EMPIRICAL VALIDATION OF OBJECT-ORIENTED METRICS FOR IMBALANCED CLASSIFICATION USING OPEN SOURCE SOFTWARE

By

JUHI JAIN

Roll No.: 2k16/Ph.D/CO/03

Under the guidance of

Prof. Ruchika Malhotra

Professor & Head of Department,

Department of Software 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

2021

Declaration

I, **Juhi Jain**, Ph.D. student Roll No.: 2k16/Ph.D/CO/03, hereby declare that the thesis entitled **"Empirical Validation of Object-Oriented Metrics for Imbalanced Classification using Open Source Software"** 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 : 8-Nov-2021

Place : Delhi

Juhi Jain
erjuhijain@gmail.com
Roll No.: 2k16/Ph.D/CO/03
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 **"Empirical Validation of Object-Oriented Metrics for Imbalanced Classification using Open Source Software"** has been completed by **Ms. Juhi Jain** Roll No.: 2k16/Ph.D/CO/03 under the guidance of **Prof. 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

Prof. RUCHIKA MALHOTRA

Professor & Head of Department

Department of Software Engineering

Delhi Technological University, Delhi 110042

"This thesis is dedicated to Three Musketeers of my Life: my mother ***Smt. Jeevan Jain****, my doting sister* ***Mrs. Ritu Jain****, and my guide* ***Prof. Ruchika Malhotra****. "*

Acknowledgment

Working as a Ph.D. student in Delhi Technological University was a magnificent as well as challenging experience to me. In all these years, many people were instrumental directly or indirectly in shaping up the academic calendar. It was hardly possible for me to thrive in my doctoral work without the support of these personalities.

First of all, I would like to thank my supervisor **Prof. Ruchika Malhotra** for introducing me to the world of Software Engineering. It was her valuable guidance, cheerful enthusiasm, and ever supporting nature that pushed me to improve and complete my research goals in a respectable manner. She always leaves a mark in her students' hearts by her incredible knowledge, hardwork, punctuality, dedication, credibility and never-give-up attitude. I feel fortunate that she chose me as her student and guides me both personally and professionally. Her unconditional support, valuable time and persistent encouragement have been of inestimable value throughout my journey of research. I am forever indebted to Ma'am for her endless patience and her innumerable lessons. I am grateful to her for being a sturdy pillar that helped me complete my research work. I could not have imagined having a better counsellor and mentor for my PhD study. I am blessed to have a guiding light for my entire life in the disguise of my Ph.D guide.

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

I am grateful to **DRC members**, **SRC members**, Delhi Technological University and other **faculty members** of the Department of Computer Science & Engineering, Delhi Technological University for their help and cooperation while carrying out this research work.

I would like to say a heartfelt thanks to my husband, **Mr. Mitesh Jain** who has been my side throughout this PhD, living every single minute of it, for his encouragement, courage and invaluable support during the course of my research work. All my love for my darlings

Vivaan and **Arha** for making my journey easy by adding their smiles and filling my life with joy and happiness.

There are no proper words to convey my deep gratitude and respect for my parents **Sh. Raja Ram Jain**, **Smt. Jeevan Jain** and my brother **Sandeep Jain** who are my constant pillars of strength and support during all these years. My Ph.D journey is generously supported by my sister **Ritu Jain**, sister-in-law **Parul Jain**, and niece-cum-friends **Anishka-Mimansha**. I can not be more grateful for their unwavering support for my decisions on the turning points in life.

Lastly, my regards and appreciation to my parents-in-law **Shri Shripal Jain**, **Smt. Bimla Jain** and my sisters-in-law **Monika Jain**, **Dr. Manisha Gupta**, and **Ekta Gupta** for providing the emotional support and making it possible for me to complete what I started.

Thank you all for allowing me to chase my dreams and for being the part of this wonderful journey.

Juhi Jain

Abstract

Software are an inextricable part of our lives. With the ever-growing complexity of software, designing and integrating changes in these software is always a tedious task for developers and software practitioners. One of the prime concerns while implementing changes is to maintain the quality of software products as there are fewer resources and rigid deadlines. If defects are uncovered in later stages of software development, the cost of detecting and removing them amplifies exponentially. This may result in poor software development processes and software quality degradation. With the constraints of strict time schedules and limited resources, it becomes the utmost requirement of software developers and practitioners to discover these defects early. Finding defects or faults in the early phases of the software development life cycle leads to better planning and reduced cost, effort, and resources [1].

Software metrics are widely used for generating defect prediction models. Different object-oriented (OO) metrics define different internal attributes of the software like cohesion, coupling, size, inheritance, encapsulation, etc. Therefore, these metrics are utilized to envisage whether a software class can be defective or not [2, 3]. Selection of relevant metrics aids in effective predictive modelling for finding defects. We evaluated the correlation-based feature selection for identifying the important metrics that are related to defect-prone areas in the software.

Various machine learning (ML) and statistical techniques have been used for developing prediction models to ascertain defect-proneness in the literature. We discovered a new category of classification techniques, search-based techniques (SBTs), that is rarely used

in the Software Defect Prediction (SDP) domain. We assessed the effectiveness of ML techniques and SBTs for developing models that predict defective classes in the OO software. We further extended the use of genetic algorithm variants for feature selection and performed the comparative analysis with Correlation Feature Selection (CFS).

One of the major issues that have been observed in software data is the imbalanced data problem. If there is a fewer number of instances of one type of class than that of another class, then data is said to have an imbalanced data problem. For our application, if in software defective classes are less than non-defective classes, then it is said to be imbalanced. We conducted a structured review to analyze the ways of tackling imbalanced data problem for developing the defect prediction models. The review results will help in identifying best practices and research gaps if any.

Imbalanced data problem can be treated either at the data level or algorithm level. At the data level, we developed ML models using resampling methods to assess their impact on defect-proneness. At the algorithm level, cost-sensitive learning is employed to tackle the imbalanced data issue. The impact of different MetaCost learners was investigated for optimum defect prediction in the software. Studies in literature have advocated the use of ensemble methodology for various software prediction tasks. We evaluated the ensemble methods after treating the data with resampling methods. The incorporation of resampling methods will alleviate the imbalanced data problem resulting in better model prediction.

We assessed the effectiveness of OO metrics, ML techniques, SBTs, resampling methods, and MetaCost learners for developing SDP models.

Contents

List of Tables

List of Figures

List of Publications

Papers Accepted/Published in International Journals

1. R. Malhotra and J. Jain, ”Predicting Software Defects for Object-oriented Softwares using Search-based Techniques”, *International Journal of Software Engineering and Knowledge Engineering*, 31(2), pp.193-215, 2021.

2. R. Malhotra and J. Jain, ”Predicting Defects in Imbalanced Data using Resampling Methods: An Empirical Investigation”, *PeerJ Computer Science*, 7:e573 DOI 10.7717/peerj-cs.573, 2021.

3. R. Malhotra and J. Jain, ”Empirical Validation of Evolutionary Feature Selection Techniques for Software Defect Prediction”, *Advances in Electrical and Computer Engineering*.

Papers Accepted/Published in International Conferences

4. R. Malhotra and J. Jain, "Predicting defects in object-oriented software using cost-sensitive classification", in *International Conference on Computational Research and Data Analytics*, 2020.

5. R. Malhotra and J. Jain, "Handling Imbalanced Data using Ensemble Learning in Software Defect Prediction", in *Confluence-2020 : 10th International Conference on Cloud Computing, Data Science & Engineering*, 2020.

6. R. Malhotra and J. Jain, "Using Class Imbalance Learning and Search - Based Techniques for Object Oriented Software Defect Prediction", in *International Conference on Recent Trends in Engineering, Technology and Business Management*, 2019.

Papers Communicated in International Journals

7. R. Malhotra and J. Jain, ”Systematic Literature Review on Imbalanced Software Defect Prediction”, *Frontiers of Computer Science*.

8. R. Malhotra and J. Jain, ”On Alleviating Class Imbalance Problem using Ensembles in Software Defect Prediction: An Empirical Investigation”, *Romanian Journal of Information Science and Technology*.

9. R. Malhotra and J. Jain, ”Using Class Imbalance Learning and Search-based Techniques for Object Oriented Software Defect Prediction”, *International Journal of System Assurance Engineering and Management*.

Abbreviations

AB	**AdaBoost**
ABM1	**Adaptive Boosting AdaboostM1**
ABNC	**AdaBoostNC**
ADSYN	**Adaptive Synthetic Sampling**
AHC	**Agglomerative Hierarchical Clustering**
AMC	**Average Method Complexity**
ANA	**Average Number of Ancestors**
Bag	**Bagging**
BIOHEL	**Bioinformatics-oriented hierarchical evolutionary learning**
BN	**Bayesian Network**
BPNN	**Back Propagation Neural Network**
Ca	**Afferent Coupling**
CAM	**Cohesion among Methods of a Class**
CART	**Classification and Regression Tree**
CBM	**Methods of a Class**
CBO	**Coupling Between Object classes**
Ce	**Efferent Coupling**
CFS	**Correlation Feature Selection**
CHC	**CHC Adaptative Search for Instance Selection**
CIS	**Class Interface Size**
CK	**Chidamber and Kemerer**

CNNTL	**Condensed Nearest Neighbor+Tomek's modification of Condensed nearest Neighbor**
CORE	**CO-Evolutionary Rule Extractor**
CPSO	**Constricted Particle Swarm Optimization**
CS	**Class Size**
CSDP	**Collaborative representation classifier based Software Defect Prediction**
DAC	**Data Abstraction Coupling**
DAM	**Data Access Metric**
DB	**DataBoost**
DCC	**Direct Class Coupling**
DIT	**Depth of Inheritance Tree**
DS	**Decision Stump**
DSC	**Design Size in Classes**
DT	**Decision Table**
ET	**Extra Tree**
FNR	**False Negative Ratio**
FPR	**False Positive Ratio**
GA_ADI	**Genetic Algorithm based Classifier System with Adaptive Discretization Intervals**
GA_INT	**Genetic Algorithm based Classifier System with Intervalar Rule**
GFS	**greedy forward selection**
GGA	**Generational Genetic Algorithm for Instance Selection**
GP	**Genetic Programming**
IBk	**K-Nearest Neighbor1**
IC	**Inheritance Coupling**
ICO	**Iterative Classifier Optimizer**
IEEE	**Institute of Electrical and Electronics Engineers**

IIVOT	**IIvotes**
ILGA	**Incremental Learning with Genetic Algorithms**
J48	**C4.5 decision tree**
KLLD	**Karhunen Loeve Decomposition**
KNN	**K-Nearest Neighbor**
KStar	**K-Nearest Neighbor2**
LARS	**Least Angle Regression**
LB	**LogitBoost**
LCOM	**Lack of Cohesion in Methods**
LDA	**linear Discriminant Analysis**
LDC	**Linear discriminant classifiers**
LDWPSO	**Linear Decreasing Weight - Particle Swarm Optimization**
LLTS	**Large Legacy Telecommunications Software system**
LMNC	**Levenberg-Marquardt feed-forward neural network**
LMT	**Logistic Model Tree**
LOO	**Leave-one-out cross-validation**
LR	**Logistic Regression**
LTR	**learning to rank**
MFA	**Method of Functional Abstraction**
ML	**Machine Learning**
MLP	**Multilayer Perceptron**
MOA	**Measure of Aggression**
MPC	**Message Pass Coupling**
MPLCS	**Memetic Pittsburgh Learning Classifier System**
MSBAG	**MSMOTEBagging**
MSMTB	**MSMOTEBoost**
NB	**Naïve Bayes**

NBR	**Negative Binomial Regression**
NCL	**Neighborhood Cleaning Rule**
NMC	**Nearest Mean Classifier**
NMSC	**Scaled Nearest Mean Classifier**
NN	**Neural Network**
NNge	**Non-Nested Generalization**
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 (methods) Overridden by a subclass**
NOP	**Number of Polymorphic Methods**
NPM	**Number of Public Methods**
OBAG	**OverBagging**
OO	**Object-Oriented**
OSS	**One Sided Selection**
PART	**Partitioning**
Parzen	**Parzen Window Classifier**
PBIL	**Population-Based Incremental Learning**
PCLD	**Linear classifier based on principle component analysis**
PERL	**linear perceptron classifier with batch processing**
QUADRC	**Quadratic classifier with regularization parameter**
QDC	**Quadratic discriminant classifiers**
QMOOD	**Quality Model for Object-Oriented Design**
RBFN	**Radial basis Function Network**
REPSO	**Real Encoding - Particle Swarm Optimization**

RF	**Random Forest**
RFC	**Response For a Class**
RIPPER	**Repeated Incremental Pruning to Produce Error Reduction**
ROC-AUC	**Area Under Receiver Operating Characteristic Curve**
ROS	**Random OverSampling**
RPART	**Recursive Partitioning**
RQ	**Research Question**
RSS	**RandomSubSpace**
RT	**Random Tree**
RUS	**Random UnderSampling**
RUSB	**RUSBoost**
RVM	**Relevance Vector Machine**
SBT	**Search-based Technique**
SDP	**Software Defect Prediction**
SGA	**Steady-State Genetic Algorithm for Instance Selection**
SGD	**Stochastic Gradient Descent**
SIA	**Supervised Inductive Algorithm**
SIX	**Specialization Index**
SL	**Simple Logistic**
SLSMT	**Safe Level Synthetic Minority Over-sampling Technique**
SMO	**Sequential Minimization Optimization**
SMT	**Synthetic Minority Over-sampling Technique**
SMTB	**SMOTEBoost**
SMTBAG	**SMOTEBagging**
SPD	**Selective Preprocessing of Imbalanced Data**
SRC	**Sparse Representation Classifier**
SUBC	**Subspace classifier**

SVM	**Support Vector Machine**
TS3WD	**three-way decisions based two-stage ranking method**
UBAG	**UnderBagging**
UBAG2	**UnderBagging2**
UCS	**sUpervised Classification System**
UDC	**Uncorrelated normal densities based quadratic Bayes classifier**
UOBAG	**UnderOverBagging**
VFI	**Voting Features Interval**
VP	**Voted Perceptron**
WEKA	**Waikato Environment for Knowledge Analysis**
WMC	**Weight Methods per Class**
XCS	**X-Classifier System**

Chapter 1

Introduction

1.1 Introduction

With the emergence of the information age and ever-growing software, people are dependent on software for day-to-day tasks in their life. Every software organization wants to deliver reliable and good quality software with the least effort and time. Providing good quality software is always a demanding task for software developers. For software quality assurance, it is mandatory to plan software testing. With a large number of modules/classes in software, it is not feasible to test every part of the code with constrained resources. Luckily, software follows the Pareto principle [5]. This fact states that usually only 20% of defective classes are responsible for all the defects in software. Hence, it is important to identify these classes for proper resource utilization. Defect-prone classes need more resources and more attention from developers. Early detection of the possibility of defective classes assists in proper resource allocation ensuing in optimizing resource utilization and increasing user satisfaction and software reliability [1].

The quality of software can be improved by mining existing software repositories for various software attributes. Many studies in the literature have established a positive re-

lationship between software metrics and predicting defects in the software. Software is comprised of classes and classes can be defined by various OO metrics. The OO metrics of software are good predictors for the development of defect prediction models.

Software quality is of utmost importance to the developer, software practitioner, manager, or researcher. Software quality can be assessed by using internal attributes of software [6]. According to the Institute of Electrical and Electronics Engineers (IEEE), software quality is defined as [7] :

- *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*

According to [1], software quality determines the **quality of design** and the **quality of conformance**. The design of the software should accommodate features for user satisfaction with the minimum number of defects. Quality of conformance is evaluated by the extent to which the software conforms to the developed design.

Software quality can be evaluated in terms of different quality attributes like functionality, maintainability, reliability, etc. The focal point of this research work is defect proneness or SDP.

1.2 Predictive Modelling

Predictive modelling a.k.a. predictive analytics is a widely used statistical technique to create a process and validate a type of model that can help in forecasting future outcomes. It is a data mining technique that helps one to answer the most frequent question of ”What might happen in the future?” Predictive modelling refers to the construction of models for determining any particular attribute. The scope of this predictive analysis doesn't stop here, it can be applied to any type of unknown event regardless of when it has occurred. They

are designed in a way to access historical data, observe trends, discover data, customer trends, and use such information to draw up prediction trends and their economic value. For instance, if a bank wants to identify which of its customers are likely to engage in large fragments of money laundering activities, predictive modelling provides the answer. A predictive model is built using the bank's customer data around the number of money transfers that they made during a particular period. The model will be taught to identify the difference between normal money transactions and money laundering. The model will be made in a way as to report the bank for the occurrence of fraud by any of the customers. Similarly, predictive modelling can be extended to any real-life application where we want to predict or classify based on previous information.

1.2.1 Steps in Predictive Modelling

Steps followed in predictive modelling are illustrated in Figure 1.1 and explained below:

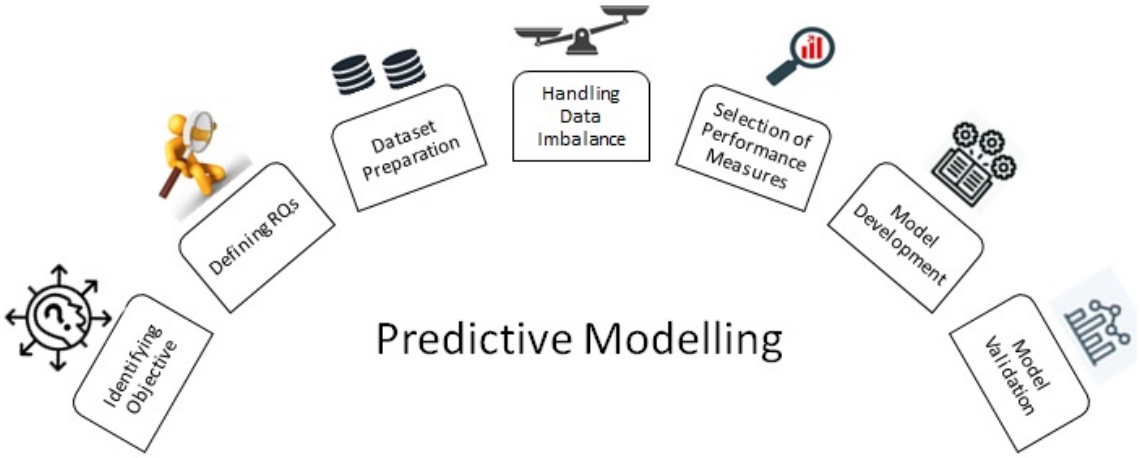

Figure 1.1: Steps in Predictive Modelling

1. **Identifying Objective:** The typical lifecycle involved in building a predictive model start with identifying the objective for any application such as the risks, revenue, fraudulent activities, defect, etc.

2. **Defining Research Questions (RQs):** It is a crucial step to define the goals to achieve the desired objective. In this step, research questions are formulated. A particular process needs to be defined to answer these RQs.

3. **Dataset Preparation:** The model needs to be trained using previous data. This data is collected in form of software metrics from the applications. In this step, we perform data cleaning and data transformation to make data fit for good quality prediction.

4. **Handling Data Imbalance:** Classification can be binary or multiclass depending on the application. If one of the classes is in majority than other classes, there is a requirement of application of imbalance learning method to balance the data. If data is imbalanced, training of the model will be infelicitous resulting in inaccurate predictions of the outcome.

5. **Selection of Performance Measures:** Performance measures are selected to evaluate and compare the prediction models. These performance measures can be computed using the confusion matrix obtained after the execution of the developed model. For imbalanced data, stable measures like G-Mean, Balance, and Area Under Receiver Operating Characteristic Curve (ROC-AUC) should be exercised.

6. **Model Development:** The next step on the way to predictive modelling is model development. For an effective model, several data analysis techniques are opted by an organization. The model is trained using training data and a test dataset is used to verify the outcomes of the model.

7. **Model Validation:** To build a sturdy model, the developed model is validated against statistical tests.

1.2.2 Issues in Predictive Modelling

One of the major issues faced by software practitioners is the lack of empirical evidence for the quality and applicability of the proposed model. This makes the course of acceptance of that method or technology challenging in the real software industry [8, 9]. Another important concern in predictive modelling is the imbalanced data problem in real-world problems. The problem of imbalanced data has gained more attention in the field of software engineering recently because of its existence in some of the predominant software quality attributes like defect prediction. Dealing with imbalanced data problem will help in the development of a reliable model resulting in quality improvement and user satisfaction. With such prominent challenges in mining software repositories, there is a need for the development of new SDP models to anticipate the future defects [10].

1.2.3 Software Defect Prediction

The explosive growth in the availability and size of software data has led to the necessity of effective software defect prediction models. Early identification of defective classes is crucial for the project to be successful. It aids in software quality assurance by effective resource allocation and accurate budget estimation. Software testing if not done competently can emanate defects in the software, ultimately raising the cost and other resource requirements [11]. With the limited resources, SDP allows assigning these resources to those portions of software that tend to embrace more defects. The SDP model can be defined as a classification model that is used to spot defect-prone zones in the software. SDP helps in prioritizing testing activities in the software. SDP models not only uncover the probable defective classes but also helps in the assessment of the influence of various software metrics on those defective classes. Many software industries like Google [12], IBM [13], Microsoft Research [14] have reported that the insights provided by the adop-

tion of SDP models are fruitful for the use of SDP models have assisted them in delivering quality software products. SDP models target to either envisage the number of defects in class or predict whether the class can be defective or non-defective. Accordingly, SDP techniques can be categorized as regression or classification techniques. This work embodies classification techniques only. As we need to predict whether the class is defective or non-defective, SDP is considered a binary problem.

1.2.3.1 Predictive Modelling in SDP

The important lesson in predictive modelling for SDP is to learn from past mistakes. Organizations can be benefitted by keeping into consideration the defect data of previous projects or previous versions of the project [15].

The technique is to plan, structure, and control a developing model in an organization. The data that is put in the model needs to be balanced in a way as to the presence of the defective and non-defective data in a balanced way to help develop a model that will be efficient in a way as to help the organization grow and know more about the trends and help remove defects.

Figure 1.2 provides the pictorial representation of the process followed in SDP through predictive modelling.

The process can be broadly divided into three phases.

- **Phase 1 Dataset Preparation:** Software metrics are collected from the software repositories for different classes and a defect label is assigned to each class. A class can be defective or non-defective. Dataset preparation also includes the cleaning of the data and the transformation of data. The quality of the model depends on the quality of the data.

- **Phase 2 Model Development:** In this phase, data is split into a training set and test set based on the cross-validation method. Data analysis techniques like machine

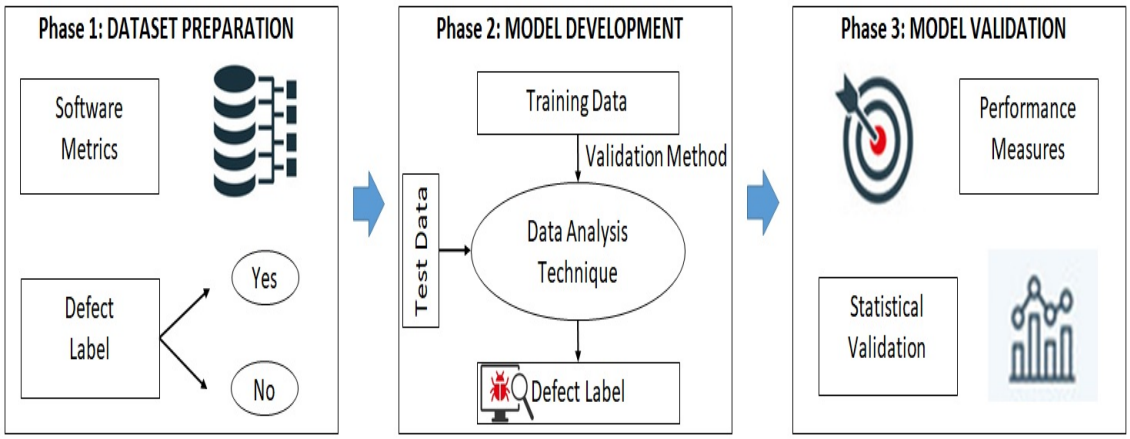

Figure 1.2: Predictive Modelling in SDP

learning techniques, statistical techniques, or search-based techniques are selected according to the nature of the data and used to train the model. The quality of the constructed model is assessed by classifying the unseen data (test data) as defective or non-defective.

- **Phase 3 Model Validation:** SDP models developed need to be evaluated based on some criteria. Therefore, in this phase performance measures, e.g., accuracy, sensitivity, precision, ROC-AUC, G-Mean, etc are selected and models are validated to predict new defects. Statistical validation of results is essential to assure the fitness of the SDP models. Hence statistical tests are carried out to validate the models.

1.2.4 Software Metrics

Many researchers make use of the static code attributes of software to build these SDP models [3, 16]. Static code attributes are metrics that describe internal quality attributes of software like size, complexity, cohesion, inheritance, etc. Metrics can be traditional metrics derived from procedural language software or OO metrics derived from OO software. Procedural metrics are incapable of capturing the properties of OO software [17]. Further, with

help of OO metrics, we can foretell defects in the early phases of the software development lifecycle, e.g., design phase [18]. Since preferred languages by developers and software practitioners are OO in nature, this thesis work focuses on OO metrics only for analyzing defect-prone areas in OO software. Details of OO metrics are provided in subsection 1.4.1.

1.3 Imbalanced Data Problem and its Solutions

This section provides a gateway to the most common problem in the underlying data used for model construction. Real-world data frequently face the data imbalance problem and measures should be taken to alleviate this problem. The more the balance amongst the type of classes, the better will be the prediction model trained on that data.

1.3.1 Imbalanced Data Problem

Many real-life problems like medical diagnosis [19, 20], fraud detection [21–23], text categorization [24, 25], sentiment analysis [26], and churn prediction [27] encounter imbalanced data problem. Imbalanced data represents the data in which instances recorded for one (or more) class type is(are) in minority as compared to other class(es). There is uneven distribution of instances of different classes. The focus of this thesis work is binary defect classification. Data is considered imbalanced if the number of defective classes is much lower than the number of non-defective classes. This imbalanced distribution of defect data misguides classifiers while learning the defective class correctly and hence results in biased and inaccurate results. With such imbalanced data defect predictors may be able to classify non-defective classes rightly, but the same is not true for defective classes. For example, let us assume a software has 1000 classes. If 10% of the total classes are defective, then we have 900 non-defective classes and 100 defective classes. Now, even if the classifier is not able to detect a single class as non-defective, still the accuracy of the

classifier will be 90%. Defective classes are underrepresented and the model depicts good accuracy. This situation is non-acceptable because the results are misleading. How can a classifier be reliable when it is incompetent to predict a single defective class? Such models that seem reliable due to their high accuracy, if implemented, will lead to huge monetary loss to the company and will tarnish the company's image. The scarcity of defective data is responsible for inefficient learning of the model resulting in inaccurate model predictions. A good defect prediction model will be the one that is trained on the similar distribution of instances of defective and non-defective classes. Therefore, it is important to handle the imbalanced data problem for building good quality SDP models and propose solutions to diminish the skewness in the distribution of defective and non-defective classes.

1.3.2 Ways to alleviate Imbalanced Data Problem

Imbalanced data problem is identified as one of the major apprehensions in data mining and several imbalance learning methods exist in literature to deal with this issue [28].

Imbalanced data problem can be handled at the following levels to treat the skewness in data distribution [29]:

Alleviating Imbalanced Data Problem at Data Level

At the data level, modification in the number of data points corresponding to a particular class is done for the training set [30]. Instances of majority class can be removed or instances of minority class can be added [31]. The former is referred to as undersampling and the latter is termed as oversampling. Oversampling and undersampling methods collectively are known as data resampling methods [32]. Oversampling may lead to overfitting and undersampling tends to have a loss of information. Therefore, to decrease negative effects and incorporate the benefits of both worlds, the combination of oversampling and undersampling can also be used.

Alleviating Imbalanced Data Problem at Algorithm Level

At the algorithm level, either a new algorithm is designed or existing algorithms are exploited to reduce the bias generated by the imbalanced data. One of the most popular solutions in this category is cost-sensitive classification. Classification is performed by penalizing the misclassification costs in the cost matrix. The main objective is to reduce the cost of wrong classifications for imbalanced data. Another effective way in this area is the use of ensemble methods. Weak learners are grouped with aim of reducing variance and bias. Several weak learners get transformed into a strong learner.

Alleviating imbalanced Data Problem using Hybrid Methods

Hybrid methods are generally an amalgamation of methods used at the data level and algorithm level. Data resampling methods can be integrated with ensemble methods to form more accurate and robust defect predictors.

In addition to this, the use of stable performance measures also assists in the development of correct and credible defect prediction models [33].

1.4 Literature Survey

Software defect prediction is an essential part of software development and aids efficient resource utilization. Researchers and software practitioners have been working hard to optimize the processes and other facets related to defect prediction. In real life, data is mostly imbalanced. This aggravates the problem and should be maneuvered for construction of reliable models. Therefore, it is crucial to conduct a comprehensive study of existing literature covering different elements and processes that require improvement. This would aid in identification of research gaps and provides motivation to work on different aspects of this area.

First, we discussed object-oriented metrics that are used to determine software defects. Next, we discussed the literature studies conducted to construct defect prediction models with different data analysis techniques, feature selection techniques, and imbalanced

data learning methods to analyze the current state of SDP. Studies that used feature selection techniques to handle the curse of dimensionality are entailed in subsection 1.4.2. Subsection 1.4.3 refers to SDP studies incorporating various machine learning methods. With a recent inclination toward search-based techniques, work done in the SDP field using these techniques is also summed in subsection 1.4.4. Subsection 1.4.5 summarized the defect-related studies that used resampling methods, cost-sensitive classification, ensemble methods, and hybrid methods used to tackle imbalanced data problem.

1.4.1 Object-oriented Metrics

Object-oriented metrics define the internal attributes of the software and provide the context-independent view of software quality. Internal quality attributes include cohesion, coupling, inheritance, encapsulation, composition, complexity, etc. Assessment of OO metrics guides to the evaluation of external software quality attributes. One of the early research that established the relationship between metrics and defect-proneness was carried out by Akiyama et al. [34]. Studies like [2, 35] have established a strong correlation between OO metrics and SDP. They proved the effectiveness of Chidamber and Kemerer (CK) metrics for software quality prediction.

Various OO metric suites have been proposed and explored by the researcher community to capture the internal quality attributes of the software.

CK metric suite [36] is a collection of six metrics that includes Response For a Class (RFC), Weight Methods per Class (WMC), Number Of Children (NOC), Coupling Between Object classes (CBO), Depth of Inheritance Tree (DIT), and Lack of Cohesion in Methods (LCOM) metrics.

Quality Model for Object-Oriented Design (QMOOD) metrics suite is proposed by Bansiya and Davis [37]. This metric suite includes Number of Methods (NOM), Data Access Metric (DAM), Number of Polymorphic Methods (NOP), Design Size in Classes

(DSC), Method of Functional Abstraction (MFA), Class Interface Size (CIS), Number of Hierarchies (NOH), Direct Class Coupling (DCC), Cohesion among Methods of a Class (CAM), Measure of Aggression (MOA), and Average Number of Ancestors (ANA).

Lorenz and Kidd [38] proposed the following OO metrics: 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).

Li and Henry [18] proposed coupling metrics and size metrics. Coupling metrics consist of Data Abstraction Coupling (DAC), Message Pass Coupling (MPC), and NOM. Two size metrics namely SIZE1 and SIZE2 were also part of this metric set.

Braid et al. [39] introduced 18 OO metrics covering majorly the coupling aspects of the software.

Martin [40] also introduced two coupling metrics namely Afferent Coupling (Ca) and Efferent Coupling (Ce).

Tang et al. [41] proposed Coupling Between Methods of a Class (CBM), Number of Object/Memory Allocation (NOMA), Average Method Complexity (AMC), and Inheritance Coupling (IC).

1.4.2 Feature Selection in Software Defect Prediction

A large number of metrics can hamper the defect prediction capabilities of ML models. The reason is the presence of redundant or irrelevant metrics. This curse of dimensionality can be reduced by using feature reduction strategies. This involves either feature selection–reducing the number of features, or feature extraction–extracting new features for existing ones. Many feature selection (FS) techniques are assessed for the construction of effective SDP models [42–45]. Malhotra in [46] has extensively investigated 18 ML techniques using ROC-AUC over 33 open-source datasets by collecting their OO metrics. She

selected relevant metrics using CFS and empirically proved that Naïve Bayes, LogitBoost, and Multilayer Perceptron built good SDP models. Zhou et al. [47] improved the defect prediction by proposing the FS using a cascade of random forests which becomes a deep forest. Xu et al. [48] inspected the effect of 32 FS techniques for the SDP problem on two NASA datasets and one AEEEM dataset, and their result analysis shows that the effectiveness of these FS methods on defect prediction performance varied significantly over all the datasets. Mutukumaran et al. [49] performed an empirical investigation of an embedded method, seven filter methods, and two wrapper methods and found their performances to be equivalent. They considered NASA data and AEEEM data for their study. Moreover, Laradji et al. [50] compared three FS techniques including CFS, and in their findings, they concluded that the wrapper method- greedy forward selection (GFS) performed better than the filter method- CFS. They used ROC-AUC as the performance measure. Ghotra et al. [42] provided the benchmark for SDP using FS techniques. They evaluated the performance of 21 ML techniques with 30 FS. They found that the CFS with the best first search outperforms the other addressed feature selection techniques but they did not explore any evolutionary methods. A review by Malhotra [51] has revealed that CFS is the most commonly used feature selection technique. Ghotra et al. [42] also explored 30 feature selection techniques and concluded CFS as the best feature predictor. They used NASA datasets and PROMISE datasets with 21 ML techniques. Balogun et al. [52] explored feature selection and feature reduction methods for five NASA datasets over four ML techniques and experimentally concluded that FS techniques did not show consistent behavior for the datasets or ML techniques. Recently, Balogun et al. [53] empirically investigated the effect of 46 feature selection techniques over 25 datasets from different sources using Naïve Bayes and decision trees. Based on the accuracy and ROC-AUC performance, they also concluded CFS as the best performer in the FSS category.

1.4.3 Software Defect Prediction using Machine Learning Techniques

SDP has been always a hot research topic. Still, analysis of different studies projects the incongruous interpretations [54].

ML techniques are commonly used to build effective defect prediction models [46, 55–58]. Malhotra and Singh [55] used artificial neural networks, Logitboost, Adaboost, Naïve Bayes, Bagging, KStar, Logistic Regression, and Random Forest to predict defects using ROC-AUC on the open-source dataset Arc. Logitboost was concluded as the best performer with an ROC-AUC of 0.806. Lessman et al. [58] developed the SDP models for ten NASA datasets and provided an empirical comparison of the performance of 22 ML techniques based on ROC-AUC. Though model performances were considered good in terms of defect prediction no statistical difference was found amongst the performance of a wide range of ML techniques.

Bowes et al. [57] analyzed the performance of Naïve Bayes, Random Forest, SVM, and RPart on 12 NASA datasets, three commercial datasets, and three open-source datasets (Ant, Tomcat, and Ivy) and calculated their effectiveness using f-measure and Matthew's coefficient of correlation. They found Random Forest as the best ML performer for defect classification.

Iqbal et al. [56] evaluated the performance of Radial Basis Function Network, Naïve Bayes, Random Forest, KStar, K Nearest Neighbor, Multi-Layer Perceptron, Support Vector Machine, Decision Tree, PART, and One Rule over 12 NASA datasets based on accuracy, precision, recall, f–measure, Matthew's coefficient of correlation and ROC-AUC but they did not execute any statistical validation of their results. They suggested performing feature selection for better and accurate predictions.

1.4.4 Software Defect Prediction using Search-based Techniques

With many advantages like easy adaptations and computational efficiency, Harman [59] advocated analyzing the suitability and applicability of evolutionary techniques in the field of SDP. Though a lot of studies in the literature have explored the nature of machine learning techniques for predicting defects [51], comparatively very few studies exist using SBTs in this direction to date. researchers are exploiting SBTs for feature extraction [60–63], parameter optimization [63–68] and building defect prediction models [4, 69–71]. A lot of recent work is done in direction of parameter tuning. In parameter optimization, for example, Cai et al. [67] used a hybrid cuckoo search approach, and Wu et al. [68] proposed a bat algorithm for parameter selection in support vector machine. Cai et al. [67] worked on 7 NASA datasets. But research studies in the other two domains are meager in number.

Singh et al. [70] interpreted the efficacy of gene expression programming for predicting defective modules in industrial software. Carvalho et al. [69] introduced multiobjective algorithmic solution for SDP. They developed multi-objective particle swarm optimization-based defect prediction model. This rule-based model illustrated the best ROC-AUC values for defect prediction in five NASA datasets as compared to the J48, Repeated Incremental Pruning to Produce Error Reduction (RIPPER), and Non-Nested Generalization (NNge). Rodriguez et al. [71] also proposed a genetic algorithm (Evolutionary Decision Rules for Subgroup Discovery) to construct rules for identifying minority classes. Chatterjee et al. [72] proposed a defect prediction model based on GA with a neuro-fuzzy approach to predict defects in two continuous datasets whereas Manjula and Florence [73] exploited GA for feature selection and classified defects in five NASA datasets by deep neural networks. Malhotra [4] has extensively analyzed 20 SBTs with their hybrid versions and machine learning techniques on open source software for SDP and empirically concluded that models developed using SBTs and their hybrid versions have the good and comparable predictive capability for defect prediction.

1.4.5 Software Defect Prediction using Imbalanced Learning Methods

In the last decade, researchers are inclined towards the building of SDP models with improved data. By improved data, the emphasis is on handling the skewness of data.

