 | 81.82 | 81.82 | **AB** | 17.65 | 41.18 | 94.12 | 94.12 | 94.12 | 100.00 | 100.00 | 100.00 |
| **RF** | 27.27 | 72.73 | 80.00 | 81.82 | 81.82 | 81.82 | 81.82 | 81.82 | **RF** | 5.88 | 41.18 | 70.59 | 82.35 | 82.35 | 76.47 | 76.47 | 76.47 |
| **BG** | 9.09 | 72.73 | 81.82 | 81.82 | 81.82 | 81.82 | 90.91 | 100.00 | **BG** | 17.65 | 29.41 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 |
| **LB** | 27.27 | 81.82 | 72.73 | 72.73 | 72.73 | 72.73 | 72.73 | 81.82 | **LB** | 17.65 | 41.18 | 88.24 | 94.12 | 94.12 | 94.12 | 94.12 | 100.00 |
| **NB** | 36.36 | 72.73 | 72.73 | 72.73 | 72.73 | 72.73 | 81.82 | 90.91 | **NB** | 29.41 | 29.41 | 47.06 | 70.59 | 100.00 | 100.00 | 100.00 | 100.00 |
| **MLP** | 27.27 | 72.73 | 81.82 | 81.82 | 90.91 | 90.91 | 90.91 | 90.91 | **MLP** | 17.65 | 29.41 | 88.24 | 100.00 | 100.00 | 100.00 | 100.00 | 100.00 |
| Org. indicates Original; CRXX indicates Cost Ratio XX | | | | | | | | | | | | | | | | | |

Table D.4: Precision Results of MetaCost Learners using ML techniques

| ML Tech. | Net | | | | | | | | ML Tech. | Log4j | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *Org.* | *CR5* | *CR10* | *CR15* | CR20 | *CR25* | *CR30* | *CR50* | | *Org.* | *CR5* | *CR10* | *CR15* | CR20 | *CR25* | *CR30* | *CR50* |
| **AB** | 61.02 | 44.55 | 39.26 | 39.04 | 38.51 | 38.00 | 38.00 | 38.16 | **AB** | 66.67 | 25.19 | 22.62 | 19.15 | 20.00 | 20.00 | 20.00 | 20.00 |
| **RF** | 57.69 | 47.83 | 43.75 | 42.37 | 41.13 | 40.77 | 40.77 | 42.22 | **RF** | 44.12 | 27.03 | 23.23 | 21.97 | 20.56 | 20.43 | 20.10 | 19.27 |
| **BG** | 63.46 | 43.12 | 39.53 | 38.85 | 37.67 | 32.78 | 32.78 | 29.35 | **BG** | 61.11 | 26.88 | 21.43 | 20.00 | 20.00 | 20.00 | 20.00 | 20.00 |
| **LB** | 64.15 | 43.36 | 41.79 | 39.58 | 39.46 | 37.75 | 37.75 | 29.74 | **LB** | 68.18 | 30.19 | 22.17 | 20.07 | 20.00 | 20.00 | 20.00 | 20.00 |
| **NB** | 61.29 | 60.78 | 53.75 | 46.15 | 45.22 | 42.52 | 42.52 | 42.31 | **NB** | 47.06 | 28.16 | 19.82 | 19.78 | 19.63 | 20.15 | 19.71 | 20.00 |
| **MLP** | 66.67 | 46.08 | 43.41 | 37.58 | 34.15 | 32.24 | 32.24 | 29.95 | **MLP** | 62.50 | 29.03 | 20.90 | 20.00 | 20.00 | 20.00 | 20.00 | 20.00 |
| **ML Tech.** | **IO** | | | | | | | | **ML Tech.** | **MMS** | | | | | | | |
| | *Org.* | *CR5* | *CR10* | *CR15* | CR20 | *CR25* | *CR30* | *CR50* | | *Org.* | *CR5* | *CR10* | *CR15* | CR20 | *CR25* | *CR30* | *CR50* |
| **AB** | 37.50 | 25.00 | 26.32 | 20.69 | 14.29 | 13.64 | 13.33 | 11.67 | **AB** | 45.28 | 45.28 | 87.76 | 39.47 | 39.13 | 38.98 | 38.33 | 36.22 |
| **RF** | 27.27 | 14.29 | 20.00 | 17.24 | 16.13 | 14.71 | 12.82 | 11.11 | **RF** | 47.50 | 47.50 | 41.90 | 40.00 | 89.80 | 39.64 | 39.29 | 38.33 |
| **BG** | 100.00 | 27.27 | 29.41 | 20.00 | 16.13 | 11.36 | 8.33 | 6.02 | **BG** | 67.86 | 67.86 | 41.12 | 38.46 | 38.14 | 39.50 | 38.52 | 32.64 |
| **LB** | 50.00 | 30.77 | 20.00 | 14.71 | 10.20 | 11.86 | 10.45 | 6.54 | **LB** | 52.94 | 52.94 | 40.57 | 37.29 | 36.00 | 35.71 | 36.51 | 33.58 |
| **NB** | 42.86 | 13.95 | 15.79 | 15.52 | 14.75 | 14.06 | 13.43 | 12.33 | **NB** | 47.14 | 47.14 | 42.86 | 43.01 | 41.67 | 42.86 | 42.86 | 45.65 |
| **MLP** | 100.00 | 33.33 | 17.95 | 13.79 | 12.86 | 12.16 | 10.11 | 8.57 | **MLP** | 56.76 | 56.76 | 40.91 | 38.94 | 39.13 | 39.13 | 39.66 | 38.98 |
| **ML Tech.** | **Bluetooth** | | | | | | | | **ML Tech.** | **Calendar** | | | | | | | |
| | *Org.* | *CR5* | *CR10* | *CR15* | CR20 | *CR25* | *CR30* | *CR50* | | *Org.* | *CR5* | *CR10* | *CR15* | CR20 | *CR25* | *CR30* | *CR50* |
| **AB** | 14.29 | 40.00 | 32.00 | 30.77 | 29.63 | 30.00 | 28.13 | 25.00 | **AB** | 76.38 | 46.67 | 16.33 | 16.16 | 16.16 | 17.00 | 17.00 | 17.00 |
| **RF** | 33.33 | 47.06 | 27.59 | 29.03 | 27.27 | 26.47 | 25.71 | 25.00 | **RF** | 80.16 | 24.14 | 16.22 | 17.50 | 17.07 | 15.66 | 15.48 | 15.12 |
| **BG** | 50.00 | 40.00 | 31.03 | 28.13 | 21.43 | 16.36 | 16.13 | 16.92 | **BG** | 78.21 | 26.32 | 17.00 | 17.00 | 17.00 | 17.00 | 17.00 | 17.00 |
| **LB** | 33.33 | 37.50 | 34.78 | 27.59 | 27.59 | 23.53 | 21.62 | 20.93 | **LB** | 76.22 | 36.84 | 17.05 | 16.49 | 16.16 | 16.16 | 16.16 | 17.00 |
| **NB** | 50.00 | 53.33 | 42.11 | 40.00 | 38.10 | 38.10 | 39.13 | 40.00 | **NB** | 52.70 | 25.00 | 22.86 | 16.44 | 17.00 | 17.00 | 17.00 | 17.00 |
| **MLP** | 50.00 | 38.10 | 30.00 | 25.71 | 24.39 | 23.26 | 22.22 | 19.23 | **MLP** | 76.39 | 25.00 | 15.46 | 17.00 | 17.00 | 17.00 | 17.00 | 17.00 |