Areas of SDP and software change prediction [74, 75] are explored to handle imbalanced data problem resulting in promising outcomes. Now, to solve this data imbalance issue, a variety of resampling methods have been proposed in the literature. Oversampling and undersampling methods are the most widely used methods for creating a balance between defective and non-defective classes. Experimentation by Pelayo and Dick [76] showed at least 23% improvement in G-Mean values for SDP when the oversampling technique SMT is used with a C4.5 decision tree classifier. Liu et al. [77] used a combination of oversampling and undersampling techniques for predicting software defects using a support vector machine. The performance of developed models was evaluated using F-Measure, G-Mean, and ROC curve. Kamei et al. [78] analyzed defect prediction power of four linear and regression models in combination with resampling methods: SMT, Random OverSampling (ROS), Random UnderSampling (RUS), and One Sided Selection (OSS) for two industrial sets. Resampling of data shows improvement in linear discriminant analysis and logistic regression analysis models using f-measure but no significant change was observed in neural network and classification tree models when the resampling method was incorporated with them. Khoshgoftaar and Gao [79] used the RUS method to handle imbalanced data problem and also used a wrapper based feature selection technique for attribute selection. They investigated four different scenarios of sampling techniques and feature selection combinations to evaluate which model has better predictive capability in terms of accuracy and ROC-AUC. Galar et al. [30] performed SDP for imbalanced data using bagging–and boosting–based ensemble techniques with C4.5 as the base classifier. The performance was evaluated using the ROC-AUC measure. Some other studies support the application of resampling methods for handling imbalanced data issue [80–83]. Shat-

wani [84] also performed an empirical comparison of defect prediction models built using oversampling techniques with three different classifiers on the eclipse dataset. Shatnawi [84] generated duplicate minority samples based on the number of defects and resulted in outstanding performance for Naïve Bayes, bayesian network, and k-nearest neighbor when ROC-AUC was considered for model comparison. Wang and Yao [85] investigated ML models built using five different resampling methods and their results support the use of these techniques for imbalanced data handling. Seiffert et al. [82] tried to investigate the effect of imbalanced data with seven resampling techniques and 11 machine learning algorithms. They also used ROC-AUC for comparing the predictive capabilities of models. RUS and Wilson's editing performed better for the most of learners in their study. They used a dataset written in ADA. Jindaluang et al.[86] proposed the undersampling technique with the k- centers clustering algorithm which proves to be effective in terms of sensitivity and F-measure, but they didn't use any stable metric for imbalanced data like G-Mean or ROC-AUC. Lingden et al. [87] have proposed a modified undersampling technique that combines CFS, RUS, and decision forest that gives promising results. Recently, Bejjanki et al. [88] used eight different classifiers namely AdaBoost, Decision Tree, Extra Tree, Gradient Boosting, KNN, Logistic Regression, Naïve Bayes, and Random Forest to build SDP models with novel class reduction methods where new samples are generated by calculating the centroids of the minority samples and evaluated based on accuracy, precision, sensitivity, F-measure and G-Mean. Malhotra and Kamal [89] inspected the impact of oversampling techniques on ML models. They performed the extensive analysis by exploiting 12 benchmark NASA datasets with six oversampling techniques on five classifiers and advocated the usage of sampling techniques for better SDP. They demonstrated the improvement in ML models with oversampling and proposed a new resampling method–SPIDER3. Though many studies have been conducted, still there is no particular set of resampling methods that can be considered the winner of all.

Cost-sensitive learning can be done either by assigning weights to minority samples or

by penalizing wrongly predicted defective classes in the cost matrix. Rodriguez [83] performed the empirical comparison of models constructed using C4.5 decision tree (J48) and Naïve Bayes with different resampling techniques, ensemble techniques (AdaBoostM1, Bagging, and Random Forest), and Metacost learner (with cost = 10). They compared performances based on MCC and ROC-AUC and found that ensemble-based models performed better than meta cost learners for NASA datasets. Siers and Islam [90] designed a reduced cost decision tree-based ensemble solution for SDP on six NASA datasets and used weighted precision and sensitivity metrics. Arar and Ayan [64] exploited neural networks to predict defects in five NASA datasets. They trained their neural network using False Negative Ratio (FNR) and False Positive Ratio (FPR) classification costs. Afterward, Siers and Islam suggested a balanced cost framework to handle class imbalance as well as cost sensitivity [91]. A balanced cost matrix was created using Standoff. Though the cost was reduced, model performances were not evaluated based on any metrics like precision, sensitivity, etc. Malhotra and Kamal [89] also explored the MetaCost learners with a cost ratio of 10, 30, and 50 and evaluated performance based on ROC-AUC, sensitivity, and precision.

Research studies are performed to explore the ensembles' performance in predicting software defects. A study by Galar et al. [30] performed an empirical comparison of RUS, ROS, Synthetic Minority Over-sampling Technique (SMT), modified SMT, and selective preprocessing of imbalanced data for bagging and boosting ensembles using C4.5 as the base classifier. The performance was evaluated using the ROC-AUC measure. They advocate the usage of ensemble classifiers for dealing with class imbalance problem. Peng et al. [92] investigated ten imbalanced NASA datasets for defect prediction using ROC-AUC and found that the C4.5 classifier with boosting provides comparably better results. Song et al. [93] provided a comprehensive investigation to study the nature of SDP models for a wide range of imbalanced datasets using Matthews's correlation coefficient for eight resampling based ensemble techniques over different classifiers. Similarly, Chen et al. [16] explored

nine highly imbalanced data by constructing a model using random undersampling and AdaBoost and compared performance with state of art models in terms of G-Mean and ROC-AUC. They also considered class overlap problems. Most of the researchers have exploited NASA datasets in this research direction.

1.5 Objectives of the Thesis

1.5.1 Vision

Improving software quality by tackling imbalanced data problem in open source software using efficient models.

1.5.2 Focus

The focus of the work is to empirically evaluate and improve the SDP models. SDP models are developed using OO metrics. In data analysis techniques, various ML techniques and SBTs are explored. This thesis strives to understand different facets of the SDP field by predictive modelling. The distinct work is done in direction of handling the skewness in data. The confidence in results is ascertained by statistical methods. Thus, the study explicitly investigates the following aspects to achieve the aforementioned vision:

1. To probe various OO metrics and establishing their association with SDP.
2. To envisage and ameliorate the efficacy of ML techniques for SDP by exercising different resampling methods.
3. To investigate the efficacy of ensemble methods for SDP by exercising different resampling methods.

4. To examine the applicability of SBTs for effective model development to predict defects.

5. To analyze the efficacy of SBTs with resampling methods for SDP.

6. To improve the SDP by using cost-sensitive classification.

7. To perform the comparative analysis of developed models with stable performance measures.

8. To conduct statistical validation of developed models.

9. To anatomize the conduct of evolutionary techniques for effective feature selection.

1.5.3 Research Questions and Goals

The sorry state of empirical studies in the SDP field motivates us to define the following research questions (RQ) and goals.

1.5.3.1 Research Questions

1. RQ1: What is the current state of software defect predictive modelling with imbalanced data?

2. RQ2: What is the comparative performance of various machine learning techniques used for detection of defects in OO software?

3. RQ3: What is the predictive performance of various machine learning-based models when resampling methods are used to handle skewness of data?

4. RQ4: What is the predictive performance of various machine learning-based models with cost-sensitive classification?

5. RQ5: What is the predictive performance of various ensembles and hybrid methods for effective defect classification?

6. RQ6: How effective are search-based techniques for model construction in the SDP domain?

7. RQ7: What is the predictive performance of various search-based models when re-sampling methods are used to handle imbalanced data problem?

8. RQ8: Do the SDP models with machine learning techniques perform better when relevant metrics are selected by an evolutionary technique instead of correlation-based filter method?

1.5.3.2 Goals

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

1. Performing Literature Review on Imbalanced Software Defect Prediction for Quality Improvement

    - Study of existing literature studies will help to understand the issues faced in predicting defects with the aim of software quality improvement in imbalanced data.
    - Survey will give the insight to which imbalance learning methods are generally addressed by researchers when they tend to envisage future defects.
    - With the gaining popularity of search-based strategies in SE, this survey will also focus on finding the extent of researchers' inclination towards search-based defect predictive modelling.
    - It will abridge the ML techniques and search-based techniques that are used for SDP in the imbalanced data domain.

- Extensive study of existing literature will guide software practitioners in devising out potential solutions to the problem faced in mining software defects for assessing the ongoing software projects.
- Exhaustive analysis of present studies will reveal the feature selection techniques, performance measures, statistical tests, and tools used in them.
- Survey will help to identify the problem areas and gaps in the existing literature.

2. Exploration of Machine Learning Techniques for Prediction of software defects

   - To empirically validate the association between OO software metrics and defect proneness.
   - To compare the performance of different ML techniques in predicting defects by experimental analysis.
   - To analyze the potential of compared ML techniques using appropriate statistical tests.
   - To engage stable performance measures like G-mean, Balance, and ROC-AUC owing to the imbalanced nature of the data.

3. Investigation of resampling methods with ML techniques to improve defect predictions.

   - To empirically assess the effect of oversampling and undersampling methods in building better defect prediction models using ML techniques.
   - To cater to the imbalanced data problem by balancing defective and non-defective classes at the data level.
   - To determine which data resampling method performs statistically better than others.

4. Examination of cost-sensitive classification and ascertain their effectiveness for developing SDP models.

    - To develop SDP models using MetaCost learners and detect future probable defects.
    - To alleviate the data imbalance problem by balancing defective and non-defective classes at the algorithm level.
    - To determine statistically whether the defect prediction models built based on cost sensitivity are better defect predictors.

5. Assessment of ensemble methods and hybrid methods for effective SDP.

    - To construct robust and reliable defect prediction models with ensemble methods.
    - To analyze the impact of ensemble methods with resampling methods to reduce the imbalanced data problem in the SDP field.
    - To statistically assess the capability of ensemble methods in tackling imbalanced data problem with help of statistical methods.

6. Exploration of search-based techniques for detection of defects in OO software.

    - To validate the competency of search-based techniques for uncovering unseen software defects.
    - To empirically evaluate the impact of search-based techniques in SDP model construction.
    - To determine which search-based technique performs statistically better than the others.

7. Investigation of resampling methods with search-based techniques for improved defect prediction.

    - To empirically gauge the usefulness of oversampling and undersampling methods in building the defect prediction models using search-based techniques.
    - To conduct an extensive empirical investigation and analyze whether developed models are efficacious in handling imbalanced data problem or not.
    - To determine which data resampling method performs statistically better than others in constructing SDP models with search-based techniques.

8. Exploration and assessment of evolutionary techniques for determining important and relevant features of the software.

    - To evaluate the fitness of evolutionary techniques for feature selection.
    - To develop SDP models based on features selected by variants of genetic algorithm and empirically compare them with correlation feature selection based models.
    - To determine statistically which of the feature selection techniques under consideration is responsible for constructing the best SDP models.

1.6 Organization of the Thesis

This section presents the organization of the thesis. **Chapter 1** presents the basic introduction of the work and the motivation of the thesis. **Chapter 2** introduces the current trends and research gaps by conducting a systematic review of existing studies in the field of SDP. In the presence of systematic literature reviews for SDP, we were motivated to explore SDP studies tackling the imbalanced data problem. **Chapter 3** expounds on the

research methodology adopted to achieve the goals. **Chapter 4** deals with the construction of SDP models using resampling methods for improved software quality. **Chapter 5** presents cost-sensitive classification to alleviate data imbalance problem. **Chapter 6** explores ensemble methods and hybrid methods in developing SDP models for imbalanced data. **Chapter 7** analyzes the applicability of search-based techniques for developing SDP models. **Chapter 8** extends the effectiveness of SBTs by involving resampling methods to tackle the imbalanced nature of underlying data. **Chapter 9** ascertains the importance of feature selection and analyzes the appositeness of evolutionary techniques for better defect prediction. **Chapter 10** states the conclusion of the thesis.

Figure 1.3 presents an overview of the work accomplished in this thesis.

Chapter 1: This chapter presents the basic concepts of software quality and software defect prediction. It describes the steps in predictive modelling and the SDP process. It provides objectives and an overview of the work performed in this thesis.

Chapter 2: This chapter includes a comprehensive systematic literature review of SDP studies conducted from 2000 to 2020 that address the data imbalance problem. RQs are formulated and studies are summarized to provide insights to datasets, feature selection methods, cross-validation methods, classification techniques, imbalance learning methods, performance measures, statistical tests, and tools used in these 48 studies. Further, research gaps are ascertained to provide future directions.

Chapter 3: This chapter entails the research methodology adopted in this thesis. It provides a brief description of datasets, independent variables, and feature selection methods used in this work. It also summarizes the cross-validation method, classification techniques, and imbalance learning methods employed to construct the SDP models in this thesis. Further performance measures used to evaluate the models and statistical tests applied to validate these models are also included.

Chapter 4: In this chapter SDP models are constructed using 15 ML techniques- Naïve Bayes (NB), Logistic Regression (LR), Simple Logistic (SL), LogitBoost (LB), multi-

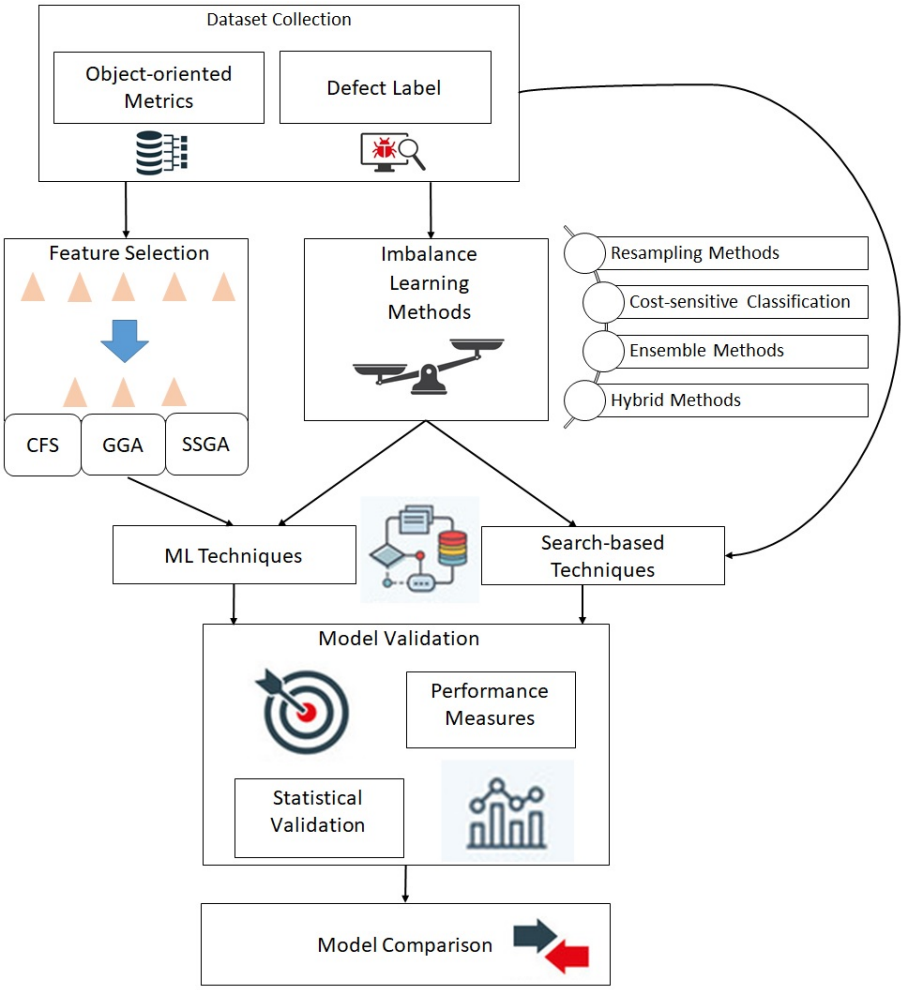

Figure 1.3: An Overview of the Thesis Work

layer perceptron (MLP), K-Nearest Neighbor1 (IBk), K-Nearest Neighbor2 (KStar), Adaptive Boosting AdaboostM1 (ABM1), Bagging (Bag), Iterative Classifier Optimizer (ICO), Logistic Model Tree (LMT), Random Tree (RT), RandomSubSpace (RSS), Partitioning (PART), and J48 on 12 datasets. As all these datasets were imbalanced, resampling methods comprising of oversampling and undersampling methods were used to tackle this problem. (i) Oversampling methods include Adaptive Synthetic Sampling (ADSYN), SMT, Safe Level Synthetic Minority Over-sampling Technique (SLSMT), Selective Preprocess-

ing of Imbalanced Data (SPD), ROS, and Agglomerative Hierarchical Clustering (AHC). (ii) In undersampling methods, RUS, Condensed Nearest Neighbor+Tomek's modification of Condensed nearest Neighbor (CNNTL), Neighborhood Cleaning Rule (NCL), and OSS were used to balance the data. Stable performance measures were used to develop and compare the models. This study also establishes the relationship of OO metrics with the defect proneness of the class.

Chapter 5: This chapter examines the effect of cost-sensitive classification in solving the data imbalance problem. Three datasets are explored with different MetaCost learners and models are developed using the J48 and ensemble methods (ABM1, Bag, RSS). Model comparison is performed using stable performance measures.

Chapter 6: Empirical validation of ensemble methods for constructing effective SDP models is executed in this chapter. The study is based on the datasets derived from 15 OO projects. Ensemble methods includes classic ensemble methods (AdaBoost (AB), AdaBoostNC (ABNC), ABM1, Bag), resampling based boosting ensemble methods (DataBoost(DB), MSMOTEBoost (MSMTB), RUSBoost (RUSB), SMOTEBoost (SMTB)), and resampling based bagging ensemble methods (MSMOTEBagging (MSBAG), OverBagging (OBAG), OverBagging2 (OBAG2), SMOTEBagging (SMTBAG), UnderBagging (UBAG), UnderBagging2 (UBAG2), UnderOverBagging (UOBAG), IIvotes (IIVOT)).

Chapter 7: This chapter investigates the applicability of SBTs for constructing SDP models. Very few studies exist in the literature exploiting the SBTs, therefore this study verifies the germaneness of SBT for SDP. This objective is achieved by using 13 datasets and 16 SBTs. SBTs include Bioinformatics-oriented hierarchical evolutionary learning (BIOHEL), Recombination and Cataclysmic Mutation Adaptative Search for Instance Selection (CHC), CO-Evolutionary Rule Extractor (CORE), Constricted Particle Swarm Optimization (CPSO), Genetic Algorithm based Classifier System with Adaptive Discretization Intervals (GA_ADI), Genetic Algorithm based Classifier System with Intervalar Rule (GA_INT), Generational Genetic Algorithm for Instance Selection (GGA), Incremental

Learning with Genetic Algorithms (ILGA), Linear Decreasing Weight - Particle Swarm Optimization (LDWPSO), Memetic Pittsburgh Learning Classifier System (MPLCS), Population-Based Incremental Learning (PBIL), Real Encoding - Particle Swarm Optimization (REPSO), Steady-State Genetic Algorithm for Instance Selection (SGA), Supervised Inductive Algorithm (SIA), sUpervised Classification System (UCS), and X-Classifier System (XCS). Stable performance measures and appropriate statistical tests are used for results verification.

Chapter 8: This chapter proposes the SDP framework to empirically evaluate the SBTs with resampling methods. This study engages ADSYN, SMT, SLSMT, SPD, ROS, CNNTL, RUS, NCL, and OSS for dealing with imbalanced data and uses eight SBTs namely GA_INT, MPLCS, UCS, BIOHEL, GA_ADI, XCS, CPSO, and LDWPSO. The chapter evaluates the obtained results using stable performance measures (G-Mean, Balance, and ROC-AUC) and non-parametric statistical tests.

Chapter 9: Further exploiting the SBTs, this chapter tests the applicability of SBTs-GGA and SGA for feature selection. We conducted an empirical comparison of defect prediction models developed using 13 datasets and 15 ML techniques when feature selection is done using GGA, SGA, and CFS. 15 ML techniques used are NB, LR, SL, LB, MLP, IBk, KStar, ABM1, Bag, ICO, LMT, RT, RSS, PART, and J48. Performance of models is assessed with help of G-Mean, Balance, and ROC-AUC. Statistical tests- Friedman test and Wilcoxon-signed rank test are employed to validate the results.

Chapter 10: This chapter summarizes the conclusion of the work performed and enlists some directions for future work.

Chapter 2

Systematic Review on Imbalanced Software Defect Prediction

2.1 Introduction

With the advent of software in day-to-day life, the reliability, and quality of the software are of utmost importance. Their ever-growing size multifold the problem of delivering effective and good quality software. The software cannot be completely defect-free and only small portions of software code are responsible for the majority of the defects. Finding and correcting these defects is time-consuming as well as a resource-draining process. SDP aids in uncovering these potential defects in the software during the early stages of its development. Defects discovered in the early stages consume fewer resources. By identifying the probable defective areas, manpower and cost budgets can be scheduled accordingly. Hence, with a powerful and correct software defect prediction model we can make good quality software with optimum resource utilization [1]. The capability of SDP models is largely affected by the nature of data. The results of good SDP models can be undependable if the data on which they are trained are imbalanced. Imbalanced defect data is the data

that have unequal distribution of defective and non-defective classes. A model trained with the majority of non-defective classes is likely to predict any class as non-defective only. This problem is referred to as the data imbalance problem. Most of the real-world data are imbalanced by their inherent nature and the problem is highly presiding in the SDP domain [94]. This necessitates the incorporation of effective solutions for data imbalance problem while constructing SDP models. The data imbalance problem has motivated researchers to propose solutions to alleviate it resulting in better and sound defect prediction models. As solving the data imbalance problem in the SDP area is an emerging field, therefore, it is vital to conduct a review of work done till now to comprehend the related accomplishments.

Catal [95] reviewed SDP studies from 1990 to 2009 and found only two studies that struggled with the data imbalance issue. The detailed insights are provided on class imbalance learning by Haixiang et al. [96] in 2017 but they performed the generic review revolving around imbalanced studies in a wide range of domains including chemical engineering, biomedical engineering, financial management, and information technology applications. They covered studies published in 2006-2016 and included only three papers related to imbalanced data problem in the SDP domain. The recent survey embarking on signs of progress in the SDP domain is conducted by Li et al. [60]. They included eight SDP studies proposing solutions to imbalanced data problem in the short period of January 2014- April 2017. The most recent survey on imbalanced data is conducted by Kaur et al. [97]. Though published in 2019, a meager number of SDP related studies are included in that survey. The authors identified 11 applications facing imbalanced data challenges but surprisingly, SDP was not able to take its place there.

This current state of the challenging imbalanced data problem in the SDP domain motivated us to conduct a review that systematically addresses this issue. The purpose of this review is to summarize the related work accomplished till now. Unlike previous reviews, this review is an attempt to provide a wider picture to the researchers regarding the proposed solutions to imbalanced data problem in SDP with the compilation of datasets,

metrics, classification techniques, validation methods, performance measures, and statistical tests addressed in them.

The chapter is organized as follows: Section 2.2 addresses the review process and research questions. Section 2.3 entails the review protocol and Section 2.4 scribes the primary studies identified in this systematic review. Further, Section 2.5 provides detailed answers to RQs. At last, Section 2.6 identifies the research gaps and brings forth future directions.

The results of the chapter are presented in [98].

2.2 Review Process

The conduct of review follows the directions provided by Kitchenham et al. [99]. The review is carried out in three phases:

1. Plan the review: This includes identification of the need for the review, designing of the RQs, and establishing the review protocol. Review protocol aims to carefully design the search string, inclusion-exclusion criteria, and quality question questionnaire.

2. Conduct the review: In this phase, a search string is formed to get the related studies. Inclusion-exclusion criteria and quality assessment questionnaire are used to filter out irrelevant and ineffective studies. Afterward, data is extracted and synthesized from the relevant studies to answer the RQs.

3. Report the review: The results of the review are portrayed with help of line charts, tables, and bar graphs.

The need for review, as mentioned in the Introduction section, is to abridge the imbalanced learning methods provided for SDP till now. The current literature lacks the review addressing this issue, particularly in the SDP application area. This review will project

the future directions in this area and acquaint developers, software practitioners, and researchers with the recent trends followed in the field. The following research questions are designed to accomplish the required objectives:

- RQ1: Which imbalanced datasets are used in SDP studies?
- RQ2: What are the different feature selection methods used in imbalanced SDP studies?
- RQ3: Which cross-validation techniques are involved in building SDP models for imbalanced data?
- RQ4: Which classification techniques are used in imbalanced SDP studies?
- RQ5: What are the different solutions proposed to solve the data imbalance problem in SDP?
- RQ6: Which performance measures are used in the analysis of SDP models generated with imbalanced data?
- RQ7: Which statistical tests are used for result validation in imbalanced SDP studies?
- RQ8: Which tools are used by researchers to assist in building effective imbalanced SDP models?

The answers to these RQs will certainly assist developers to employ the appropriate metrics, cross-validation, imbalance learning solutions, classification techniques, performance evaluators, and statistical tests for SDP model construction while dealing with imbalanced data.

2.3 Review Protocol

Review protocol sets the rules for the selection of primary studies for the survey. The studies are searched in the period from January 2000 to September 2020 and carefully chosen using a three-level selection procedure.

2.3.1 Three-level selection procedure

1. Level 1 Selection: It requires constituting the search string for the problem. Initially, an informal search was made using strings like software defect OR bug OR fault prediction in imbalanced data, class imbalance problem in software defect OR bug OR fault prediction. The studies emerged as the outcome of these strings helped to design the formal search string. All the synonyms for the imbalanced, software, defect, prediction, classification techniques are considered. Next, the following string is formed with the help of Boolean AND and Boolean OR connectors.

   (“undersampling” OR “oversampling” OR “resampling” OR “sampling” OR “cost sensitive” OR “cost-sensitive” OR “hybrid” OR “ensemble” OR “hybrid” OR “balancing” OR “imbalance”) AND (“software” OR “software quality” OR “open source” OR “datasets” OR “system” OR “software quality”) AND (“defect” OR “fault” OR “bug” OR “defective”) AND (“prediction” OR “proneness” OR “classification” OR “impact” OR “classifier” OR “empirical” OR “learning”) AND (“machine learning” OR “learning” OR “statistical” OR “search based” OR “search-based” OR “evolutionary”)

   The search was performed on distinguished digital libraries like IEEE Explore, ACM, Wiley, Springer, Science Direct, and SCOPUS.

2. Level 2 Selection: At level 2 of study selection, we need to formulate the inclusion-

exclusion criteria for the selected studies in level 1.

Inclusion Criteria: The empirical studies on software defect prediction dealing with class or data imbalance issue in machine learning are included. The studies that compare two or more classification techniques are included. We included studies that compare two or more solutions for the data imbalance problem in SDP. All the empirical studies that provide at least a way of handling data imbalance problem using machine learners, statistical classifiers, search-based techniques, or evolutionary techniques are included as well. We manually screened the selected papers by first examining the abstract and conclusion. If required, the entrails of the studies were also inspected.

Exclusion Criteria: We excluded the review studies done in the area. Studies that do not hold any empirical evidence or are based on any learning paradigm other than supervised learning like [100] are excluded. Cross-project studies were also discarded. We excluded change-based studies like [74, 75] that were selected at Level 1 as we need to address defects not change in the software. We excluded studies that do not consider software engineering or NASA datasets like [30, 101]. Based on these inclusion and exclusion criteria, studies were selected prudently.

3. Level 3 Selection: Now, we need to compose the quality questions so that the papers selected at Level 2 can be judged against their importance. Quality check of Level-2 papers is completed by filling its questionnaire. Three possible values for a particular question is ‘Yes’, ‘No’, ‘PARTLY’ and their corresponding values are ‘1’, ‘0’, and ‘0.5’ respectively. The studies with a quality score of less than 50% were dropped. The decision regarding the quality of the paper is taken by both authors. At any point of disagreement, the final value is assigned only after qualitative and quantitative discussion among them.

2.3.2 Designing Quality Questions

The quality questions are designed to select relevant studies required to seek answers to RQs. Therefore, RQs guide to making quality questions. The higher the quality score of the paper, the higher will be the relevance of the paper concerning the survey. We identified 11 quality questions to evaluate the goodness of the paper. These quality assessment criteria are listed in Table 2.1. Both the authors addressed these questions separately. The sum of their scores for each study was averaged and assigned as its quality score. The studies whose mean quality score was less than 5.5 were rejected.

Table 2.1: Quality Assessment Questions

S.No.	Question	Yes	Partly	NO
1	Are the research questions clearly determined?			
2	Are the dependent and independent variables properly specified?			
3	Does the study states the imbalance ratio in datasets properly?			
4	Is the related work clearly stated in the paper?			
5	Does the study states experimental settings clearly?			
6	Does the study performs the comparative analysis amongst classification techniques?			
7	Are the solutions for data imbalance problem described properly?			
8	Has the study used an appropriate statistical test?			
9	If more than one solution is carried out for the data imbalance problem, does the study comparatively analyzed their performance?			
10	Are the performance measures that are used to compare models for the imbalanced data stated clearly?			
11	If feature selection is done, does the study report selected features properly?			

2.3.3 Data Extraction and Synthesis

Once the primary studies are finalized, it is important to organize the information retrieved from them in a structured manner. For this purpose, we designed data extraction forms. Data extraction forms hold related information in several fields including authors, the title

of the paper, year of publication, name of journal/conference, datasets, classification technique(s), data imbalance problem solutions, performance measures, statistical tests, and feature selection technique(s). These data extraction forms are filled disjointedly by the authors for all the primary studies. Next, data collected in forms is required to be synthesized. The analysis of forms is performed individually by authors and the answers to RQs are framed collectively with discussion. At any point of disagreement, the concerned RQ is responded only after qualitative and quantitative discussion among them.

2.4 Primary Studies of the Review

Following the rigorous process of implementing review protocol by [99], we determined and shortlisted the most relevant studies for our survey. After executing this step, we recognized 48 literature studies as primary studies. Table 2.2 enlists the primary studies with their unique identifier. The studies are associated with an imbalance problem in SDP. Therefore they are prefixed with "IS" denoting "Imbalanced Study" before its numeric value.

Table 2.2: Primary Studies scrutinized for the survey

PS Identifier	Name	Reference	PS Identifier	Name	Reference
IS1	kaminsky2004	[102]	IS25	xiang2017	[63]
IS2	pelayo2007	[76]	IS26	shatnawi2017	[103]
IS3	riquelme2008	[80]	IS27	akour2017	[104]
IS4	khoshgoftaar2009	[79]	IS28	yang2017	[105]
IS5	pelayo2012	[81]	IS29	yohannese2017	[106]
IS6	shatnawi2012	[84]	IS30	ibrahim2017	[107]
IS7	gao2012	[108]	IS31	bennin2017	[109]
IS8	wang2013	[85]	IS32	song2018	[93]
IS9	wahono2013	[110]	IS33	chen2018	[16]
IS10	liu2014	[111]	IS34	kalsoom2018	[112]
IS11	rodriguez2014	[83]	IS35	huda2018	[113]
IS12	seiffert2014	[82]	IS36	mousavi2018	[114]
IS13	li2014	[115]	IS37	miholca2018	[116]
IS14	wahono2014a	[117]	IS38	suntoro2018	[118]

PS Identifier	Name	Reference	PS Identifier	Name	Reference
IS15	wahono2014b	[119]	IS39	lingden2019	[87]
IS16	laradji2015	[50]	IS40	khuat2019	[120]
IS17	hussain2015	[121]	IS41	balogun2019b	[122]
IS18	siers2015	[90]	IS42	alsaeedi2019	[123]
IS19	arar2015	[64]	IS43	cai2020	[67]
IS20	jing2016	[124]	IS44	pan2019	[125]
IS21	wu2016	[126]	IS45	maruf2019	[127]
IS22	bennin2016	[128]	IS46	bejjanki2020	[88]
IS23	li2016	[129]	IS47	bal2020	[130]
IS24	siers2016	[91]	IS48	pandey2020	[131]

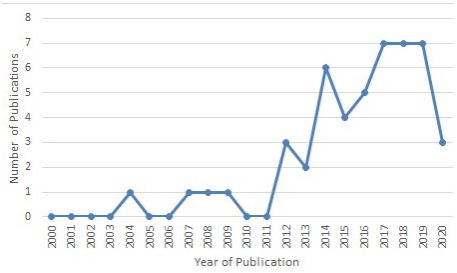

Figure 2.1: Year-wise Publication Trends

The year-wise publication of primary studies is represented in Figure 2.1. Though work geared up from 2014,the striking interest of the researchers for handling data imbalance is conspicuous since 2017.

It is worth noting from Figure 2.2 that 26.5% of studies were published in reputed conferences whereas 73.5% of primary studies were issued in the journals.

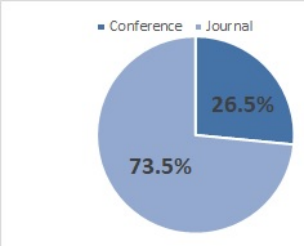

Figure 2.2: Venue-wise Primary Studies distribution

Table 2.3 accords the top five venues of publication of primary studies. None of the conferences was able to acquire any of the top five positions.

Table 2.3: Top Journal/Conference Venues of Publication

Identifier	Name	%age of IS	IS
J1	IEEE Transactions on Reliability	8.2%	IS10, IS47, IS5, IS8
J2	Information and Software Technology	4.1%	IS16, IS28
J3	IEEE Transactions on Software Engineering	4.1%	IS20, IS32
J4	Advanced Science Letters	4.1%	IS14, IS15
J5	Information Sciences	4.1%	IS12, IS37

2.5 Review Results and Discussion

2.5.1 Results specific to RQ1

From 2000 to 2020, researchers have exploited a wide range of datasets for model validation while tackling the data imbalance issue. Datasets used in the selected primary studies are of varied size with different imbalance ratios. The primary studies have used either open-source or proprietary software. Approximate 94% of studies have used open-source software. Proprietary software are used only by three studies [79, 82, 108].

The categorization of datasets used in the primary studies is represented in Table 2.4.

Table 2.4: Datasets Used in the Primary Studies

Datasets	PS Identifier	Count	%age
NASA	IS1, IS2, IS3, IS4, IS5, IS8, IS9, IS10, IS11, IS13, IS14, IS15, IS16, IS18, IS19, IS20, IS21, IS23, IS24, IS25, IS27, IS29, IS30, IS33, IS34, IS35, IS36, IS38, IS39, IS41, IS42, IS43, IS48	33	68.8%
Apache	IS16, IS17, IS20, IS26, IS31, IS32, IS34, IS37, IS39, IS40, IS41, IS44, IS45, IS46, IS47	15	31.25%
Eclipse	IS6, IS23, IS28, IS32, IS39	5	10.42%
Mozilla	IS5, IS28, IS39	3	6.25%
SOFTLAB	IS20, IS37	2	4.17%
Other OSS	IS4, IS7, IS12, IS17, IS20, IS22, IS28, IS31, IS39, IS45	7	14.6%
Proprietary	IS4, IS7, IS12	3	6.3%

Open-source software: These software are publicly available and have open access to all. Many software repositories provide metric details at the class level or method level for these software. In the primary studies, the majority of the datasets were downloaded from the PROMISE repository. The open-source software can be subcategorized as:

- NASA datasets: The most popular datasets are extracted from the NASA MDP program and are available on PROMISE repositories. 68.8% of primary studies have exploited NASA datasets to build SDP models. 32 out of 48 studies used NASA datasets. Different datasets used from NASA are CM1, JM1, AR1, AR3, AR4, AR5, AR6, KC1, KC2, KC3, MW1, MC2, PC1, PC2, PC3, PC4, and PC5.

- Eclipse: Eclipse is the toolset for software management. Metric details of Eclipse software can be accessed by the AEEEM repository. The Eclipse software used in the studies are JDTCore, platform, PDE UI, and IDE. Five primary studies making 10.42% of the total studies used these datasets.

- SOFTLAB: Datasets from SOFTLAB are available on the PROMISE repository. These datasets are extracted from the embedded goods manufacturing system. Five datasets used from SOFTLAB in primary studies are AR1, AR3, AR4, AR5, AR6. IS37 used all these five datasets whereas IS20 used three datasets- AR3, AR4, AR5.

- Apache: Apache software used in the primary studies are HTTP Server, Ant, Camel, Ivy, Synapse, Lucene, Mylyn, Poi, Velocity, Xerces. HTTP server data is available on ReLink, Lucene and Mylyn are accessible at AEEEM; while the rest of Apache software are available on the PROMISE repository.

- Mozilla: Mozilla-delta is used in IS5. Mozilla is also used in IS28 and IS39.

- Other OSS: There are some other open-source software used in 14.6% of studies. Examples of such datasets are- learning, Forrest, Zuzel, Berek, Pbean, OpenIntents

Safe ZXing, equinox, Clam Antivirus, eCos, Helma, NetBSD, OpenBSD, OpenCms, OpenNMS, Scilab, Spring, Kalkulator, Nieruchomosci, Prop, Bugzilla, Columba, and PostgreSQL software.

Proprietary software: Proprietary software are industry-specific. This category of software cannot be used without the consent of the company or organization that owns it. Only 6.3% of primary studies used proprietary software. Four datasets corresponding to large legacy telecommunications software system (LLTS) are used in IS4 and IS7. Another software that is used by IS12 is CCCS and it is the military control and communication system.

Figure 2.3 shows the graphical representation of the imbalanced datasets used with respect to the percentage of the studies.