Org. indicates Original; CRXX indicates Cost Ratio XX

Table D.5: Cost Values of MetaCost Learners using ML Techniques

| ML Tech. | Net | | | | | | | ML Tech. | Log4j | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *CR5* | *CR10* | *CR15* | CR20 | *CR25* | *CR30* | *CR50* | | *CR5* | *CR10* | *CR15* | CR20 | *CR25* | *CR30* | *CR50* |
| **AB** | 111 | 142 | 119 | 131 | 143 | 153 | 144 | **AB** | 216 | 241 | 273 | 228 | 228 | 228 | 228 |
| **RF** | 123 | 163 | 203 | 233 | 227 | 257 | 178 | **RF** | 216 | 329 | 420 | 543 | 623 | 695 | 926 |
| **BG** | 122 | 158 | 160 | 171 | 121 | 121 | 142 | **BG** | 228 | 247 | 228 | 228 | 228 | 228 | 228 |
| **LB** | 114 | 108 | 117 | 109 | 144 | 154 | 187 | **LB** | 199 | 252 | 249 | 228 | 228 | 228 | 228 |
| **NB** | 160 | 197 | 221 | 203 | 198 | 223 | 275 | **NB** | 214 | 302 | 275 | 297 | 268 | 284 | 274 |
| **MLP** | 115 | 103 | 138 | 168 | 124 | 124 | 138 | **MLP** | 193 | 222 | 228 | 228 | 228 | 228 | 228 |
| **ML Tech.** | **IO** | | | | | | | **ML Tech.** | **MMS** | | | | | | |
| | *CR5* | *CR10* | *CR15* | CR20 | *CR25* | *CR30* | *CR50* | | *CR5* | *CR10* | *CR15* | CR20 | *CR25* | *CR30* | *CR50* |
| **AB** | 47 | 74 | 98 | 136 | 163 | 189 | 253 | **AB** | 144 | 46 | 99 | 110 | 97 | 104 | 131 |
| **RF** | 58 | 80 | 114 | 146 | 179 | 214 | 340 | **RF** | 161 | 91 | 111 | 65 | 142 | 158 | 124 |
| **BG** | 48 | 72 | 110 | 146 | 189 | 216 | 206 | **BG** | 149 | 93 | 102 | 113 | 72 | 75 | 97 |
| **LB** | 44 | 80 | 119 | 164 | 152 | 180 | 300 | **LB** | 161 | 103 | 119 | 120 | 131 | 110 | 141 |
| **NB** | 62 | 68 | 79 | 92 | 105 | 118 | 164 | **NB** | 157 | 132 | 158 | 196 | 181 | 206 | 300 |
| **MLP** | 46 | 72 | 95 | 101 | 115 | 140 | 196 | **MLP** | 146 | 85 | 114 | 110 | 120 | 100 | 122 |
| **ML Tech.** | **Bluetooth** | | | | | | | **ML Tech.** | **Calendar** | | | | | | |
| | *CR5* | *CR10* | *CR15* | CR20 | *CR25* | *CR30* | *CR50* | | *CR5* | *CR10* | *CR15* | CR20 | *CR25* | *CR30* | *CR50* |
| **AB** | 27 | 47 | 63 | 79 | 71 | 83 | 127 | **AB** | 58 | 92 | 98 | 103 | 83 | 83 | 83 |
| **RF** | 24 | 41 | 52 | 64 | 75 | 86 | 127 | **RF** | 72 | 112 | 111 | 128 | 170 | 191 | 273 |
| **BG** | 27 | 40 | 53 | 73 | 96 | 82 | 54 | **BG** | 74 | 83 | 83 | 83 | 83 | 83 | 83 |
| **LB** | 25 | 45 | 66 | 81 | 101 | 119 | 134 | **LB** | 62 | 93 | 96 | 103 | 108 | 113 | 83 |
| **NB** | 22 | 41 | 57 | 73 | 88 | 74 | 65 | **NB** | 75 | 117 | 136 | 83 | 83 | 83 | 83 |
| **MLP** | 28 | 41 | 56 | 51 | 58 | 65 | 92 | **MLP** | 75 | 102 | 83 | 83 | 83 | 83 | 83 |
| CRXX indicates Cost Ratio XX | | | | | | | | | | | | | | | |

# Appendix E

# Evolution Patterns of OO Metrics

This is an Appendix to Chapter 11, which analyzes the evolution patterns of OO metrics.

## E.1 Observed Median Values of OO metrics

Tables E.1, E.2 and E.3 respectively state the median values of metrics analyzed for size, cohesion and coupling dimensions respectively over all the five investigated versions (4.0.2-4.3.1) of Android software for CC and UCC. As stated in Chapter 11, the trends were analyzed in Gallery2 and Contacts package.

As observed in table E.1, the median values of size metrics depict higher values for CC as compared to UCC. For instance, the median WMC for CC was 13 in the Gallery2 package, while it was only 6 for UCC. The trends were similar for both the metrics in both the investigated application packages.

Table E.1: Median values of Size Metrics

| Application package | WMC (CC) | WMC (UCC) | SLOC (CC) | SLOC (UCC) |
|---|---|---|---|---|
| Gallery2 | 13 | 6 | 83 | 33 |
| Contacts | 10 | 5 | 67 | 30 |

As depicted in table E.2, there was a large difference in the median LCOM values of UCC (15) for the Gallery2 application package as compared to the LCOM values of CC (78). Similar trend was observed in Contacts application package.

Table E.2: Median values of Cohesion Metrics

| Application package | LCOM (CC) | LCOM (UCC) |
|---|---|---|
| Gallery2 | 78 | 15 |
| Contacts | 45 | 10 |

As depicted in table E.3, the CC exhibited higher coupling values than the UCC in both the application packages.

Table E.3: Median values of Coupling Metrics

| Application package | CBO (CC) | CBO (UCC) | RFC (CC) | RFC (UCC) |
|---|---|---|---|---|
| Gallery2 | 3 | 2 | 14 | 7 |
| Contacts | 1 | 0 | 11 | 6 |

# Bibliography

[1] M. O. Elish and M. Al-Rahman Al-Khiaty, "A suite of metrics for quantifying historical changes to predict future change-prone classes in object-oriented software," *Journal of Software: Evolution and Process*, vol. 25, no. 5, pp. 407–437, 2013.

[2] A. G. Koru and H. Liu, "Identifying and characterizing change-prone classes in two large-scale open-source products," *Journal of Systems and Software*, vol. 80, no. 1, pp. 63–73, 2007.

[3] A. G. Koru and J. Tian, "Comparing high-change modules and modules with the highest measurement values in two large-scale open-source products," *IEEE Transactions on Software Engineering*, vol. 31, no. 8, pp. 625–642, 2005.

[4] D. Romano and M. Pinzger, "Using source code metrics to predict change-prone java interfaces," *Proceedings of the 27th IEEE International Conference on Software Maintenance (ICSM)*, pp. 303–312, 2011.

[5] R. Malhotra and M. Khanna, "Investigation of relationship between object-oriented metrics and change proneness," *International Journal of Machine Learning and Cybernetics*, vol. 4, no. 4, pp. 273–286, 2013.

[6] M. Harman, P. McMinn, J. T. De Souza, and S. Yoo, "Search based software engineering: Techniques, taxonomy, tutorial," *Empirical Software Engineering and Verification*, vol. 7007, pp. 1–59, 2012.

[7] M. Harman and J. Clark, "Metrics are fitness functions too," *Proceedings of the 10th International Symposium on Software Metrics*, pp. 58–69, 2004.

[8] Y. Singh and R. Malhotra, *Object-oriented software engineering*. PHI Learning Pvt. Ltd., 2012.

[9] R. Malhotra, *Empirical research in software engineering: concepts, analysis, and applications*. CRC Press, 2016.

[10] J. Wen, S. Li, Z. Lin, Y. Hu, and C. Huang, "Systematic literature review of machine learning based software development effort estimation models," *Information and Software Technology*, vol. 54, no. 1, pp. 41–59, 2012.

[11] E. N. Regolin, G. A. de Souza, A. R. Pozo, and S. R. Vergilio, "Exploring machine learning techniques for software size estimation," *Proceedings of the 23rd International Conference of the Chilean Computer Science Society*, pp. 130–136, 2003.

[12] R. Malhotra, "An empirical framework for defect prediction using machine learning techniques with android software," *Applied Soft Computing*, vol. 49, pp. 1034–1050, 2016.

[13] K. Aggarwal and Y. Singh, *Software Engineering*. New Age International (P) Limited, 2008. [Online]. Available: https://books.google.co.in/books?id=ISh5PwAACAAJ

[14] T. DeMarco, *Controlling software projects: Management, measurement, and estimates*. Prentice Hall PTR, 1986.

[15] B. Henderson-Sellers, *Object-oriented metrics: measures of complexity*. Prentice-Hall, Inc., 1995.

[16] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476–493, 1994.

[17] R. Malhotra, "A systematic review of machine learning techniques for software fault prediction," *Applied Soft Computing*, vol. 27, pp. 504–518, 2015.

[18] W. Li and S. Henry, "Object-oriented metrics that predict maintainability," *Journal of Systems and Software*, vol. 23, no. 2, pp. 111–122, 1993.

[19] M. Lorenz and J. Kidd, *Object-oriented software metrics: a practical guide*. Prentice-Hall, Inc., 1994.

[20] F. B. e Abreu and W. Melo, "Evaluating the impact of object-oriented design on software quality," *Proceedings of the 3rd International Software Metrics Symposium*, pp. 90–99, 1996.

[21] J. M. Bieman and B.-K. Kang, "Cohesion and reuse in an object-oriented system," *ACM SIGSOFT Software Engineering Notes*, vol. 20, no. SI, pp. 259–262, 1995.

[22] L. C. Briand, J. W. Daly, and J. K. Wust, "A unified framework for coupling measurement in object-oriented systems," *IEEE Transactions on software Engineering*, vol. 25, no. 1, pp. 91–121, 1999.

[23] R. C. Martin, *Agile software development: principles, patterns, and practices*. Prentice Hall, 2002.

[24] Y.-S. Lee, "Measuring the coupling and cohesion of an object-oriented program based on information flow," *Proceedings of the International Conference on Software Quality*, pp. 81–90, 1995.

[25] J. Bansiya and C. G. Davis, "A hierarchical model for object-oriented design quality assessment," *IEEE Transactions on Software Engineering*, vol. 28, no. 1, pp. 4–17, 2002.

[26] M.-H. Tang, M.-H. Kao, and M.-H. Chen, "An empirical study on object-oriented metrics," *Proceedings of the International Software Metrics Symposium*, pp. 242–249, 1999.

[27] S. Eski and F. Buzluca, "An empirical study on object-oriented metrics and software evolution in order to reduce testing costs by predicting change-prone classes," *Proceedings of the IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pp. 566–571, 2011.

[28] H. Lu, Y. Zhou, B. Xu, H. Leung, and L. Chen, "The ability of object-oriented metrics to predict change-proneness: a meta-analysis," *Empirical Software Engineering*, vol. 17, no. 3, pp. 200–242, 2012.

[29] Y. Zhou, H. Leung, and B. Xu, "Examining the potentially confounding effect of class size on the associations between object-oriented metrics and change-proneness," *IEEE Transactions on Software Engineering*, vol. 35, no. 5, pp. 607–623, 2009.

[30] E. Giger, M. Pinzger, and H. C. Gall, "Can we predict types of code changes? an empirical analysis," *Proccedings of the 9th IEEE Working Conference on Mining Software Repositories (MSR)*, pp. 217–226, 2012.