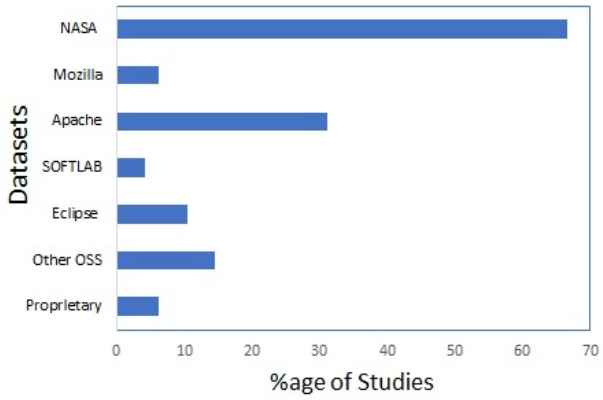

Figure 2.3: Imbalanced Datasets used for constructing SDP models

Table 2.5 depicts the languages in which the discussed software are written. The majority of the studies exploited project metrics for the JAVA and C software. 81.25% of the primary studies used JAVA projects and 72.92% of studies used C projects. C++ projects are used in 27 studies out of total studies.

Table 2.5: Languages of the Software used in the Primary Studies

Language	PS Identifier	Count	%
C	IS1, IS2, IS3, IS4, IS5, IS8, IS9, IS10, IS11, IS13, IS14, IS15, IS16, IS18, IS19, IS20, IS21, IS22, IS23, IS24, IS25, IS27, IS29, IS30, IS33, IS34, IS35, IS36, IS37, IS38, IS39, IS41, IS42, IS43, IS48	35	72.92%
Protel	IS4, IS7	2	4.17%
C++	IS2, IS3, IS5, IS8, IS9, IS11, IS13, IS14, IS15, IS16, IS18, IS19, IS20, IS22, IS23, IS27, IS29, IS33, IS34, IS35, IS36, IS38, IS39, IS41, IS42, IS43, IS48	27	56.25%
JAVA	IS2, IS3, IS6, IS8, IS9, IS10, IS11, IS13, IS14, IS15, IS16, IS17, IS18, IS19, IS20, IS22, IS23, IS24, IS26, IS28, IS29, IS31, IS32, IS33, IS34, IS35, IS36, IS37, IS38, IS39, IS40, IS41, IS42, IS43, IS44, IS45, IS46, IS47, IS48	39	81.25%
ADA	IS12	1	2.08%

Figure 2.4 illustrates that there is a single study that used a project in Ada language and only two studies used a project in Protel language. This is due to the fact that these projects belong to the proprietary category and are closed source software.

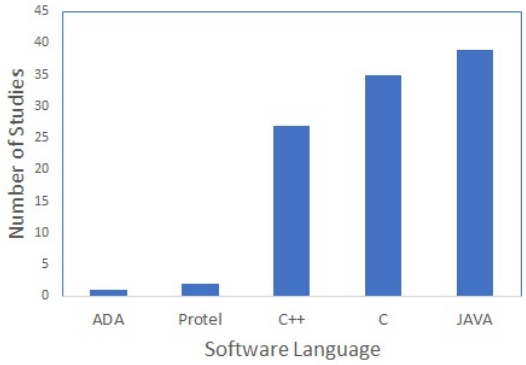

Figure 2.4: Languages of software used in Primary Studies

2.5.2 Results specific to RQ2

With the plethora of metrics, it becomes difficult to develop effective SDP models. The more the number of metrics, the more will be the training time required by the model. Additionally, metrics can be redundant or irrelevant. Redundant metrics may increase bias. Therefore, it becomes important to apply some feature reduction techniques to scrutinize important features. This RQ addresses whether the primary studies for imbalanced SDP employed feature reduction techniques or not. If applied, then which feature reduction techniques are used? As depicted in Figure 2.5, only 40% of studies have used the reduced set of features.

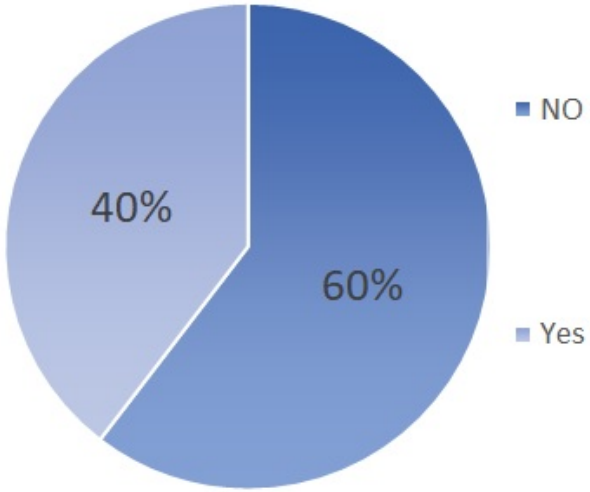

Figure 2.5: Feature Reduction

Table 2.6 summarizes the primary studies in which feature selection is done or not done. A total of 19 studies used feature reduction techniques while the remaining 29 studies did not perform this step during model development.

Table 2.6: Feature Reduction in Primary Studies

Feature Reduction	PS Identifier	Count	%
Done	IS4, IS7, IS9, IS10, IS14, IS16, IS17, IS19, IS20, IS25, IS26, IS28, IS29, IS30, IS34, IS36, IS37, IS38, IS39	19	39.60%
Not Done	IS1, IS2, IS3, IS5, IS6, IS8, IS11, IS12, IS13, IS15, IS18, IS21, IS22, IS23, IS24, IS27, IS31, IS32, IS33, IS35, IS40, IS41, IS42, IS43, IS44, IS45, IS46, IS47, IS48	29	60.40%

Next, we explored the different feature reduction techniques involved in the primary studies. Broadly, the feature reduction or dimensionality reduction techniques can be divided into feature selection and feature extraction.

1. Feature Selection: Feature selection methods tend to select the features that add the most information to classify the defects in the model. To accomplish this task, features election techniques can be further divided into two subcategories:

    - Filter Methods: Features are selected irrespective of the model using scoring methods and are robust to overfitting. Ten different filter methods are used in the considered studies for imbalanced SDP. CFS and Information Gain are the most popular among them. 14 studies used the filter methods and details are provided in Table 2.7.
    - Wrapper Methods: Different subsets of features are tested against the classification technique and features that generate the best model are selected. In this category, selected studies have used either machine learning or evolutionary techniques. Three evolutionary techniques: Genetic Algorithm, Particle Swarm Optimization (PSO), and Bat algorithm are used in primary studies to select important features. IS14 and IS25 used a genetic algorithm to generate the subset of features. Details are in Table 2.7.

2. Feature Extraction: Instead of selecting features, original features are transformed into new features. These derived features led to more accurate prediction models with less training time and reduced overfitting risk. In the primary studies, only two papers exploited feature extraction techniques. IS19 used Principal Component Analysis which is unsupervised feature extraction. IS34 used supervised feature extraction with Fisher's Linear Discriminant Analysis.

3. Others: IS10 selected relevant features by cost-sensitive approach. IS26 exploited AUC values to generate the useful subset of features and proposed the AUCThresholding method for feature selection. IS17 simply selected CK metrics of the projects and build SDP models based on them. IS28, unlike other studies, chose change metrics as features for model development.

Table 2.7: Distribution of Feature Reduction Techniques in the Primary Studies

Feature Reduction	Method	Technique
Feature Selection	Filter	Chi-square (IS7, IS26, IS37), CFS (IS16, IS19, IS36, IS39), Information Gain (IS7, IS19, IS26, IS29), Average Weight Information Gain (IS38), Fisher's Criterion (IS16), Forward Feature Selection (IS26), Gain Ratio (IS7), Greedy Forward Selection (IS16), ReliefF (IS7), Symmetrical Uncertainty (IS7)
	Wrapper	Support Vector Machine (IS4), Particle Swarm Optimization (IS9), Genetic Algorithm (IS14, IS25), Bat Algorithm (IS30), Forward Feature Selection (IS25), Backward Feature Elimination (IS25)
Feature Extraction	Supervised	Fisher Linear Discriminant Analysis (IS34)
	Unsupervised	Principal Component Analysis (IS19, IS20)
Others		Cost-sensitive (IS10), CK metrics (IS17), AUCThresholding (IS26), Change metrics (IS28)

2.5.3 Results specific to RQ3

It is important to incorporate a cross-validation method to evaluate how well the machine learning or other classification technique generalizes the unseen data. Cross-validation reduces the bias and variance of the model. 22.9% of studies did not report the validation method used. Figure 2.6 depicts the validation methods used with the number of studies. The cross-validation methods employed in the primary studies are represented in Table 2.8 and can be categorized as below:

- k-fold cross-validation: In k-fold cross-validation, the dataset is divided into k partitions. One partition acts as testing data and the remaining partitions act as training data. This process is repeated k times and finally, performance values are averaged. 87% of primary studies have used k- fold cross-validation. researchers have used its three versions: ten-fold, five-fold and 20-fold. In these versions also, few researchers have used either stratified or repeated forms of validation. Stratified ten-fold cross-validation is done by IS2, IS9, IS14, IS15, IS16, and IS24. Repeated ten-fold cross-validation is employed in IS12, IS23, IS26, IS32, and IS44.

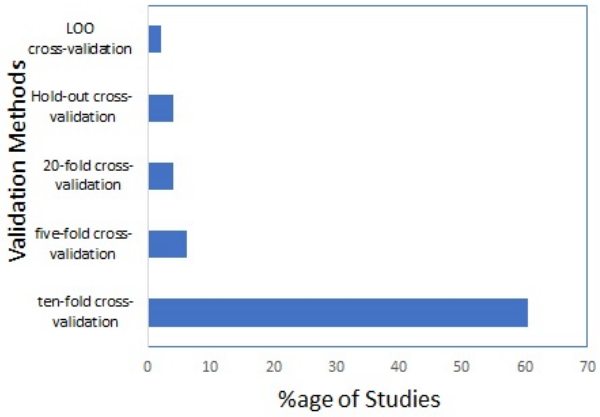

Figure 2.6: Validation methods used in Primary Studies

Table 2.8: Cross-validation Methods used in the Primary Studies

Cross-validation	PS Identifier	Count	%
ten-fold cross-validation	IS2, IS3, IS5, IS6, IS8, IS9, IS10, IS12, IS14, IS15, IS16, IS17, IS19, IS22, IS23, IS24, IS26, IS27, IS28, IS29, IS30, IS31, IS32, IS41, IS42, IS44, IS45, IS46, IS48	29	60.42%
five-fold cross-validation	IS4, IS11, IS35	3	6.25%
20-fold cross-validation	IS21, IS33	2	4.17%
Hold-out cross-validation	IS36, IS38	2	4.17%
Leave-one-out cross-validation	IS37	1	2.08%

- Hold-out cross-validation: In hold-out cross-validation, data is partitioned into two subsets. The larger subset is used for training the model and the smaller subset is used as a training subset. IS36 employed an 80:20 ratio and IS38 performed stratified hold-out validation with a 70:30 ratio.

- Leave-one-out (LOO) cross-validation: It works similar to k-fold cross-validation with k=1. Only one of the primary studies (IS37) has used this validation method.

As depicted by Table 2.8, the most stable cross-validation method is ten-fold cross-validation. Dividing data into 10 parts results in unbiased prediction, unlike hold-out cross-validation. LOO cross-validation is also not preferred by the software community because it is very time consuming for mid-sized or large projects.

2.5.4 Results specific to RQ4

Various classification techniques have been proposed in the literature for accurate predictions in SDP considering the imbalanced nature of datasets. These techniques exploit static features of the datasets and establish their relationship with the dependent variable defect.

The earliest study that we found in this area was conducted by Kaminsky and Boetticher [102]. They used the data equalization term for data balancing. The original KC2 dataset contained 379 samples (after data preprocessing) whereas the equalized dataset had 3013 samples and Genetic Programming (GP) was used to test their performance.

Ma et al. [132] suggested using G-Mean and F-measure for evaluating SDP models based on imbalanced data. They also experimentally shown that the defects in the large imbalanced datasets can be correctly detected by balanced random forests (RF).

Based on the primary studies, classification techniques are divided into following categories:

- Statistical Techniques
- Instance based learning
- Bayesian Learning
- Tree based Learning
- Ranking based Learning
- Support Vector Machines (SVM)
- Neural Networks (NN)
- Rule based Learning
- Extreme Learning
- Evolutionary Learning
- Hybrid Learning

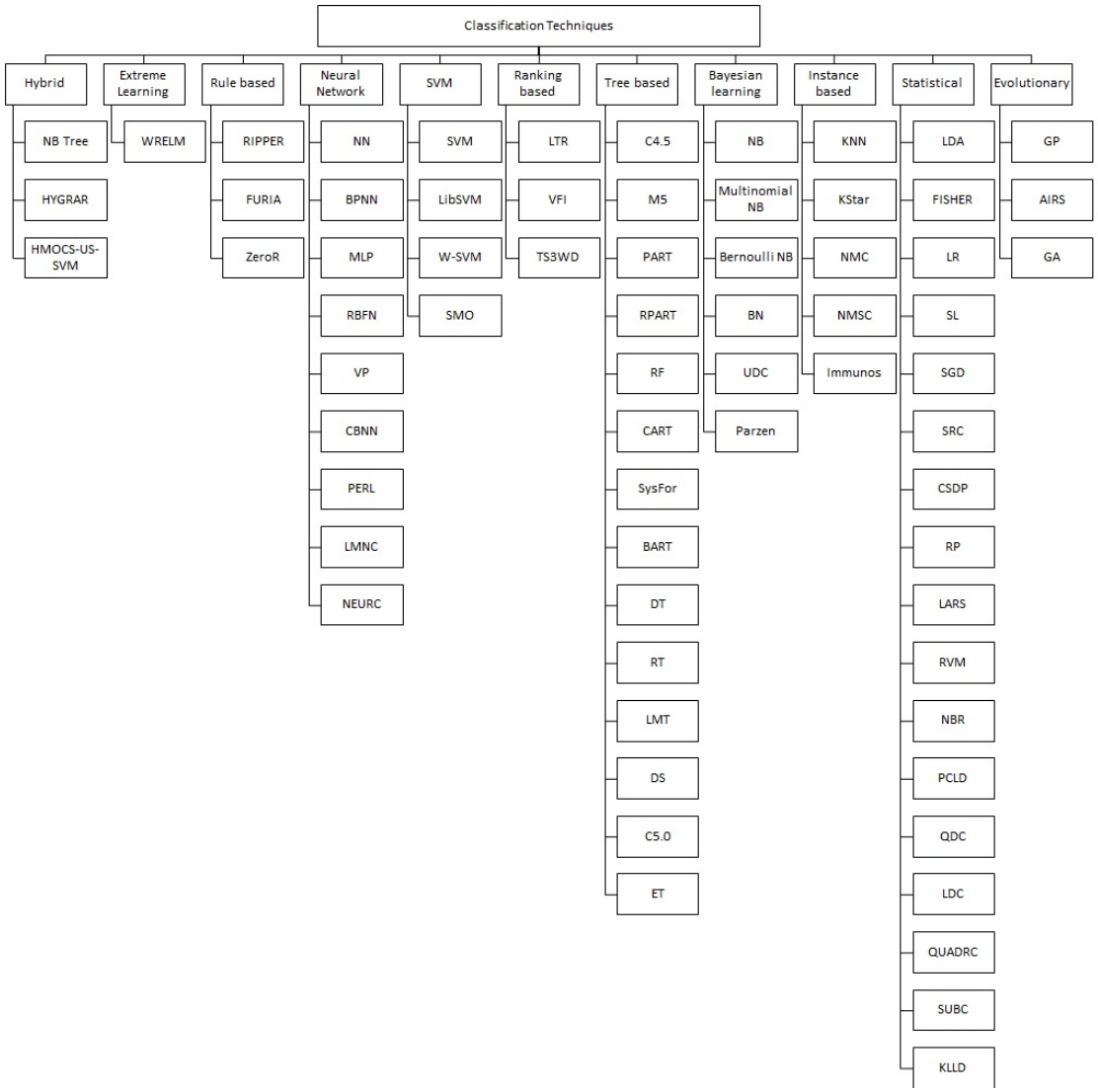

Figure 2.7: Taxanomy of Classification Techniques used in Primary Studies

This taxonomy of classification techniques with their subcategories (techniques used in a particular category) is hierarchically presented in Figure 2.7.

Table 2.9 mentions the techniques used in the primary studies within the identified categories of SDP.

A lot of techniques are applied in the literature. In statistical techniques, LR is used maximum in 14 studies. Other statistical techniques used for assessing the predictive

capability are Linear Discriminant Analysis (LDA), FISHER, LR, SL, sparse representation classifier (SRC), Stochastic Gradient Descent (SGD), collaborative representation classifier based software defect prediction (CSDP), Least Angle Regression (LARS), Relevance Vector Machine (RVM), Negative Binomial Regression (NBR), Linear classifier based on principle component analysis (PCLD), Quadratic discriminant classifiers (QDC), Linear discriminant classifiers (LDC), Quadratic classifier with regularization parameter (QUADRC), Subspace classifier (SUBC), and Karhunen Loeve Decomposition (KLLD).

In instance-based learning, researchers have majorly built models using K-Nearest Neighbor (KNN), KStar, Nearest Mean Classifier (NMC), Scaled Nearest Mean Classifier (NMSC), and Immunos.

Table 2.9: Distribution of Classification Techniques used in Primary Studies for Imbalanced SDP

Classification Category	Technique used
Statistical	LDA (IS9, IS14, IS28), FISHER (IS36), LR (IS9, IS12, IS14, IS16, IS20, IS22, IS23, IS25, IS26, IS32, IS41, IS42, IS44, IS46), SL(IS13, IS17, IS19, IS36), SGD (IS16), SRC (IS21), CSDP (IS21), LARS (IS22), RVM(IS22), NBR (IS23), PCLD (IS36), QDC (IS36), LDC (IS36),QUADRC (IS36), SUBC (IS36), KLLD(IS36)
Instance based	1NN (IS6, IS12, IS27, IS36), 5NN (IS6, IS7, IS12, IS26, IS32, IS34, IS36), 10NN (IS36), KNN(IS9, IS13, IS14, IS22, IS31, IS40, IS41, IS46, IS48), KStar (IS9, IS14, IS22, IS30), NMC (IS36), NMSC (IS36), Immunos (IS19)
Bayesian Learning	NB (IS3, IS6, IS9, IS11, IS12, IS13, IS14, IS17, IS19, IS21, IS25, IS26, IS27, IS28, IS30, IS32, IS34, IS36, IS38, IS41, IS43, IS44, IS46, IS48), Multinomial NB (IS16), Bernoulli NB (IS16), BN (IS6, IS27, IS40, IS41), UDC (IS36), Parzen (IS36)
Tree-based	C4.5 (IS2, IS3, IS5, IS9, IS11, IS12, IS13, IS14, IS17, IS18, IS19, IS24, IS25, IS26, IS27, IS28, IS29, IS31, IS32, IS36, IS40, IS41, IS44, IS46, IS48), M5 (IS22), RPART (IS22, IS23), PART(IS27, IS48), RF (IS8, IS9, IS12, IS14, IS16, IS19, IS20, IS21, IS23, IS27, IS30, IS31, IS32, IS35, IS39, IS41, IS42, IS43, IS44, IS45, IS46, IS48), CART (IS9, IS14), SysFor (IS18, IS24), DT (IS27), RT(IS27, IS41), LMT (IS41), DS (IS42, IS48), C5.0 (IS45), ET (IS46)
Ranking-based	LTR (IS23), VFI (IS27), TS3WD (IS23)
Evolutionary	GP (IS1), AIRS (IS19), GA (IS36)
SVM	SVM (IS4, IS7, IS9, IS12, IS13, IS14, IS16, IS18, IS20, IS21, IS28, IS31, IS32, IS34, IS36, IS40, IS42, IS48), LibSVM (IS9), W-SVM (IS16), SMO (IS17, IS27)

Classification Category	Technique used
Neural Networks	NN (IS6, IS15, IS19, IS20, IS22, IS28, IS31, IS34), BPNN (IS9, IS10, IS14, IS36), MLP (IS12, IS30, IS37, IS40, IS45, IS48), RBFN (IS12, IS36, IS37, IS44), VP (IS17), PERL(IS36), LMNC (IS36), NEURC(IS36)
Rule-based	RIPPER (IS12, IS32, IS45, IS48), FURIA (IS30), ZeroR (IS34)
Extreme Learning	WRELM (IS47)
Hybrid	NBTree (IS27), BART (IS23), HYGRAR(GRAR+NN) (IS37), HMOCS-US-SVM (IS43)

In Table 2.9 1NN represents KNN with K=1, 5NN denotes KNN with K=5, 10NN signifies KNN with K=10. KNN in Table 2.9 denotes the studies that used the KNN technique but did not report the value of K.

From Bayesian learning, studies have used NB, Multinomial NB, Bernoulli NB, Bayesian Network (BN), Uncorrelated normal densities based quadratic Bayes classifier (UDC), and Parzen Window Classifier (Parzen).

Tree-based classifiers used in primary studies include C4.5 or J48, M5, PART, Recursive Partitioning (RPART), RF, Classification and Regression Tree (CART), Decision Table (DT), RT, LMT, Decision Stump (DS), C5.0, and Extra Tree (ET).

Ranking-based techniques involved learning to rank (LTR), Voting Features Interval (VFI), and three-way decisions based two-stage ranking method (TS3WD).

SVM is also one of the most frequently used machine learning techniques in research studies. Its variants include LibSVM, W-SVM, and Sequential Minimization Optimization (SMO).

Researchers have applied NN for defect prediction. Its variants cover Back Propagation Neural Network (BPNN), MLP, Radial basis Function Network (RBFN), Voted Perceptron (VP), Linear perceptron classifier with batch processing (PERL), Levenberg-Marquardt feed-forward neural network (LMNC), Automatic Levenberg-Marquardt feed-forward neural network classifier (NEURC).

Six studies have also investigated rule-based classifiers like RIPPER, Fuzzy Unordered

Rule Induction Algorithm (FURIA), and Zero Rule (ZeroR). In these classifiers, RIPPER is used in four primary studies.

Recently, Bal and Kumar [130] have tried extreme learning for predicting defects in imbalanced datasets. They used a weighted regularization extreme learning machine (WRELM) to achieve the purpose.

Similarly, evolutionary techniques are also emerging solutions for effective defect prediction. We found three studies that used GP, Artificial Immune Recognition System (AIRS), and Genetic Algorithm (GA) for the early detection of defects.

Hybrid techniques are a combination of two or more techniques. Primary studies have used Bayesian Additive Regression Trees (BART), NBTree, HYGRAR, and HMOCS-US-SVM.

Table 2.10 summarizes the top 5 techniques that are used in literature. C4.5 is the most widely used classification technique. It is used in 52% of total studies. NB, RF, SVM, and KNN are also amongst the preferred techniques in the literature.

Table 2.10: Top five Classification Techniques used in the Primary Studies

Classification Techniques	PS Identifier	Count	%
C4.5	(IS2, IS3, IS5, IS9, IS11, IS12, IS13, IS14, IS17, IS18, IS19, IS24, IS25, IS26, IS27, IS28, IS29, IS31, IS32, IS36, IS40, IS41, IS44, IS46, IS48	25	52.08%
NB	IS3, IS6, IS9, IS11, IS12, IS13, IS14, IS17, IS19, IS21, IS25, IS26, IS27, IS28, IS30, IS32, IS34, IS36, IS38, IS41, IS43, IS44, IS46, IS48	24	50.00%
RF	IS8, IS9, IS12, IS14, IS16, IS19, IS20, IS21, IS23, IS27, IS30, IS31, IS32, IS35, IS39, IS41, IS42, IS43, IS44, IS45, IS46, IS48	22	45.8%
SVM	IS4, IS7, IS9, IS12, IS13, IS14, IS16, IS18, IS20, IS21, IS28, IS31, IS32, IS34, IS36, IS40, IS42, IS48	18	37.5%
KNN	IS6, IS7, IS9, IS12, IS13, IS14, IS22, IS26, IS27, IS31, IS32, IS34, IS36, IS40, IS41, IS46, IS48	17	35.4%

2.5.5 Results specific to RQ5

With the uneven distribution of defective and non-defective classes in the projects, the model prediction results may not be accurate. Therefore, it is recommended to incorporate some ways to tackle this issue. This RQ reflects the ways that are proposed to handle the imbalanced data problem in the SDP field. Figure 2.8 illustrates the usage of different imbalance learning techniques in the SDP literature.

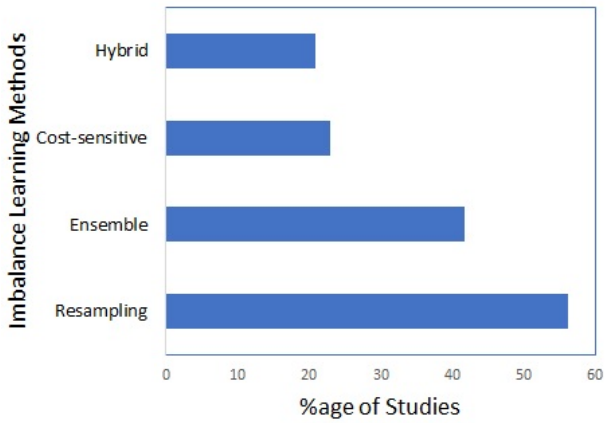

Figure 2.8: Imbalanced Learning Methods proposed in the Primary Studies

Imbalanced Learning Methods used in primary studies are stated in Table 2.11 and are enumerated below:

1. Resampling Methods: The most commonly used imbalance learning techniques by the researchers in SDP are resampling methods. 56.25% of primary studies have explored this method. Resampling methods tackle the data imbalance problem at the data level.

Table 2.11: Study-wise Imbalanced Learning Methods in literature

Imbalanced Learning Methods	PS Identifier	Count	%
Resampling	IS1, IS2, IS3, IS4, IS5, IS6, IS8, IS11, IS12, IS13, IS20, IS22, IS25, IS26, IS29, IS30, IS31, IS32, IS34, IS35, IS39, IS40, IS41, IS42, IS43, IS44, IS46	27	56.25%
Ensemble	IS8, IS9, IS11, IS13, IS14, IS15, IS16, IS17, IS20, IS21, IS27, IS29, IS30, IS32, IS35, IS36, IS40, IS45, IS46, IS48	20	41.67%
Cost-sensitive	IS8, IS10, IS11, IS19, IS20, IS21, IS23, IS24, IS32, IS38, IS47	11	22.92%
Hybrid	IS7, IS11, IS13, IS18, IS28, IS32, IS33, IS36, IS43, IS48	10	20.83%

Resampling methods can be further recognized as:

- Oversampling methods: The minority samples (defective classes) are increased to create a balance between minority and majority classes. Out of 48 studies, 23 studies have used oversampling methods to alleviate the data imbalance problem.
- Undersampling methods: Some of the majority samples (non-defective classes) are removed so that the datasets have an almost equal number of samples of both classes. Out of 48 studies, 15 studies have used undersampling methods to alleviate the data imbalance problem.
- Underover resampling methods: This subcategory involves both undersampling and oversampling methods to balance the data. Hence, it includes the benefits of undersampling as well as oversampling methods. IS5 and IS32 have used the amalgamation of these methods.

This categorization with PS identifiers is provided in Table 2.12.

Table 2.12: Category-wise Distribution of Resampling and Hybrid Methods in Primary Studies

IL Methods	Subcategory	PS Identifier	Count	%age
Resampling method	Oversampling	IS1, IS2, IS3, IS5, IS6, IS8, IS11, IS12, IS13, IS20, IS22, IS25, IS26, IS29, IS30, IS31, IS32, IS34, IS35, IS41, IS42, IS44, IS46	23	47.9%
	Undersampling	IS3, IS4, IS5, IS8, IS11, IS12, IS20, IS22, IS26, IS31, IS32, IS39, IS40, IS41, IS43	15	31.3%
	Underover resampling	IS5, IS32	2	4.2%
Hybrid Method	oversampling + ensemble	IS11, IS13, IS32, IS36, IS48	5	10.4%
	undersampling + Ensemble	IS7, IS11, IS28, IS32, IS33, IS43	6	12.5%
	underover + ensemble	IS32	1	2.1%
	Cost-sensitive + ensemble + oversampling	IS18	1	2.1%
	useofstablemetric	IS37	1	2.1%

2. Ensemble methods: Multiple classifiers are combined to produce better prediction results than any single model. 41.67% of primary studies have used ensemble methods in imbalanced SDP.

3. Cost-sensitive based models: In this method, we deal with imbalanced data problem at the algorithm level. Predictions are improved by exploiting the misclassification costs. 11 studies have employed cost-sensitive learning for solving data imbalance issue.

4. Hybrid methods: This includes designing models by merging any two or more imbalance learning methods. Researchers have used oversampling+ensemble, undersampling+ensemble, underoversampling+ensemble, and cost-sensitive+ oversampling+ensemble methods. Primary studies that employed these hybrid methods can be referred to from Table 2.12.

2.5.6 Results specific to RQ6

Once the models are constructed, one needs to evaluate the performance of them. For this purpose, many performance measures are evaluated by the researcher community so that correct and reliable assessments can be executed. Various performance measures used in the primary studies are scribed in Table 2.13. Graphical representation of measures used is provided in Figure 2.9 for easy interpretation. It represents the percentage of studies that employ a particular measure.

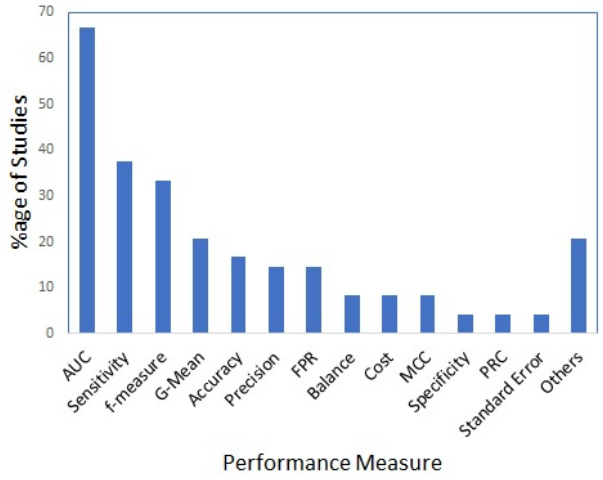

Figure 2.9: Performance Measures used in the Primary Studies

Table 2.13: Performance Measures used in the Primary Studies

Performance Measures	PS Identifier	Count
ROC-AUC	IS3, IS4, IS6, IS7, IS8, IS9, IS11, IS12, IS13, IS14, IS15, IS16, IS17, IS19, IS20, IS21, IS22, IS26, IS27, IS29, IS30, IS31, IS33, IS34, IS35, IS36, IS37, IS38, IS41, IS42, IS45, IS48	32
Sensitivity	IS10, IS13, IS17, IS18, IS19, IS20, IS21, IS27, IS31, IS33, IS34, IS35, IS36, IS37, IS40, IS42, IS43, IS46	18
F-measure	IS11, IS17, IS20, IS21, IS22, IS23, IS27, IS28, IS34, IS35, IS39, IS40, IS42, IS44, IS46, IS48	16

Performance Measures	PS Identifier	Count
G-Mean	IS2, IS8, IS16, IS22, IS31, IS33, IS36, IS43, IS45, IS46	10
Accuracy	IS4, IS10, IS13, IS19, IS29, IS35, IS42, IS46	8
Precision	IS18, IS27, IS34, IS37, IS40, IS42, IS46	7
FPR	IS19, IS20, IS21, IS31, IS33, IS36, IS43	7
Balance	IS8, IS19, IS31, IS36	4
Cost	IS10, IS18, IS24, IS28	4
MCC	IS11, IS32, IS44, IS48	4
Specificity	IS37, IS46	2
PRC	IS4, IS48	2
Standard Error	IS1, IS45	2
Kappa Statistic	IS5	1
Coverage	IS23	1
AAE	IS47	1
AM	IS4	1
CV	IS41	1
G-measure	IS44	1
ARE	IS47	1
FNR	IS35	1
NECM	IS19	1
Fault-Percentile-Average	IS23	1

The most widely used performance measure for SDP with imbalanced data is the ROC-AUC. Two-third of the total studies have considered ROC-AUC for model assessment. Figure 2.9 shows the performance measures used in terms of the percentage of imbalanced SDP studies. 37.5% of studies employed sensitivity and 33.3% of studies exploited f-measure for imbalanced defect classification.

Researchers also used G-Mean, accuracy, precision, FPR, Balance, cost, Matthew's Correlation Coefficient (MCC), specificity, and Precision-Recall Curve (PRC).

Some other measures are used by only a single study amongst all studies under consideration like Standard Error, Kappa Statistic, Coverage, Average Absolute Error (AAE), Arithmetic Mean (AM), coefficient of variation (CV), G-measure, Average Relative Error

(ARE), FNR, Normalized Expected Cost of Misclassification (NECM) and Fault-Percentile-Error. The performance measures that are used only in one study are clubbed together and denoted as others in Figure 2.9.

2.5.7 Results specific to RQ7

Statistical Tests are important to quantify the reliability of the results. Statistical validation enhances confidence in the conclusions of the research. Table 2.14 shows whether the study supported statistical validation or not. Only 58.33% of studies have conducted statistical tests to strengthen their results.

Table 2.14: Statistical Validation in the Primary Studies

Statistical Test	PS Identifier	Count	%
NO	IS2, IS3, IS10, IS13, IS16, IS17, IS18, IS22, IS24, IS25, IS27, IS29, IS30, IS34, IS35, IS39, IS40, IS41, IS42, IS43	20	41.67%
YES	IS1, IS4, IS5, IS6, IS7, IS8, IS9, IS11, IS12, IS14, IS15, IS19, IS20, IS21, IS23, IS26, IS28, IS31, IS32, IS33, IS36, IS37, IS38, IS44, IS45, IS46, IS47, IS48	28	58.33%

Statistical tests can be further categorized into parametric and non-parametric tests. Details of statistical tests used in the primary studies are presented in Table 2.15. The percent-wise distribution of statistical tests used in primary studies is illustrated in Figure 2.10.

- Parametric Tests: These tests can be conducted if we have a normal distribution of data. These tests may have additional requirements to be fulfilled before their application. Many parametric tests are used by the selected primary studies like T-test, ANOVA, and Welch's F-Test. 25% of the primary studies have employed the T-test. IS4, IS5, and IS12 have used ANOVA to statistically validate the results and carried out post-hoc analysis using Tukey's Test.

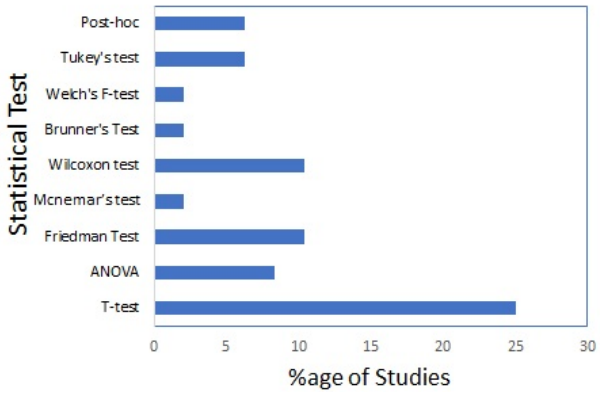

Figure 2.10: Statistical Tests used in the Primary Studies

Table 2.15: Statistical Tests employed in the Primary Studies

	Statistical Test	PS Identifier	Count	%
Parametric Tests	T-test	IS1, IS6, IS7, IS8, IS9, IS11, IS14, IS15, IS23, IS36, IS37, IS38	12	25.0%
	ANOVA	IS4, IS5, IS12, IS46	4	8.33%
	Welch's F-test	IS45	1	2.08%
Non-parametric Tests	Friedman Test	IS19, IS20, IS33, IS44, IS47	5	10.42%
	Wilcoxon signed rank test	IS26, IS28, IS32, IS48, IS47	5	10.42%
	Mcnemar Test	IS21	1	2.08%
	Brunner's Test	IS31	1	2.08%
Post-hoc Analysis				
Parametric Tests	Tukey's Test	IS4, IS5, IS12	3	6.25%
Non-parametric Tests	Bonferroni Correction	IS20, IS28, IS44	3	6.25%

- Non-parametric Tests: These tests do not require any underlying assumption to be met, hence have wider applicability. Friedman Test and Wilcoxon signed-rank test are used by 10.42% of studies each. Other non-parametric tests used in the primary studies are Brunner's test and the Mcnemar test. Brunner's test is used in IS31 and McNemar Test is used in IS21. After the Friedman testing, post-hoc analysis in IS44 and IS20 is carried out using the Friedman test followed by Bonferroni correction. IS28 applied the Wilcoxon signed-rank test with Bonferroni correction.

It is important to carry post-hoc analysis after conducting the statistical test to eradicate family-wise errors. This ensures that results achieved are not by chance and hence strengthens the validation of results. Very few studies have incorporated post-hoc analysis. Researchers should understand the importance of statistical validation and also conduct post-hoc analysis for unbiased predictions.

2.5.8 Results specific to RQ8

The tools reported in the primary studies are recorded in Table 2.11. We identified three categories of tools used in the primary studies:

1. Tools used for Model Construction:

    - WEKA: The most commonly used tool by the primary studies is the Waikato Environment for Knowledge Analysis (WEKA). WEKA is open-source software that provides an integrated platform for predictive classification with a wide range of machine learning techniques. 21 studies out of the total used WEKA classifiers for building models.
    - RapidMiner 5.2: Like WEKA, RapidMiner is also predictive analytics software that supports data mining data preprocessing, machine learning, and deep learning. IS9, IS14, and IS15 employed RapidMiner 5.2.