[31] R. Malhotra and A. J. Bansal, "Predicting software change in an open source software using machine learning algorithms," *International Journal of Reliability, Quality and Safety Engineering*, vol. 20, no. 6, pp. 1 350 025–1–1 350 025–14, 2013.

[32] N. Tsantalis, A. Chatzigeorgiou, and G. Stephanides, "Predicting the probability of change in object-oriented systems," *IEEE Transactions on Software Engineering*, vol. 31, no. 7, pp. 601–614, 2005.

[33] L. Kumar, S. K. Rath, and A. Sureka, "Empirical analysis on effectiveness of source code metrics for predicting change-proneness," *Proceedings of the 10th Innovations in Software Engineering Conference*, pp. 4–14, 2017.

[34] D. Azar, "A genetic algorithm for improving accuracy of software quality predictive models: a search-based software engineering approach," *International Journal of Computational Intelligence and Applications*, vol. 9, no. 02, pp. 125–136, 2010.

[35] D. Azar and J. Vybihal, "An ant colony optimization algorithm to improve software quality prediction models: Case of class stability," *Information and Software Technology*, vol. 53, no. 4, pp. 388–393, 2011.

[36] A. Bansal, "Empirical analysis of search based algorithms to identify change prone classes of open source software," *Computer Languages, Systems & Structures*, vol. 47, pp. 211–231, 2017.

[37] A. B. De Carvalho, A. Pozo, and S. R. Vergilio, "A symbolic fault-prediction model based on multiobjective particle swarm optimization," *Journal of Systems and Software*, vol. 83, no. 5, pp. 868–882, 2010.

[38] M. Harman, S. Islam, Y. Jia, L. L. Minku, F. Sarro, and K. Srivisut, "Less is more: Temporal fault predictive performance over multiple hadoop releases," *International Symposium on Search Based Software Engineering*, pp. 240–246, 2014.

[39] F. Ferrucci, P. Salza, and F. Sarro, "Using hadoop mapreduce for parallel genetic algorithms: A comparison of the global, grid and island models," *Evolutionary Computation*, pp. 1–33, 2017.

[40] X. Xia, D. Lo, S. J. Pan, N. Nagappan, and X. Wang, "Hydra: Massively compositional model for cross-project defect prediction," *IEEE Transactions on software Engineering*, vol. 42, no. 10, pp. 977–998, 2016.

[41] S. Hosseini, B. Turhan, and M. Mäntylä, "A benchmark study on the effectiveness of search-based data selection and feature selection for cross project defect prediction," *Information and Software Technology*, 2017.

[42] T. G. Dietterich, "Ensemble methods in machine learning," *Proceedings of the International Workshop on Multiple Classifier Systems*, pp. 1–15, 2000.

[43] L. I. Kuncheva and C. J. Whitaker, "Measures of diversity in classifier ensembles and their relationship with the ensemble accuracy," *Machine Learning*, vol. 51, no. 2, pp. 181–207, 2003.

[44] L. Jonsson, M. Borg, D. Broman, K. Sandahl, S. Eldh, and P. Runeson, "Automated bug assignment: Ensemble-based machine learning in large scale industrial contexts," *Empirical Software Engineering*, vol. 21, no. 4, pp. 1533–1578, 2016.

[45] M. O. Elish, H. Aljamaan, and I. Ahmad, "Three empirical studies on pre-

dicting software maintainability using ensemble methods," *Soft Computing*, vol. 19, no. 9, pp. 2511–2524, 2015.

[46] S. Di Martino, F. Ferrucci, C. Gravino, and F. Sarro, "A genetic algorithm to configure support vector machines for predicting fault-prone components," *Proceedings of the International Conference on Product Focused Software Process Improvement*, pp. 247–261, 2011.

[47] F. Ferrucci, C. Gravino, R. Oliveto, and F. Sarro, "Genetic programming for effort estimation: an analysis of the impact of different fitness functions," *Second International Symposium on Search Based Software Engineering (SSBSE)*, pp. 89–98, 2010.

[48] U. Bhowan, M. Johnston, and M. Zhang, "Developing new fitness functions in genetic programming for classification with unbalanced data," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 42, no. 2, pp. 406–421, 2012.

[49] M. W. Aslam, "Selection of fitness function in genetic programming for binary classification," *Proceedings of the Science and Information Conference (SAI)*, pp. 489–493, 2015.

[50] J. Petrić, D. Bowes, T. Hall, B. Christianson, and N. Baddoo, "Building an ensemble for software defect prediction based on diversity selection," *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, p. 46, 2016.

[51] A. Panichella, R. Oliveto, and A. De Lucia, "Cross-project defect prediction models: L'union fait la force," *Proceedings of the IEEE Conference on Soft-*

*ware Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE)*, pp. 164–173, 2014.

[52] Y. Zhang, D. Lo, X. Xia, and J. Sun, "An empirical study of classifier combination for cross-project defect prediction," *Proccedings of the IEEE 39th Annual Computer Software and Applications Conference (COMPSAC)*, vol. 2, pp. 264–269, 2015.

[53] H. I. Aljamaan and M. O. Elish, "An empirical study of bagging and boosting ensembles for identifying faulty classes in object-oriented software," *Proccedings of the IEEE Symposium on Computational Intelligence and Data Mining*, pp. 187–194, 2009.

[54] A. T. Mısırlı, A. B. Bener, and B. Turhan, "An industrial case study of classifier ensembles for locating software defects," *Software Quality Journal*, vol. 19, no. 3, pp. 515–536, 2011.

[55] I. H. Laradji, M. Alshayeb, and L. Ghouti, "Software defect prediction using ensemble learning on selected features," *Information and Software Technology*, vol. 58, pp. 388–402, 2015.

[56] D. Di Nucci, F. Palomba, R. Oliveto, and A. De Lucia, "Dynamic selection of classifiers in bug prediction: An adaptive method," *IEEE Transactions on Emerging Topics in Computational Intelligence*, vol. 1, no. 3, pp. 202–212, 2017.

[57] R. Shatnawi, "Improving software fault-prediction for imbalanced data," *Proceedings of the International Conference on Innovations in Information Technology (IIT)*, pp. 54–59, 2012.

[58] N. Seliya and T. M. Khoshgoftaar, "The use of decision trees for cost-sensitive classification: an empirical study in software quality prediction," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 1, no. 5, pp. 448–459, 2011.

[59] M. Liu, L. Miao, and D. Zhang, "Two-stage cost-sensitive learning for software defect prediction," *IEEE Transactions on Reliability*, vol. 63, no. 2, pp. 676–686, 2014.

[60] D. Rodriguez, I. Herraiz, R. Harrison, J. Dolado, and J. C. Riquelme, "Preliminary comparison of techniques for dealing with imbalance in software defect prediction," *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, p. 43, 2014.

[61] Ö. F. Arar and K. Ayan, "Software defect prediction using cost-sensitive neural network," *Applied Soft Computing*, vol. 33, pp. 263–277, 2015.

[62] M. Tan, L. Tan, S. Dara, and C. Mayeux, "Online defect prediction for imbalanced data," *Proceedings of the 37th International Conference on Software Engineering*, pp. 99–108, 2015.

[63] M. Acharya and B. Robinson, "Practical change impact analysis based on static program slicing for industrial software systems," *Proceedings of the 33rd International Conference on Software Engineering*, pp. 746–755, 2011.

[64] M.-A. Jashki, R. Zafarani, and E. Bagheri, "Towards a more efficient static software change impact analysis method," *Proceedings of the 8th ACM SIGPLAN-SIGSOFT workshop on Program analysis for software tools and engineering*, pp. 84–90, 2008.

[65] H. Cai and R. Santelices, "A comprehensive study of the predictive accuracy of dynamic change-impact analysis," *Journal of Systems and Software*, vol. 103, pp. 248–265, 2015.

[66] M. C. O. Maia, R. A. Bittencourt, J. C. A. de Figueiredo, and D. D. S. Guerrero, "The hybrid technique for object-oriented software change impact analysis," *Proceedings of the 14th European Conference on Software Maintenance and Reengineering (CSMR)*, pp. 252–255, 2010.

[67] T. Zimmermann, A. Zeller, P. Weissgerber, and S. Diehl, "Mining version histories to guide software changes," *IEEE Transactions on Software Engineering*, vol. 31, no. 6, pp. 429–445, 2005.

[68] H. Abdeen, K. Bali, H. Sahraoui, and B. Dufour, "Learning dependency-based change impact predictors using independent change histories," *Information and Software Technology*, vol. 67, pp. 220–235, 2015.

[69] G. Canfora and L. Cerulo, "Fine grained indexing of software repositories to support impact analysis," *Proceedings of the International Workshop on Mining software Repositories*, pp. 105–111, 2006.

[70] G. Antoniol, G. Canfora, G. Casazza, and A. De Lucia, "Identifying the starting impact set of a maintenance request: A case study," *Proceedings of the Fourth European Software Maintenance and Reengineering*, pp. 227–230, 2000.

[71] M. Gethers, B. Dit, H. Kagdi, and D. Poshyvanyk, "Integrated impact analysis for managing software changes," *Proceedings of the 34th International Conference on Software Engineering*, pp. 430–440, 2012.

[72] M. B. Zanjani, G. Swartzendruber, and H. Kagdi, "Impact analysis of change requests on source code based on interaction and commit histories," *Proceedings of the 11th Working Conference on Mining Software Repositories*, pp. 162–171, 2014.

[73] R. Jindal, R. Malhotra, and A. Jain, "Mining defect reports for predicting software maintenance effort," *Proceedings of the International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pp. 270–276, 2015.

[74] R. Jindal, R. Malhotra, and A. Jain, "Predicting software maintenance effort using neural networks," *Proceedings of the 4th International Conference on Reliability, Infocom Technologies and Optimization (ICRITO)(Trends and Future Directions)*, pp. 1–6, 2015.

[75] M. P. Basgalupp, R. C. Barros, T. S. da Silva, and A. C. de Carvalho, "Software effort prediction: A hyper-heuristic decision-tree based approach," *Proceedings of the 28th Annual ACM Symposium on Applied Computing*, pp. 1109–1116, 2013.

[76] M. P. Basgalupp, R. C. Barros, and D. D. Ruiz, "Predicting software maintenance effort through evolutionary-based decision trees," *Proceedings of the 27th Annual ACM Symposium on Applied Computing*, pp. 1209–1214, 2012.

[77] G. Balogh, A. Z. Vegh, and A. Beszedes, "Software development modification effort enhanced by a genetic algorithm," *Symposium on Search based Software Engineering (MSR)*, pp. 1–6, 2012.

[78] J. M. Bieman, D. Jain, and H. J. Yang, "Oo design patterns, design structure,

and program changes: an industrial case study," *Proceedings of IEEE International Conference on Software Maintenance*, pp. 580–589, 2001.

[79] K. El Emam, W. Melo, and J. C. Machado, "The prediction of faulty classes using object-oriented design metrics," *Journal of Systems and Software*, vol. 56, no. 1, pp. 63–75, 2001.

[80] T. Gyimothy, R. Ferenc, and I. Siket, "Empirical validation of object-oriented metrics on open source software for fault prediction," *IEEE Transactions on Software engineering*, vol. 31, no. 10, pp. 897–910, 2005.

[81] S. Kpodjedo, F. Ricca, P. Galinier, Y.-G. Guéhéneuc, and G. Antoniol, "Design evolution metrics for defect prediction in object oriented systems," *Empirical Software Engineering*, vol. 16, no. 1, pp. 141–175, 2011.

[82] Y. Singh, A. Kaur, and R. Malhotra, "Empirical validation of object-oriented metrics for predicting fault proneness models," *Software Quality Journal*, vol. 18, no. 1, pp. 3–35, 2010.

[83] H. M. Olague, L. H. Etzkorn, S. Gholston, and S. Quattlebaum, "Empirical validation of three software metrics suites to predict fault-proneness of object-oriented classes developed using highly iterative or agile software development processes," *IEEE Transactions on Software Engineering*, vol. 33, no. 6, pp. 402–419, 2007.

[84] D. Radjenović, M. Heričko, R. Torkar, and A. Živkovič, "Software fault prediction metrics: A systematic literature review," *Information and Software Technology*, vol. 55, no. 8, pp. 1397–1418, 2013.

[85] P. Sentas, L. Angelis, I. Stamelos, and G. Bleris, "Software productivity and

effort prediction with ordinal regression," *Information and Software Technology*, vol. 47, no. 1, pp. 17–29, 2005.

[86] D. W. Hosmer Jr, S. Lemeshow, and R. X. Sturdivant, *Applied logistic regression*. John Wiley & Sons, 2013, vol. 398.

[87] S. Haykin and N. Network, "A comprehensive foundation," *Neural networks*, vol. 2, no. 2004, p. 41, 2004.

[88] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: an update," *ACM SIGKDD explorations newsletter*, vol. 11, no. 1, pp. 10–18, 2009.

[89] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, no. 2, pp. 123–140, 1996.

[90] J. Friedman, T. Hastie, R. Tibshirani *et al.*, "Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors)," *The annals of statistics*, vol. 28, no. 2, pp. 337–407, 2000.

[91] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[92] K. P. Murphy, "Naive bayes classifiers," *University of British Columbia*, vol. 18, 2006.

[93] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.

[94] R. Eberhart and J. Kennedy, "A new optimizer using particle swarm theory," *Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, pp. 39–43, 1995.

[95] M. Clerc and J. Kennedy, "The particle swarm-explosion, stability, and convergence in a multidimensional complex space," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 1, pp. 58–73, 2002.

[96] J. Bacardit and J. M. Garrell, "Evolving multiple discretizations with adaptive intervals for a pittsburgh rule-based learning classifier system," *Genetic and Evolutionary Computation Conference*, pp. 1818–1831, 2003.

[97] J. S. Aguilar-Ruiz, J. C. Riquelme, and M. Toro, "Evolutionary learning of hierarchical decision rules," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 33, no. 2, pp. 324–331, 2003.

[98] E. Bernadó-Mansilla and J. M. Garrell-Guiu, "Accuracy-based learning classifier systems: models, analysis and applications to classification tasks," *Evolutionary Computation*, vol. 11, no. 3, pp. 209–238, 2003.

[99] M. V. Butz, T. Kovacs, P. L. Lanzi, and S. W. Wilson, "How xcs evolves accurate classifiers," *Proceedings of the 3rd Annual Conference on Genetic and Evolutionary Computation*, pp. 927–934, 2001.

[100] J. Bacardit and N. Krasnogor, "Performance and efficiency of memetic pittsburgh learning classifier systems," *Evolutionary computation*, vol. 17, no. 3, pp. 307–342, 2009.

[101] C. Ferreira, "Gene expression programming: a new adaptive algorithm for solving problems," *Complex Systems*, vol. 13, no. 2, pp. 87–129, 2001.

[102] D. R. Carvalho and A. A. Freitas, "A hybrid decision tree/genetic algorithm method for data mining," *Information Sciences*, vol. 163, no. 1-3, pp. 13–35, 2004.

[103] J. Otero and L. Sánchez, "Induction of descriptive fuzzy classifiers with the logitboost algorithm," *Soft Computing*, vol. 10, no. 9, pp. 825–835, 2006.

[104] R. Durbin and D. E. Rumelhart, "Product units: A computationally powerful and biologically plausible extension to backpropagation networks," *Neural Computation*, vol. 1, no. 1, pp. 133–142, 1989.

[105] F. J. Martínez-Estudillo, C. Hervás-Martínez, P. A. Gutiérrez, and A. C. Martínez-Estudillo, "Evolutionary product-unit neural networks classifiers," *Neurocomputing*, vol. 72, no. 1-3, pp. 548–561, 2008.

[106] R. Malhotra, N. Pritam, K. Nagpal, and P. Upmanyu, "Defect collection and reporting system for git based open source software," *Proceedings of the International Conference on Data Mining and Intelligent Computing (ICDMIC)*, pp. 1–7, 2014.

[107] R. Malhotra and A. Agrawal, "Cms tool: calculating defect and change data from software project repositories," *ACM SIGSOFT Software Engineering Notes*, vol. 39, no. 1, pp. 1–5, 2014.

[108] V. Barnett and T. Lewis, *Outliers in statistical data*. Wiley, 1974.

[109] M. A. Hall, "Correlation-based feature selection of discrete and numeric class machine learning," *Proceedings of the 17th International Conference on Machine Learning*, pp. 359–366, 2000.

[110] K. Gao, T. M. Khoshgoftaar, and A. Napolitano, "Combining feature subset selection and data sampling for coping with highly imbalanced software data," *Proceedings of the Software Engineering Knowledge Engineering Conference*, pp. 439–444, 2015.

[111] M. A. Hall and G. Holmes, "Benchmarking attribute selection techniques for discrete class data mining," *IEEE Transactions on Knowledge and Data engineering*, vol. 15, no. 6, pp. 1437–1447, 2003.

[112] E. Arisholm, L. C. Briand, and E. B. Johannessen, "A systematic and comprehensive investigation of methods to build and evaluate fault prediction models," *Journal of Systems and Software*, vol. 83, no. 1, pp. 2–17, 2010.

[113] M. Stone, "Cross-validatory choice and assessment of statistical predictions," *Journal of the royal statistical society. Series B (Methodological)*, pp. 111–147, 1974.

[114] G. J. Pai and J. B. Dugan, "Empirical analysis of software fault content and fault proneness using bayesian methods," *IEEE Transactions on software Engineering*, vol. 33, no. 10, pp. 675–686, 2007.

[115] Z. He, F. Shu, Y. Yang, M. Li, and Q. Wang, "An investigation on the feasibility of cross-project defect prediction," *Automated Software Engineering*, vol. 19, no. 2, pp. 167–199, 2012.

[116] F. Peters, T. Menzies, and A. Marcus, "Better cross company defect prediction," *Proceedings of the 10th Working Conference on Mining Software Repositories*, pp. 409–418, 2013.

[117] T. Fawcett, "An introduction to roc analysis," *Pattern recognition letters*, vol. 27, no. 8, pp. 861–874, 2006.

[118] H. He and E. A. Garcia, "Learning from imbalanced data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 21, no. 9, pp. 1263–1284, 2009.

[119] T. Menzies, A. Dekhtyar, J. Distefano, and J. Greenwald, "Problems with precision: A response to" comments on'data mining static code attributes to learn

defect predictors'"," *IEEE Transactions on Software Engineering*, vol. 33, no. 9, pp. 637–640, 2007.

[120] M. Kubat, S. Matwin *et al.*, "Addressing the curse of imbalanced training sets: one-sided selection," *Proceedings of 14th International Conference on Machine Learning*, pp. 179–186, 1997.

[121] T. Menzies, J. Greenwald, and A. Frank, "Data mining static code attributes to learn defect predictors," *IEEE Transactions on Software Engineering*, no. 1, pp. 2–13, 2007.

[122] M. Li, H. Zhang, R. Wu, and Z.-H. Zhou, "Sample-based software defect prediction with active and semi-supervised learning," *Automated Software Engineering*, vol. 19, no. 2, pp. 201–230, 2012.