2. Tool used for Metric Extraction: IS5 used Krakatau Professional metrics tool to extract class-level metrics from Mozilla-Delta.

3. Tools used for conducting Statistical Tests: Two tools ASTASTA and SPSS were reported by one primary study each. ASTASTA tool was used by IS44 to carry out the Friedman test and post-hoc Bonferroni test. IS46 has reported usage of the SPSS tool for single-factor ANOVA to perform multiple comparisons.

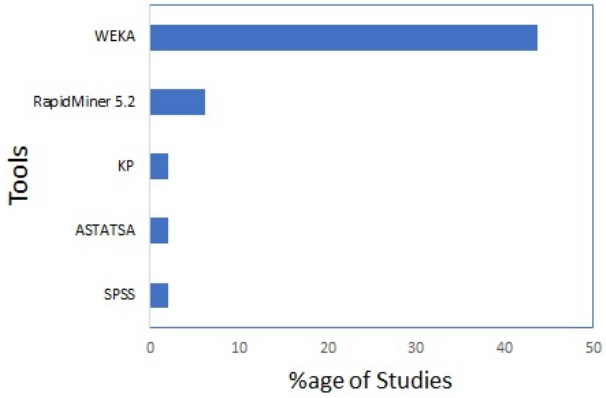

Figure 2.11: Tools used in the Primary Studies

Table 2.16: Tools

Purpose	Tool Used	PS Identifier	Count	%
Model Construction	WEKA	IS3, IS4, IS5, IS6, IS7, IS8, IS11, IS12, IS13, IS17, IS23, IS27, IS29, IS30, IS32, IS34, IS36, IS40, IS41, IS47, IS48	21	43.75%
	RapidMiner 5.2	IS9, IS14, IS15	3	6.25%
Metric Extraction	Krakatau Professional (KP)	IS5	1	2.08%
Statistical Test	ASTATSA	IS44	1	2.08%
	SPSS	IS46	1	2.08%

2.6 Discussion and Future Directions

An extensive literature review was performed to assess the current state-of-art of SDP studies with an imbalanced data problem. Studies from 2000 to 2020 were examined. 48 studies were selected by following the systematic procedure. Analysis of these selected studies was performed to answer various RQs. Discussion of these RQs paved the way for this thesis work by identifying the potential research gaps.

The information retrieved for each RQ from this survey is listed below:

- RQ1: Out of 48 studies, 45 studies have used open-source datasets. Approximate 70% of studies are based on NASA datasets. Studies lack proprietary software as their code has copyright issues. Apart from NASA datasets, other open-source software are used in fewer studies.

- RQ2: Dimensionality reduction is not carried out by 60% of studies. Only 19 studies have used some strategy to get a reduced set of metrics. Studies that have used feature selection techniques include filter and wrapper methods. The most widely used amongst all are filter methods that are involved in 10 primary studies. Wrapper methods include evolutionary techniques like PSO, GA, and Bat algorithm. Feature extraction is observed by three studies only. They addressed Fisher LDA and Principal Component Analysis (PCA). One study explored a cost-sensitive approach while another one makes use of AUC thresholding to scrutinize important features.

- RQ3: The majority of studies have employed the ten-fold cross-validation for model validation. Two studies have used hold-out validation whereas only one of the primary studies conducted LOO cross-validation to validate the model.

- RQ4: Varied techniques are used for defect prediction. We categorized these techniques under 11 headings and provide the taxonomy for SDP studies that deal with

imbalanced data. The prominent techniques used in the literature are C4.5, NB, RF, SVM, and KNN.

- RQ5: Solutions to the data imbalance problem in literature comprise of resampling methods, ensemble methods, cost-sensitive methods, and hybrid methods. 56.25% of studies have used the resampling methods to deal with the data imbalance problem and successfully resolved the problem to an extent. 20 studies depicted using ensembles for alleviating the problem. The cost-sensitive approach is applied only by 11 studies. Hybrid methods are also proposed as imbalanced learning methods by 20.83% of the studies.

- RQ6: ROC-AUC being the stable metric is used in 66.67% of the primary studies for performance evaluation. Developed models were evaluated with help of sensitivity and f-measure in 18 and 16 studies respectively. G-Mean is used only by 10 studies. Balance is also used which is a promising measure in the imbalanced domain but it is assessed by only 4 studies. Other measures include accuracy, precision, FPR, cost, MCC, specificity, etc.

- RQ7: Statistical validation is performed only in 58.33% of the primary studies. From these studies, 60.7% of studies have used parametric tests. Studies used T-test, ANOVA, and Welch's F-test in the parametric category. From the non-parametric domain, the Friedman test and Wilcoxon signed-rank test are the most prominently used. Mcnemar test and Brunner's test are also observed in one study each. Post-hoc analysis is done after both parametric and nonparametric tests in three studies each. Parametric tests used Tukey's test whereas non-parametric tests used Bonferroni correction for the post-hoc analysis. 20 studies did not report for any statistical test.

- RQ8: This RQ revealed that the use of WEKA is reported by 43.8% of the primary

studies. Three studies also used RapidMiner 5.2 for developing classification models. One study reported feature extraction from Krakatau Professional. ASTASTA and SPSS are used to apply the statistical test in one study each.

After taking into account the result discussion specific to each RQ in the preceding section, we propose the following future directions.

- In literature studies, approximately 70% of studies have used data imbalance problem with NASA datasets. Datasets related to the software engineering domain are not explored much. Therefore, more studies should be carried out using the imbalanced open-source software data so that results can be generalized.

- Feature reduction techniques are employed by approximately 40% of the primary studies only. The reduced set of features save time and assist in delivering effective and accurate SDP models. Therefore, future studies should consider effective feature selection techniques to reduce the dimensionality of data.

- The results of the systematic review indicate that only four studies have determined the useful metrics with the help of search-based techniques. Future work should focus on conducting more studies with search-based feature selection. Literature holds some studies that empirically prove CFS as the worthy feature selection technique in filter and wrapper methods. But it lacks studies that can assess and compare model performances when features are selected by CFS and any of evolutionary techniques. Thus, in the future, such studies need to be performed for new findings.

- Less use of search-based techniques in feature selection intrigued us to probe its use in model development. Not so surprising, there were very few studies [4, 70] achieving this target. Future work should encompass software defect classification using search-based techniques.

- In continuation with the above observation, researchers should investigate the effect of imbalance learning methods like resampling methods with the search-based classification for varied datasets and establish their applicability for SDP. There is no such study in the literature that involved resampling methods with search-based techniques for SDP to date.

- Even though resampling methods are the most commonly used imbalanced learning method, future studies should include their assessment with open-source datasets. For example, Apache datasets are analyzed with resampling methods only in 20.8% of the total primary studies. Hence, there arises a need to reexamine the oversampling and undersampling methods with ML techniques for such datasets in the context of solving the data imbalance problem.

- Similar reflections were made for ensemble methods. Out of all the studies that addressed ensemble methods as the solution to the data imbalance problem, 76.5% of studies exploited NASA datasets. Only 12.5% of total primary studies evaluated ensemble-based models with Apache datasets. Also, there is only one study IS32 that used a hybrid method that combined the resampling method and ensemble method for better classification in Apache datasets. Future work should focus on the development of machine learning models with ensemble methods and resampling-based ensembles with datasets other than NASA datasets.

- It has been observed that only three studies used Apache datasets for providing a cost-sensitive solution to the data imbalance problem. Therefore, future research work should include studies examining the cost-sensitive models for SDP in imbalanced data.

- The systematic review has revealed that approximately 42% of concerned studies did not statistically verify their results. Future studies should include the statistical

validation of their results as it increases the credibility of the research done.

Chapter 3

Research Methodology

3.1 Introduction

After the identification of the research gaps in Section 2.6 and objectives in subsection 1.5.3.2, there is a need to design the research methodology to achieve these objectives. The research methodology is the systematic outline of steps to be carried out to perform robust and reliable empirical experimentations. It is a structured and organized approach followed to achieve the desired objectives. The roadmap is conceived for empirical investigation of the research problem and experimentally sound, reasonable solutions are proposed.

This chapter is organized as follows: Section 3.2 portrays the research process complied in designing and execution of empirical experiments. Further sections explain the steps involved in the research process. Research problem and research questions are established in Section 3.3. Section 3.4 presents the summary of existing literature. Next, Section 3.5 defines the independent and dependent variables used in this research work. Section 3.6 gives the source and brief description of the datasets used. The preprocessing steps applied to these datasets are scribed in Section 3.7. Imbalance learning methods used to tackle data imbalanced data problem are mentioned in Section 3.8. Section 3.9 abridges the data

analysis techniques used in subsequent chapters to develop models. Section 3.10 briefly outlines the SDP model development and cross-validation method used in this research work. Performance measures used for model evaluation are addressed in Section 3.11. Finally, Section 3.12 introduces the statistical tests used in proceeding chapters to validate the results.

3.2 Research Process

The research process elaborates on all the necessary steps required for conducting the research work. Figure 3.1 gives a thumbnail sketch of these steps that are elaborated in the following sections of this chapter.

3.3 Identification of Research Problem

To initiate the research, we need to formulate the research problem. In this step, we identify the research problem and state it in form of RQs. RQs are carefully designed to fulfill research objectives.

RQs addressed in this thesis are stated in subsection 1.5.3.1.

3.4 Reviewing the Literature

A survey of existing related studies is required for a better understanding of the problem. It provides us with information about the extent to which the research problem has been investigated by previous studies. Many empirical studies have been successfully executed in the SDP domain establishing a strong relationship between OO metrics and defect proneness of classes [2, 3, 35, 133–138]. Various ML techniques are exploited to build SDP models and a broad picture of their usage in the literature world is provided by many studies

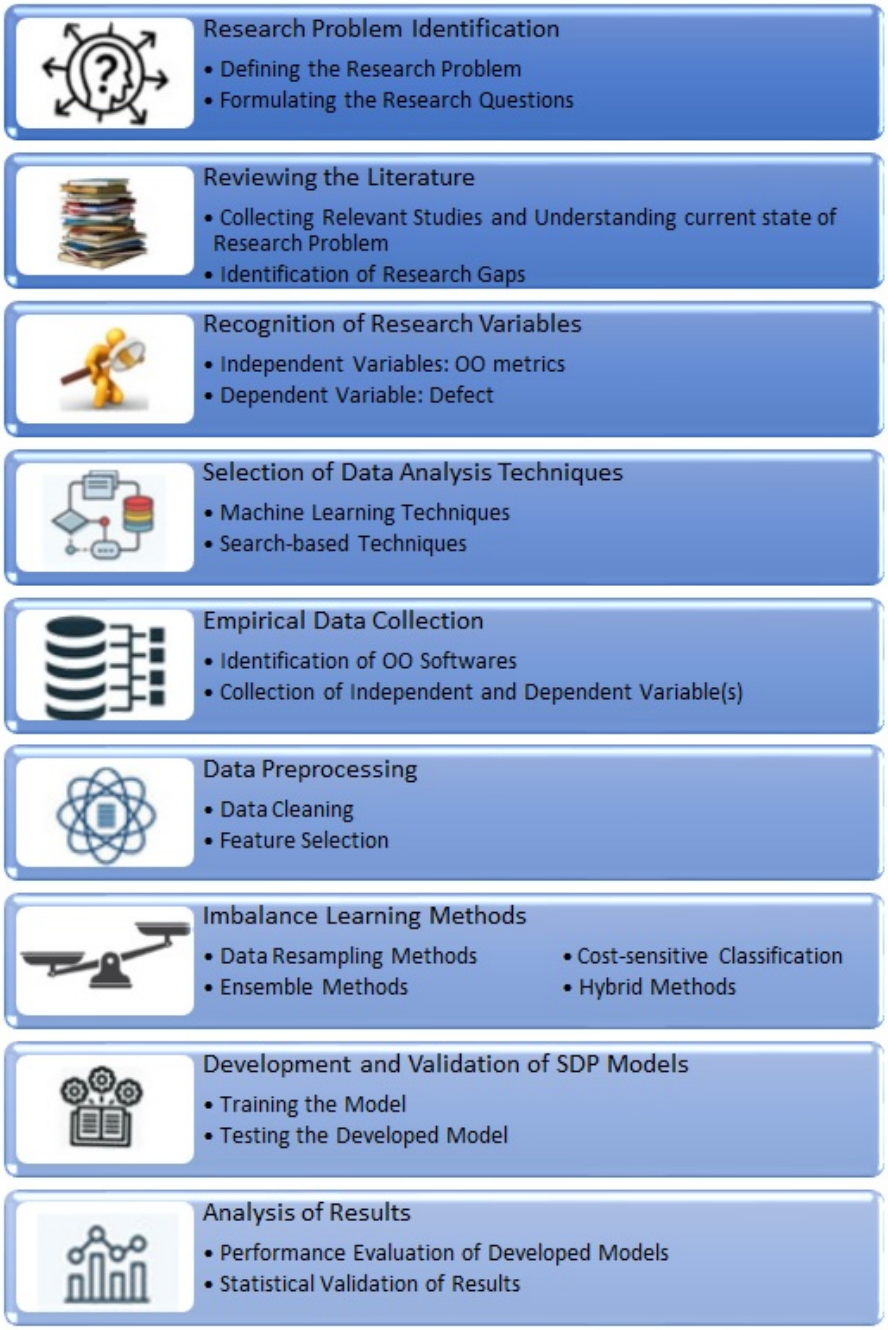

Figure 3.1: Research Process

[51, 57, 139–141].

We performed the systematic literature review concerning software defect prediction in Chapter 2. According to Koru and Tian [5] software data follows the Pareto principle and only 20% of classes are responsible for all defects present in the software. Keeping this in consideration, the survey was done with attention to imbalanced data. With the help of directions provided by conducted review and existing reviews in literature, further course of action is set to seek answers for RQs set in Section 3.3.

3.5 Recognition of Research Variables

There are two kinds of research variables that need to be identified for empirical investigations of any study:

- *Independent Variables:* Independent variables are the predictor variables of the system. These variables help the researcher to predict the dependent variable of the study. Independent variables need to be independent of each other to reduce the model bias. We have used OO metrics of the software as the independent variables in the research work. Details are provided in subsection 3.5.1.

- *Dependent Variable:* Dependent variable is the variable that we need to predict. It is also known as the target variable or response variable. In this thesis, the defect is the dependent variable and is explained in subsection 3.5.2.

3.5.1 Independent Variables

The independent variables used in this thesis are different OO metrics characterizing a software system from various aspects. The OO metrics used in this thesis work include–

- Chidamber and Kemerer (CK) Metric suite [36]: Six popularly used metrics namely

WMC, DIT, NOC, LCOM, RFC, and CBO are incorporated in this metric suite. This metric suite has been validated in many empirical studies for developing SDP models.

- Quality Model for Object-Oriented Design metric suite (QMOOD) [37]: The study uses few metrics from this metric suite namely NPM, DAM, MFA, MOA, and CAM along with CK metrics for developing defect prediction models. These metrics are also well exploited in related studies to develop effective software quality prediction models.

- Metrics proposed by Tang et al. [41]: Three metrics CBM, AMC, and IC that were proposed by Tang et al. [41] after analysis of CK metrics are used to predict defects in this thesis.

- Metrics proposed by Martin [142]: In this thesis, two coupling metrics are explored that are suggested by Martin [142]. These metrics are Ca and Ce.

- Li and Henry Metric [143]: LCOM3 metric is a variant of LCOM and was proposed in [143]. This metric is also utilized to find the possibility of probable defects in the early phases of software development.

- McCabe's Metric: Cyclomatic complexity (CC) is proposed by McCabe and is included in the set of independent variables. Two variants of Cyclomatic complexity, Maximum Cyclomatic complexity (Max_CC) and average cyclomatic complexity (Avg_CC), are considered.

- Lines of Code (LOC), a popular size metric, is also a part of the independent variables' set in addition to the above metrics. LOC, CBM, AMC, and LCOM3 are explained by Henderson-Sellers [18].

  These metrics have been widely used by the researcher community and software practitioners as independent variables. These metrics are summarized in Table 3.1.

Table 3.1: Independent Variables

S. No.	OO Metric	Definition	Source
1	Coupling between Objects (CBO)	It signifies the count of classes coupled to a particular class. Interdependence between classes exist because of inheritance, method calls, arguments, etc.	[36]
2	Response for a Class (RFC)	It is computed as the count of different methods that can be executed on receiving message by any object of that class	[36]
3	Coupling Between Methods (CBM)	It represents the total count of new or redefined methods to which all the inherited methods are coupled.	[41]
4	Inheritance Coupling (IC)	It provides the count of parent classes to which a given class is coupled.	[41]
5	Afferent coupling (Ca)	For a class, Ca is defined as the count of classes that uses or calls it.	[40]
6	Efferent coupling (Ce)	For a class, Ce signifies the count of classes that are used by that class.	[40]
7	Number of Children (NOC)	It provides the count of immediately derived classes for the base class.	[36]
8	Depth of Inheritance Tree (DIT)	It computes the levels of inheritance a class depicts in object hierarchy.	[36]
9	Measure of Functional Abstraction (MFA)	It is computed as the ratio of count of inherited methods in a class to the count of accessible methods (that are accessible by member methods) in the class.	[37].
10	Weighted methods per class (WMC)	It is calculated as the addition of its methods' complexities.	[36]
11	Number of Public Methods (NPM)	It represents the count of public methods in the class.	[37].
12	Lines of Code (LOC)	It is a size metric and accounts for number of fields, methods, and instructions in every method of given class.	[41]
13	Average Method Complexity (AMC)	This metric measures the average method size for each class.	[41]
14	Lack of cohesion in methods (LCOM)	This metric determine the sets of class methods that are not related through the sharing of some of the class's fields.	[36]
15	Cohesion Among Methods of Class (CAM)	This metric computes the relatedness among methods of a class based upon the parameter list of the methods.	[37].

S. No.	OO Metric	Definition	Source
16	Lack of cohesion in methods (LCOM3)	LCOM3 gives the count of connected methods in a class. Methods are connected iff thet share atleast one instance variable. It varies between 0 and 2. $$LCOM3 = \frac{\frac{1}{a}(\sum_{j=1}^{a} \mu(Aj) - m}{1 - m} \quad (3.1)$$ where m - number of methods in class, a - number of variables in class, μ(A) - number of methods that access a variable	[143]
17	Measure of Aggregation (MOA)	This metric is a count of number of data fields in a class whose type is user-defined.	[37].
18	Data Access Metric (DAM)	It is computed as the ratio of the number of private or protected attributes to the total number of attributes declared in the class.	[37].
19	Average Cyclomatic Complexity (Avg_CC)	CC is the count of independent paths in a method of a class. Avg_CC represents the mean of CC of all methods in a class.	[144]
20	Maximum Cyclomatic Complexity (Max_CC)	Max_CC signifies the highest CC gained by all methods in a class.	[144]

These OO metrics are related to internal quality attributes (IQAs) of the software. Different IQAs covered through the OO metrics used in this thesis are coupling, inheritance, size, cohesion, composition, encapsulation, and complexity. Table 3.2 provides these IQAs with their definitions and related OO metrics.

Table 3.2: IQAs and related OO Metrics

S.No.	IQA	Definition	OO Metrics
1	Coupling	Coupling represents the extent to which one class is dependent on other classes in the software. Low coupling is preferred in good quality software.	CBO, RFC, CBM, IC, Ca, Ce
2	Inheritance	Inheritance signifies the tendency of one class to gain characteristics from another class(es). The class that inherits is called Derived class and class whose properties are inherited is called Base class	NOC, DIT, MFA

S.No.	IQA	Definition	OO Metrics
3	Size	Size of a class represents the way to measure it. It can be determined by the number of data members, methods, or code length.	WMC, NPM, LOC, AMC
4	Cohesion	Cohesion defines the extent to which data and methods in a class are interrelated. High cohesion is desired in good quality software.	LCOM, CAM, LCOM3
5	Composition	Composition defines the has-a relationship between a class and a user-defined class that is used as an instance of former.	MOA
6	Encapsulation	Encapsulation refers to hiding the internal details of a class from the classes that use it.	DAM
7	Complexity	Complexity is the extent to which any class or component of the software is difficult to understand and manage.	Avg_CC, Max_CC

3.5.2 Dependent Variable

Datasets collected contain a continuous variable 'bug' representing the number of defects in a particular class. A class can be either defective or non-defective. While there is no defect observed in non-defective classes, a defective class can have one or more defects. In this research work, we address the binary defect prediction problem in OO software. Therefore, 'bug' is converted into a binary variable by replacing '0' with 'No' and natural numbers with 'Yes'. The designed binary dependent variable is named 'defect'. It can have two possible values- 'Y' and 'N' that reflect whether the software class is defective or not.

3.6 Empirical Data Collection

Data for empirical validation may be collected from industrial software, open-source software, or academic software. Industrial software are proprietary software that are not easily available to the researcher community. Academic software are student projects in uni-

versities and colleges, and hence not reliable because of lack of expertise. Over the last few years, therefore open-source software have been popular and widely used for empirical validation. The main reasons for their increased usage are cost-effectiveness, ease of availability, and ease of replicability. This thesis also employed open-source software for developing SDP models.

Datasets of JAVA open-source projects were collected from the Promise repository (http://openscience.us/repo). Jureczko and Madeyski [145] extracted 20 OO metrics discussed in subsection 3.5.1 of these software with the help of the CKJM and Bugzilla tool. Additionally, the number of defects in each class is required to be recorded. Data, then, need to be preprocessed and require data cleaning. Data is checked for the existence of any missing data or redundant classes. Jureczko and Madeyski [145] provided good quality datasets as there is no missing data and redundant class in them.

Empirical data is collected from 21 versions of nine OO software systems. Apache datasets used in this work are -Ivy, Tomcat, Xerces, Synapse, Camel, Ant, Log4j, and Xalan. Jedit metrics were also collected.

A brief description of the software is provided below:

- Apache Ivy is an application package to manage project dependencies. It includes tracking, recording, and resolving project dependencies and is characterized by its flexibility. Above all, it is very easy and simple to use. Being highly flexible, extensible, and easily integrable with Apache Ant, it is very useful and popular.

- Apache Tomcat is an open-source container for servlets and JavaServer Pages. It is a lightweight application server and has captured 60% of the market.

- Apache Synapse is a service-oriented architecture framework that provides web services. It fulfills web requests and can simultaneously handle load balancing, protocol switching, and routing.

- Apache Xerces provides a library for an XML parser. It allows parsing, manipulating, and validating XML data. Xerces enables the applications to read and write XML documents.

- Apache Camel provides a single platform to work with different kinds of transports, pluggable components, and messaging models. Transports are executed like components of Camel. It exploits a uniform resource identifier to accomplish this task. Version Camel1.6 resolved 169 issues including two new components and three new data formats with respect to the previous version.

- Apache Ant is used to build applications in JAVA language. It has built-in support for compiling, testing, and running the concerned application.

- Apache Log4j provides utilities for asynchronous logging services. It is a fast and reliable logging package with three primary components: logger, appender, and layout. The logger captures the logging information. Appender is accountable for publishing the logging information, whereas the layout component formats the information in different formats.

- Apache Xalan is an XSLT processor and has food library support. It is used for converting XML documents into HTML, text, or other XML document types.

- Jedit is a text editor software with strong macros and plugin support. It comes with easy configuration and customization features.

Table 3.3 describes the addressed datasets with their statistics. #Total represents the total number of classes, #ND represents the number of non-defective classes, #D corresponds to the number of defective classes in the particular dataset. %age#D symbolizes the percentage of defective classes in the corresponding software. #IR determines the imbalance ratio (IR) of the particular version of the software.

IR is calculated as:

$$IR = \frac{No.\ of\ Non-Defective\ classes}{No.\ of\ Defective\ classes} \tag{3.2}$$

Table 3.3: Description of Datasets

Dataset Name	#Total	#ND	#D	%age#D	IR
Jedit4.3	492	481	11	2.23	43.73
Ivy1.4	241	225	16	6.64	14.06
Tomcat6.0	858	781	77	9.85	10.14
Synapse1.0	157	141	16	10.19	8.81
Ivy2.0	352	312	40	11.36	7.8
Jedit4.2	367	319	48	13.08	6.65
Xalan2.4	723	613	110	15	5.57
Xerces1.3	453	384	69	15.23	5.57
Xerces1.2	440	369	71	16.13	5.2
Camel1.4	872	727	145	16.63	5.01
Camel1.6	965	777	188	19.48	4.13
Ant1.7	745	580	166	22.28	3.49
Jedit4.0	306	231	75	24.51	3.08
Log4j1.0	135	101	34	25.18	2.97
Jedit4.1	312	233	79	25.3	2.95
Synapse1.1	222	162	60	27.02	2.7
Jedit3.2	272	182	90	33.1	2.02
Synapse1.2	256	170	86	33.5	1.98
Log4j1.1	109	72	37	33.95	1.95
Xalan2.6	885	474	411	46.44	1.15
Xalan2.5	803	416	387	48.19	1.07

3.7 Data Preprocessing

This section describes the descriptive statistics of OO metrics of the datasets and data preprocessing steps. Datasets are thoroughly checked for redundant data or missing data. If present, such data need to be removed to achieve accurate and precise model predictions. In software engineering predictive modelling, data preprocessing is the crucial step.

3.7.1 Descriptive Statistics

Empirical data need to be understood properly, therefore descriptive statistics are calculated for each dataset. The descriptive statistics for each dataset are presented in Appendix A.

The following descriptive statistics are reported for OO metrics.

- *Minimum*: The minimum statistic for an OO metric reports the minimum value of the corresponding metric in the dataset.
- *Maximum*: The maximum statistic for an OO metric reports the maximum value of the corresponding metric in the dataset.
- *Mean*: The mean statistic states the average value for an OO metric in the dataset.
- *Median*: The median statistic gives facts about the frequency distribution of the classes for an OO metric in the dataset. Median is preferable to mean when data has outliers.
- *Standard Deviation (SD)*: It projects the central tendency of the OO metric and measures the dispersion in data.
- *Skewness*: The measure of skewness tells about the data normality of the OO metric. For normal data, skewness $= 0$.
- *Standard Error of Skewness (SES)*: It accounts for the normality check for an OO metric. It is calculated as the ratio of skewness to its standard error.

Cumulative descriptive statistics of datasets are reported in Table 3.4.

Table 3.4: Descriptive Statistics of Cumulative Datasets

Metric	Minimum	Maximum	Mean	Median	SD	Skewness	SES
WMC	0.00	413	10.82	6	17.57	8.6	0.025
DIT	0.00	8	2.2	2	1.51	1.22	0.025
NOC	0.00	102	0.5	0	2.71	13.42	0.025

Metric	Minimum	Maximum	Mean	Median	SD	Skewness	SES
CBO	0.00	499	11.16	7	18.63	8.96	0.025
RFC	0.00	583	29.78	18	39.82	4.64	0.025
LCOM	0.00	41713	127.76	4	944.76	25.74	0.025
Ca	0.00	498	5.61	2	16.74	11.37	0.025
Ce	0.00	101	5.79	3	7.69	3.03	0.025
NPM	0.00	231	8.25	4	12.99	5.79	0.025
LCOM3	0.00	2	1.11	0.87	0.68	0.28	0.025
LOC	0.00	23683	311.24	108	779.54	12.3	0.025
DAM	0.00	1	0.51	0.63	0.47	-0.07	0.025
MOA	0.00	41	0.79	0	1.79	5.84	0.025
MFA	0.00	1	0.43	0.43	0.43	0.13	0.025
CAM	0.00	1	0.48	0.44	0.25	0.61	0.025
IC	0.00	5	0.54	0	0.87	1.79	0.025
CBM	0.00	33	1.48	0	3.25	3.65	0.025
AMC	0.00	2052	28.41	14	72.16	11.16	0.025
Max_CC	0.00	167	4.39	2	7.65	8.59	0.025
Avg_CC	0.00	28.67	1.37	1	1.26	5.47	0.025

The following observations have been made after analyzing the descriptive statistics.

- Median of NOC is 0 and DIT is 2. MFA also has a low median value of 0.43. This signifies that inheritance was not preferred in these software. Inheritance metrics, therefore, will have a low impact on building defect prediction models.

- Mean of CBO, RFC, CBM, and IC is 11.16, 29.78, 1.48, and 0.54 respectively. These values are different from corresponding median values. Therefore, the fact signifies that underlying data does not have a normal distribution. The same observation was made for other metrics except for DIT, LCOM3, DAM, MFA, and CAM.

- SD is maximum for LCOM and is computed as 944.76. The more the SD, the more is the dispersion of data from its mean value.

- For the normal distribution, skewness is 0. Only three metrics- LCOM3, DAM, and MFA have low skewness values of 0.28, -0.07, and 0.13 respectively. The mean of negatively skewed data will be less than the median. Except for DAM, every metric has a mean greater than the median, signifying the long tail in the right direction.

3.7.2 Feature Selection

In predictive modelling, it is important to identify the most relevant and important features. The models are constructed by using a reduced set of features. In our research problem, features are OO metrics. Applying feature selection techniques will aid in building better SDP models in terms of accuracy with less computation time. Feature selection can be accomplished by implementing either filter methods or wrapper methods.

3.7.2.1 Filter Methods

Filter methods detect the association of features and the dependent variable based on some statistics or measures. There are several filter methods used in literature like Chi-square, infogain, ReliefF, consistency-based FS, etc. Though in the literature some researchers find it difficult to empirically validate any single FS technique as the best, still CFS is the most used one.

- Correlation Feature Selection: CFS [146] finds the subset of features based on Pearson's correlation coefficient. It is independent of any learning technique. It identifies the features that have a high correlation with the class label (defect) and a low correlation with other features. It performs the univariate analysis. The predictive capability of an individual feature is explored from the subsets of features and the best subset wins. In this thesis work, CFS is used because it is the most preferred FS technique according to the survey in 2015 [51] and recently Ghotra et al. [42] also concluded CFS to be the best FS strategy after extensive analysis of 30 FS techniques. Parameters defined for CFS in our experimentation are local predictive = True, missingSeparate = False, numThreads = 1, poolSize = 1 and preComputeCorrelationMatrix = False.

  Table 3.5 reports the OO metrics selected by CFS in the datasets used in this thesis.

Table 3.5: Feature Selection with CFS

Dataset	OO Metrics selected by CFS
Tomcat6.0	CBO, RFC, LCOM, LOC, MOA, AMC, Max_CC, Avg_CC
Synapse1.0	RFC, LCOM, Ce, LOC, DAM, MFA, CAM, IC, AMC, Max_CC, Avg_CC
Ivy2.0	RFC, CE, LOC, MOA, AMC, NPM, CBO, WMC, LCOM, LCOM3, CAM
Jedit4.2	RFC, CA, CE, NPM, LCOM3, CAM, Max_CC, CBM, LCOM, MOA, LOC, CBO, AMC
Xalan2.4	RFC, LCOM, LOC, CBM, Max_CC, WMC, AMC, CBO, LCOM3, Avg_CC
Xerces1.3	WMC, DAM, IC, CBM, AMC, CE, LCOM, MOA
Camel1.4	WMC,CBO,RFC,Ca,Ce,NPM,IC,CBM
Camel1.6	DIT, NOC, CBO, LCOM, Ca, NPM, LCOM3, CAM, IC, CBM, AMC, Max_CC, Avg_CC
Ant1.7	CAM, RFC, LOC, AMC, Max_CC, LCOM, Ce, CBO, MOA
Jedit4.0	WMC, LCOM, LOC, DAM, MOA, Max_CC, RFC, CBM, CA, CE, NPM, LCOM3, DIT, CBO, Avg_CC
Log4j1.0	CBO, CA, NPM, WMC, RFC, LCOM, CE, DAM, Avg_CC, LOC, CAM
Jedit4.1	RFC, CE, LOC, MOA, CAM, LCOM, Max_CC, WMC, IC, DAM, Avg_CC
Synapse1.1	CBO, CE, LCOM, DAM, DIT, RFC, CAM, MFA, Max_CC
Jedit3.2	DIT, RFC, LCOM, CE, DAM, MOA, IC, CBM, Max_CC, CBO, LCOM3, LOC, MFA, NPM, AMC
Synapse1.2	CBO, RFC, CE, LOC, AMC, CA, WMC, MOA, CAM, CBM, Max_CC
Log4j1.1	WMC, RFC, LCOM, CE, NPM, LCOM3, MFA, MOA
Xalan2.6	NPM, LCOM3, LOC, AMC, Avg_CC

3.7.2.2 Wrapper Methods

Wrapper methods find an effective subset of features with the help of learning techniques involved and cross-validation. Many ML techniques like J48, k-Nearest Neighbor (KNN), Naïve Bayes, Logistic Regression have been used in literature for selecting the important metrics of the software. One of the aims of this thesis is to explore the search-based wrapper method to find their competency in the area of feature selection in SDP. Two variants of the popular genetic algorithm used to gain the purpose are GGA and SGA. These search-based techniques are inspired by evolution, therefore also referred to as evolutionary techniques. Individuals are selected from the population and a new population is created by crossover and mutation operations.

- Generational Genetic Algorithm (GGA) [147] : Generational Genetic Algorithm is a Pittsburgh style learning classifier system. In GGA, during evolution, numerous

candidate chromosomes are crossovered to generate new offsprings which are then mutated and added into a temporary population. The selection of candidate chromosomes from the main population is done through elitism and when the temporary population reaches its maximum number, then it replaces the old population. Fitness function is based on accuracy and cost and is defined as:

$$fitness(x) = acc(x) - \frac{cost(x)}{acc(x) + 1} + max\_cost \tag{3.3}$$

Ten fold cross-validation is done and each partition is used for training GGA for five times. Parameters are set as: Cross Probability = 0.7, Mutation Probability = 0.01, Population Size = 50, Number of Evaluations = 10000, Beta Equilibrate Factor = 0.99, Number of Neighbors in KNN = 1, Use Elitism = Yes.

- Steady-State Genetic Algorithm (SGA) [148]: SGA is Michigan style learning classifier system and is much simpler than GGA. Only two individuals are selected in one iteration to create offsprings. Out of two parents and two children, only two individuals that have the best fitness values are populated back in the pool. It requires a replacement algorithm as no intermediate population can be generated and offsprings are added to the same population. In SGA, the number of features selected is limited explicitly. In this study, the number of features selected is set to be 8 by varying it from 3 to 10. The fitness function of SGA is defined by precision obtained by KNN.

  Parameters of algorithm are set as: k in KNN = 1, number of evaluations = 5000, popLength = 100, number of features = 8.

Features selected by GGA and SGA are discussed in detail in Ch 9.

3.8 Imbalance Learning Methods

Uneven distribution of defective and non-defective classes in the software results in biased training of the classification model. This will deteriorate the software quality as the model predictions may not be accurate and reliable. Hence, in this section, we have incorporated imbalance learning methods employed in this thesis to solve imbalanced classification problem and attain the set objectives.

3.8.1 Resampling Methods

In resampling methods, the number of datapoints is changed to create a balance between majority and minority classes. If we increase the minority class samples, it is known as oversampling and if we decrease the majority class samples, it is called undersampling.

3.8.1.1 Oversampling Methods

1. ADAptive SYNthetic Sampling (ADSYN) [149]: In ADSYN, synthetic samples are generated by finding the density distribution of minority classes. Density distribution is computed using a k-nearest neighbor with Euclidian distance. It is an extension of the Synthetic Minority Oversampling Technique. It focuses on the samples that are hard to classify.

2. Synthetic Minority Over-sampling Technique (SMT) [150]: The number of minority class samples is increased by generating artificial samples in direction of k nearest neighbors of minority class samples. If one neighbor is selected, then one synthetic sample is generated corresponding to that original minority sample resulting in 100% oversampling of minority classes. In this study, k=5 results in a 500% oversampling of minority classes.

3. Safe Level Synthetic Minority Over-sampling Technique (SLSMT) [151]: Unlike the SMT version, where synthetic samples are generated randomly, in SLSMT first safe levels are calculated that helps in determining safe positions for generating the synthetic samples. If the safe level value is close to 0, it is considered noise and if the safe level value is close to k for k nearest neighbor implementation, then it is considered safe resulting in producing synthetic minority samples there.

4. Selective Preprocessing of Imbalanced Data (SPD) [152]: This technique categorizes the sample as either safe or noise based on the nearest neighbor rule where distance measurement is done using a heterogeneous value distance metric. If an instance is accurately classified by its k nearest neighbors, it is considered safe otherwise it is considered noise and then discarded.

5. Random OverSampling (ROS) [153]: ROS is a very simple oversampling technique in which minority class instances are replicated at random with the sole aim of creating a balance between majority and minority class instances.

6. Agglomerative Hierarchical Clustering (AHC) [154]: In AHC, each class is decomposed into sub-clusters and synthetic samples are generated corresponding to cluster prototypes. Since artificial samples are created as centroids of sub-clusters of classes, they, therefore extract the characteristics of that class and represent better samples than randomly generated samples.

3.8.1.2 Undersampling Methods

1. Random UnderSampling (RUS) [153] : RUS, like ROS, is a non-heuristic technique. But in this instead of replicating minority class instances, majority class instances are removed with aim of creating a balance between majority and minority class instances. The problem with this technique is that some important or useful data may

be rejected as it is based on random selection.

2. Condensed Nearest Neighbor (CNN) + Tomek's modification of Condensed Nearest Neighbor (CNNTL) [153]: First CNN is applied to find a consistent subset of samples that helps to eliminate majority class samples that are far from the decision border. Then Tomek links [46] are made between samples. If it exists between any two samples, then either both are borderline samples or one of them is noise. Samples with Tomek links that fit in majority classes are removed.