[123] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking classification models for software defect prediction: A proposed framework and novel findings," *IEEE Transactions on Software Engineering*, vol. 34, no. 4, pp. 485–496, 2008.

[124] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *Journal of Machine Learning Research*, vol. 7, no. Jan, pp. 1–30, 2006.

[125] M. Friedman, "A comparison of alternative tests of significance for the problem of m rankings," *The Annals of Mathematical Statistics*, vol. 11, no. 1, pp. 86–92, 1940.

[126] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics bulletin*, vol. 1, no. 6, pp. 80–83, 1945.

[127] J. Pallant, *SPSS survival manual*. McGraw-Hill Education (UK), 2013.

[128] J. Cohen, *Statistical power analysis for the behavioral sciences. 2nd*. Hillsdale, NJ: erlbaum, 1988.

[129] R. Malhotra and M. Khanna, "Software change prediction: A systematic review and future research directions," *Journal of Information and Processing Systems*, 2018.

[130] B. A. Kitchenham, D. Budgen, and P. Brereton, *Evidence-based software engineering and systematic reviews*. CRC Press, 2015.

[131] X. Zhu, Q. Song, and Z. Sun, "Automated identification of change-prone classes in open source software projects." *Journal of Software*, vol. 8, no. 2, pp. 361–366, 2013.

[132] M. Lindvall, "Are large c++ classes change-prone? an empirical investigation," *Software: Practice and Experience*, vol. 28, no. 15, pp. 1551–1558, 1998.

[133] M. Lindvall, "Measurement of change: stable and change-prone constructs in a commercial c++ system," *Proceedings of Sixth International Software Metrics Symposium*, pp. 40–49, 1999.

[134] Y. Liu and T. M. Khoshgoftaar, "Genetic programming model for software quality classification," *Proceedings of the International Symposium on High Assurance Systems Engineering*, pp. 127–136, 2001.

[135] R. Malhotra and M. Khanna, "Mining the impact of object oriented metrics for change prediction using machine learning and search-based techniques," *Proceedings of the International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pp. 228–234, 2015.

[136] T. M. Khoshgoftaar, N. Seliya, and Y. Liu, "Genetic programming-based decision trees for software quality classification," *Proceedings of the 15th IEEE International Conference on Tools with Artificial Intelligence*, pp. 374–383, 2003.

[137] G. Catolino, F. Palomba, A. De Lucia, F. Ferrucci, and A. Zaidman, "Developer-related factors in change prediction: an empirical assessment," *Proceedings of the 25th International Conference on Program Comprehension*, pp. 186–195, 2017.

[138] A. R. Sharafat and L. Tahvildari, "Change prediction in object-oriented software systems: A probabilistic approach," *Journal of Software*, vol. 3, no. 5, pp. 26–39, 2008.

[139] M. Al-Khiaty, R. Abdel-Aal, and M. O. Elish, "Abductive network ensembles for improved prediction of future change-prone classes in object-oriented software." *International Arab Journal of Information Technology*, vol. 14, no. 6, pp. 803–811, 2017.

[140] A.-R. Han, S.-U. Jeon, D.-H. Bae, and J.-E. Hong, "Measuring behavioral dependency for improving change-proneness prediction in uml-based design models," *Journal of Systems and Software*, vol. 83, no. 2, pp. 222–234, 2010.

[141] L. Kumar, S. K. Rath, and A. Sureka, "Using source code metrics to predict change-prone web services: A case-study on ebay services," *IEEE Workshop on Machine Learning Techniques for Software Quality Evaluation (MaLTeSQuE)*, pp. 1–7, 2017.

[142] L. Kumar, R. K. Behera, S. Rath, and A. Sureka, "Transfer learning for cross-project change-proneness prediction in object-oriented software systems: A

feasibility analysis," *ACM SIGSOFT Software Engineering Notes*, vol. 42, no. 3, pp. 1–11, 2017.

[143] R. Malhotra and R. Jangra, "Prediction & assessment of change prone classes using statistical & machine learning techniques." *Journal of Information Processing Systems*, vol. 13, no. 4, pp. 778–804, 2017.

[144] M. Yan, X. Zhang, C. Liu, L. Xu, M. Yang, and D. Yang, "Automated change-prone class prediction on unlabeled dataset using unsupervised method," *Information and Software Technology*, vol. 92, pp. 1–16, 2017.

[145] G. Canfora and L. Cerulo, "How software repositories can help in resolving a new change request," *Proceedings of the Workshop on Empirical Studies in Reverse Engineering*, p. 99, 2005.

[146] G. Canfora and L. Cerulo, "Impact analysis by mining software and change request repositories," *IEEE International Symposium on Software Metrics*, pp. 1–9, 2005.

[147] R. Malhotra and A. J. Bansal, "Cross project change prediction using open source projects," *Proceedings of the International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pp. 201–207, 2014.

[148] R. Malhotra and M. Khanna, "Analyzing software change in open source projects using artificial immune system algorithms," *Proceedings of the International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pp. 2674–2680, 2014.

[149] M. H. Asl and N. Kama, "A change impact size estimation approach during the software development," *Proceedings of the 22nd Australian Software Engineering Conference (ASWEC)*, pp. 68–77, 2013.

[150] C. Marinescu, "How good is genetic programming at predicting changes and defects?" *Proceedings of the 16th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, pp. 544–548, 2014.

[151] B. Dit, M. Wagner, S. Wen, W. Wang, M. Linares-Vásquez, D. Poshyvanyk, and H. Kagdi, "Impactminer: A tool for change impact analysis," *Proceedings of the 36th International Conference on Software Engineering*, pp. 540–543, 2014.

[152] C. G. Weng and J. Poon, "A new evaluation measure for imbalanced datasets," *Proceedings of the 7th Australian Data Mining Conference*, pp. 27–32, 2008.

[153] A. L. Oliveira, P. L. Braga, R. M. Lima, and M. L. Cornélio, "Ga-based method for feature selection and parameters optimization for machine learning regression applied to software effort estimation," *Information and Software Technology*, vol. 52, no. 11, pp. 1155–1166, 2010.

[154] K. Aggarwal, Y. Singh, A. Kaur, and R. Malhotra, "Application of artificial neural network for predicting maintainability using object-oriented metrics," *Transactions on Engineering, Computing and Technology*, vol. 15, pp. 285–289, 2006.

[155] R. Malhotra and M. Khanna, "An empirical study to evaluate the relationship of object-oriented metrics and change proneness," *International Arab Journal of Information Technology*, vol. 15, no. 6, pp. 1016–1023, 2018.

[156] R. Malhotra and M. Khanna, "Examining the effectiveness of machine learning algorithms for prediction of change prone classes," *Proceedings of the International Conference on High Performance Computing & Simulation (HPCS)*, pp. 635–642, 2014.

[157] M. Harman and B. F. Jones, "Search-based software engineering," *Information and Software Technology*, vol. 43, no. 14, pp. 833–839, 2001.

[158] M. Harman, "The relationship between search based software engineering and predictive modeling," *Proceedings of the 6th International Conference on Predictive Models in Software Engineering*, p. 1, 2010.

[159] M. Harman, "Why the virtual nature of software makes it ideal for search based optimization," *International Conference on Fundamental Approaches to Software Engineering*, pp. 1–12, 2010.

[160] M. Harman and J. Clark, "Metrics are fitness functions too," *International Symposium on Software Metrics*, pp. 58–69, 2004.

[161] L. C. Briand and J. Wüst, "Empirical studies of quality models in object-oriented systems," *Advances in Computers*, vol. 56, pp. 97–166, 2002.

[162] R. Malhotra and M. Khanna, "The ability of search-based algorithms to predict change-prone classes," *Software Quality Professional*, vol. 17, no. 1, p. 17, 2014.

[163] R. Malhotra and M. Khanna, "Threats to validity in search-based predictive modelling for software engineering," *IET Software*, vol. 12, no. 4, pp. 293–305, 2018.

[164] R. Malhotra, M. Khanna, and R. R. Raje, "On the application of search-based techniques for software engineering predictive modeling: A systematic review and future directions," *Swarm and Evolutionary Computation*, vol. 32, pp. 85–109, 2017.

[165] S. Xanthakis, C. Ellis, C. Skourlas, A. Le Gall, S. Katsikas, and K. Karapoulios, "Application of genetic algorithms to software testing," *Proceedings*

*of the 5th International Conference on Software Engineering and Applications*, pp. 625–636, 1992.

[166] J. J. Dolado and L. Fernandez, "Genetic programming, neural networks and linear regression in software project estimation," *Proceedings of the International Conference on Software Process Improvement, Research, Education and Training*, pp. 157–171, 1998.

[167] J. J. Dolado, "A validation of the component-based method for software size estimation," *IEEE Transactions on Software Engineering*, vol. 26, no. 10, pp. 1006–1021, 2000.

[168] Y. Liu and T. Khoshgoftaar, "Reducing overfitting in genetic programming models for software quality classification," *Proceedings of the International Symposium on High Assurance Systems Engineering*, pp. 56–65, 2004.

[169] K. K. Shukla, "Neuro-genetic prediction of software development effort," *Information and Software Technology*, vol. 42, no. 10, pp. 701–713, 2000.

[170] A. B. de Carvalho, A. Pozo, S. Vergilio, and A. Lenz, "Predicting fault proneness of classes trough a multiobjective particle swarm optimization algorithm," *Proceedings of the 20th IEEE International Conference on Tools with Artificial Intelligence*, pp. 387–394, 2008.

[171] C. J. Burgess and M. Lefley, "Can genetic programming improve software effort estimation? a comparative evaluation," *Information and Software Technology*, vol. 43, no. 14, pp. 863–873, 2001.

[172] A. Tsakonas and G. Dounias, "Predicting defects in software using grammar-guided genetic programming," *Proceedings of the Hellenic Conference on Artificial Intelligence*, pp. 413–418, 2008.

[173] J. J. Dolado, "On the problem of the software cost function," *Information and Software Technology*, vol. 43, no. 1, pp. 61–72, 2001.

[174] O. Vandecruys, D. Martens, B. Baesens, C. Mues, M. De Backer, and R. Haesen, "Mining software repositories for comprehensible software fault prediction models," *Journal of Systems and software*, vol. 81, no. 5, pp. 823–839, 2008.

[175] C. Kirsopp, M. Shepperd, and J. Hart, "Search heuristics, case-based reasoning and software project effort prediction," *Proceedings of the 4th Annual Conference on Genetic and Evolutionary Computation*, pp. 1367–1374, 2002.

[176] C. Catal and B. Diri, "Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem," *Information Sciences*, vol. 179, no. 8, pp. 1040 – 1058, 2009.

[177] Y. Shan, R. I. McKay, C. J. Lokan, and D. L. Essam, "Software project effort estimation using genetic programming," *Proceedings of the IEEE 2002 International Conference on Communications, Circuits and Systems and West Sino Expositions*, pp. 1108–1112, 2002.

[178] Y. Singh, A. Kaur, and R. Malhotra, "Prediction of software quality model using gene expression programming," *Proceedings of the International Conference on Product-Focused Software Process Improvement*, pp. 43–58, 2009.

[179] M. Lefley and M. J. Shepperd, "Using genetic programming to improve software effort estimation based on general data sets," *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 2477–2487, 2003.

[180] W. Afzal, "Using faults-slip-through metric as a predictor of fault-proneness,"

*Proceedings of the 17th Asia Pacific Software Engineering Conference (APSEC'10)*, pp. 414–422, 2010.

[181] E. N. Regolin, G. A. de Souza, A. R. Pozo, and S. R. Vergilio, "Exploring machine learning techniques for software size estimation," *Proceedings of the 23rd International Conference of the Chilean Computer Science Society*, pp. 130–136, 2003.

[182] C. Lokan, "What should you optimize when building an estimation model?" *Proceedings of the 11th IEEE International Symposium Software Metrics*, p. 10, 2005.

[183] C. Jin, E.-M. Dong, and L.-N. Qin, "Software fault prediction model based on adaptive dynamical and median particle swarm optimization," *Proceedings of the Second International Conference on Multimedia and Information Technology (MMIT)*, vol. 1, pp. 44–47, 2010.

[184] S.-J. Huang and N.-H. Chiu, "Optimization of analogy weights by genetic algorithm for software effort estimation," *Information and Software Technology*, vol. 48, no. 11, pp. 1034–1045, 2006.

[185] Y. Liu, T. M. Khoshgoftaar, and N. Seliya, "Evolutionary optimization of software quality modeling with multiple repositories," *IEEE Transactions on Software Engineering*, vol. 36, no. 6, pp. 852–864, 2010.

[186] A. F. Sheta, "Estimation of the cocomo model parameters using genetic algorithms for nasa software projects," *Journal of Computer Science*, vol. 2, no. 2, pp. 118–123, 2006.

[187] P. C. Pendharkar, "Exhaustive and heuristic search approaches for learning

a software defect prediction model," *Engineering Applications of Artificial Intelligence*, vol. 23, no. 1, pp. 34–40, 2010.

[188] N.-H. Chiu and S.-J. Huang, "The adjusted analogy-based software effort estimation based on similarity distances," *Journal of Systems and Software*, vol. 80, no. 4, pp. 628–640, 2007.

[189] N.-H. Chiu, "Combining techniques for software quality classification: An integrated decision network approach," *Expert Systems with Applications*, vol. 38, no. 4, pp. 4618–4625, 2011.

[190] F. Ahmed, S. Bouktif, A. Serhani, and I. Khalil, "Integrating function point project information for improving the accuracy of effort estimation," *Proceedings of the Second International Conference on Advanced Engineering Computing and Applications in Sciences*, pp. 193–198, 2008.

[191] P. L. Braga, A. L. Oliveira, and S. R. Meira, "A ga-based feature selection and parameters optimization for support vector regression applied to software effort estimation," *Proceedings of the 2008 ACM symposium on Applied computing*, pp. 1788–1792, 2008.

[192] L. Yu, "An evolutionary programming based asymmetric weighted least squares support vector machine ensemble learning methodology for software repository mining," *Information Sciences*, vol. 191, pp. 31–46, 2012.

[193] S.-J. Huang, N.-H. Chiu, and L.-W. Chen, "Integration of the grey relational analysis with genetic algorithm for software effort estimation," *European Journal of Operational Research*, vol. 188, no. 3, pp. 898–909, 2008.

[194] D. Rodríguez, R. Ruiz, J. C. Riquelme, and J. S. Aguilar-Ruiz, "Searching for

rules to detect defective modules: A subgroup discovery approach," *Information Sciences*, vol. 191, pp. 14–30, 2012.

[195] A. Tsakonas and G. Dounias, "Application of genetic programming in software engineering empirical data modelling," *Proceedings of the International Conference on Software and Data Technologies*, pp. 295–300, 2008.

[196] F. Sarro, S. Di Martino, F. Ferrucci, and C. Gravino, "A further analysis on the use of genetic algorithm to configure support vector machines for inter-release fault prediction," *Proceedings of the 27th annual ACM symposium on applied computing*, pp. 1215–1220, 2012.

[197] F. Ferrucci, C. Gravino, R. Oliveto, and F. Sarro, "Using tabu search to estimate software development effort," *International Workshop on Software Measurement*, pp. 307–320, 2009.

[198] H. Can, X. Jianchun, Z. Ruide, L. Juelong, Y. Qiliang, and X. Liqiang, "A new model for software defect prediction using particle swarm optimization and support vector machine," *Proceedings of the 25th Chinese Control and Decision Conference (CCDC)*, pp. 4106–4110, 2013.

[199] Y.-F. Li, M. Xie, and T. Goh, "A study of mutual information based feature selection for case based reasoning in software cost estimation," *Expert Systems with Applications*, vol. 36, no. 3, pp. 5921–5931, 2009.

[200] G. Canfora, A. De Lucia, M. Di Penta, R. Oliveto, A. Panichella, and S. Panichella, "Multi-objective cross-project defect prediction," *Proceedings of the 2013 IEEE Sixth International Conference on Software Testing, Verification and Validation*, pp. 252–261, 2013.

[201] Y.-F. Li, M. Xie, and T. N. Goh, "A study of project selection and feature weighting for analogy based software cost estimation," *Journal of Systems and Software*, vol. 82, no. 2, pp. 241–252, 2009.

[202] G. Abaei and A. Selamat, "A survey on software fault detection based on different prediction approaches," *Vietnam Journal of Computer Science*, vol. 1, no. 2, pp. 79–95, 2014.

[203] A. Tsakonas and G. Dounias, "Deriving models for software project effort estimation by means of genetic programming," *Proceedings of the International Conference on Knowledge Discovery and Information Retrieval (KDIR)*, pp. 34–42, 2009.

[204] F. Ferrucci, C. Gravino, R. Oliveto, F. Sarro, and E. Mendes, "Investigating tabu search for web effort estimation," *Proceedimgs of the 36th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*, pp. 350–357, 2010.

[205] K. Li, C. Chen, W. Liu, X. Fang, and Q. Lu, "Software defect prediction using fuzzy integral fusion based on ga-fm," *Wuhan University Journal of Natural Sciences*, vol. 19, no. 5, pp. 405–408, 2014.

[206] R. Malhotra, "Comparative analysis of statistical and machine learning methods for predicting faulty modules," *Applied Soft Computing*, vol. 21, pp. 286–297, 2014.

[207] F. Ferrucci, C. Gravino, R. Oliveto, and F. Sarro, "Estimating software development effort using tabu search," *Proceedings of the International Conference on Enterprise Information Systems*, pp. 236–241, 2010.

[208] C. Jin and S.-W. Jin, "Prediction approach of software fault-proneness based on hybrid artificial neural network and quantum particle swarm optimization," *Applied Soft Computing*, vol. 35, pp. 717–725, 2015.

[209] A. F. Sheta, A. Ayesh, and D. Rine, "Evaluating software cost estimation models using particle swarm optimisation and fuzzy logic for nasa projects: a comparative study," *International Journal of Bio-Inspired Computation*, vol. 2, no. 6, pp. 365–373, 2010.

[210] Y. Abdi, S. Parsa, and Y. Seyfari, "A hybrid one-class rule learning approach based on swarm intelligence for software fault prediction," *Innovations in Systems and Software Engineering*, vol. 11, no. 4, pp. 289–301, 2015.

[211] F. S. Alaa and A. Al-Afeef, "A gp effort estimation model utilizing line of code and methodology for nasa software projects," *Proceedings of the 10th International Conference on Intelligent Systems Design and Applications (ISDA)*, pp. 290–295, 2010.

[212] W. Afzal and R. Torkar, "Towards benchmarking feature subset selection methods for software fault prediction," *Computational intelligence and quantitative software engineering*, pp. 33–58, 2016.

[213] S. Aljahdali, "Development of a software effort estimation model using differential evolution," *Journal of Electronics and Computer Science*, vol. 12, no. 1, pp. 1–8, 2010.

[214] L. Kumar and S. K. Rath, "Application of genetic algorithm as feature selection technique in development of effective fault prediction model," *Proceedings of the International Conference on Electrical, Computer and Electronics Engineering (UPCON)*, pp. 432–437, 2016.