3. Neighborhood Cleaning Rule (NCL) [155]: NCL uses the edited nearest-neighbor (ENN) rule to remove majority class samples. For each training sample S_i, first it finds three nearest neighbors. If S_i= majority class sample, then discard it if more than two nearest neighbors incorrectly classifies it. If S_i = minority class sample, then discard the nearest neighbors if they incorrectly classify it.

4. One Sided Selection (OSS) [156] : OSS and CNNTL have similar working. The difference lies in the order of the application of CNN and the determination of Tomek links. OSS identifies unsafe samples using Tomek links and then applying CNN. Noisy and borderline samples are considered unsafe. Small noise may result in the flipping of the decision border of the borderline samples; therefore, they are also considered unsafe. CNN eliminates the majority of samples that are far away from decision boundaries.

3.8.2 Cost Sensitive Learning

Cost-sensitive classification can be conducted in two ways; either by adding weights to samples or by using a cost matrix to penalize type-I and type-II errors. Meta cost learners were proposed by [157] in which penalization is done in cost matrix and training instance is relabeled based on the majority voting. These errors are actually false positives and false

negatives in the model prediction. In this work, meta-cost classifiers are used and the cost penalization of wrongly predicted defective classes is done at different levels- 5, 10,15, 20, 20, 40, and 50 times the cost of wrongly predicted non-defective classes.

3.8.3 Ensemble Methods

Ensemble learning involves combining several ML techniques into one predictive model to reduce bias or variance and build robust learners. Ensemble learners explored in this thesis can be categorized as :(1) Classic ensemble methods (2) Resampling-based boosting ensembles methods and (3) Resampling-based bagging ensemble methods.

Description of Ensemble Methods

1. Classic Ensemble Methods

    - AdaBoost (AB) [158]: AB significantly enhances the efficacy of weak classifiers by performing the plurality of learning iterations. The training set generated by a particular iteration acts as an input for the next iteration, thereby multiple learning iterations generate a better classifier by using misclassified weak classifiers.
    - AdaBoostNC (ABNC) [159] : ABNC demonstrates low computation cost and much better classification accuracy than negative correlation learning (NCL) algorithms. ABNC reduces error correlation and is mostly preferred in the case of recognizing minority classes.
    - AdaBoostM1 (ABM1) [160, 161] : ABM1 is an algorithm for binary classification to train the learners sequentially. It is an ensemble technique where numbers of weak classifiers are used iteratively to improve the overall performance. It augments the performance of weak learners by adjusting the weak hypothesis returned by the weak learner. The base decision tree used in ABM1 is J48.

J48 learns from the previous trees about misclassified instances and calculates the weighted average. NumIterations in parameters represent the number of classifiers involved in this ensemble. This technique helps in reducing the bias in the model. It handles primarily nominal class which drastically improves performance. However, ABM1 lacks when the model overfits.

- Bagging (Bag) [162, 163]: Bagging or bootstrap aggregation is also one of an ensemble technique that improves the predictive capability of base classifiers by making bags of training data. Bag involves bootstrapped replicas referring to the original training data set, to be substantively used for the creation of different classifiers. Furthermore, the replicas are used to bag a weak learner such as a decision tree. The bagged tree efficacy is enhanced by the random selection of predictors. Models work in parallel and their results are averaged. Bagging reduces the variance. The default number of bags is 10 and the base classifier used is J48.

2. Resampling-based Boosting Ensemble Methods

   - DataBoost (DB) [164] : DB creates synthetic samples for both the majority class and minority class by identifying hard samples. It rebalances the weights of both the classes resulting in improved predictions for the majority as well as the minority class. It blends data generation and boosting.
   - MSMOTEBoost (MSMTB) [165] : MSMTB is a combination of AB and MSMOTE. It is preferred when a highly accurate prediction is required for the minority class.
   - RUSBoost (RUSB) [166]: RUSB combines boosting and random undersampling method to alleviate the imbalanced classification problem. Before each iteration of boosting, some of the majority class instances are randomly removed to construct a Balance between both classes.

- SMOTEBoost (SMTB) [167]: Minority class samples, which are not properly classified by the base classifier during the learning iteration phase, are efficiently oversampled by SMTB. More data corresponding to minority class can be added during any learning iteration.

3. Resampling-based Bagging Ensemble Methods

- MSMOTEBagging (MSBAG) [168] : MSBAG is tweaked from SMOTE for efficient usage of the feature space. It creates multiple synthetic instances of the minority class by selecting nearest neighbors.
- OverBagging (OBAG) [168]: OBAG employs one or more oversampling methods while preprocessing data. In other words, oversampling of the minority class observations are performed in each iteration. To enhance variability among training sets, majority classes are considered.
- OverBagging2 (OBAG2) [168]: OBAG2 involves preprocessing of different data through sampling method. The sampling method can be oversampling rather than random sampling. Such an instance involves an increase in the minority data set.
- SMOTEBagging (SMTBAG) [168]: SMTBAG is a merger of selective ensemble and BAG, and it is applied for binary classification. It enhances the count of minority instances to Balance the outspreading of classes. SMTBAG aims to equate instances of majority and minority.
- UnderBagging (UBAG) [169]: UBAG provides compiled weak learners which are developed over undersampled training sets. The undersampled data is randomly selected.
- UnderBagging2 (UBAG2) [168]: UBAG2 is used to multiply the size of positive instances. In particular, this bagging implementation involves the resam-

pling model to double the size of positive instances that are retrieved from a set of data values.

- UnderOverBagging (UOBAG) [167]: UOBAG applies the bagging technique to the instances of every bag by using any resampling mechanism. For an instance the resampling mechanism includes oversampling or undersampling.
- IIvotes (IIVOT) [170]: IIVOT is implemented for selective data preprocessing by using the SPD method for selective data preprocessing into the adaptive Ivotes ensemble. The SPD is effective while preprocessing particular learning samples as this enhances the accuracy of derived classifiers.

3.9 Selection of Data Analysis Techniques

This work was empirically carried out using several data analysis techniques including statistical techniques, ML techniques, and SBTs. In this work, we analyzed 15 ML. Statistical and ensemble techniques are covered under the umbrella of ML. Further, the performances of SDP models were evaluated with 16 SBTs.

3.9.1 Machine Learning Techniques

3.9.1.1 Statistical Techniques

- Naïve Bayes (NB) [171]: Naïve Bayes is a probability-based classifier that works on Bayes theorem. It is an instance-based learner that computes class wise conditional probabilities. Features need to be conditionally independent with each other. It provides fair results even in violation of this assumption. This ML technique works well for both categorical and numerical variables.
- Logistic Regression (LR) [172]: Logistic Regression is also a probabilistic classifier

used for dichotomous variables and assumes that the data follows Gaussian distribution. It works well in case of assumption desecration. During training coefficient values are minimized by ridge estimator to solve multicollinearity and this makes the model simpler. The algorithm runs until it converged.

- Simple Logistic (SL) [173]: Simple Logistic uses LogitBoost to construct logistic regression models. LogitBoost uses the logit transform to predict the probabilities. With each repetition, one simple regression model is added for each class. The process terminates when there is no more reduction in classification error.

- LogitBoost (LB) [174]: LogitBoost is an additive logistic regression with a decision stump as the base classifier. It maximizes the likelihood and, therefore, generalizes the linear logistic model. The base classifier taken is the decision stump which considers entropy for classification.

3.9.1.2 Neural Networks

- MultiLayerPerceptron (MLP) [175]: It is a backpropagation neural network that uses sigmoid function as the activation function. The number of hidden layers in the network is determined by the average of the number of attributes and total classes for a particular dataset. The error is backpropagated in every epoch and reduced via gradient descent. The network is then learned based on revised weights.

3.9.1.3 Nearest Neighbors

- IBk [176]: IBk is an instance-based K-nearest neighbor learner. It calculates the Euclidian distance measure of the test sample with all the training samples to find its 'k' nearest neighbors. It then assigns the class label to the testing instance based on the majority classification of nearest neighbors. Only one nearest neighbor is

determined with k=1 and the class label of that nearest neighbor is assigned to the testing instance.

- KStar [177]: Like Ibk, KStar is also an instance-based learning algorithm. The difference between the two techniques is about the similarity measures they use. IBk exploits Euclidian distance and KStar uses the similarity measure based on entropy. KStar exhibits good classification competence for noisy and imbalanced data.

3.9.1.4 Ensemble Methods

Ensemble methods are promising ML techniques because in addition to classification they also assist in solving imbalance issue as discussed above. Apart from ABM1 and Bag, we use following ensemble methods for defect classification.

- Iterative Classifier Optimizer (ICO): LogitBoost is used as the iterative classifier in this technique. Cross-validation is utilized for its optimization. In the experiments conducted, it goes through 50 iterations to decide for the best cross-validation.

- Logistic Model Tree (LMT) [173, 178]: Logistic Model Tree is a meta-learning algorithm that uses logistic regression at leaf nodes for classification. A combination of linear logistic regression and decision tree helps in dealing with the bias-variance tradeoff. This technique is robust to missing values and can handle numeric as well as nominal attributes.

- Random Tree (RT) [179]: Random Tree is an ensemble-based supervised learner where different trees are constructed from the same population. Random samples of the population are generated to form different trees with a random selection of features. After bags are constructed, models are developed and majority voting is performed to classify the class.

- Random SubSpace method (RSS) [180]: Random SubSpace is used to construct random forest. Randomly feature subsets are selected to generate multiple trees. Bagging is performed with Reptree. Reptree is faster than the basic decision tree and generates multiple trees in each iteration. It then selects the tree whose performance is the best.

3.9.1.5 Decision Trees

- Pruning rule-based classification tree (PART) [181]: PART is a rule-based learning algorithm that exploits partial C4.5 decision trees and generates rules at each iteration. PART stands for a pruning rule-based classification tree. The rule that results in the best classification is selected. MDL is used to find the optimal split. smaller the confidence factor more will be the pruning done.

- J48 [162]: J48 is a JAVA implementation of the C4.5 decision tree. It follows the greedy technique to build a decision tree and uses the gain ratio as splitting criteria. Leaf nodes are the classification labels- defective and non-defective and rules can be derived by traversing from root to leaf node. By default, it generates a binary tree, and one-third of the data is used for reduced error pruning.

Parameter settings of these ML techniques are scribed in Table 3.6.

Table 3.6: Parameter Settings of ML Techniques

Category	ML technique	Parameter Settings
Statistical ML techniques	NB	useKernelEstimator = false, displayModelInOldFormat = false, useSupervisedDiscretization = false
	LR	Ridge: 1.0E-8, useConjugateGradientDescent = false, maxIts = -1
	SL	Heuristic Stop = 50, Max Boosting Iterations = 500, useCrossValidation = True, weightTrimBeta = 0.0

Category	ML technique	Parameter Settings
	LB	Zmax = 3.0, likelihoodThreshold = -1.7976931348623157E308, numIterations = 10, numThreads = 1, poolSize = 1, seed = 1, shrinkage = 1.0, useResampling = False, weightThreshold = 100
Neural Networks	MLP	Hidden layer = a, Learning rate = 0.3, Momentum = 0.2, Training time = 500, Validation threshold =20
Nearest Neighbour	IBk	KNN = 1, nearestNeighbourSearchAlgorithm = LinearNNSearch,
	KStar	globalBlend = 20, entropicAutoBlend = False
Methods Ensemble Methods	ABM1	numIterations = 10, weightThreshold = 100, seed =1, classifier = J48: confidenceFactor = 0.25, minNumObj = 2, numFolds = 3, seed = 1, subtreeRaising = True, useMDLcorrection = True
	Bag	numIterations = 10, numExecutionSlots = 1, seed =1, bagSizePercent = 100, classifier = J48: confidenceFactor = 0.25, minNumObj = 2, numFolds = 3, seed = 1, subtreeRaising = True, useMDLcorrection = True
	ICO	evaluationMetric = RMSE, lookAheadIterations = 50, numFolds = 10, numRuns = 1, numThreads = 1, poolsize = 1, seed = 1, stepSize = 1, iterativeClassifier = LogitBoost
	LMT	errorOnProbabilities = False, fastRegression = True, minNumInstances = 15, numBoostingIterations = -1, weightTrimBeta = 0.0
	RT	KValue = 0, breakTiesRandomly = False, maxDepth = 0, minNum = 1, minVarianceProp = 0.001, numFolds = 0, seed = 1
	RSS	numExecutionSlots = 1, numIterations = 10, seed = 1, subSpaceSize = 0.5, classifier = Reptree: initialCount = 0.0, maxDepth = -1, minNum = 2.0, minVarianceProp = 0.001, numFolds = 3, seed = 1
Decision Tree	PART	confidenceFactor = 0.25, minNumObj = 2, numFolds = 3, reducedErrorPruning = False, seed = 1, useMDLcorrection = True
	J48	confidenceFactor = 0.25, minNumObj = 2, numFolds = 3, seed = 1, subtreeRaising = True, useMDLcorrection = True

3.9.2 Search-based Techniques

3.9.2.1 Genetic Algorithm based SBTs

- Genetic Algorithm based Classifier System with Adaptive Discretization Intervals (GA_ADI) [182] is a genetic-based classifier system that follows the Pittsburgh ap-

proach. Instead of being static, discretization intervals are adaptive in this SBT.

- Genetic Algorithm based Classifier System with Intervalar Rule (GA_INT) [182] is also a genetic-based Pittsburgh style classifier system where the fitness function is a combination of complexity and accuracy. Following the minimum description length (MDL) principle, it keeps control of the length of candidate chromosomes.

- In the Generational Genetic Algorithm for Instance Selection (GGA) [183] parent selection, crossover, and mutation are performed on population till the optimal solution is obtained. A large population is replaced by fitter chromosomes in each generation.

- Steady-State Genetic Algorithm for Instance Selection (SGA) [183] is a plain flavor of the GA where only a few chromosomes are replaced in each generation when crossover and mutation are performed. It is simpler than GGA but requires more generations to converge.

- Incremental Learning with Genetic Algorithms (ILGA) [184] exploits the GA by providing incremental learning to accommodate changes in the number of classes and the addition of new training examples or predictors. Agents can learn from co-agents (cooperative learning) or their environment.

- Memetic Pittsburgh Learning Classifier System (MPLCS) [182] hybridizes GAssist with local search algorithms to find the set with the minimum rules that yield the maximum accuracy while training. Incremental learning makes it to converge faster.

- Linear Decreasing Weight - Particle Swarm Optimization (LDWPSO) [185] is the modification of PSO implemented with a GA, similar to CPSO. The difference lies in its implementation strategy. It uses linear decreasing weight function instead of constriction coefficients while exploring for a solution in search space.

- sUpervised Classification System (UCS) [186] is an accuracy based classification system developed for supervised learning. The fitness function is calculated as the ratio of the number of correct classification to the number of matches for a rule. It ignores incorrect rule sets and applies GA only on correct rule sets.

- X –Classifier System (XCS) [187] uses a GA with reinforcement learning. For each action by the system, a suitable reward is given for the correct action. The algorithm evolves as a population of classifiers. Each classifier has a rule and some parameters for estimating the rule's quality. The parameter settings of XCS and UCS are the same.

- Bioinformatics-oriented hierarchical evolutionary learning (BIOHEL) [188] amalgamates features of GAssist with iterative rule learning. It works on the Pittsburgh-style learning classifier system.

3.9.2.2 Particle Swarm Optimization based SBTs

- Constricted Particle Swarm Optimization (CPSO) [185] is the variant of PSO and GA. CPSO uses constriction coefficients efficiently to avoid the explosion of search space while searching for an optimum solution and therefore results in comparatively less execution time.

- In Real Encoding - Particle Swarm Optimization (REPSO) [189], rules are determined with the assistance of particle swarm optimization for the continuous and categorical data type. It is a type of Michigan style classifier system where the only single rule is determined in each generation.

3.9.2.3 Instance-based SBTs

- Adaptative Search for Instance Selection (CHC) [183] uses a unique recombination operator that swaps half of the bits of parents that are different. The best chromosome achieved by this process in a generation is used to seed the population for the next generation.
- Population-Based Incremental Learning (PBIL) [183] is an effective evolutionary technique for problems with binary search space. The probability vector contains probability values which in turn are responsible for the creation of a solution vector.

3.9.2.4 Rule-based SBT

CO-Evolutionary Rule Extractor (CORE) [190] uses the Michigan approach for population generation and Pittsburgh approach for coding chromosomes in co-population, therefore, utilizing the benefits of both approaches and supporting cooperative coevolution. The limit of rules in the rule-set defines the number of co-populations.

3.10 Development and Validation of SDP Models

3.10.1 Model Development

This thesis develops SDP models to uncover probable defects in future software based on historic data. Supervised models are constructed with various data analysis techniques that are described in Section 3.9. As discussed in Chapter 1, defect prediction models are designed as illustrated in Figure 1.3. Once the model is developed, it is tested against unseen data points and these data points are classified as defective or non-defective. Classes that have a high probability to be defective will be assigned more resources allocated to the project. We need to validate the empirically driven model and there are many ways to

achieve this like hold-out validation, LOO validation, and ten-fold cross-validation. In this thesis, we have used a ten-fold cross-validation method.

3.10.2 Validation Method

Ten-fold cross-validation is carried out to reduce the partitioning bias. Data is divided into ten partitions. Nine partitions are used for the training part and the remaining one partition is used for the testing part [191]. Then performance measures are averaged across ten folds. Applying ten-fold cross-validation reduces the bias of the models [134].

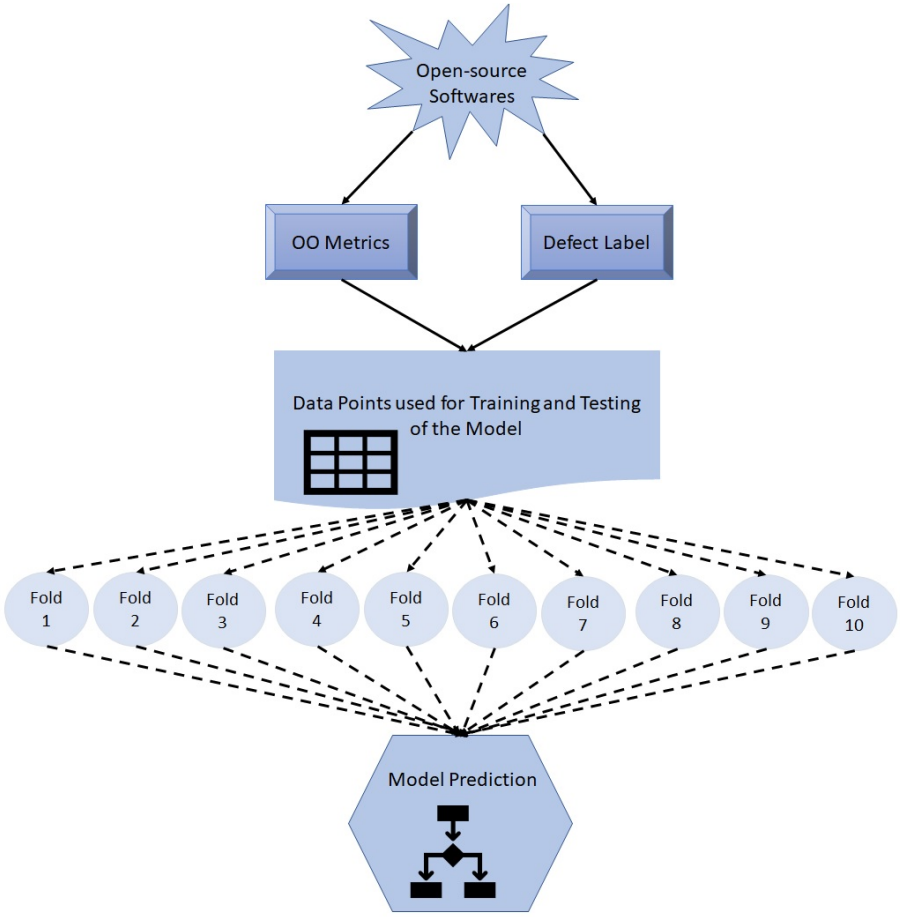

Figure 3.2: Ten-fold Cross-validation

3.11 Performance Measures

Performance measures are the metrics that are used to quantify the performance of constructed defect prediction models. When we build any defect prediction model, it predicts the probability of whether the concerned class is defective or non-defective. Broadly, we can categorize them into two headings:

- Performance Measures that are dependent on the threshold: Accuracy, sensitivity, specificity, G-Mean, Balance, etc.
- Performance Measures that are independent of threshold: ROC-AUC

For threshold-dependent metrics, the probability of defect-proneness is calculated for a particular class. Generally, if it is greater than 0.5, the class is predicted as defective, otherwise not. These performance measures can be calculated by using the confusion matrix shown in Table 3.7.

Table 3.7: Confusion Matrix

	Predicted Positive	Predicted Negative
Actual Positive	True Positive (TP)	False Negative (FN)
Actual Negative	False Positive (FP)	True Negative (TN)

TP represents the number of defective classes predicted correctly. TN represents the number of non-defective classes predicted correctly. FP represents the number of non-

defective classes that are wrongly predicted as defective classes. FN represents the number of defective classes that are wrongly predicted as non-defective classes.

When it comes to imbalanced data, the selection of appropriate performance evaluators plays a critical role. The use of accuracy to evaluate performance is specious when data is imbalanced. Instead, robust performance evaluators like ROC-AUC, G-Mean, and Balance should be used in the class imbalance framework [192, 193]. This thesis work employs ROC-AUC because it is a threshold independent metric and a stable metric [58, 194]. In addition to this, ROC-AUC is capable of handling skewness of data [74, 195]. The use of G-Mean and ROC-AUC is advocated by [63] to handle the class imbalance problem. Balance is also one of the reliable measures used in defect prediction studies [196].

Various performance measures can be defined as follows:

- Sensitivity: The sensitivity indicates the probability of correctly predicted defective classes out of total defective classes. whereas specificity refers to the probability of identifying non-defective classes correctly. Sensitivity or True Positive Rate (TPR) is defined as –

$$Sensitivity = \frac{TP}{TP + FN} X100 \tag{3.4}$$

- Specificity: It is defined as the percentage of correctly predicted non-defective classes amongst actual non-defective classes. Specificity is calculated as –

$$Specificity = \frac{TN}{TN + FP} X100 \tag{3.5}$$

- False Positive Rate (FPR): FPR is the probability of false alarm. It exemplifies the proportion of non-defective classes that are misclassified as defective classes

amongst actual non-defective classes. It is computed as –

$$FPR = \frac{FP}{FP + TN} X 100 \tag{3.6}$$

- G-Mean: G-Mean is defined as the geometric mean of sensitivity and specificity for any classifier. G-Mean maintains a balance between both these accuracies [196]. Therefore, it is wise to use G-Mean as an effective measure to assess imbalanced data. It is calculated as–

$$GMean = \sqrt{SensitivityXSpecificity} \tag{3.7}$$

- Balance: Balance corresponds to the Euclidean distance between a pair of sensitivity and FPR [196]. It is desired to have high sensitivity and low FPR. Balance can be computed as–

$$Balance = 1 - \sqrt{\frac{(0 - FPR)^2 + (1 - Sensitivity)^2}{2}} \tag{3.8}$$

- Area under the curve (ROC-AUC): ROC-AUC is widely accepted as a consistent and robust performance evaluator for predictions in imbalanced data [74, 195]. It is threshold independent and can handle skewed data. It is a measure to distinguish between the two classes. The range of ROC-AUC is (0, 1). Higher the ROC-AUC value, the better the prediction model. ROC-AUC value of 0.5 signifies that the model cannot differentiate between the two classes. ROC-AUC values from 0.7 to 0.8 are considered acceptable. ROC-AUC values greater than 0.8 are considered excellent.

3.12 Result Analysis and Statistical Validation

The results need to be statistically verified because without the involvement of statistics results may be misleading [58]. Statistical tests strengthen the conclusion validity of the study. The survey conducted also revealed that only 40% of relevant SDP studies used statistical tests. Therefore, to affirm the resultant model predictions we performed statistical tests. Statistical tests used are non-parametric. We preferred non-parametric statistical tests because the data under consideration does not follow a normal distribution as discussed in subsection 3.7.1. Results are statistically validated using Friedman test [197] and Wilcoxon signed-rank test [198].

- Friedman Test: Friedman test is used to find the rankings of performance of k techniques with multiple datasets [197]. 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 significantly different from each other.
  - Alternate Hypothesis (H_a): The performance of different techniques is significantly different from each other.

  The nonparametric tests are exercised in this thesis because software data do not follow normal distribution [199]. These tests work well for data that have outliers or cases where data distribution is not normal [199]. The Friedman test is executed for different performance evaluators for establishing the statistical difference amongst the performance of developed SDP models. We need to compare several ML models built for several datasets. Therefore, Friedman rankings are computed using the Friedman test.

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

$$\chi^2 = \frac{12}{nk(nk+1)} \sum_{i=1}^{k} T_i^2 - 3n(k+1) \tag{3.9}$$

where T_i is the rank total of i^{th} technique, n is the number of total datasets, $k-1$ is degree of freedom of the Friedman test.

Friedman test statistic is computed at $\alpha = 0.05$. Empirical results must be 95% significant to strengthen conclusion validity. Friedman test is used in many defect prediction studies [200, 201]. Demsar recommends using it as a nonparametric alternative to the parametric ANOVA test [199].

- Wilcoxon-signed Rank test: If the Friedman test results tend to be positive, post hoc analysis is carried by Wilcoxon signed-rank test to find pair-wise significant differences. It reduces the family-wise error.

  This test is exploited in this thesis in two ways: (i) post-hoc test after the Friedman test results come to be significant and (ii) an independent test performed to compare the pairwise performance of two techniques. The Wilcoxon signed-rank test is a pair test applicable only when two different techniques are evaluated on the same set of datasets [202]. Wilcoxon signed-rank test is one of the most preferred tests when SDP studies are performed [200].

  The hypothesis tested against Wilcoxon signed-rank test is defined as follows:

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

The wilcoxon signed-rank test statistic W is also computed at $\alpha = 0.05$ using the following formula:

$$W = \frac{Q - \frac{1}{4}n_r(n_r + 1)}{\sqrt{\frac{1}{24}n_r(n_r + 1)(2n_r + 1)}} \tag{3.10}$$

where Q is the minimum of S^+ and S^-, S^+ is the sum of ranks where the difference is positive, S^- is the sum of ranks where the difference is negative, n_r is number of reduced pairs.

We can reject the null hypothesis if W statistic is less than 0.05. This will conclude in the significant difference between the two techniques that are compared with each other. 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 BonC value.

$$BonC = \frac{0.05}{Numberof Comparisons} \tag{3.11}$$

To include family-wize errors, we reject the null hypothesis if W statistic is less than BonC value.

Results must be statistically analyzed and reported to increase their reliability.

Chapter 4

Tackling Class Imbalance Problem at Data Level: Resampling Methods

4.1 Introduction

SDP deals with uncovering the probable future defects. With the increasing complexity of software, early prediction of defects, and assurance of good software quality of projects becomes a difficult task. Efficient defect prediction helps in the timely identification of areas in software which can lead to defects in software owing to better resource utilization [1]. As discussed in previous chapter source code metrics give useful insights to software quality attributes like cohesion, coupling, size, inheritance, etc, and are extensively used in developing software defect models [2, 3, 137]. Effective models can be generated using the OO metrics. To achieve this task of SDP, various ML techniques have been used by several researchers from the past two decades [51] but software defect data is mostly imbalanced [203]. This issue of SDP has recently gained a high interest in researchers in the software engineering community. As discussed earlier, data is imbalanced if the number of minority classes, in our case, defective classes, is much lower than the majority class,

i.e., non-defective classes. This imbalanced distribution of data misguides classifiers while learning the defective class correctly and hence results in biased and inaccurate results. A good defect prediction model will be the one that is trained on the similar distribution of instances of defective and non-defective classes. In this chapter, we addressed the imbalanced ratio of defective and non-defective classes of software and provided its solution by modifying these ratios by either increasing samples of minority classes or decreasing samples of majority classes.

Many real-world problems encounter this imbalanced data problem [19–22, 24–26]. Researchers are actively participating in finding solutions for imbalanced classification in the software engineering domain for the last two decades. One of the prominent solutions is using resampling methods comprising of undersampling and oversampling methods [28].

A large number of software metrics adds to the problem of constructing good SDP models. This curse of dimensionality is handled by selecting important and distinct features using CFS. CFS is selected as it is widely accepted FS technique and emerged as an effective FS technique in a benchmark study [42]. This study deals with the application of resampling methods to handle imbalanced data problem for 12 Apache datasets and performing a comparative analysis of various SDP models developed using ML techniques. In this study, six oversampling methods and four undersampling methods are explored on different datasets to deal with their imbalanced nature. Oversampling methods used are SMT, SLSMT, SPD, ADSYN, ROS, and AHC. Undersampling methods considered here are CNNTL, RUS, NCL, OSS. The main objective of this chapter is to ascertain the importance of dealing with imbalanced data problem using (1) resampling methods and (2) stable performance measures to build correct and effective SDP models. The research work done in this chapter also identifies the important IQAs of software on which developers need to focus on. RQs to achieve the aforementioned objectives are designed as follows:

- RQ1: Which features are repeatedly selected by CFS in software engineering datasets?

- RQ2: What is the performance of ML techniques on imbalanced data while building SDP models?
- RQ3a: What is the comparative performance of various SDP models developed using resampling methods?

  RQ3b: Is there any improvement in the performance of SDP models developed using resampling methods?
- RQ4: Which resampling method outperforms the addressed undersampling and oversampling methods for building an efficient SDP model?
- RQ5: Which ML technique performs the best for SDP in imbalanced data?

The answers to these questions are explored by building ML models on CFS selected features using ten-fold cross-validation. Predictive performances of developed models are evaluated using stable performance evaluators like sensitivity, G-Mean, Balance, and ROC-AUC. Statistical validation is carried out using the Friedman test followed by post-hoc analysis that is performed using the Wilcoxon signed-rank test with Bonferroni correction. The conducted study will acquaint developers with useful resampling methods and performance evaluators that will assist them to solve imbalanced data problem. This study also guides developers and software practitioners about the important metrics that affect the SDP potential of ML models. The result examination ascertained ROS-based ML models as the best defect predictors for datasets related to the software engineering domain. With ROS as a resampling method, nearest neighbors and ensemble methods gave a comparable performance in SDP.

This chapter is organized as follows: Section 4.2 presents the research background. The design framework for the empirical study is explained in Section 4.3. Section 4.4 summarizes the results with their interpretation. Section 4.5 presents the conclusions with potential future directions.

The results of this work are communicated in [204].

4.2 Research Background

This section describes the research elements and framework established to build a classification model for defect prediction from dataset collection to model validation.

4.2.1 Independent and Dependent Variable(s)

The independent variables are 20 OO metrics described in Table 3.1. The dependent variable is defect and is explained in subsection 3.5.2.

4.2.2 Empirical Data Collection

Datasets mined from the promise repository are used for empirical predictive modelling and validation. We have used 12 datasets to perform the experiments. Datasets used are Tomcat6.0, Synapse1.0, Ivy2.0, Jedit4.2, Xerces1.3, Camel1.6, Ant1.7, Jedit4.0, Log4j1.0, Synapse1.1, Synapse1.2, and Log4j1.1. Details of these datasets can be referred from Section 3.6. The percentage of defective classes in addressed projects in the study varies from 9.85% to 33.9%. This low percentage represents the imbalanced ratio of defective and non-defective classes in the datasets. The statistics of datasets used in this chapter are listed in the Table 3.3 of Section 3.6.

4.2.3 Feature Selection

To achieve the objectives set in this chapter, we need to build SDP models that are trained on good quality data. With large number of metrics used to train model, the results can be of less credibility because of some extraneous features. We employed CFS to eradicate

these unnecessary metrics because it is widely accepted technique in literature for feature selection [51]. Recently many studies like [42, 53] have empirically proved CFS to be the best feature selection technique. Features that are highly correlated with the dependent variable 'Defect' are selected. A list of metrics selected by CFS for each dataset is presented in Table 3.5.

4.2.4 Model Development and Performance Measures

Model development involves applying resampling methods for treating imbalanced data problem. Six oversampling methods and four undersampling methods were applied to create a balance between majority and minority classes. Details of these methods are given in subsection 3.8.1. These methods are implemented using a knowledge extraction tool based on evolutionary learning (KEEL) [205]. Default parameter settings are used in the execution of models so that study can be easily repeated and replicated.

We developed models based on 15 ML techniques that are explained in subsection 3.9.1.

Talking of imbalanced data, the selection of appropriate performance measures plays a critical role. The predictive performance of the developed models are realized using four performance measures- sensitivity, Balance, G-Mean and ROC-AUC. These measures are summarized in Section 3.11.

4.3 Experimental Framework

This section discusses the design framework established to build a classification model for defect prediction from dataset collection to model validation.

Figure. 4.1 explains the experimental setup for the study.

Ten-fold cross-validation is carried out to reduce the partitioning bias. The working of ten-fold cross-validation is mentioned in subsection 3.10.2.

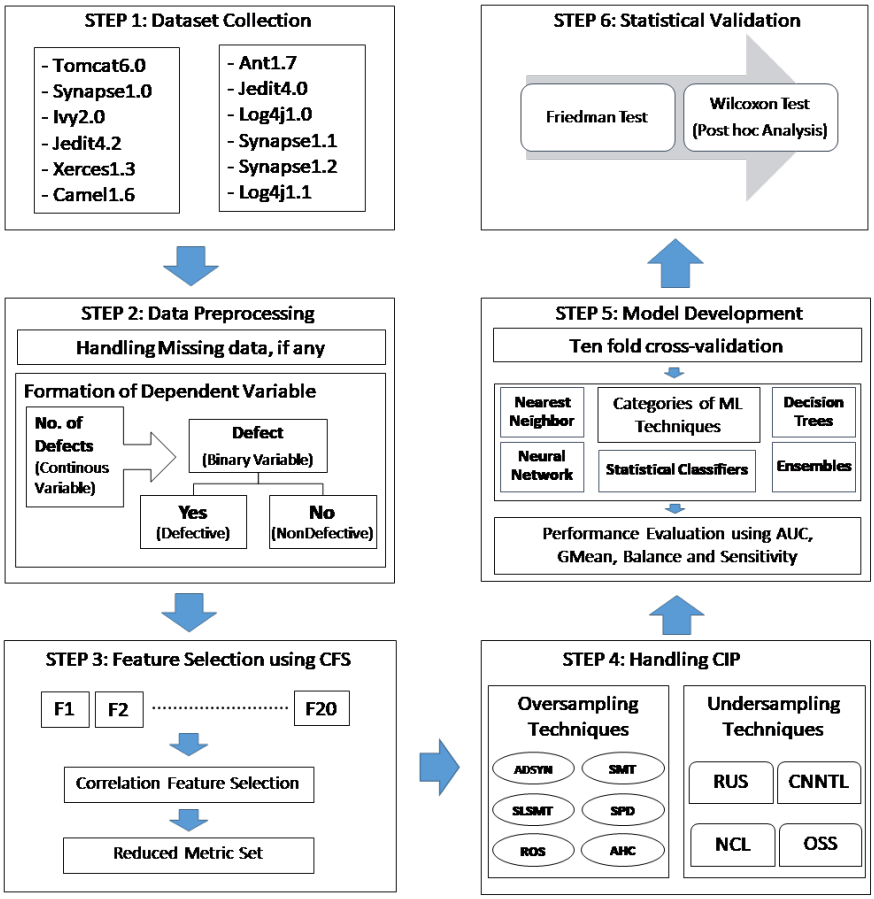

Figure 4.1: Experimental framework for SDP in imbalanced data

4.3.1 Statistical Analysis and Hypothesis Evaluation

The results need to be statistically verified because results may be misleading without the involvement of statistics [58]. The results of the chapter are statistically evaluated using two non-parametric test Friedman test and Wilcoxon signed-rank test. Bonferroni correction is included to remove family-wize error. These tests are explained in Section 3.12.

4.4 Experimental Results and Analysis

This section describes the answers to the RQs of the chapter and discusses the obtained results.

4.4.1 RQ1: Which features are repeatedly selected by CFS in software engineering datasets?

Software engineering datasets refer to software engineering research data or software development related data. OO metrics of these software act as the predictor in predictive modelling. 20 OO metrics of datasets fundamentally define the IQAs of the software and can be grouped into cohesion, coupling, size, complexity, inheritance, encapsulation, and composition metrics as explained in subsection 3.5.1.

Proportion selection of IQAs and CFS selected metrics are scribed in Table 4.1. #Selected denotes the number of times a particular metric is selected by CFS for all datasets. LCOM was selected by 11 datasets whereas the Ce metric was selected by 10 datasets. This RQ contemplates the metrics that are important for SDP. The weightage of each metric that was selected by CFS for 12 datasets is considered and their proportion selection was determined for each IQA.

In cohesion metrics, LCOM, CAM, and LCOM3 were chosen by 91.67%, 66.67%, and 41.67% of datasets respectively. The cumulative proportion of the selection of cohesion metrics is 66.7%. This shows that cohesion metrics are important for SDP and while developing the software, developers can focus more on LCOM and CAM values. Similarly, the composition metric, MOA, was selected by 66.67% of the datasets.

Exploring Table 4.1, though the cumulative proportion of coupling metrics is 55.6% its significance can be judged by selecting the top three selected metrics–RFC, Ca, and Ce. RFC is picked by 10 datasets whereas Ca and Ce are opted by 9 datasets each. Considering

only these three metrics, the proportion selection of coupling metrics raises from 55.65% to 77.78%.

In size metrics, LOC and AMC are more preferred software metrics for defect prediction. The least selected metrics belong to the inheritance category for all datasets. Therefore, resource investment can be done wisely by developers. The number of times any metric is selected for all the datasets guides developers and software practitioners in determining its worth for SDP.

Table 4.1: Proportion Selection of IQAs and CFS selected metrics

IQA	OO Metric	#Selected	Proportion Selection
Cohesion	LCOM	11	0.667
	CAM	8	
	LCOM3	5	
Composition	MOA	8	0.667
Size	WMC	6	0.583
	NPM	6	
	LOC	8	
	AMC	8	
Coupling	Ca	9	0.556
	Ce	10	
	CBO	5	
	RFC	9	
	CBM	3	
	IC	4	
Complexity	Avg_CC	4	0.5
	Max_CC	8	
Encapsulation	DAM	5	0.462
Inheritance	NOC	1	0.194
	MFA	3	
	DIT	3	

4.4.2 RQ2: What is the performance of ML techniques on imbalanced data while building SDP models?

Figure 4.2 presents the boxplots depicting predictive capability of ML models on imbalanced data in terms of ROC-AUC, Balance, G-Mean, and sensitivity.

Tables 4.2, 4.3, 4.4, and 4.5 presents the dataset wise predictive capability of ML mod-

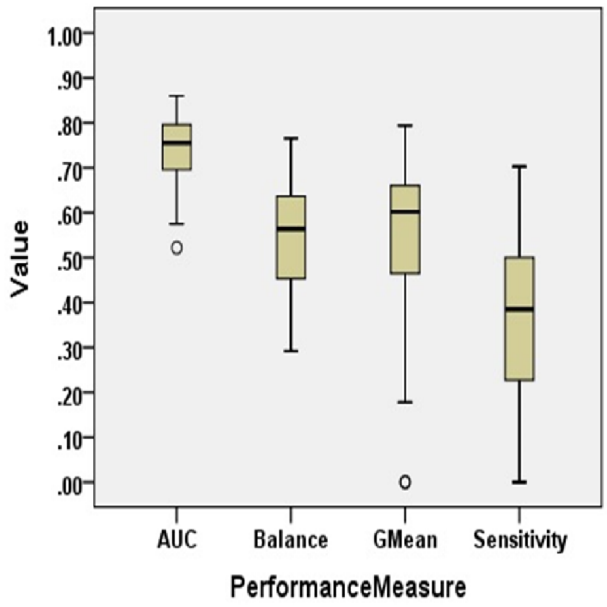

Figure 4.2: Boxplot representation of ML Models' Performance with Original Data (NS)

els on imbalanced data in terms of ROC-AUC, Balance, G-Mean, and sensitivity.

ROC-AUC Analysis:

The ROC-AUC of ML models of imbalanced data varies from 0.52 to 0.86 with a median value of 0.76. Only 55.6% of models have ROC-AUC greater than 0.75. ROC-AUC is less than 0.85 in 97.8% of cases. Log4j1.1 depicted the best ROC-AUC value of 0.86 with NB. Analysis of Table 6 reveals that statistical techniques, NB and SL, have performed fairly good in terms of ROC-AUC, depicting the highest ROC-AUC values for Log4j1.1, Ant1.7, Jedit4.2, Log4j1.0, and Tomcat6.0. Similarly, LMT models were able to predict the highest ROC-AUC values for five datasets. Overall, 25% of ML models have ROC-AUC less than 0.7.

Table 4.2: ROC-AUC values of SDP models with imbalanced data-Without Resampling methods

Dataset	NB	LR	SL	LB	MLP	IBk	KStar	ABM1	Bag	ICO	LMT	RT	RSS	PART	J48
Ant1.7	0.8	0.82	0.83	0.8	0.81	0.69	0.78	0.78	0.8	0.79	0.83	0.67	0.8	0.76	0.74
Camel1.6	0.68	0.69	0.68	0.73	0.69	0.64	0.67	0.71	0.72	0.73	0.68	0.59	0.69	0.68	0.59

	NB	LR	SL	LB	MLP	IBk	KStar	ABM1	Bag	ICO	LMT	RT	RSS	PART	J48
Ivy2.0	0.79	0.79	0.76	0.77	0.73	0.68	0.73	0.73	0.81	0.72	0.73	0.61	0.83	0.72	0.72
Jedit4.0	0.76	0.78	0.76	0.78	0.79	0.72	0.75	0.78	0.74	0.76	0.77	0.66	0.77	0.71	0.67
Jedit4.2	0.83	0.8	0.84	0.82	0.82	0.66	0.74	0.75	0.82	0.83	0.84	0.59	0.84	0.84	0.69
Log4j1.0	0.83	0.82	0.85	0.78	0.73	0.64	0.73	0.71	0.77	0.76	0.85	0.62	0.78	0.73	0.65
Log4j1.1	0.86	0.84	0.83	0.83	0.82	0.75	0.82	0.76	0.82	0.81	0.81	0.68	0.85	0.8	0.72
Synapse1.0	0.74	0.72	0.67	0.73	0.74	0.66	0.67	0.64	0.59	0.73	0.68	0.58	0.59	0.52	0.59
Synapse1.1	0.75	0.74	0.77	0.69	0.77	0.7	0.76	0.78	0.77	0.7	0.77	0.66	0.75	0.67	0.66
Synapse1.2	0.78	0.8	0.79	0.78	0.76	0.67	0.77	0.75	0.8	0.76	0.82	0.67	0.76	0.73	0.74
Tomcat6.0	0.78	0.8	0.81	0.79	0.78	0.64	0.72	0.73	0.77	0.81	0.81	0.58	0.78	0.75	0.67
Xerces1.3	0.78	0.76	0.73	0.81	0.72	0.77	0.82	0.76	0.8	0.79	0.76	0.68	0.83	0.77	0.6

Balance Analysis:

The range of Balance in NS is from 29.20 to 76.55. In Tomcat6.0, only 59.01 value is achieved as the maximum value by NB. Balance values achieved by RSS, PART, and LMT are comparatively lower than other ML techniques. The median value for Balance for all models in the NS case is 56.35. Ivy2.0, Synape1.0, and Tomcat6.0 attained the maximum Balance value with NB. IBk also resulted in maximum Balance values for Jedit4.0, Synapse1.1, and Xerces1.3. 68.3% of datasets have a Balance value of less than 65. Only 3.3% of datasets have a Balance value greater than 75.

Table 4.3: Balance values of SDP models with imbalanced data-Without Resampling methods

	NB	LR	SL	LB	MLP	IBk	KStar	ABM1	Bag	ICO	LMT	RT	RSS	PART	J48
Ant1.7	64.16	56.81	55.17	67.82	61.32	63.66	58.32	62.07	63.32	69.46	55.17	63.27	62.13	65.82	68.55
Camel1.6	43.82	36.42	36.04	37.9	38.21	52.16	47.66	51.13	44.95	38.65	37.54	52.48	31.54	38.64	42.22
Ivy2.0	58.88	45.18	38.12	43.4	41.65	57.25	48.55	50.35	46.84	43.33	38.1	51.69	32.82	38.12	38.12
Jedit4.0	50.67	57.26	56.48	60.06	58.23	67.65	57.81	60.81	62.86	65.56	57.33	62.14	57.26	53.61	62.24
Jedit4.2	57.05	48.4	45.47	48.37	51.35	55.58	49.72	55.58	51.31	46.92	45.47	49.28	38.12	44	45.39
Log4j1.0	66.61	66.13	64.4	65.68	55.97	59.29	47.63	59.29	68.17	59.99	64.4	56.54	54.16	58.12	60.09
Log4j1.1	76.55	74.46	74.84	73.63	75.07	74.33	69.03	69.92	66.33	72.11	74.67	66.06	69.36	70.73	70.52
Synapse1.0	68.04	60.17	42.51	42.49	51.26	50.95	50.95	42.33	42.49	42.53	42.51	46.5	29.29	29.2	33.62
Synapse1.1	63.68	61.93	56.22	53.22	61.99	65.25	61.39	62.43	61.65	52.13	56.22	64.51	51.59	58.69	58.89
Synapse1.2	67.44	62.4	59.83	67.12	70.13	64.95	68.76	66.83	69.14	68.49	64.57	64.78	62.36	67.56	69.04
Tomcat6.0	55.6	42.13	38.47	43.06	36.63	45.53	41.17	44.78	43.04	40.3	38.47	45.57	32.05	35.72	33.88

	NB	LR	SL	LB	MLP	IBk	KStar	ABM1	Bag	ICO	LMT	RT	RSS	PART	J48
Xerces1.3	56.67	44.62	39.51	56.87	53.74	63.66	60.95	58.71	56.84	53.84	51.79	59.77	47.73	50.72	54.8

G-Mean Analysis:

As depicted from Table 4.4, NB achieved the highest G-Mean values for 58.33% of datasets. When no resampling method is used, G-Mean ranges from 0 to 0.79.

Table 4.4: G-Mean values of SDP models with imbalanced data-Without Resampling methods

	NB	LR	SL	LB	MLP	IBk	KStar	ABM1	Bag	ICO	LMT	RT	RSS	PART	J48
Ant1.7	0.68	0.61	0.59	0.71	0.65	0.66	0.61	0.65	0.67	0.72	0.59	0.65	0.66	0.69	0.71
Camel1.6	0.44	0.31	0.31	0.34	0.35	0.54	0.49	0.53	0.46	0.36	0.34	0.54	0.18	0.36	0.42
Ivy2.0	0.62	0.47	0.35	0.44	0.41	0.61	0.51	0.53	0.49	0.44	0.35	0.54	0.22	0.35	0.35
Jedit4.0	0.53	0.61	0.61	0.64	0.62	0.7	0.61	0.64	0.67	0.69	0.61	0.64	0.61	0.57	0.65
Jedit4.2	0.61	0.51	0.47	0.51	0.55	0.59	0.52	0.59	0.55	0.49	0.47	0.51	0.35	0.45	0.47
Log4j1.0	0.71	0.69	0.69	0.68	0.59	0.62	0.49	0.62	0.71	0.63	0.69	0.58	0.58	0.62	0.64
Log4j1.1	0.79	0.77	0.78	0.75	0.79	0.75	0.73	0.71	0.69	0.74	0.78	0.67	0.74	0.74	0.73
Synapse1.0	0.71	0.65	0.43	0.43	0.55	0.53	0.53	0.42	0.43	0.43	0.43	0.47	0	0	0.24
Synapse1.1	0.66	0.66	0.6	0.55	0.66	0.66	0.64	0.65	0.65	0.54	0.6	0.65	0.55	0.61	0.61
Synapse1.2	0.7	0.66	0.63	0.69	0.71	0.66	0.71	0.68	0.71	0.7	0.67	0.66	0.65	0.68	0.7
Tomcat6.0	0.59	0.42	0.36	0.44	0.32	0.46	0.4	0.46	0.44	0.39	0.36	0.47	0.2	0.3	0.25
Xerces1.3	0.6	0.46	0.38	0.61	0.57	0.67	0.66	0.63	0.61	0.58	0.56	0.64	0.51	0.54	0.59

The median value of G-Mean for all datasets is observed as 0.6. Considering the models for all datasets with 15 different ML techniques, only 14.4% of models achieved G-Mean greater than 0.7. 30.6% of models have G-Mean value less than 0.5.

Sensitivity Analysis:

Models depicted low predictability in terms of sensitivity also. Sensitivity is less than 60% in 93.9% of cases. The median value of sensitivity is only 0.39. This corroborates the low predictive capability of developed models when imbalanced data problem is not handled. Sensitivity values lie between 0 and 0.63. Only 0.6% of cases have sensitivity greater than 70% which is not an acceptable achievement for any prediction model. The maximum sensitivity value obtained in the NS case is 0.7 by IBk, the nearest neighbor

technique, in the Log4j1.1 dataset.

Table 4.5: Sensitivity values of SDP models with imbalanced data (Without Resampling methods)

	NB	LR	SL	LB	MLP	IBk	KStar	ABM1	Bag	ICO	LMT	RT	RSS	PART	J48
Ant1.7	0.5	0.39	0.37	0.55	0.46	0.51	0.42	0.48	0.49	0.58	0.37	0.51	0.47	0.52	0.57
Camel1.6	0.21	0.1	0.1	0.12	0.13	0.34	0.27	0.32	0.22	0.13	0.12	0.35	0.03	0.13	0.19
Ivy2.0	0.43	0.23	0.13	0.2	0.18	0.4	0.28	0.3	0.25	0.2	0.13	0.33	0.05	0.13	0.13
Jedit4.0	0.31	0.4	0.39	0.44	0.41	0.56	0.41	0.45	0.48	0.52	0.4	0.49	0.4	0.35	0.48
Jedit4.2	0.4	0.27	0.23	0.27	0.31	0.38	0.29	0.38	0.31	0.25	0.23	0.29	0.13	0.21	0.23
Log4j1.0	0.53	0.53	0.5	0.53	0.38	0.44	0.26	0.44	0.56	0.44	0.5	0.41	0.35	0.41	0.44
Log4j1.1	0.68	0.65	0.65	0.65	0.65	0.7	0.57	0.62	0.54	0.62	0.65	0.57	0.57	0.59	0.59
Synapse1.0	0.56	0.44	0.19	0.19	0.31	0.31	0.31	0.19	0.19	0.19	0.19	0.25	0	0	0.06
Synapse1.1	0.5	0.47	0.38	0.35	0.47	0.55	0.47	0.48	0.47	0.33	0.38	0.55	0.32	0.43	0.43
Synapse1.2	0.56	0.48	0.44	0.57	0.63	0.55	0.58	0.58	0.58	0.58	0.51	0.55	0.49	0.59	0.6
Tomcat6.0	0.38	0.18	0.13	0.19	0.1	0.23	0.17	0.22	0.19	0.16	0.13	0.23	0.04	0.09	0.06
Xerces1.3	0.39	0.22	0.14	0.39	0.35	0.49	0.45	0.42	0.39	0.35	0.32	0.43	0.26	0.3	0.36

Thus, the overall performance of SDP models developed using ML techniques on imbalanced data is not satisfactory for high-quality predictions.

4.4.3 RQ3a: What is the comparative performance of various SDP models developed using resampling methods?

RQ3b: Is there any improvement in the performance of SDP models developed using ML techniques on the application of resampling methods?

To answer these questions, we exploited performance measures–Sensitivity, G-Mean, Balance, and ROC-AUC values that are calculated with help of a confusion matrix obtained by ten-fold cross-validation trained models developed using resampling methods. The boxplots in Figure 4.3 depicts the predictive capability of ML models on resampled data in terms of ROC-AUC, Balance, G-Mean, and sensitivity.

Values for all these metrics for resampling methods corresponding to each dataset are

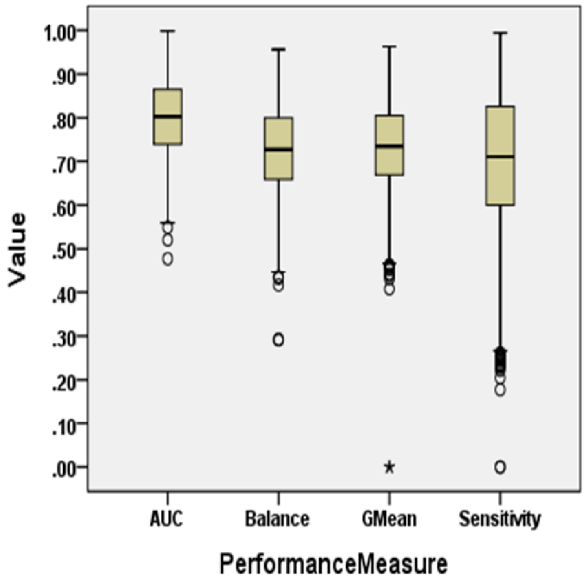

Figure 4.3: Boxplot representation of ML Models' Performance with Resampled Data (SS)

recorded in Appendix A. We developed 1800 resampling based ML models and analyzed their performance based on mean, median, minimum, and maximum values obtained for the cumulative ML techniques of considered performance measures.

4.4.3.1 Comparative Performance of various SDP Models developed using Resampling Methods

The result summary of models developed using resampling methods based on ROC-AUC, G-Mean, Balance, and Sensitivity are abridged in Tables 4.6 4.7, 4.8, and 4.9 . For all the ML techniques, maximum (max), minimum (min), mean, and median values are noted for resampling methods. The maximum value achieved row-wise is boldfaced. NS case is included to provide fair comparison only.

Table 4.6: ROC-AUC Performance of SDP models for imbalanced data (With Resampling methods)

		NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
Ant1.7	Max	0.83	0.85	0.88	0.83	0.96	**0.98**	0.96	0.83	0.79	0.90	0.79
	Min	0.67	0.74	0.74	0.69	0.77	**0.81**	0.79	0.66	0.62	0.78	0.66
	Mean	0.78	0.79	0.83	0.79	0.86	**0.89**	0.86	0.77	0.73	0.86	0.75
	Median	0.80	0.78	0.83	0.81	0.87	**0.88**	0.84	0.8	0.75	0.88	0.77
Camel1.6	Max	0.73	0.83	0.83	0.69	0.95	**0.98**	0.94	0.68	0.72	0.84	0.72
	Min	0.59	0.67	0.68	0.57	**0.69**	0.68	0.68	0.57	0.57	**0.69**	0.57
	Mean	0.68	0.74	0.75	0.64	0.82	**0.84**	0.81	0.64	0.65	0.77	0.65
	Median	0.68	0.73	0.74	0.65	0.85	**0.89**	0.81	0.64	0.65	0.78	0.65
Ivy2.0	Max	0.83	0.93	0.93	0.86	0.97	**0.99**	0.98	0.81	0.77	0.86	0.74
	Min	0.61	0.79	0.82	0.77	0.79	0.84	**0.84**	0.61	0.58	0.61	0.55
	Mean	0.74	0.85	0.86	0.82	0.89	**0.93**	0.91	0.75	0.70	0.80	0.64
	Median	0.73	0.85	0.86	0.83	0.9	**0.93**	0.90	0.76	0.72	0.81	0.63
Jedit4.0	Max	0.79	0.86	0.87	0.79	0.96	**0.98**	0.95	0.78	0.73	0.87	0.74
	Min	0.66	0.69	0.73	0.65	0.74	**0.78**	0.77	0.66	0.59	0.72	0.60
	Mean	0.75	0.77	0.80	0.73	0.87	**0.89**	0.86	0.73	0.66	0.82	0.68
	Median	0.76	0.76	0.78	0.74	**0.88**	0.88	0.84	0.75	0.67	0.83	0.69
Jedit4.2	Max	0.84	0.9	0.9	0.84	0.98	**1.00**	0.99	0.83	0.78	0.89	0.72
	Min	0.59	0.78	0.79	0.75	0.81	0.85	**0.85**	0.61	0.59	0.71	0.60
	Mean	0.78	0.83	0.84	0.80	0.89	**0.93**	0.91	0.75	0.71	0.84	0.66
	Median	0.82	0.83	0.83	0.80	0.88	**0.94**	0.92	0.77	0.73	0.85	0.67
Log4j1.0	Max	0.85	0.83	0.86	0.79	0.96	**0.97**	0.96	0.82	0.89	0.92	0.87
	Min	0.62	0.70	0.75	0.61	0.82	**0.85**	0.80	0.65	0.69	0.81	0.70
	Mean	0.75	0.80	0.82	0.70	0.87	**0.90**	0.86	0.74	0.80	0.87	0.80
	Median	0.76	0.80	0.82	0.71	0.87	**0.90**	0.87	0.73	0.79	0.88	0.83
Log4j1.1	Max	0.86	0.84	0.89	0.84	**0.97**	0.95	0.93	0.84	0.86	0.95	0.86
	Min	0.68	0.73	0.75	0.70	0.76	**0.80**	0.79	0.69	0.70	0.79	0.69
	Mean	0.80	0.78	0.84	0.79	**0.89**	0.87	0.85	0.77	0.79	0.87	0.77
	Median	0.82	0.78	0.84	0.79	**0.92**	0.87	0.84	0.80	0.78	0.87	0.78
Synapse1.0	Max	0.74	0.92	0.92	0.85	0.96	**0.99**	0.97	0.79	0.79	0.88	0.72
	Min	0.52	0.81	0.81	0.76	0.83	**0.88**	0.87	0.59	0.59	0.61	0.48
	Mean	0.66	0.86	0.87	0.82	0.91	**0.94**	0.92	0.71	0.72	0.77	0.61
	Median	0.67	0.86	0.87	0.83	0.92	**0.94**	0.92	0.73	0.73	0.77	0.61
Synapse1.1	Max	0.78	0.83	0.84	0.72	**0.95**	0.95	0.92	0.79	0.72	0.86	0.79
	Min	0.66	0.69	0.73	0.60	0.70	**0.77**	0.75	0.57	0.6	0.69	0.59
	Mean	0.73	0.76	0.77	0.67	0.84	**0.85**	0.83	0.70	0.67	0.80	0.68
	Median	0.75	0.77	0.75	0.68	**0.87**	0.85	0.83	0.72	0.67	0.80	0.67
Synapse1.2	Max	0.82	0.80	0.83	0.79	0.93	**0.94**	0.93	0.79	0.80	0.91	0.83

		NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
	Min	0.67	0.68	0.73	0.63	**0.82**	0.76	0.76	0.64	0.64	0.77	0.68
	Mean	0.76	0.74	0.77	0.73	0.86	0.84	0.82	0.74	0.75	**0.86**	0.78
	Median	0.76	0.75	0.77	0.75	0.85	0.84	0.82	0.76	0.77	**0.87**	0.80
Tomcat6.0	Max	0.81	0.93	0.91	0.85	0.97	**1.00**	0.98	0.80	0.77	0.88	0.71
	Min	0.58	0.78	0.79	0.77	0.77	**0.80**	0.79	0.64	0.63	0.71	0.57
	Mean	0.75	0.85	0.84	0.80	0.89	**0.92**	0.90	0.74	0.70	0.84	0.64
	Median	0.78	0.84	0.84	0.80	0.92	**0.95**	0.90	0.75	0.71	0.86	0.63
Xerces1.3	Max	0.83	0.92	0.92	0.81	0.95	**0.97**	0.97	0.82	0.84	0.87	0.84
	Min	0.60	0.73	0.78	0.73	0.73	0.79	**0.80**	0.66	0.68	0.74	0.65
	Mean	0.76	0.83	0.86	0.77	0.85	**0.90**	0.90	0.75	0.77	0.82	0.75
	Median	0.77	0.86	0.87	0.77	0.86	**0.94**	0.90	0.76	0.79	0.83	0.75

ROC-AUC Analysis:

Investigation of Table 4.6 reveals the fact that 51.4% of models have ROC-AUC greater than 0.8. From Table 4.6, it is visible that ROS performance is the best amongst others. ROS attained the highest mean and median value for Ant1.7, Camel1.6, Ivy2.0, Jedit4.0, Jedit4.2, Log4j1.0, Synapse1.0, Tomcat6.0, and Xerces1.3, It also showed the highest mean value for Synapse1.1. Median values for ROC-AUC for the best resampling method and NS Models are recorded in Figure 4.4.

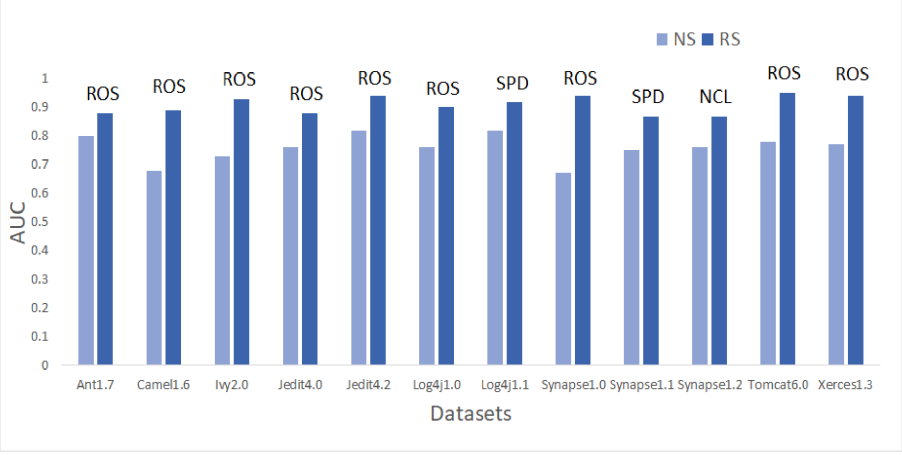

Figure 4.4: Comparison of median ROC-AUC value for NS Models and Models with Resampling Methods

Other resampling methods like AHC, SMT, and SPD also has depicted good performance in terms of ROC-AUC. In undersampling methods, only NCL results can be considered progressive. NCL was able to manage to secure the highest median value for only one dataset, i.e., Synapse1.2. The highest ROC-AUC value of 1 is achieved by Jedit4.2 and Tomcat6.0. The highest median values of Log4j1.1 and Synapse1.1 are depicted by another oversampling technique–SPD. 13.9% of models have ROC-AUC greater than 90%, which is a remarkable improvement.

Balance Analysis:

According to the value of Balance performance measures in Table 4.7. ROS performance seems to outperform the other resampling methods. ADSYN could maximum achieve 86.24 Balance value for Synapse1.0. There were only three resampling methods, ROS, AHC, and SPD, that could achieve the highest Balance value greater than 90. ROS was able to generate a Balance value of 95.58 for Ivy2.0. This remarkable performance of ROS in predicting defects make them good contender of preferable resampling method while developing SDP models.

Table 4.7: Balance Performance of SDP models for imbalanced data (With Resampling methods)

		NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
Ant1.7	Max	69.46	77.18	79.98	75.23	90.68	**90.89**	86.66	75.3	74.99	81.51	71.41
	Min	55.17	59.67	65.92	64.48	59.94	64.46	65.1	64.76	61.68	**71.04**	63.16
	Mean	62.47	71.83	75.71	71.77	76.99	**81.81**	78.4	70.66	68.81	77.85	68.99
	Median	63.27	73.44	75.95	72.91	81.29	**85.75**	79.42	70.5	69.11	78.39	69.87
Camel1.6	Max	52.48	75.81	75.28	64.47	88.87	**89.11**	85.41	61.7	67.37	80.29	66.68
	Min	31.54	44.7	45.65	41.66	**47.94**	46.05	43.61	46.32	46.86	45.29	45.35
	Mean	41.96	67.01	67.96	59.21	72.5	**76.03**	73.34	56.93	59.41	60.92	55.7
	Median	38.65	69.19	70.1	61.41	81.02	**83.32**	78.4	56.47	60.42	63.87	56.39
Ivy2.0	Max	58.88	84.53	84.38	79.84	92.62	**95.58**	92.76	74.25	73.27	78.53	67.14
	Min	32.82	67.68	67.04	**68.46**	56.19	62.58	67.1	58.76	54.29	57.52	48.79
	Mean	44.83	79.95	79.6	77.51	79.01	**87.16**	84.92	65.86	60.95	66.9	55.43
	Median	43.4	81.56	80.7	77.98	86.17	**91.13**	87.93	65.53	60.14	66.35	54.76
Jedit4.0	Max	67.65	77.16	80.05	75.95	90.77	**91.27**	87.72	72.98	69.06	80.21	72.45
	Min	50.67	47.89	54.72	52.16	52.14	54.49	52.2	**57.64**	52.49	55.22	57.08

		NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
	Mean	59.33	70.07	72.25	68.75	80.14	**81.69**	78.31	67.11	59	72.67	63.84
	Median	58.23	71.09	72.82	71.53	86.56	**86.63**	80.92	68.12	58.04	72.78	63.8
Jedit4.2	Max	57.05	81.63	84.33	76.1	93.90	**95.10**	90.66	70.79	72.41	78.7	66.19
	Min	38.12	59.88	**60.33**	55.49	53.31	58.73	58.94	59.68	57.61	55.61	54.99
	Mean	48.8	75.58	77.12	72.52	78.76	**86.56**	84.45	66.95	65.41	67.3	58.72
	Median	48.4	76.27	77.73	73.85	87.6	**91.34**	87.45	67.46	66.42	67.37	57.37
Log4j1.0	Max	68.17	77.31	81.55	71.15	89.36	**91.98**	88.66	76.77	82.54	83.22	80.02
	Min	47.63	63.46	70.26	60.86	67.2	**74.08**	72.48	61.39	63.2	70.13	61.21
	Mean	60.43	72.69	77.11	65.39	80.87	**84.70**	79.99	68.59	72.93	79.1	72.68
	Median	59.99	72.31	77.3	64.4	84.38	**84.70**	80.22	69.15	73.20	80.54	72.33
Log4j1.1	Max	76.55	75.82	80.86	78.40	**88.9**	87.84	85.68	75.42	79.79	84.87	78.01
	Min	66.06	61.77	72.77	68.09	64.68	71.83	69.94	66.15	54.20	**75.04**	64.50
	Mean	71.84	70.08	76.55	72.59	**80.14**	79.85	76.98	71.20	63.45	79.93	70.71
	Median	72.11	69.94	77.07	72.72	**83.14**	81.21	76.7	71.41	61.79	80.86	68.66
Synapse1.0	Max	68.04	86.24	84.46	81.26	93.38	**94.38**	92.14	76.64	81.01	73.13	67.01
	Min	29.2	76.89	77.54	73.17	60.44	81.12	**84.05**	55.39	62.12	29.29	29.06
	Mean	44.99	82.06	81.8	77.69	83.45	**89.84**	88.01	68.92	71.3	59.99	53.95
	Median	42.51	81.48	82.15	78.36	86.92	**90.45**	88.3	70.63	70.83	60.15	54.79
Synapse1.1	Max	65.25	76.12	78.65	69.01	**88.66**	87.90	84.33	77.36	67.50	85.28	69.91
	Min	51.59	64.18	**67.35**	58.83	56.69	66.87	66.45	51.35	55.05	66.85	57.66
	Mean	59.32	71.29	72.12	64.26	77.27	**79.89**	77.61	65.22	61.89	74.01	63.38
	Median	61.39	71.65	71.46	64.47	**82.80**	81.82	77.10	66.19	61.85	72.59	63.57
Synapse1.2	Max	70.13	73.71	76.05	71.99	**85.18**	84.21	83.46	75.05	72.88	84.39	74.37
	Min	59.83	60.63	64.09	61.85	69.23	66.41	63.68	63.90	43.22	**75.50**	61.81
	Mean	66.23	68.64	70.90	68.31	79.82	77.72	75.42	69.33	61.69	**80.89**	68.71
	Median	67.12	68.37	70.68	69.51	**81.34**	80.84	75.53	71.07	62.09	81.33	70.82
Tomcat6.0	Max	55.60	86.08	84.67	78.73	93.61	**94.99**	91.89	68.2	64.78	73.11	58.92
	Min	32.05	58.27	57.18	56.29	51.03	56.58	56.78	**60.47**	53.95	54.08	45.33
	Mean	41.09	78.17	76.99	74.15	75.70	**86.07**	83.59	64.82	60.93	62.14	51.22
	Median	41.17	81.25	77.58	75.54	84.95	**91.66**	88.98	65.27	60.83	62.31	50.97
Xerces1.3	Max	63.66	85.50	86.00	72.60	88.30	**92.79**	92.52	72.60	77.79	74.91	72.53
	Min	39.51	55.21	63.66	57.55	59.32	**68.53**	67.53	55.57	63.67	58.82	59.69
	Mean	54.01	77.10	80.14	68.57	73.99	**85.05**	84.34	68.61	70.66	66.92	66.03
	Median	54.80	79.55	82.53	69.17	74.49	**89.94**	88.48	69.15	71.63	67.03	64.99

Median values of Balance value for the best resampling method with NS scenario are recorded in Figure 4.5.

Comparatively, in undersampling methods, RUS and OSS, had a maximum Balance

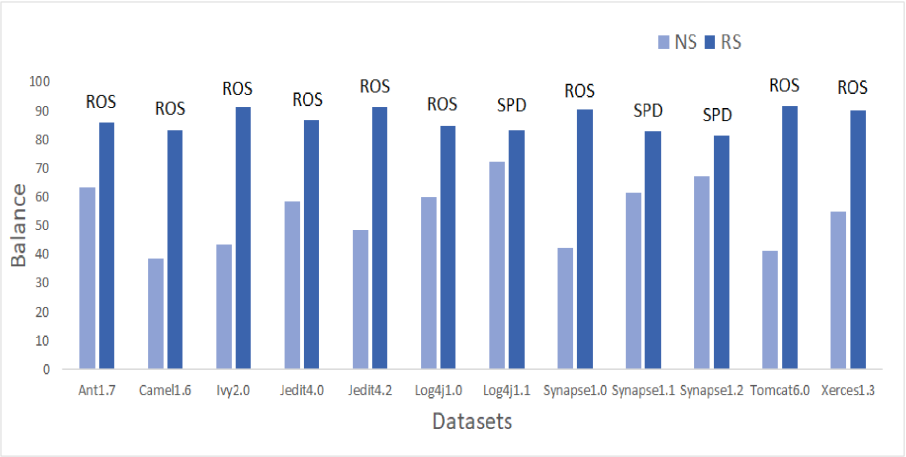

Figure 4.5: Comparison of median Balance value for NS Models and Models with Resampling Methods

value of only 77.36 and 80.02 respectively. The highest median and mean values for all datasets except Synapse1.2 are attained by either SPD or ROS. Synapse 1.2 got the highest mean value with NCL and undersampling method. 60.9% of cases have a Balance greater than 70. Therefore, there is 357% of growth in median values of Balance when resampling is done as compared to NS.

G-Mean Analysis:

Similar patterns are observed in G-Mean also. Bold values in Table 4.8 portrays the better predictive capabilities of ML techniques with ROS. The highest mean and median values are attained by ROS and SPD for all datasets except Synapse1.2. NCL gave the best results for mean and median values of Synapse1.2. 80.7% of cases have G-Mean greater than 0.65 for resampling methods. AHC, though not having any maximum values, have consistent performance for all the datasets. With NS, only 3.9% of cases could achieve a G-Mean value greater than 0.75. With resampling methods, the number of cases for G-Mean values greater than or equal to 0.75 has elevated to 42.7% with 1182% of improvement.

Median values of Balance value for the best resampling method with NS scenario are recorded in Figure 4.6.