[215] A. Chavoya, C. Lopez-Martin, and M. Meda-Campa, "Applying genetic programming for estimating software development effort of short-scale projects," *Proceedings of the International Conference on Information Technology: New Generations (ITNG)*, pp. 174–179, 2011.

[216] D. Ryu and J. Baik, "Effective multi-objective naïve bayes learning for cross-project defect prediction," *Applied Soft Computing*, vol. 49, pp. 1062–1077, 2016.

[217] R. D. A. Araújo, A. L. Oliveira, S. Soares, and S. Meira, "An evolutionary morphological approach for software development cost estimation," *Neural Networks*, vol. 32, pp. 285–291, 2012.

[218] F. Sarro, F. Ferrucci, and C. Gravino, "Single and multi objective genetic programming for software development effort estimation," *Proceedings of the 27th annual ACM symposium on Applied Computing*, pp. 1221–1226, 2012.

[219] V. K. Bardsiri, D. N. A. Jawawi, S. Z. M. Hashim, and E. Khatibi, "A pso-based model to increase the accuracy of software development effort estimation," *Software Quality Journal*, vol. 21, no. 3, pp. 501–526, 2013.

[220] R. C. Barros, M. P. Basgalupp, R. Cerri, T. S. da Silva, and A. C. de Carvalho, "A grammatical evolution approach for software effort estimation," *Proceedings of the 15th annual conference on Genetic and Evolutionary Computation*, pp. 1413–1420, 2013.

[221] G. Mauša and T. G. Grbac, "Co-evolutionary multi-population genetic programming for classification in software defect prediction: An empirical case study," *Applied Soft Computing*, vol. 55, pp. 331–351, 2017.

[222] A. Corazza, S. Di Martino, F. Ferrucci, C. Gravino, F. Sarro, and E. Mendes, "Using tabu search to configure support vector regression for effort estimation," *Empirical Software Engineering*, vol. 18, no. 3, pp. 506–546, 2013.

[223] A. A. B. Baqais, M. Alshayeb, and Z. A. Baig, "Hybrid intelligent model for software maintenance prediction," *Proceedings of the World Conference on Engineering*, 2014.

[224] L. L. Minku and X. Yao, "Software effort estimation as a multiobjective learning problem," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 22, no. 4, p. 35, 2013.

[225] R. Malhotra and A. Chug, "Application of evolutionary algorithms for software maintainability prediction using object-oriented metrics," *Proceedings of the 8th International Conference on Bioinspired Information and Communications Technologies*, pp. 348–351, 2014.

[226] L. L. Minku and X. Yao, "An analysis of multi-objective evolutionary algorithms for training ensemble models based on different performance measures in software effort estimation," *Proceedings of the 9th International Conference on Predictive Models in Software Engineering*, p. 8, 2013.

[227] L. Kumar, D. K. Naik, and S. K. Rath, "Validating the effectiveness of object-oriented metrics for predicting maintainability," *Procedia Computer Science*, vol. 57, pp. 798–806, 2015.

[228] Z. Dan, "Improving the accuracy in software effort estimation: Using artificial neural network model based on particle swarm optimization," *Proceedings of the IEEE International Conference on Service Operations and Logistics, and Informatics (SOLI)*, pp. 180–185, 2013.

[229] A. Jain, S. Tarwani, and A. Chug, "An empirical investigation of evolutionary algorithm for software maintainability prediction," *Proceedings of the IEEE Students' Conference on Electrical, Electronics and Computer Science (SCEECS)*, pp. 1–6, 2016.

[230] V. K. Bardsiri, D. N. Jawawi, S. Z. M. Hashim, and E. Khatibi, "A flexible method to estimate the software development effort based on the classification of projects and localization of comparisons," *Empirical Software Engineering*, vol. 19, no. 4, pp. 857–884, 2014.

[231] M. Azzeh, A. B. Nassif, and S. Banitaan, "A better case adaptation method for case-based effort estimation using multi-objective optimization," *Proceedings of the 13th International Conference on Machine Learning and Applications*, pp. 409–414, 2014.

[232] F. Sarro, A. Petrozziello, and M. Harman, "Multi-objective software effort estimation," *Proceedings of the IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, pp. 619–630, 2016.

[233] T. R. Benala and R. Mall, "Dabe: Differential evolution in analogy-based software development effort estimation," *Swarm and Evolutionary Computation*, vol. 38, pp. 158–172, 2018.

[234] J. Murillo-Morera, C. Quesada-López, C. Castro-Herrera, and M. Jenkins, "A genetic algorithm based framework for software effort prediction," *Journal of Software Engineering Research and Development*, vol. 5, no. 1, p. 4, 2017.

[235] D. Wu, J. Li, and C. Bao, "Case-based reasoning with optimized weight derived by particle swarm optimization for software effort estimation," *Soft Computing*, vol. 22, no. 16, pp. 5299–5310, 2018.

[236] R. Hochman, T. M. Khoshgoftaar, E. B. Allen, and J. P. Hudepohl, "Using the genetic algorithm to build optimal neural networks for fault-prone module detection," *Proceedings of the Seventh International Symposium on Software Reliability Engineering*, pp. 152–162, 1996.

[237] R. Hochman, T. M. Khoshgoftaar, E. B. Allen, and J. P. Hudepohl, "Evolutionary neural networks: a robust approach to software reliability problems," *Proceedings of the 8th International Symposium on Software Reliability Engineering*, pp. 13–26, 1997.

[238] S. Ali, L. C. Briand, H. Hemmati, and R. K. Panesar-Walawege, "A systematic review of the application and empirical investigation of search-based test case generation," *IEEE Transactions on Software Engineering*, vol. 36, no. 6, pp. 742–762, 2010.

[239] T. Foss, E. Stensrud, B. Kitchenham, and I. Myrtveit, "A simulation study of the model evaluation criterion mmre," *IEEE Transactions on Software Engineering*, vol. 29, no. 11, pp. 985–995, 2002.

[240] W. B. Langdon, J. Dolado, F. Sarro, and M. Harman, "Exact mean absolute error of baseline predictor, marp0," *Information and Software Technology*, vol. 73, pp. 16–18, 2016.

[241] M. Shepperd and S. MacDonell, "Evaluating prediction systems in software project estimation," *Information and Software Technology*, vol. 54, no. 8, pp. 820–827, 2012.

[242] H. Zhang and X. Zhang, "Comments on" data mining static code attributes to learn defect predictors"," *IEEE Transactions on Software Engineering*, vol. 33, no. 9, 2007.

[243] S. D. Conte, H. E. Dunsmore, and Y. Shen, *Software engineering metrics and models*. Benjamin-Cummings Publishing Co., Inc., 1986.

[244] A. Arcuri and L. Briand, "A practical guide for using statistical tests to assess randomized algorithms in software engineering," *Proceedings of the 4th International Conference on Software Engineering (ICSE)*, pp. 1–10, 2011.

[245] M. Shepperd and S. MacDonell, "Evaluating prediction systems in software project estimation," *Information and Software Technology*, vol. 54, no. 8, pp. 820–827, 2012.

[246] P. A. Whigham, C. A. Owen, and S. G. Macdonell, "A baseline model for software effort estimation," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 24, no. 3, p. 20, 2015.

[247] A. A. Neto and T. Conte, "A conceptual model to address threats to validity in controlled experiments," *Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering*, pp. 82–85, 2013.

[248] A. Arcuri and G. Fraser, "On parameter tuning in search based software engineering," *International Symposium on Search Based Software Engineering*, pp. 33–47, 2011.

[249] A. Arcuri and G. Fraser, "Parameter tuning or default values? an empirical investigation in search-based software engineering," *Empirical Software Engineering*, vol. 18, no. 3, pp. 594–623, 2013.

[250] J. B. Peñarroya, "Pittsburgh genetic-based machine learning in the data mining era: representations, generalization, and run-time," Ph.D. dissertation, Universitat Ramon Llull, 2004.

[251] L. Di Geronimo, F. Ferrucci, A. Murolo, and F. Sarro, "A parallel genetic algorithm based on hadoop mapreduce for the automatic generation of junit test suites," *Proceedings of Fifth International Conference on Software Testing, Verification and Validation (ICST)*, pp. 785–793, 2012.

[252] D. R. White, "Cloud computing and sbse," *International Symposium on Search Based Software Engineering*, pp. 16–18, 2013.

[253] M. Harman, S. A. Mansouri, and Y. Zhang, "Search-based software engineering: Trends, techniques and applications," *ACM Computing Surveys (CSUR)*, vol. 45, no. 1, p. 11, 2012.

[254] J. Clarke, J. J. Dolado, M. Harman, R. Hierons, B. Jones, M. Lumkin, B. Mitchell, S. Mancoridis, K. Rees, M. Roper *et al.*, "Reformulating software engineering as a search problem," *IEE Proceedings-software*, vol. 150, no. 3, pp. 161–175, 2003.

[255] R. Malhotra, "Search based techniques for software fault prediction: current trends and future directions," *Proceedings of the 7th International Workshop on Search-Based Software Testing*, pp. 35–36, 2014.

[256] C. Grosan and A. Abraham, "Hybrid evolutionary algorithms: methodologies, architectures, and reviews," *Hybrid Evolutionary Algorithms*, pp. 1–17, 2007.

[257] R. Malhotra and M. Khanna, "An exploratory study for software change prediction in object-oriented systems using hybridized techniques," *Automated Software Engineering*, vol. 24, no. 3, pp. 673–717, 2017.