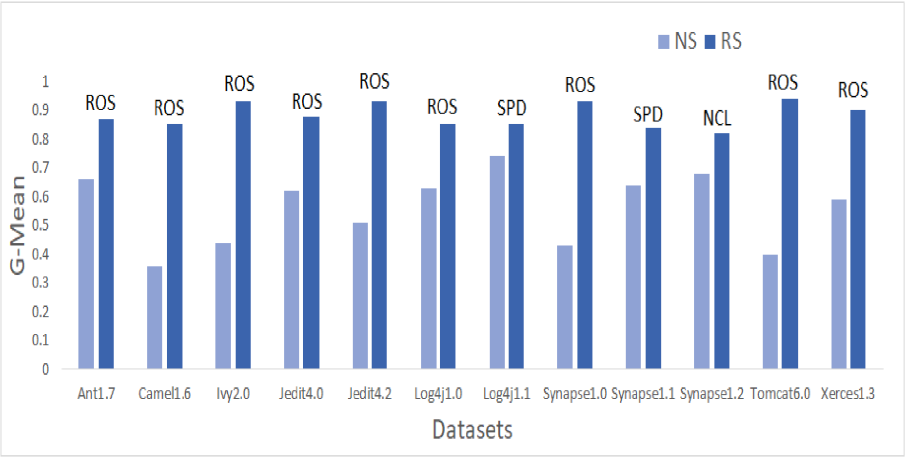

Figure 4.6: Comparison of median G-Mean value for NS Models and Models with Resampling Methods

Table 4.8: G-Mean Performance of SDP models for imbalanced data (With Resampling methods)

		NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
Ant1.7	Max	0.72	0.77	0.80	0.75	0.91	**0.92**	0.88	0.75	0.75	0.83	0.73
	Min	0.59	0.62	0.69	0.67	0.64	0.68	0.68	0.65	0.62	**0.74**	0.64
	Mean	0.66	0.72	0.76	0.72	0.79	**0.83**	0.79	0.71	0.69	0.80	0.69
	Median	0.66	0.74	0.76	0.73	0.83	**0.87**	0.79	0.72	0.69	0.8	0.70
Camel1.6	Max	0.54	0.76	0.75	0.64	0.89	**0.91**	0.86	0.62	0.67	0.81	0.67
	Min	0.18	0.45	0.47	0.41	**0.50**	0.47	0.44	0.48	0.48	0.46	0.47
	Mean	0.40	0.67	0.68	0.59	0.74	**0.77**	0.74	0.57	0.60	0.63	0.57
	Median	0.36	0.70	0.70	0.61	0.81	**0.85**	0.79	0.57	0.60	0.66	0.58
Ivy2.0	Max	0.62	0.85	0.85	0.80	0.93	**0.96**	0.94	0.75	0.73	0.82	0.67
	Min	0.22	0.69	0.69	**0.71**	0.59	0.65	0.70	0.60	0.55	0.62	0.49
	Mean	0.45	0.80	0.80	0.78	0.81	**0.88**	0.86	0.67	0.62	0.71	0.56
	Median	0.44	0.82	0.81	0.79	0.86	**0.93**	0.89	0.67	0.61	0.71	0.56
Jedit4.0	Max	0.70	0.77	0.8	0.76	0.91	**0.93**	0.89	0.74	0.69	0.81	0.73
	Min	0.53	0.49	0.58	0.55	0.55	0.57	0.55	**0.60**	0.54	0.58	0.57
	Mean	0.63	0.71	0.73	0.69	0.81	**0.83**	0.79	0.68	0.60	0.74	0.64
	Median	0.62	0.71	0.73	0.72	0.87	**0.88**	0.81	0.68	0.59	0.74	0.64
Jedit4.2	Max	0.61	0.82	0.85	0.76	0.94	**0.96**	0.92	0.72	0.75	0.81	0.67
	Min	0.35	0.63	**0.63**	0.59	0.57	0.62	0.63	0.63	0.59	0.59	0.56
	Mean	0.51	0.77	0.78	0.73	0.80	**0.88**	0.85	0.68	0.67	0.71	0.60
	Median	0.51	0.78	0.78	0.74	0.88	**0.93**	0.88	0.68	0.68	0.72	0.58
Log4j1.0	Max	0.71	0.77	0.82	0.72	0.90	**0.93**	0.89	0.78	0.83	0.86	0.80

		NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
	Min	0.49	0.67	0.74	0.61	0.71	**0.77**	0.74	0.62	0.63	0.73	0.62
	Mean	0.64	0.73	0.77	0.66	0.82	**0.86**	0.80	0.69	0.73	0.80	0.73
	Median	0.63	0.73	0.77	0.66	0.84	**0.85**	0.80	0.69	0.74	0.82	0.74
Log4j1.1	Max	0.79	0.76	0.81	0.79	**0.91**	0.88	0.86	0.77	0.81	0.85	0.81
	Min	0.67	0.66	0.73	0.68	0.65	0.72	0.70	0.66	0.59	**0.75**	0.65
	Mean	0.74	0.70	0.77	0.73	**0.81**	0.81	0.77	0.72	0.66	0.8	0.71
	Median	0.74	0.70	0.77	0.73	**0.85**	0.81	0.77	0.72	0.65	0.81	0.69
Synapse1.0	Max	0.71	0.86	0.85	0.83	0.93	**0.95**	0.92	0.78	0.81	0.77	0.69
	Min	0	0.77	0.78	0.74	0.64	0.81	**0.84**	0.56	0.64	0.00	0.00
	Mean	0.42	0.82	0.82	0.79	0.85	**0.91**	0.88	0.70	0.72	0.61	0.53
	Median	0.43	0.83	0.83	0.80	0.88	**0.93**	0.89	0.72	0.71	0.65	0.56
Synapse1.1	Max	0.66	0.76	0.79	0.69	**0.90**	0.89	0.85	0.77	0.68	0.85	0.70
	Min	0.54	0.65	0.68	0.59	0.59	**0.68**	0.68	0.51	0.57	0.67	0.58
	Mean	0.62	0.72	0.73	0.65	0.78	**0.80**	0.78	0.65	0.63	0.75	0.64
	Median	0.64	0.72	0.72	0.65	**0.84**	0.83	0.77	0.66	0.63	0.74	0.64
Synapse1.2	Max	0.71	0.74	0.76	0.72	**0.86**	0.85	0.84	0.75	0.73	0.84	0.75
	Min	0.63	0.62	0.66	0.62	0.71	0.68	0.65	0.64	0.43	**0.77**	0.64
	Mean	0.68	0.69	0.71	0.69	0.80	0.78	0.76	0.70	0.63	**0.81**	0.70
	Median	0.68	0.69	0.71	0.70	0.82	0.81	0.76	0.71	0.64	**0.82**	0.71
Tomcat6.0	Max	0.59	0.87	0.85	0.79	0.94	**0.96**	0.93	0.70	0.66	0.77	0.60
	Min	0.20	0.61	0.60	0.59	0.54	0.60	0.60	**0.63**	0.56	0.59	0.46
	Mean	0.39	0.79	0.78	0.75	0.77	**0.87**	0.84	0.67	0.63	0.67	0.53
	Median	0.40	0.82	0.78	0.76	0.85	**0.94**	0.89	0.67	0.63	0.68	0.52
Xerces1.3	Max	0.67	0.86	0.86	0.73	0.89	**0.93**	0.93	0.73	0.78	0.78	0.74
	Min	0.38	0.58	0.67	0.61	0.64	**0.72**	0.71	0.59	0.66	0.63	0.62
	Mean	0.57	0.78	0.81	0.69	0.77	**0.86**	0.85	0.69	0.71	0.71	0.68
	Median	0.59	0.81	0.83	0.69	0.77	**0.90**	0.89	0.70	0.72	0.71	0.67

Comparing the oversampling and undersampling methods, oversampling methods thrived in predicting defects efficiently.

Sensitivity Analysis: Sensitivity values are illustrated in Table 4.9. CNNTL worked best for Log4j1.1 and Synapse1.2 with an average sensitivity value of 0.85 and 0.83 respectively. For other datasets, ROS outperformed other resampling methods. Median values of Balance value for the best resampling method with NS scenario are recorded in Figure **??**.

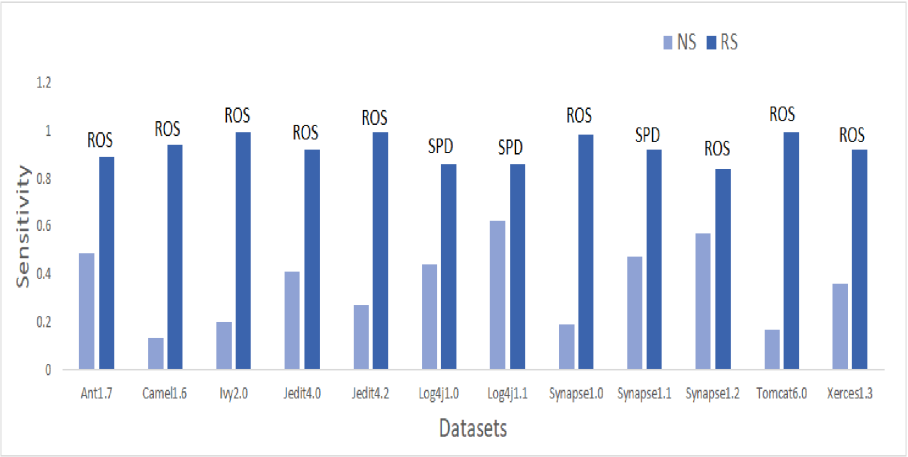

Figure 4.7: Comparison of median Sensitivity value for NS Models and Models with Resampling Methods

Table 4.9: Sensitivity Performance of SDP models for imbalanced data (With Resampling methods)

		NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
Ant1.7	Max	0.58	0.83	0.83	0.78	0.91	**0.97**	0.94	0.72	0.78	0.76	0.73
	Min	0.37	0.44	0.53	0.51	0.44	0.51	0.52	0.56	0.49	**0.60**	0.49
	Mean	0.48	0.72	0.74	0.70	0.71	**0.83**	0.78	0.65	0.72	0.70	0.65
	Median	0.49	0.76	0.75	0.71	0.75	**0.89**	0.80	0.64	0.75	0.70	0.66
Camel1.6	Max	0.35	0.81	0.80	0.66	0.94	**0.98**	0.95	0.59	0.68	0.75	0.62
	Min	0.03	0.22	0.23	0.18	**0.27**	0.24	0.20	0.24	0.26	0.23	0.23
	Mean	0.18	0.70	0.68	0.56	0.69	**0.79**	0.75	0.48	0.55	0.46	0.42
	Median	0.13	0.75	0.74	0.6	0.85	**0.94**	0.81	0.50	0.57	0.51	0.45
Ivy2.0	Max	0.43	0.94	0.88	0.89	0.91	**0.99**	0.98	0.68	0.73	0.70	0.65
	Min	0.05	0.57	0.55	**0.57**	0.39	0.48	0.55	0.45	0.40	0.40	0.30
	Mean	0.22	0.82	0.81	0.79	0.72	**0.90**	0.86	0.57	0.52	0.54	0.42
	Median	0.20	0.84	0.86	0.83	0.83	**0.99**	0.92	0.58	0.48	0.53	0.40
Jedit4.0	Max	0.56	0.85	0.81	0.75	0.95	**0.98**	0.94	0.68	0.87	0.76	0.72
	Min	0.31	0.27	0.37	0.33	0.33	0.36	0.33	0.41	0.37	0.37	**0.43**
	Mean	0.43	0.73	0.70	0.65	0.77	**0.84**	0.79	0.61	0.74	0.65	0.66
	Median	0.41	0.76	0.71	0.66	0.90	**0.92**	0.80	0.63	0.76	0.64	0.68
Jedit4.2	Max	0.40	0.90	0.88	0.86	0.95	**0.99**	0.98	0.67	0.65	0.71	0.58
	Min	0.13	0.45	**0.45**	0.38	0.34	0.42	0.42	0.44	0.42	0.38	0.38
	Mean	0.28	0.81	0.79	0.73	0.73	**0.89**	0.86	0.58	0.56	0.54	0.46
	Median	0.27	0.85	0.83	0.76	0.86	**0.99**	0.91	0.58	0.56	0.54	0.44
Log4j1.0	Max	0.56	0.89	0.82	0.68	0.94	**0.98**	0.95	0.71	0.82	0.81	0.79

		NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
	Min	0.26	0.49	0.59	0.46	0.54	**0.64**	0.62	0.50	0.59	0.59	0.50
	Mean	0.45	0.73	0.75	0.62	0.80	**0.86**	0.79	0.64	0.77	0.73	0.69
	Median	0.44	0.74	0.76	0.64	**0.86**	0.85	0.80	0.65	0.79	0.74	0.68
Log4j1.1	Max	0.70	0.78	0.81	0.77	**0.99**	0.91	0.90	0.76	0.97	0.84	0.76
	Min	0.54	0.46	0.64	0.62	0.55	0.62	0.67	0.59	**0.70**	**0.70**	0.62
	Mean	0.62	0.69	0.73	0.68	0.82	0.79	0.75	0.66	**0.85**	0.77	0.71
	Median	0.62	0.70	0.72	0.68	**0.86**	0.78	0.74	0.65	0.84	0.76	0.70
Synapse1.0	Max	0.56	0.91	0.89	0.93	0.96	**0.99**	0.98	0.69	0.81	0.63	0.56
	Min	0.00	0.75	0.74	0.70	0.44	0.81	**0.85**	0.44	0.50	0.00	0.00
	Mean	0.23	0.86	0.85	0.83	0.79	**0.95**	0.91	0.60	0.66	0.44	0.38
	Median	0.19	0.88	0.87	0.81	0.82	**0.98**	0.91	0.63	0.69	0.44	0.38
Synapse1.1	Max	0.55	0.83	0.84	0.65	**0.95**	0.94	0.89	0.75	0.87	0.85	0.75
	Min	0.32	0.57	0.60	0.51	0.41	0.57	0.57	0.52	**0.63**	0.58	0.57
	Mean	0.44	0.72	0.69	0.59	0.79	**0.83**	0.77	0.62	0.76	0.68	0.62
	Median	0.47	0.74	0.67	0.60	**0.92**	0.88	0.77	0.62	0.75	0.67	0.62
Synapse1.2	Max	0.63	0.83	0.78	0.7	0.91	0.91	0.9	0.71	**0.93**	0.86	0.86
	Min	0.44	0.47	0.52	0.54	0.59	0.56	0.52	0.53	**0.69**	0.67	0.66
	Mean	0.55	0.69	0.69	0.64	0.79	0.79	0.76	0.64	**0.83**	0.80	0.79
	Median	0.57	0.70	0.73	0.65	0.80	**0.84**	0.77	0.66	0.83	0.80	0.79
Tomcat6.0	Max	0.38	0.91	0.89	0.87	0.94	**0.99**	0.98	0.58	0.57	0.62	0.45
	Min	0.04	0.42	0.40	0.39	0.31	0.39	0.40	**0.45**	0.36	0.35	0.23
	Mean	0.17	0.79	0.77	0.74	0.68	**0.88**	0.84	0.53	0.48	0.47	0.33
	Median	0.17	0.87	0.83	0.77	0.82	**0.99**	0.91	0.53	0.48	0.47	0.32
Xerces1.3	Max	0.49	0.91	0.86	0.79	0.86	0.94	**0.95**	0.71	0.77	0.65	0.67
	Min	0.14	0.37	0.49	0.41	0.43	**0.56**	0.55	0.38	0.49	0.42	0.43
	Mean	0.35	0.79	0.77	0.66	0.64	**0.84**	0.83	0.61	0.65	0.54	0.55
	Median	0.36	0.86	0.81	0.69	0.65	**0.92**	0.90	0.62	0.65	0.54	0.54

After applying resampling methods, sensitivity increases above 0.9 for 11.9% of the developed models. 53% of resampling-based models have sensitivity greater than 0.7 as compared to only 0.6% of cases in the NS scenario. This accounts for a whopping 9440% of improvement in ML models having a sensitivity value greater than 0.7. Therefore, it can be concluded that the predictive capability of ML models has immensely improved after treating imbalanced data properly using resampling methods.

4.4.3.2 Comparison of Resampling-based ML Models with NoSampling (NS) Models

A comparison of the results of models developed without employing resampling methods and models developed using resampling methods provides an answer to RQ3b. 1980 ML models were constructed for 12 datasets to achieve objectives of this chapter. The results are compared based on the maximum value attained and averaged median value achieved in each dataset. These results are summarized in Table 4.10 and Table 4.11. The last column 'All' in both the tables epitomizes the maximum and median values attained with cumulative datasets. 'NS' represents no sampling scenario and 'RS' signifies the ten resampling methods that are used in this study. 'GR' indicates the percentage growth in maximum and median values of RS as compared to NS. Analysis of Table 4.10 and Table 4.11 respectively shows that there is a positive increment in all the values of maximum or median for the four performance measures when resampling methods are employed to build SDP models. This proves that there is a definite improvement in resampling-based ML models than the NS models when evaluated based on ROC-AUC, Balance, G-Mean, and sensitivity.

For ROC-AUC, the overall percentage growth for the median value is 6.3%. The maximum ROC-AUC value achieved in the NS case is 0.86 for Log4j1.1 which increases to 0.97 when resampling methods are applied to it. Jedit4.2 and Tomcat6.0 were able to attain a maximum ROC-AUC value of 1 showing 18.2% and 22.2% of the increase. For Synapse1.0, on the application of resampling methods, the maximum value depicts the incremental growth of 33.8% and the respective median growth corresponds to 24.3%. The increase in median values of Balance and G-Mean for all the datasets is 29.1% and 21.2% respectively with resampling methods. Synapse1.0, Tomcat6.0, Ivy2.0, and Camel1.6 has illustrated more than 60% of improvement in Balance median values and more than 70% improvement in G-Mean median values. The maximum Balance value gained by models with resampling methods is 95.58 for Ivy2.0 which was earlier 58.88. Similarly, the sensi-

tivity median values have shown a remarkable improvement of 84.6%. The median value of NS was 0.39 when all datasets were considered together. This value was raised to 0.71 on the application of resampling methods.

Answer to RQ3b: The results verify that there is an improvement in the performance of SDP models developed using ML techniques on the application of resampling methods.

Table 4.10: Comparison of Maximum of NoSampling (NS) and Resampling-based ML models (RS) for ROC-AUC, Balance, G-Mean, and Sensitivity

		D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12	All
ROC-AUC Max	NS	0.83	0.73	0.83	0.79	0.84	0.85	0.86	0.74	0.78	0.82	0.81	0.83	0.86
	RS	0.98	0.98	0.99	0.98	1	0.97	0.97	0.99	0.95	0.94	1	0.97	1
	GR	18.5	34.3	19	23.8	18.2	14.3	12.6	33.8	22.8	15.1	22.2	17.3	16
Balance Max	NS	69.46	52.48	58.88	67.65	57.05	68.17	76.55	68.04	65.25	70.13	55.6	63.66	76.55
	RS	90.89	89.11	95.58	91.27	95.1	91.98	88.9	94.38	88.66	85.18	94.99	92.79	95.58
	GR	30.9	69.8	62.3	34.9	66.7	34.9	16.1	38.7	35.9	21.5	70.8	45.8	24.9
G-Mean Max	NS	0.72	0.54	0.62	0.7	0.61	0.71	0.79	0.71	0.66	0.71	0.59	0.67	0.79
	RS	0.92	0.91	0.96	0.93	0.96	0.93	0.91	0.95	0.9	0.86	0.96	0.93	0.96
	GR	27	69.7	54.4	32.4	57.9	30.2	14.8	34.9	34.9	20.4	62.7	38.1	21.2
Sensitivity Max	NS	0.58	0.35	0.43	0.56	0.4	0.56	0.7	0.56	0.55	0.63	0.38	0.49	0.7
	RS	0.97	0.98	0.99	0.98	0.99	0.98	0.99	0.99	0.95	0.93	0.99	0.95	0.99
	GR	68.4	180.5	132.8	74.4	150.9	75.1	40.2	76.4	73.3	48.1	164	92	41.5

Table 4.11: Comparison of Averaged Median values of NoSampling (NS) and Resampling-based ML models (RS) for ROC-AUC, Balance, G-Mean, and Sensitivity

		D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12	All
ROC-AUC Median	NS	0.8	0.68	0.73	0.76	0.82	0.76	0.82	0.67	0.75	0.76	0.78	0.77	0.76
	RS	0.81	0.71	0.83	0.77	0.83	0.83	0.82	0.83	0.75	0.79	0.81	0.81	0.8
	GR	1.9	3.8	14.3	1.9	1.3	8.7	0.6	24.3	0.7	3.3	4.7	5.8	6.3
Balance Median	NS	63.27	38.65	43.4	58.23	48.4	59.99	72.11	42.51	61.39	67.12	41.17	54.8	56.35
	RS	73.66	62.34	75.8	71.98	72.37	75.56	73.7	79.4	69.41	71.93	73.15	72.29	72.73
	GR	16.4	61.3	74.7	23.6	49.5	26	2.2	86.8	13.1	7.2	77.7	31.9	29.1
G-Mean Median	NS	0.66	0.36	0.44	0.62	0.51	0.63	0.74	0.43	0.64	0.68	0.4	0.59	0.6
	RS	0.74	0.62	0.77	0.73	0.74	0.76	0.75	0.8	0.7	0.72	0.74	0.73	0.73
	GR	13.1	73.7	73.5	16.9	44.7	20.1	0.4	88.3	9	5.5	83.5	24.2	22
Sensitivity Median	NS	0.49	0.13	0.2	0.41	0.27	0.44	0.62	0.19	0.47	0.57	0.17	0.36	0.39
	RS	0.71	0.59	0.7	0.71	0.69	0.74	0.74	0.81	0.68	0.76	0.66	0.68	0.71
	GR	45.7	341.9	250	72.2	154.8	66.7	18.9	330	46.5	32.8	293.3	88.8	84.6

4.4.4 RQ4: Which resampling method outperforms the addressed undersampling and oversampling methods for building an efficient SDP model?

This RQ addresses the effectiveness of 10 resampling methods that are investigated in this study for constructing good SDP models. For the experiments conducted, is there any particular resampling method that can be considered the best? In this direction, we conducted Friedman tests on performance evaluators to provide rankings to models built using resampling methods. The case when no resampling method was used (NS) is also included. The Friedman test is used to find the difference between methods statistically. Four hypotheses are formed for four different performance measures. The hypothesis formed to achieve this objective is stated as:

- H_{i0} (Null Hypothesis): There is no significant statistical difference between the performance of any of the defect prediction models developed after using resampling methods and models developed using original data, in terms of PM_j.
- H_{ia} (Alternate Hypothesis): There is a significant statistical difference between the performance of any of the defect prediction models developed after using resampling methods and model developed using original data, in terms of PM_j.

where i = 1 to 4 denoting H_1, H_2, H_3 and H_4 hypothesis and j = 1 to 4. PM_1 = ROC-AUC, PM_2 = Balance, PM_3 = G-Mean, and PM_4 = Sensitivity. Table 4.12 provides the desired ranking of SDP models developed in this study for ROC-AUC, Balance, G-Mean, and Sensitivity. The mean ranks of each resampling method and NS are shown in parentheses. NS represents the scenario when no sampling technique is used, so, it represents the performance with original data. We evaluated the hypothesis at the 0.05 level of significance, i.e., 95% of confidence level. Rank 1 is the best rank and rank 11

is the worst rank. The p-values achieved for all performance measures is 0.000. As p-values are less than 0.05, we reject the null hypothesis and declare that there is a significant difference between methods applied to developed SDP models.

Table 4.12 shows that ROS and AHC have unanimously scored Rank 1 and Rank 2 respectively for all the performance measures. Out of four undersampling methods, only one, i.e., NCL can make its space in the first seven positions for the reliable performance measures–ROC-AUC, Balance, and G-Mean. OSS, CNNTL, and RUS have acquired positions in the last four ranks with ROC-AUC, Balance, and G-Mean. These rankings clearly state the supremacy of oversampling methods over undersampling methods. For Balance, G-Mean, and Sensitivity, NS (no sampling scenario) is ranked last. Therefore, results statistically approved the visualization in RQ3 that usage of resampling methods improved the predictive power of SDP models.

Table 4.12: Friedman Rankings for NS and Resampling methods with ROC-AUC, Balance, G-Mean, and Sensitivity

Rank	ROC-AUC	Balance	G-Mean	Sensitivity
Rank 1	**ROS (10.11)**	**ROS (10.08)**	**ROS (10.15)**	**ROS (9.89)**
Rank 2	AHC (9.13)	AHC (9.01)	AHC (9.01)	AHC (8.83)
Rank 3	SPD (8.58)	SPD (7.93)	SPD (7.91)	SMT (7.06)
Rank 4	NCL (8.04)	SMT (7.56)	SMT (7.46)	SPD (7.06)
Rank 5	SMT (7.06)	NCL (6.60)	NCL (6.95)	ADSYN (7.01)
Rank 6	ADSYN (5.55)	ADSYN (6.23)	ADSYN (6.11)	CNNTL (6.48)
Rank 7	NS (4.30)	SLSMT (5.09)	SLSMT (4.88)	NCL (5.29)
Rank 8	SLSMT (4.05)	RUS (4.42)	RUS (4.36)	SLSMT (5.20)
Rank 9	RUS (3.37)	CNNTL (3.85)	CNNTL (3.66)	RUS (4.01)
Rank 10	CNNTL (3.08)	OSS (3.41)	OSS (3.30)	OSS (3.90)
Rank 11	OSS (2.68)	NS (1.77)	NS (2.16)	NS (1.22)
p-value	**0.000**	**0.000**	**0.000**	**0.000**

The Friedman test tells whether there is an overall difference or not in the model performances. But if there is a difference, it fails to further identify the pairwise difference, i.e, exactly which group is different from each other. For this, post-hoc analysis is conducted

on overall datasets using Wilcoxon signed-rank test and comparative pairwise performance of all the resampling methods was evaluated with ROS. The Wilcoxon signed-rank test was carried out with bonferroni correction at the α = 0.05 level of significance. total number of comparisons are 10. Therefore, BonC value becomes 0.005. Table 4.13 summarizes the Wilcoxon signed-rank test results with Bonferroni Correction for ROC-AUC, G-Mean, Balance, and Sensitivity. 'S+' represents 'significantly better' results and their p-values are scribed in parenthesis. The p-value obtained for all the comparisons is 0.000. It is less than 0.005, there we reject the null hypothesis.

Table 4.13: Wilcoxon Signed-Rank Test Results for Resampling Methods based on ROS

Pair	ROC-AUC	Balance	G-Mean	Sensitivity
ROS - NS	**S+ (0.000)**	**S+ (0.000)**	**S+ (0.000)**	**S+ (0.000)**
ROS - ADSYN	**S+ (0.000)**	**S+ (0.000)**	**S+ (0.000)**	**S+ (0.000)**
ROS - SMT	**S+ (0.000)**	**S+ (0.000)**	**S+ (0.000)**	**S+ (0.000)**
ROS - SLSMT	**S+ (0.000)**	**S+ (0.000)**	**S+ (0.000)**	**S+ (0.000)**
ROS - SPD	**S+ (0.000)**	**S+ (0.000)**	**S+ (0.000)**	**S+ (0.000)**
ROS - AHC	**S+ (0.000)**	**S+ (0.000)**	**S+ (0.000)**	**S+ (0.000)**
ROS - RUS	**S+ (0.000)**	**S+ (0.000)**	**S+ (0.000)**	**S+ (0.000)**
ROS - CNNTL	**S+ (0.000)**	**S+ (0.000)**	**S+ (0.000)**	**S+ (0.000)**
ROS - NCL	**S+ (0.000)**	**S+ (0.000)**	**S+ (0.000)**	**S+ (0.000)**
ROS - OSS	**S+ (0.000)**	**S+ (0.000)**	**S+ (0.000)**	**S+ (0.000)**

It can be inferred from Table 4.13 that ROS exhibits statistically better performance

than all the compared scenarios based on ROC-AUC, Balance, G-Mean, and sensitivity.

Answer to RQ4: Oversampling methods resulted in better SDP models than undersampling methods. Random OverSampling methods emerged as the best resampling method in terms of ROC-AUC, Balance, G-Mean, and sensitivity.

4.4.5 RQ5: Which ML technique performs the best for SDP in imbalanced data?

To answer this RQ, we performed the Friedman test for 15 ML techniques by considering ROS-based models. The Friedman rankings are recorded in Table 4.14 with Rank 1 as the best rank. The Friedman test was held at the 0.05 significance level with a degree of freedom of 14. The p-value for each performance measures was 0.000. Therefore, the results are considered 95% significant. 'Kendall C' in Table 4.14 shows the value for Kendall's coefficient of concordance. Its value ranges from 0 to 1. It reflects the degree of agreement and for all the four performance measures, Kendall's coefficient is greater than or equal to 0.835. This signifies that rankings for different datasets are approximate 83.5% similar and hence, increases the reliability and credibility of the Friedman statistical results. This can be observed from Table 4.14 that KStar, IBk, ABM1, RT, and RSS techniques incited better prediction models than other ML techniques. KStar and IBk are the variants of the nearest neighbor techniques. These ML techniques provide good results when there is a large number of samples irrespective of the data distribution. Nearest neighbors are instance-based fast learners. Mean ranks are inscribed in brackets. The mean rank of KStar is 14.41 for ROC-AUC which is the highest in the pool. The rank of IBk is second when the models are evaluated based on Balance, G-Mean, and sensitivity. The mean rank of IBk is 12.66, 12.91, and 12.33 for Balance, G-Mean, and sensitivity respectively.

ABM1, RT, Bag, and RSS have also attained high mean ranks. ABM1 is the first ranker in Balance and G-Mean with a mean rank of 13.41 and 13.5 respectively. ABM1 got 2nd

rank with ROC-AUC and 4th rank with sensitivity. For the stable performance measures, ABM1, Bag, RT, and RSS appeared in the first six ranks, proving their competency in the ML world. These four techniques are ensembles that are considered robust in dealing with imbalanced data. The crux of ensemble methods is to cover the weakness of the base ML technique and combine them to reduce bias or variance and enhance its predictive capability.

The last three ranks for all the performance measures are grabbed by the statistical learners: Naïve Bayes, Simple Logistic, and Logistic Regression. These ML techniques, in contrast, were able to generate good prediction models when no resampling method was involved in model construction. The balancing of both the classes boosted the performance of other classifiers, especially ensembles and nearest neighbors, and resulted in their unacceptable performance.

Table 4.14: Friedman Rankings for ML techniques with ROC-AUC, Balance, G-Mean, and Sensitivity

Rank	ROC-AUC	Balance	G-Mean	Sensitivity
Rank 1	**KStar (14.41)**	**ABM1 (13.41)**	**ABM1 (13.50)**	**RT (12.62)**
Rank 2	ABM1 (13.75)	IBk (12.66)	IBk (12.91)	IBk (12.54)
Rank 3	Bag (13.37)	RT (12.16)	RT (12.58)	KStar (12.33)
Rank 4	RSS (12.29)	Bag (12.12)	Bag (11.75)	ABM1 (12.20)
Rank 5	IBk (9.83)	KStar (11.08)	KStar (11.25)	LMT (10.95)
Rank 6	RT (9.20)	RSS (10.33)	RSS (9.58)	PART (9.95)
Rank 7	LMT (8.58)	PART (9.04)	PART (9.04)	J48 (9.83)
Rank 8	J48 (7.62)	LMT (9.00)	LMT (9.00)	Bag (9.54)
Rank 9	PART (7.58)	J48 (8.58)	J48 (8.70)	RSS (8.75)
Rank 10	LB (6.08)	MLP (5.45)	MLP (5.75)	LB (5.25)
Rank 11	ICO (5.54)	ICO (4.87)	ICO (4.87)	MLP (4.91)
Rank 12	MLP (4.75)	LB (4.83)	LB (4.7)	ICO (4.83)
Rank 13	LR (2.79)	SL (2.75)	SL (2.66)	SL (2.66)
Rank 14	SL (2.5)	LR (2.41)	LR (2.33)	LR (2.41)
Rank 15	NB (1.66)	NB (1.25)	NB (1.33)	NB (1.16)
p-value	**0.000**	**0.000**	**0.000**	**0.000**
Kendall C	**0.876**	**0.835**	**0.844**	**0.838**

Answer to RQ5: Ensemble methods and nearest neighbors performed the best for SDP in imbalanced data with Random OverSampling method.

4.5 Discussion

This study evaluates the effect of resampling methods on various ML models for defect prediction using Apache software. In total 1980 models were built and the performances of models were empirically compared using stable performance measures (G-Mean, Balance, and ROC-AUC).

In this chapter, we first find the proportion selection of metrics related to IQAs and established the coherence of CFS-selected OO metrics with defect proneness. Coupling and cohesion metrics are concluded to be the most related to defect prediction. With CFS, LCOM is selected in 91.67% of the datasets while Ca and RFC are selected in 75% of the datasets. Software practitioners are, therefore, advised to design software classes such that they exhibit high cohesion and low coupling.

The results of the chapter show that the performance of ML techniques significantly improved after incorporating resampling methods. This study reinsures that SDP models developed with resampling methods enhance their predictive capability as compared to models developed without resampling methods. The balancing of datasets resulted in following observations:

- Median values of sensitivity have improved by 45.7%, 341.9%, 250%, 72.2%, 154.8%, 66.7%, 18.9%, 330%, 46.5%, 32.8%,293.3%, and 88.8% for Ant1.7, Camel1.6, Ivy2.0, Jedit4.0, Jedit4.2, Log4j1.0, Log4j1.1, Synapse1.0, Synapse1.1, Synapse1.2, Tomcat6.0, and Xerces1.3 respectively after applying resampling methods.

- Median values of Balance have improved by 16.4%, 61.3%, 74.7%, 23.6%, 49.5%, 26%, 2.2%, 86.8%, 13.1%, 7.2%,77.7%, and 31.9% for Ant1.7, Camel1.6, Ivy2.0,

Jedit4.0, Jedit4.2, Log4j1.0, Log4j1.1, Synapse1.0, Synapse1.1, Synapse1.2, Tomcat6.0, and Xerces1.3 respectively after applying resampling methods.

- Median values of G-Mean have improved by 13.1%, 73.7%, 73.5%, 16.9%, 44.7%, 20.1%, 0.4%, 88.3%, 9%, 5.5%, 83.5%, and 24.2% for Ant1.7, Camel1.6, Ivy2.0, Jedit4.0, Jedit4.2, Log4j1.0, Log4j1.1, Synapse1.0, Synapse1.1, Synapse1.2, Tomcat6.0, and Xerces1.3 respectively after applying resampling methods.

- Median values of ROC-AUC have improved by 1.9%, 3.8%, 14.3.7%, 1.9%, 1.3%, 8.7%, 0.6%, 24.3%, 0.7%, 3.3%, 4.7%, and 5.8% for Ant1.7, Camel1.6, Ivy2.0, Jedit4.0, Jedit4.2, Log4j1.0, Log4j1.1, Synapse1.0, Synapse1.1, Synapse1.2, Tomcat6.0, and Xerces1.3 respectively after applying resampling methods.

The chapter ascertains that the ML models developed with ROS method exhibit the better prediction capability than other resampling methods. The use of statistical tests reinforces the correctness of results. The results of the Friedman test strongly advocate the use of the ROS method for the improved predictive capability of SDP classifiers for imbalanced data. Apart from ROS, AHC and SMT also demonstrated good predictive capability for uncovering defects. Wilcoxon signed-rank test with Bonferroni correction eradicates family-wise error and concluded ROS to be significantly and statistically better than other resampling methods.

Models developed using oversampling methods illustrated better defect prediction capability than that of undersampling methods. Handling imbalanced data problem using ROS will aid developers and software practitioners in detecting defects effectively in the early stages of software development reducing testing cost and effort. The pair-wise comparison of the performance of ROS with other resampling methods used in this chapter indicates that the performance of ROS was statistically better than all other resampling methods.

In ML techniques ABM1, Bag, RT, and RSS secured ranks in first six positions. The rest of the ML techniques seem to have comparable performance. We conclude that the nearest neighbors and ensemble methods improvise the predictive capability of SDP models.

Next, we would like to design the hybrid framework and investigate the impact of resampling methods with ensemble techniques for predicting defects in OO software.

Chapter 5

Tackling Class Imbalance Problem using Ensemble Methods

5.1 Introduction

With the emergence of the information age and ever-growing software providing good quality, software is always a demanding task for software developers. As mentioned earlier, early detection of the possibility of defective classes assists in proper resource allocation ensuing in optimizing resource utilization and increasing user satisfaction and software reliability [1]. However, the nature of data in software projects gives rise to one of the crucial problems in the SDP of imbalanced data [28]. When any machine learning-based model will be built on such kind of data, the model will learn on the majority classes and tend to predict the majority class more often, considering the minority classes as noise. Such defect prediction models may be of less practical usage as defective classes cannot be predicted correctly. The results of the previous chapter confirmed that the employment of resampling methods proved to be useful in reducing the effect of imbalanced data and improved the performance of the models developed using ML techniques significantly. The previous

chapter also concluded with the better predictive capability of tree-based ensemble methods for SDP.

Ensemble methods combine and consolidate the results of multiple base learners resulting in optimal predictions [33]. If base learners are implemented in parallel, the output predictions of various base learners are averaged. If base learners are implemented in sequence, learnings from one base classifier are used in training of next classifier. In parallel implementation, the variance is improved whereas in sequential implementation bias is reduced. Ensemble methods are used by approximate 40% of related studies according to the survey conducted in Chapter 2. Their predictions can be enhanced by involving resampling methods in it. The solution to the imbalanced classification problem may lie in the merger of resampling methods and ensemble methods.

Working in this direction, this study explored four classic ensemble methods, five boosting-based ensemble methods, and nine bagging-based ensemble methods and tried to find answers to the following research questions with the help of OO metrics of 15 open-source JAVA projects:

- RQ1a: Do resampling-based boosting ensemble methods perform better than classic boosting ensemble methods in case of imbalanced data for predicting software defects?
- RQ1b: Which resampling-based boosting ensemble method is the best classifier for predicting software defects using imbalanced data?
- RQ2a: Do resampling-based bagging ensemble methods perform better than classic bagging ensemble methods in case of imbalanced data?
- RQ2b: Which resampling-based bagging ensemble method is the best classifier for predicting software defects using imbalanced data?

- RQ3: Which amongst addressed ensemble methods is the best for predicting software defects in an imbalanced data domain?

We provided an extensive statistically validated evaluation of 17 ensemble methods with fifteen real open-source JAVA projects. The datasets used in this study are derived from the software industry. As researchers tend to design hybridization of resampling methods with ensemble methods, we extended the thought to present a single platform for performing their empirical comparison with datasets other than NASA datasets so that results can be generalized and be useful to the software industry. The SDP models are constructed using ten-fold cross-validation with the C4.5 decision tree as a base classifier and their predictive capabilities are measured with stable performance metrics G-Mean, Balance, and Area under Curve (ROC-AUC). Only a few studies use statistical tests for result validation which is an important part of any empirical study [1]. The Friedman test with Wilcoxon post-hoc analysis is carried out to statistically validate the developed models. The results show that the application of bagging-based ensemble methods and boosting-based ensemble methods with resampling methods do alleviate the imbalanced classification problem in the SDP domain. Both the variations worked better than the classic ensemble models because resampling methods increased the percentages of defective classes in a particular software leading to better learning of the model. The resampling-based bagging ensemble methods are the preferred ones for building SDP models.

The rest of the paper organization is as follows: The experimental research framework is elaborated in Section 5.2. Section 5.3 highlights the research methodology. The performances of designed models are presented and analyzed in Section 5.4. Section 5.5 discusses the key findings of this chapter.

The part of this work is published in [206] and results are communicated in [207].

5.2 Experimental Research Framework

5.2.1 Dependent and Independent Variables of the Study

SDP models are developed based on datasets containing independent variables and dependent variables. Independent variables represent the internal characteristics of the considered software and based on these independent variables, conclusions are drawn for the dependent variable. The research carried out in this chapter used 20 OO metrics as independent variables and defect as the dependent variable. A detailed description of these research variables is available in Section 3.5.

5.2.2 Dataset Collection and Preprocessing

Datasets belong to 15 open-source JAVA projects that are downloaded from the Promise repository [145, 208]. 15 datasets include Apache projects- three versions of Synapse, four versions of Jedit, two versions of Camel and Ivy each, one ant version, one Tomcat version, one Log4j version, and one Xalan version. Dataset details are provided in Section 3.6. Instead of NASA datasets that are widely used, we preferred industrial projects so that developers can utilize the findings of this empirical analysis with confidence while making new software or versions of the software. Considering the importance of feature selection for better prediction results, CFS was conducted on all datasets and CFS selected features for used datasets can be referred to in subsection 3.7.2. CFS helps in recognizing the relevant set of features to develop SDP models [146]. Out of 15 datasets, WMC and RFC are selected in 10 datasets, and LOC by 8 datasets. 7 datasets found CBO, MOA, and Ca as relevant features. The selection of pertinent metrics results in developing better defect prediction models. Based on these results, it is reasonable to claim that developers can plan and execute testing by focusing resources on the defect-prone parts of the design

and code.