[258] S.-W. Lin, K.-C. Ying, S.-C. Chen, and Z.-J. Lee, "Particle swarm optimization for parameter determination and feature selection of support vector ma-

chines," *Expert Systems with Applications*, vol. 35, no. 4, pp. 1817–1824, 2008.

[259] C.-L. Huang and J.-F. Dun, "A distributed pso–svm hybrid system with feature selection and parameter optimization," *Applied Soft Computing*, vol. 8, no. 4, pp. 1381–1391, 2008.

[260] S.-W. Lin and S.-C. Chen, "Psolda: A particle swarm optimization approach for enhancing classification accuracy rate of linear discriminant analysis," *Applied Soft Computing*, vol. 9, no. 3, pp. 1008–1015, 2009.

[261] E. Arisholm and L. C. Briand, "Predicting fault-prone components in a java legacy system," *Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering*, pp. 8–17, 2006.

[262] L. Kumar and A. Sureka, "Using structured text source code metrics and artificial neural networks to predict change proneness at code tab and program organization level," *Proceedings of the 10th Innovations in Software Engineering Conference*, pp. 172–180, 2017.

[263] L. K. Hansen and P. Salamon, "Neural network ensembles," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, no. 10, pp. 993–1001, 1990.

[264] J. Kennedy and W. M. Spears, "Matching algorithms to problems: an experimental test of the particle swarm and some genetic algorithms on the multimodal problem generator," *Proceedings of the IEEE World Congress on Computational Intelligence*, pp. 78–83, 1998.

[265] M. Abdelhalim and S.-D. Habib, "Particle swarm optimization for hw/sw partitioning," *Particle Swarm Optimization*, 2009.

[266] R. Hassan, B. Cohanim, O. De Weck, and G. Venter, "A comparison of particle swarm optimization and the genetic algorithm," *Proceedings of the 46th AIAA/ASME/ASCE/AHS/ASC structures, structural dynamics and materials conference*, p. 1897, 2005.

[267] T. Sousa, A. Silva, and A. Neves, "Particle swarm based data mining algorithms for classification tasks," *Parallel Computing*, vol. 30, no. 5-6, pp. 767–783, 2004.

[268] R. Moussa and D. Azar, "A pso-ga approach targeting fault-prone software modules," *Journal of Systems and Software*, vol. 132, pp. 41–49, 2017.

[269] R. Malhotra and M. Khanna, "Software change prediction using voting particle swarm optimization based ensemble classifier," *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 311–312, 2017.

[270] R. Malhotra and M. Khanna, "Particle swarm optimization-based ensemble learning for software change prediction," *Information and Software Technology*, vol. 102, pp. 65–84, 2018.

[271] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, "Automated parameter optimization of classification techniques for defect prediction models," *Proceedings of the 38th International Conference on Software Engineering (ICSE)*, pp. 321–332, 2016.

[272] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal, *Data Mining: Practical machine learning tools and techniques*. Morgan Kaufmann, 2016.

[273] A. Tosun, B. Turhan, and A. Bener, "Validation of network measures as indicators of defective modules in software systems," *Proceedings of the 5th*

*international conference on predictor models in software engineering*, p. 5, 2009.

[274] R. Malhotra and M. Khanna, "Dynamic selection of fitness function for software change prediction using particle swarm optimization," *Information and Software Technology*, 2018.

[275] A. Vargha and H. D. Delaney, "A critique and improvement of the cl common language effect size statistics of mcgraw and wong," *Journal of Educational and Behavioral Statistics*, vol. 25, no. 2, pp. 101–132, 2000.

[276] J. Ruscio and B. L. Gera, "Generalizations and extensions of the probability of superiority effect size estimator," *Multivariate Behavioral Research*, vol. 48, no. 2, pp. 208–219, 2013.

[277] A. Elmishali, R. Stern, and M. Kalech, "An artificial intelligence paradigm for troubleshooting software bugs," *Engineering Applications of Artificial Intelligence*, vol. 69, pp. 147–156, 2018.

[278] A. Lamkanfi, S. Demeyer, Q. D. Soetens, and T. Verdonck, "Comparing mining algorithms for predicting the severity of a reported bug," *Proceedings of the 15th European Conference on Software Maintenance and Reengineering (CSMR)*, pp. 249–258, 2011.

[279] T. Menzies and A. Marcus, "Automated severity assessment of software defect reports," *Proceedings of the IEEE International Conference on Software Maintenance*, pp. 346–355, 2008.

[280] Y. Tian, D. Lo, X. Xia, and C. Sun, "Automated prediction of bug report priority using multi-factor analysis," *Empirical Software Engineering*, vol. 20, no. 5, pp. 1354–1383, 2015.

[281] R. Malhotra and M. Khanna, "A novel framework for software bug categorization using change impact and maintenance effort," *Journal of Systems and Software*, 2018.

[282] F. Sebastiani, "Machine learning in automated text categorization," *ACM Computing Surveys*, vol. 34, no. 1, pp. 1–47, 2002.

[283] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?" *Proceedings of the 28th International Conference on Software Engineering*, pp. 361–370, 2006.

[284] R. Malhotra and M. Khanna, "An empirical study for software change prediction using imbalanced data," *Empirical Software Engineering*, vol. 22, no. 6, pp. 2806–2851, 2017.

[285] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: synthetic minority over-sampling technique," *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, 2002.

[286] V. López, A. Fernández, S. García, V. Palade, and F. Herrera, "An insight into classification with imbalanced data: Empirical results and current trends on using data intrinsic characteristics," *Information Sciences*, vol. 250, pp. 113–141, 2013.

[287] P. Domingos, "Metacost: A general method for making classifiers cost-sensitive," *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 155–164, 1999.

[288] J. Van Hulse, T. M. Khoshgoftaar, A. Napolitano, and R. Wald, "Feature selection with high-dimensional imbalanced data," *Proceedings of the IEEE International Conference on Data Mining*, pp. 507–514, 2009.

[289] Y. Liu, A. An, and X. Huang, "Boosting prediction accuracy on imbalanced datasets with svm ensembles," *Proceedings of the Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 107–118, 2006.

[290] R. Malhotra and M. Khanna, "Prediction of change prone classes using evolution-based and object-oriented metrics," *Journal of Intelligent & Fuzzy Systems*, vol. 34, no. 3, pp. 1755–1766, 2018.

[291] R. Malhotra and M. Khanna, "Analyzing evolution patterns of object-oriented metrics: A case study on android software," *International Journal of Rough Sets and Data Analysis*.

[292] M. Alenezi and M. Zarour, "Modularity measurement and evolution in object-oriented open-source projects," *Proceedings of the International Conference on Engineering & MIS*, p. 16, 2015.

[293] E. Nasseri, S. Counsell, and M. Shepperd, "An empirical study of evolution of inheritance in java oss," *Proceedings of the 19th Australian Conference on Software Engineering*, pp. 269–278, 2008.

# Supervisor's Biography


**Dr. Ruchika Malhotra**

Associate Head & Associate Professor
Discipline of Software Engineering
Department of Computer Science & Engineering
Delhi Technological University
Email: ruchikamalhotra@dtu.ac.in


**Educational Qualifications:**

Ph.D.(Information Technology), MCA, BIS(H)

Ruchika Malhotra is an Associate Professor in Discipline of Software Engineering, Department of Computer Science & Engineering, Delhi Technological University (formerly Delhi College of Engineering), Delhi, India. She is Associate Dean in Industrial Research and Development, Delhi Technological University. he has been awarded prestigious UGC Raman Postdoctoral Fellowship by the Indian government for pursuing postdoctoral research from the Department of Computer and Information Science, Indiana University-Purdue University Indianapolis (2014-15),

Indianapolis, Indiana, USA. She received her master's and doctorate degree in software engineering from the University School of Information Technology, Guru Gobind Singh Indraprastha University, Delhi, India. She was an Assistant Professor at the University School of Information Technology, Guru Gobind Singh Indraprastha University, Delhi, India. She has received IBM Faculty Award 2013. She is the recipient of Commendable Research Award by Delhi Technological University for her research in the year 2017 and 2018. She is the author of book titled "Empirical Research in Software Engineering" published by CRC press and co-author of a book on Object Oriented Software Engineering published by PHI Learning. Her research interests are in software testing, improving software quality, statistical and adaptive prediction models, software metrics and the definition and validation of software metrics. Her h-index is 26 as reported by Google Scholar. She has published more than 160 research papers in international journals and conferences.

# Author's Biography

**Ms. Megha Ummat**
Research Scholar
Department of Computer Science & Engineering
Delhi Technological University
Assistant Professor
Sri Guru Gobind Singh College of Commerce, University of Delhi
Email: meghakhanna@gmail.com

**Educational Qualifications:**

MCA (SE), B.Sc. (H) Computer Science

Megha Ummat is currently pursuing her doctoral degree from Delhi Technological University. She is currently working as Assistant Professor in Sri Guru Gobind Singh College of Commerce, University of Delhi. She completed her master's degree in software engineering in 2010 from the University School of Information Technology, Guru Gobind Singh Indraprastha University, India. She received her graduation degree in computer science (Hons.) in 2007 from Acharya Narendra Dev College, University of Delhi. She is the recipient of Commendable Research Award by Delhi

Technological University for her research in the year 2017 and 2018. She was also awarded the ”Research Incentive” for her research in the year 2018 by the Governing body of Sri Guru Gobind Singh College of Commerce. Her research interests are in software quality improvement, applications of machine learning techniques in change prediction, and the definition and validation of software metrics. Her h-index is 7 as reported by Google Scholar. She has various publications in international conferences and journals.