5.2.3 Experimental Design

This section describes the framework designed for achieving the objectives of this study. The complete framework is illustrated in Figure 5.1.

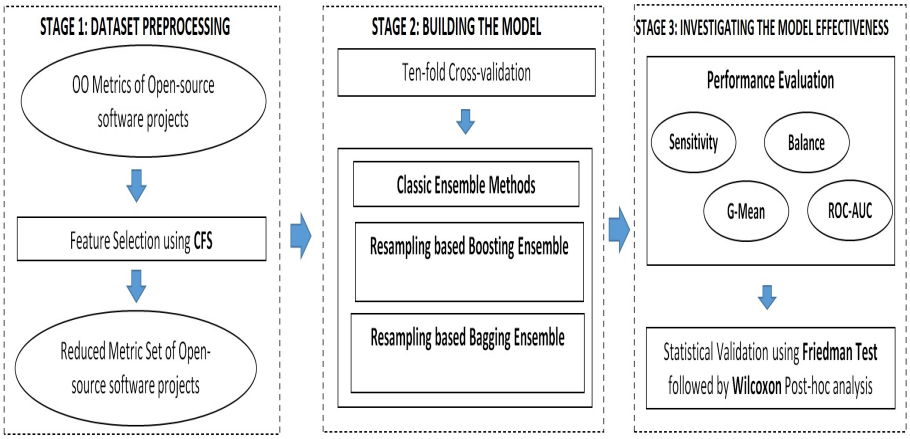

Figure 5.1: Experimental Framework of the Study

Experiments are planned to be conducted in three stages:

- *Stage 1*: Datasets are collected and preprocessed as explained earlier.
- *Stage 2*: Models are validated using ten-fold cross-validation and developed using ensemble methods and their hybrids. Ten-fold cross-validation method is explained in subsection 3.10.2. The models are constructed in the KEEL environment [205] with default parameter settings. Default parameter settings are used to assist researchers to reproduce the study.
- *Stage 3*: Model effectiveness is investigating employing stable performance measures and statistical tests. Performance measures used are Sensitivity, G-Mean, Bal-

ance, and ROC-AUC. Friedman test and Wilcoxon signed-rank test used for statistically validating the results.

External validity is supported by the use of open-source datasets and publicly available tools. Default experimental settings are used and this enhances the external validity of the study as the study can be replicated and can be applied in similar situations.

5.2.4 Performance Measures

In this study, by convention positives are the number of classes that have any defect(s) and negatives are the number of classes that do not have any defect. Appropriate performance measures should be opted to evaluate and assess the developed models. How can the model result in a good performance when it is trained on data that has a disproportionate ratio of outcome class values? Inefficient training is responsible for the inaccurate model resulting in insignificant predictions. Therefore, we used Sensitivity, G-Mean, Balance, and ROC-AUC. Section 3.11 explains the significance and derivations of these measures. Internal validation threat is minimized by incorporating stable performance evaluators like G-Mean, Balance, and ROC-AUC.

5.2.5 Statistical Validation

In this research work, Friedman test [197] is conducted followed by post-hoc analysis using the Wilcoxon signed-rank test [209] to statistically validate the results. These tests are summarized earlier in Section 3.12. Friedman test provides mean ranking of ensemble methods and their hybrid versions used in this study and the Wilcoxon signed-rank test is then used to calculate the pair-wise significant difference between the performance of the best Friedman ranker and other ensemble methods based on the respective performance measure. The results are discussed in the next section.

5.3 Research Methodology

This study exploits 17 ensemble methods taking the C4.5 decision tree [162] as the base classifier. Many researchers have used C4.5 as a machine learner for defect prediction. C4.5 is chosen because it is identified as one of the top ten data mining algorithms [210] and has been concluded as a good learner for imbalanced data [93]. Addressed ensemble methods include four classic ensemble methods (AB, ABNC, ABM1, Bag), and 13 hybrid ensemble methods. Hybrid methods comprise four resampling-based boosting methods and nine resampling-based bagging methods. Subsection 3.8.3 briefly elucidates all these ensemble methods.

5.4 Performance Analysis and Evaluation

This study focuses on finding the importance of ensemble methods for developing effective machine learning models with imbalanced data. Therefore, it is needed to inspect the predictive capability of these models by using various performance measures that are derived from the confusion matrix obtained for a particular classifier. As discussed, applying a statistical test is an important step that every researcher should focus on while validating the results.

5.4.1 Performance of Boosting-based Ensemble methods

This part of the chapter answers the RQ1 and explores the procured results.

5.4.1.1 RQ1a: Do resampling-based boosting ensemble methods perform better than classic boosting ensemble methods in case of imbalanced data for predicting software defects?

Different SDP models based on classic boosting ensembles and resampling-based boosting ensemble methods are evaluated based on Sensitivity, G-Mean, Balance, and ROC-AUC. Tables 5.1, 5.2, 5.3, and 5.4 demonstrates the values of Sensitivity, G-Mean, Balance, and ROC-AUC for three classic boosting ensemble methods- AB, ABNC, ABM1, and four resampling-based boosting ensemble methods-DB, MSMTB, RUSB, SMTB. The values are rounded up to two decimal places. The maximum value of particular performance measures corresponding to each dataset is boldfaced typed for better readability.

Sensitivity Analysis:

On analyzing Table 5.1, we find that the mean Sensitivity of classic ensembles increased by 50.5% when resampling methods were applied. The maximum Sensitivity value achieved by the classic ensemble is 0.56 which is achieved by the ABM1 classifier for Synapse1.2. In resampling-based boosting ensemble methods, the maximum Sensitivity achieved is 0.75 by RUSB. Experimental analysis reveals that ABM1 and AB have similar performance in terms of Sensitivity for Synapse1.1 (0.5), Log4j1.1 (0.5), Jedit4.0 (0.45), Xerces1.3 (0.38), Jedit4.2 (0.32), and Ivy2.0 (0.21). The values in brackets are the maximum Sensitivity that is achieved by the classic ensembles for the particular datasets. ABNC got maximum Sensitivity in classic ensemble category for four datasets- Log4j1.0 (0.52), Jedit4.1 (0.48), Ant1.7 (0.26), and Synapse1.0 (0.5). Striking increment in Sensitivity is observed when resampling methods are combined with boosting. In hybrid ensemble methods that are based on boosting, RUSB showed exemplary performance irrespective of the datasets involved. It has acquired the maximum Sensitivity value in Synapse1.2 (0.71), Jedit4.1 (0.67), Jedit4.0 (0.57), Camel1.6 (0.59), Camel1.4 (0.57), Xerces1.3 (0.61), Xalan2.4 (0.7), Jedit4.2 (0.68), Ivy2.0 (0.74), Synapse1.0 (0.75), and Tomcat6.0 (0.71).

ROSB was able to gain the maximum Sensitivity in two datasets, Log4j1.1 (0.65) and Ant1.7 (0.58). MSB and SMT got maximum Sensitivity value for only one dataset each.

Table 5.1: Sensitivity Results of Boosting-based Ensemble Methods

Dataset	AB	ABNC	ABM1	DBIM	MSMTB	RUSB	SMTB	ROSB
Synapse1.2	0.53	0.47	0.56	0.56	0.65	**0.71**	0.62	0.59
Synapse1.1	0.5	0.34	0.5	0.59	0.64	0.57	**0.66**	0.57
Log4j1.0	0.41	0.53	0.44	0.47	**0.65**	0.62	0.56	0.38
Jedit4.1	0.47	0.48	0.47	0.54	0.61	**0.67**	0.56	0.51
Log4j1.1	0.50	0.45	0.50	0.4	0.45	**0.60**	0.60	**0.65**
Jedit4.0	0.45	0.30	0.45	0.47	0.53	**0.57**	0.53	0.55
Ant1.7	0.24	0.26	0.24	0.48	**0.58**	**0.58**	0.53	**0.58**
Camel1.6	0.23	0.14	0.24	0.36	0.40	**0.59**	0.54	0.37
Camel1.4	0.15	0.09	0.14	0.18	0.38	**0.57**	0.38	0.29
Xerces1.3	0.38	0.38	0.38	0.48	0.54	**0.61**	0.54	0.37
Xalan2.4	0.21	0.07	0.19	0.00	0.40	**0.70**	0.46	0.45
Jedit4.2	0.32	0.16	0.32	0.28	0.40	**0.68**	0.40	0.46
Ivy2.0	0.21	0.11	0.21	0.05	0.37	**0.74**	0.47	0.28
Synapse1.0	0.33	0.50	0.33	0.42	0.50	**0.75**	0.42	0.13
Tomcat6.0	0.29	0.18	0.26	0.48	0.49	**0.71**	0.57	0.38

RUSB provided the best Sensitivity for 73% of datasets. There is not a single dataset that can attain a Sensitivity of 0.56 or more with classic ensemble methods. This target was achieved by 38.7% of datasets when resampling methods were involved. RUSB uniformly performed better than others as classification performed with all the datasets yields Sensitivity greater than or same as 0.56. the mean value of classic ensemble methods is either 0.3 for ABNC or 0.35 for AB and ABM1. The range of ensemble boosting methods with the changed proportion of defective and non-defective classes ranges from 0.38 to 0.64. Therefore, though classic ensemble methods perform better than other machine learning techniques (proved in the previous chapter), we should incorporate resampling methods with them to predict defects better with imbalanced data.

G-Mean Analysis:

Analysis of Table 5.2 points to the fact that there is no single dataset that has the maxi-

mum value of G-Mean for three classical boosting ensembles. This is a clear indication that the resampling-based boosting ensemble methods performed better than classical ensemble methods for the imbalanced data problem in the SDP domain. The mean value of G-Mean comes to be 0.53 for classic boosting ensembles whereas it is 0.63 for resampling-based boosting ensemble methods. Therefore, employing resampling methods with boosting ensemble methods certainly accounts for discovering more defects than the classic ensemble methods. 40% of cases in classic boosting ensemble classification have G-Mean value less than 50% as compared to only 6.7% cases of resampling-based boosting ensembles. 58.7% of resampling-based boosting ensemble models have G-Mean greater than 65%. On the contrary, as expected, only 15.6% of classic ensembles of boosting have the G-Mean value greater than 65%. In the case of classic ensemble models, the maximum value of G-Mean is depicted by ABNC for the Log4j1.0 dataset which is 0.69. In contrast to this, the maximum value reported by the boosting-based ensemble method is 0.77 by RUSB for the Ivy2.0 dataset. RUSB exemplifies the best performance for 10 out of 15 datasets for the G-Mean. SMTB also proved to be a good ensemble method giving the highest value of G-Mean for 4 datasets. It can be deduced from Table 5.2 that hybrid ensemble methods are exceptionally better performers than their classic versions.

Table 5.2: G-Mean Results of Boosting-based Ensemble Methods

Dataset	AB	ABNC	ABM1	DBIM	MSMTB	RUSB	SMTB	ROSB
Synapse1.2	0.66	0.63	0.68	0.68	0.71	**0.73**	0.68	0.67
Synapse1.1	0.67	0.57	0.66	0.7	0.71	0.65	**0.75**	0.69
Log4j1.0	0.59	0.69	0.61	0.64	**0.73**	0.69	0.68	0.55
Jedit4.1	0.63	0.65	0.63	0.67	0.71	**0.72**	0.68	0.65
Log4j1.1	0.65	0.60	0.65	0.58	0.57	0.67	0.69	**0.73**
Jedit4.0	0.65	0.53	0.65	0.63	0.65	0.66	0.67	**0.70**
Ant1.7	0.47	0.49	0.47	0.64	0.68	0.68	0.66	**0.70**
Camel1.6	0.46	0.37	0.47	0.55	0.55	0.59	**0.65**	0.57
Camel1.4	0.37	0.29	0.36	0.40	0.56	**0.63**	0.56	0.50
Xerces1.3	0.6	0.6	0.60	0.66	**0.70**	**0.70**	0.69	0.58
Xalan2.4	0.44	0.27	0.43	0.00	0.59	**0.72**	0.62	0.63

Dataset	AB	ABNC	ABM1	DBIM	MSMTB	RUSB	SMTB	ROSB
Jedit4.2	0.55	0.40	0.55	0.52	0.6	**0.74**	0.59	0.64
Ivy2.0	0.45	0.32	0.45	0.23	0.58	**0.77**	0.65	0.49
Synapse1.0	0.55	0.68	0.55	0.62	0.68	**0.73**	0.59	0.34
Tomcat6.0	0.52	0.42	0.50	0.66	0.67	**0.75**	0.71	0.59

The minimum G-mean value is predicted on Xalan2.4 with the value of 0.27. This value is extremely low and is increased to 0.72 when RUSB is applied to that dataset. G-Mean value considers both the correctly classified defective classes and correctly classified non-defective classes. This metric is, therefore, useful for the allocation of resources properly.

Balance Analysis:

The Balance value should be at least greater than 50% for good model predictions. The reason supporting this fact is data considered in building models is highly imbalanced. In 42.2% of the classic boosting cases, Balance values are less than 50%. On analyzing Table 5.3, we found that Balance values for classic boosting ensembles ranged from 34.52% to 66.37%. The maximum value of Balance is achieved by classic ensemble methods on the Synapse1.2 dataset by ABM1. The averaged Balance value in classic ensemble methods and resampling-based boosting ensemble methods is 52.08% and 62.03% respectively. The mean value of Balance has increased by approximately 20%.

The Balance values of the models developed using the RUSB are ranged from 59.16% to 76.59%. MSB and SMTB also seem to provide an acceptable range of Balance value. MSB gave a minimum Balance value of 54.14% on the Camel1.4 dataset and a maximum value of 72.05% on the Log4j1.0 dataset. SMT also depicted a similar range with a minimum of 54.36% on Camel1.4 and the maximum of 73.57% on Synapse1.1. RUSB gave the highest values for Balance on Synapse1.2 (72.71%), Jedit4.1 (71.88%), Camel1.4 (62.51%), Xerces1.3 (68.83%), Xalan2.4 (72.15%), Jedit4.2 (73.61%), Ivy2.0 (76.59%), Synapse1.0 (73.24%), and Tomcat6.0 (75.04%). The only 9.3% of datasets have Balance values less than 50% in resampling-based boosting methods as compared to classic 42.2%,

showing an improvement of approximate 354% in this particular case.

Table 5.3: Balance Results of Boosting-based Ensemble Methods

Dataset	AB	ABNC	ABM1	DBIM	MSMTB	RUSB	SMTB	ROSB
Synapse1.2	64.27	61.07	66.37	66.37	70.2	**72.71**	67.44	66.55
Synapse1.1	63.75	53.24	63.53	68.94	70.19	64.84	**73.57**	67.33
Log4j1.0	56.74	66	58.73	61.48	**72.05**	68.17	66.35	53.92
Jedit4.1	61.1	62.46	61.1	65.46	69.46	**71.88**	66.29	63.23
Log4j1.1	62.85	58.77	62.85	56.35	56.54	66.95	68.06	**72.06**
Jedit4.0	60.62	50.26	60.62	60.79	64.11	65.28	65.3	**67.18**
Ant1.7	45.9	47.48	45.9	61.91	67.09	67.09	64.7	**68.39**
Camel1.6	45.21	39.25	45.81	53.15	54.53	59.16	**64.14**	54.69
Camel1.4	39.62	35.31	38.9	41.65	54.14	**62.51**	54.36	48.98
Xerces1.3	55.76	55.79	55.76	62.64	66.47	**68.83**	66.22	55.03
Xalan2.4	43.94	34.52	42.9	29.29	56.9	**72.15**	60.43	60.2
Jedit4.2	51.8	40.59	51.83	49	57.05	**73.61**	56.79	61.04
Ivy2.0	44.05	36.72	44.12	33.01	54.9	**76.59**	62.06	48.12
Synapse1.0	52.27	64.36	52.27	58.33	64.15	**73.24**	57.27	37.84
Tomcat6.0	49.38	42.13	47.55	62.69	63.5	**75.04**	68.42	55.6

In classic boosting ensemble methods, only one model has a Balance value greater the 65%. This statistic changes to 45.3% with resampling methods. Where there is no single dataset that has a Balance value greater than 0.67 for classic methods, the involvement of resampling methods has improved the status and the concerned value is increased by 32%.

ROC-AUC Analysis: Similar behavior is emulated by the ROC-AUC performance metric as can be concluded by Table 5.4. There is again no evidence of any classic boosting ensemble method that provided the best results for any of the datasets. RUSB has the best predictive capability to discover defects for 8 out of 15 datasets. MSMTB and ROSB performed the best for three datasets each. The mean value of ROC-AUC increases to 0.70 for resampling-based boosting ensembles' performance from 0.63 of classing boosting ensembles. The only 4.4% of cases of classic boosting models have ROC-AUC greater than 70% which demonstrates their low predictive power for the detection of software defects. In contrast, 29.3% of cases of resampling-based boosting ensemble models have ROC-AUC

greater than 70%. RUSB based models have attained maximum ROC-AUC for Synapse1.2 (0.73), Jedit4.1(0.73), Camel1.4 (0.63), Xalan2.4 (0.72), Jedit4.2 (0.75), Ivy2.0 (0.77), Synapse1.0 (0.73), and Tomcat6.0 (0.75). In classic boosting ensemble methods, only 35.6% models have ROC-AUC greater than 65%. In contradiction to this, approximate 75% of cases of resampling-based boosting methods have ROC-AUC greater than 65%. Therefore, we can conclude that the hybrid boosting models performed better than classic boosting ensembles in terms of ROC-AUC also.

Table 5.4: ROC-AUC Results of Boosting-based Ensemble Methods

Dataset	AB	ABNC	ABM1	DBIM	MSMTB	RUSB	SMTB	ROSB
Synapse1.2	0.67	0.66	0.69	0.69	0.71	**0.73**	0.68	0.67
Synapse1.1	0.69	0.64	0.68	0.71	0.71	0.66	**0.75**	0.70
Log4j1.0	0.62	0.71	0.63	0.67	**0.73**	0.69	0.68	0.58
Jedit4.1	0.66	0.69	0.66	0.68	0.71	**0.73**	0.69	0.67
Log4j1.1	0.67	0.63	0.67	0.63	0.59	0.68	0.69	**0.73**
Jedit4.0	0.69	0.62	0.69	0.66	0.67	0.67	0.69	**0.72**
Ant1.7	0.58	0.60	0.58	0.67	0.69	0.69	0.68	**0.72**
Camel1.6	0.57	0.55	0.57	0.6	0.58	0.59	**0.66**	0.62
Camel1.4	0.53	0.52	0.53	0.54	0.60	**0.63**	0.60	0.58
Xerces1.3	0.66	0.64	0.66	0.70	**0.72**	0.70	0.71	0.67
Xalan2.4	0.58	0.52	0.57	0.5	0.64	**0.72**	0.65	0.66
Jedit4.2	0.64	0.58	0.65	0.62	0.66	**0.75**	0.66	0.68
Ivy2.0	0.60	0.54	0.61	0.50	0.66	**0.77**	0.70	0.58
Synapse1.0	0.62	0.72	0.62	0.68	**0.73**	**0.73**	0.65	0.53
Tomcat6.0	0.62	0.58	0.61	0.70	0.70	**0.75**	0.73	0.66

5.4.1.2 RQ1b: Which resampling-based boosting ensemble method is the best classifier for predicting software defects using imbalanced data?

It is evident from the above explanation that resampling-based boosting ensembles outperforms the classic boosting ensembles. Resampling-based ensembles are hybrid models that incorporate the benefits of resampling as well as ensembles. Therefore, their performances are better than classic ones. To determine the best resampling-based boosting ensemble

method, the study employs the Friedman rank test at significance level $\alpha = 0.05$ with a degree of freedom of seven to ascertain whether there is a significant difference between the predictive capability of the classic boosting ensemble methods and resampling-based boosting methods or not.

The following hypotheses are evaluated to answer this RQ:

- *Null Hypothesis (H_{0i})*: There is no difference between predictive capabilities of classic ensemble methods and resampling-based boosting ensemble methods for SDP with imbalanced data in terms of PM_i.

- *Alternate Hypothesis (H_{ai})*: There is a difference between predictive capabilities of classic ensemble methods and resampling-based boosting ensemble methods for SDP with imbalanced data in terms of PM_i.

  where $i = 1to4$ and PM_1 = Sensitivity, PM_2 = G-Mean, PM_3 = Balance, and PM_4 = ROC-AUC.

The hypothesis was designed and tested against the four performance measures. Rankings obtained are reported in Table 5.5 with their p-values.

Table 5.5: Friedman Rankings for SDP Models developed using Boosting based Ensemble Methods

Rank	Sensitivity	G-Mean	Balance	ROC-AUC
Rank 1	RUSB (7.53)	RUSB (7.03)	RUSB (7.23)	RUSB (6.56)
Rank 2	SMTB (6.26)	SMTB (6.13)	SMTB (6.4)	SMTB (6.2)
Rank 3	MSB (6.1)	MSB (5.9)	MSB (5.96)	MSB (5.83)
Rank 4	ROSB (4.73)	ROSB (4.8)	ROSB (4.93)	ROSB (4.86)
Rank 5	DB (3.6)	DB (3.7)	DB (3.63)	DB (3.83)
Rank 6	ABM1 (2.76)	ABM1 (3.03)	ABM1 (2.76)	ABM1 (3.26)
Rank 7	AB (2.73)	AB (2.8)	AB (2.66)	AB (3.06)
Rank 8	ABNC (2.26)	ABNC (2.6)	ABNC (2.4)	ABNC (2.36)
p-value	**0.000**	**0.000**	**0.000**	**0.000**
Kendall C	0.651	0.489	0.598	0.423

The mean ranks of ensemble methods are also recorded in Table 5.5. p-value comes to be 0.000 for Sensitivity, G-Mean, Balance, and ROC-AUC which is less than 0.05. Therefore, we refute the null hypotheses and conclude that there is a significant difference between the predictive capabilities of these methods based on all four measures. As depicted in Table 5.5, classic ensemble methods have least ranks owing to their low predictive capability for SDP in imbalanced datasets. The unanimous top ranker is RUSB followed by SMTB and MSB for Sensitivity, G-Mean, Balance, and ROC-AUC. RUSB has attained a mean rank of 7.53, 7.03, 7.23, and 6.56 for Sensitivity, G-Mean, Balance, and ROC-AUC.

5.4.2 Performance of Bagging-based Ensemble methods

Performance of classic bagging ensembles is compared with resampling-based bagging ensembles over 15 datasets that are imbalanced.

5.4.2.1 RQ2a: Do resampling-based bagging ensemble methods perform better than classic bagging ensemble methods in case of imbalanced data for predicting software defects?

Table 5.6, 5.7, 5.8, and 5.9 reports the Sensitivity, G-Mean, Balance, and ROC-AUC values of developed bagging-based ensemble models for defect prediction respectively. Here performances of one classic bagging ensemble and nine resampling-based bagging ensembles are compared. The maximum value of all performance measures is boldfaced typed for each dataset. As expected, the close examination of Table 5.6, 5.7, 5.8, and 5.9 reveals that Bag has never achieved maximum value of any of performance measures.

Sensitivity Analysis:

In 15 datasets, the classic bagging ensemble models have attained the lowest Sensitivity value for 11 datasets accounting for their low predictive capability as compared to other ensemble models that are resampling-based. Table 5.6 shows the Sensitivity results obtained

for SDP models developed using bagging and resampling-based bagging methods. The range of Bag models is from 0.03 to 0.5. This range is too low and signifies the extent to which wrong predictions are made for defective classes. UBAG2 and UBAG made better prediction models for all the 15 datasets. Sensitivity values for UBAG2 ranges from 0.53 to 0.83 and UBAG Sensitivity value ranges from 0.62 to 0.78. The drastic improvement in Sensitivity values can be monitored on the application of resampling methods.

Table 5.6: Sensitivity Results of Bagging-based Ensemble Methods

Dataset	Bag	MSBAG	OBAG	OBAG2	SMTBAG	UBAG	UBAG2	UOBAG	IIVOT	ROSBAG
Synapse1.2	0.49	0.76	0.63	0.71	0.72	**0.78**	0.74	0.68	0.65	0.64
Synapse1.1	0.39	**0.68**	0.50	0.55	0.61	0.66	0.64	0.61	0.57	0.62
Log4j1.0	0.50	0.71	0.47	0.59	0.68	0.71	**0.74**	0.68	0.65	0.56
Jedit4.1	0.46	0.68	0.43	0.57	0.63	**0.72**	0.67	0.68	0.66	0.58
Log4j1.1	0.40	0.60	0.35	0.50	0.60	**0.65**	**0.65**	0.60	0.50	0.59
Jedit4.0	0.40	0.62	0.49	0.53	0.55	**0.62**	**0.62**	0.51	0.36	0.57
Ant1.7	0.22	0.64	0.55	0.55	0.65	**0.70**	0.62	0.62	0.48	0.63
Camel1.6	0.09	0.5	0.33	0.38	0.62	**0.78**	0.61	0.56	0.32	0.41
Camel1.4	0.12	0.42	0.22	0.32	0.43	**0.69**	0.53	0.34	0.28	0.39
Xerces1.3	0.36	0.61	0.39	0.49	0.62	**0.78**	0.64	0.57	0.57	0.52
Xalan2.4	0.03	0.54	0.36	0.42	0.67	**0.78**	0.7	0.57	0.33	0.46
Jedit4.2	0.16	0.56	0.28	0.40	0.64	0.76	**0.80**	0.56	0.24	0.58
Ivy2.0	0.11	0.53	0.37	0.32	**0.68**	**0.68**	**0.68**	0.58	0.26	0.35
Synapse1.0	0.17	0.67	0.33	0.42	0.58	0.75	**0.83**	0.50	0.50	0.19
Tomcat6.0	0.16	0.64	0.43	0.45	**0.78**	**0.78**	0.70	0.65	0.31	0.38

Analysis of Table 5.6 tells that there is not a single dataset with Bag that can achieve Sensitivity greater than 0.50 whereas 68% of datasets with resampling-based ensembles have Sensitivity greater than 0.50. The mean value of Sensitivity obtained by Bag over 15 datasets is 0.27. The mean value for Bag ranges from 0.41 to 0.72. The highest mean Sensitivity value of 0.72 is gained by UBAG and an approximate 167% increase is monitored in this as compared to the Bag mean value. UBAG achieved highest Sensitivity values for Synapse1.2 (0.78), Jedit4.1 (0.74), Log4j1.1 (0.65), Jedit4.0 (0.62), Ant1.7 (0.70), Camel1.6 (0.78), Camel1.4 (0.69), Xerces1.3 (0.78), Xalan2.4 (0.78), Ivy2.0 (0.68), and

Tomcat6.0 (0.78). UBAG2 and MSBAG also were able to attain maximum Sensitivity value for a few datasets.

G-Mean Analysis:

Similar to Sensitivity analysis, Bag did not perform better as compared to other resampling-based models. The minimum G-Mean value achieved by BAG is 0.17 for Xalan2.4. The minimum G-mean range for resampling-based bagging ensemble methods is from 0.55 to 0.73. UBAG2 depicted maximum G-Mean value in Log4j1.0, Jedit4.1, Log4j1.1, Jedit4.0, and Jedit4.2 of 0.76, 0.74, 0.72, 0.72, and 0.79 respectively. UBAG and MSBAG predicted the maximum G-Mean values for three datasets each. Percentage improvement in Xalan2.4 on applying resampling method is minimum 223% and maximum 339%. Similar patterns were observed in other datasets. In the previous chapter, ROS turned to be a better sampling technique with ML techniques. When hybrid ensembles were considered, though it did not perform that well, but managed to grab the highest G-Mean value in Synapse1.1 and Ant1.7 with a maximum of 0.72 and 0.74 respectively.

Table 5.7: G-Mean Results of Bagging-based Ensemble Methods

Dataset	Bag	MSBAG	OBAG	OBAG2	SMTBAG	UBAG	UBAG2	UOBAG	IIVOT	ROSBAG
Synapse1.2	0.65	0.76	0.72	0.74	0.74	0.75	**0.76**	0.72	0.71	0.70
Synapse1.1	0.60	**0.72**	0.66	0.67	0.71	0.69	0.69	0.68	0.68	**0.72**
Log4j1.0	0.68	0.76	0.63	0.71	0.73	0.72	**0.76**	0.75	0.72	0.69
Jedit4.1	0.65	0.73	0.61	0.68	0.70	0.71	**0.74**	0.74	0.73	0.69
Log4j1.1	0.6	0.67	0.52	0.6	0.66	0.66	**0.72**	0.67	0.60	0.69
Jedit4.0	0.62	0.69	0.65	0.68	0.68	0.65	**0.72**	0.66	0.56	0.71
Ant1.7	0.46	0.73	0.68	0.68	0.71	0.73	0.70	0.71	0.64	**0.74**
Camel1.6	0.29	0.62	0.53	0.56	**0.65**	0.62	0.63	0.64	0.53	0.59
Camel1.4	0.34	0.57	0.43	0.52	0.57	**0.63**	0.60	0.53	0.50	0.57
Xerces1.3	0.59	0.73	0.59	0.66	0.71	**0.74**	0.71	0.69	0.72	0.68
Xalan2.4	0.17	0.67	0.57	0.60	0.70	**0.73**	0.72	0.68	0.55	0.62
Jedit4.2	0.40	0.70	0.51	0.61	0.72	0.74	**0.79**	0.69	0.47	0.72
Ivy2.0	0.32	0.69	0.58	0.52	**0.72**	0.71	0.71	0.7	0.50	0.56
Synapse1.0	0.40	**0.76**	0.54	0.60	0.65	0.69	0.74	0.60	0.66	0.41
Tomcat6.0	0.39	0.74	0.62	0.64	**0.78**	0.77	0.75	0.74	0.54	0.59

SMTBAG and UBAG have the maximum G-Mean value for 3 datasets each. The results could not succeed to get a clear picture of any one method. UBAG2, UBAG, SMTBAG, and MSBAG may have comparable predictive capabilities that can be further analyzed by performing post-hoc analysis. The observed averaged value of G-Mean for classic bagging ensemble is 0.47 whereas it is 0.66 for resampling-based bagging ensemble models. 53.3% of cases of classic bagging ensemble have G-Mean less than 50% as compared to meager 3% of cases of models generated by resampling-based bagging ensemble methods. Merely 6.7% of classic ensembles of bagging have the G-Mean value greater than 0.65 whereas 66.7% of resampling-based bagging ensemble models have G-Mean greater than 0.65. There are approximately 40% of cases that have G-Mean value greater than 70% when resampling-based bagging ensembles are used. On the contrary, not a single dataset got its G-Mean value greater than 70% when the classic bagging ensemble method is used to train and test the data. Thus, these facts and observations confirmed that hybrid ensembles (with resampling) are better defect predictors than the classic ensembles.

Balance Analysis:

Table 5.8 reports the Balance values for bagging ensemble methods. The maximum value of Balance is 79.49 portrayed by UBAG2 for the Jedit4.2 dataset for resampling-based bagging ensemble methods whereas the maximum value observed by the classic bagging ensemble is only 64.31 for the Log4j1.0 dataset. The performance of bagging-based ensemble methods in terms of Balance is similar to that of G-Mean for maximum value achieved by datasets.

Table 5.8: Balance Results of Bagging-based Ensemble Methods

Dataset	BAG	MSBAG	OBAG	OBAG2	SMTBAG	UBAG	UBAG2	UOBAG	IIVOT	ROSBAG
Synapse1.2	62.32	**76.39**	70.93	73.59	73.49	74.63	75.79	71.65	70.94	70.01
Synapse1.1	56.36	**71.82**	63.53	65.61	69.66	68.81	68.39	67.58	66.74	70.78
Log4j1.0	64.31	75.68	61.12	69.05	72.42	71.9	**76.2**	74.53	71.04	67.08
Jedit4.1	61.04	72.61	58.5	67	69.46	71.03	**73.47**	73.46	72.39	67.83
Log4j1.1	56.83	66.34	51.34	59.68	65.71	65.56	**71.15**	66.34	59.68	67.77

Dataset	BAG	MSBAG	OBAG	OBAG2	SMTBAG	UBAG	UBAG2	UOBAG	IIVOT	ROSBAG
Jedit4.0	57.79	68.57	62.7	65.6	66.03	64.99	**70.6**	63.66	53.76	68.72
Ant1.7	44.84	71.73	66.22	66.3	70.59	**72.75**	68.97	69.85	61.86	72.15
Camel1.6	35.58	60.91	51.53	54.6	**64.65**	60.53	63.4	63.5	51.01	57.02
Camel1.4	37.62	56.21	43.6	50.81	56.24	**63.24**	60.13	51.66	48.54	55.18
Xerces1.3	54.85	70.84	56.24	63.11	70.16	**73.83**	70.36	67.51	68.55	65.35
Xalan2.4	31.39	65.46	54	57.7	69.85	**72.9**	72.4	67.04	52.04	60.46
Jedit4.2	40.6	67.62	48.8	57.2	71.27	73.78	**79.49**	67.4	46.02	69.35
Ivy2.0	36.73	65.7	54.82	50.61	**71.74**	70.48	71.03	68.03	47.73	53.4
Synapse1.0	41.03	**74.8**	52.15	57.8	64.75	69.13	73.43	59.43	63.71	42.06
Tomcat6.0	40.3	72.19	59.04	60.8	**78.45**	77.1	74.89	73	51	55.47

UBAG2 was able to generate a maximum Balance value of 76.20, 73.47, 71.15, 70.60, and 79.49 for Log4j1.0, Jedit4.1, Log4j1.1, Jedit4.0, and Jedit4.2 respectively. Camel1.4, Xerces1.3, and Xalan2.4 experienced the best prediction based on Balance with UBAG. Like with G-Mean, SMTBAG was also able to attain the highest Balance value for three datasets- Camel1.6, Ivy2.0, and Tomcat6.0.

The averaged value of Balance over datasets for Bag comes to be 48.11. This value increased to 65.17 when resampling methods are embedded in ensemble methods. 53.3% of models have a Balance value less than 50% when Bag is used to identify defects. This state is not acceptable as such models have very low predictive capabilities. Hybrid ensembles that combine Bagging with resampling methods reduce the proportion of models to 4.4% with the Balance value less than 50%. With classic bagging, no model can secure a Balance value greater than 65% whereas 61.5% of models have attained a Balance value greater than 65%. This is a remarkable improvement in model prediction which is achieved by using hybrid versions of bagging ensembles.

ROC-AUC Analysis:

Now, by analyzing Table 5.9, we ascertain analogous conduct of ROC-AUC performance metric for SDP using bagging-based ensemble methods.

Table 5.9: ROC-AUC Results of Bagging-based Ensemble Methods

Dataset	Bag	MSBAG	OBAG	OBAG2	SMTBAG	UBAG	UBAG2	UOBAG	IIVOT	ROSBAG
Synapse1.2	0.67	**0.76**	0.72	0.74	0.73	0.75	**0.76**	0.72	0.72	0.71
Synapse1.1	0.65	0.72	0.68	0.68	0.71	0.69	0.68	0.68	0.68	**0.73**
Log4j1.0	0.71	0.77	0.66	0.72	0.73	0.72	**0.77**	0.76	0.72	0.70
Jedit4.1	0.69	0.73	0.64	0.69	0.70	0.71	**0.75**	0.74	0.73	0.70
Log4j1.1	0.64	0.67	0.56	0.61	0.66	0.65	**0.72**	0.67	0.61	0.69
Jedit4.0	0.68	0.69	0.68	0.70	0.69	0.65	**0.73**	0.68	0.61	**0.73**
Ant1.7	0.59	0.73	0.70	0.70	0.71	0.73	0.70	0.72	0.67	0.74
Camel1.6	0.53	0.63	0.60	0.61	**0.65**	0.63	0.63	**0.65**	0.59	0.63
Camel1.4	0.54	0.60	0.54	0.58	0.59	**0.64**	0.61	0.57	0.59	0.60
Xerces1.3	0.67	0.74	0.64	0.69	0.72	**0.74**	0.71	0.71	0.74	0.71
Xalan2.4	0.51	0.69	0.63	0.64	0.7	**0.73**	0.72	0.7	0.61	0.65
Jedit4.2	0.57	0.73	0.60	0.67	0.75	0.74	**0.80**	0.71	0.58	0.73
Ivy2.0	0.55	0.72	0.65	0.60	**0.73**	0.71	0.72	0.72	0.62	0.62
Synapse1.0	0.58	**0.79**	0.62	0.64	0.66	0.70	0.73	0.61	0.69	0.55
Tomcat6.0	0.57	0.74	0.67	0.68	**0.79**	0.77	0.75	0.75	0.62	0.65

With no dataset having the maximum ROC-AUC value in the classic bagging-based model, one can comment that the resampling-based bagging ensemble methods performed much better than the classic bagging ensemble method. There is a mixed bag of resampling-based bagging ensemble methods that scored the maximum ROC-AUC values for individual datasets. UBAG2 demonstrates the maximum ROC-AUC value for six datasets followed by UBAG and SMTBAG which have the maximum ROC-AUC values for three datasets each. UOBAG and ROSBAG also delivered maximum ROC-AUC value for two datasets each. The mean value for resampling-based bagging ensembles increases to 0.68 from 0.61 of classing bagging ensemble. Approximate 53.3% of classic bagging models have ROC-AUC less than 60% as compared to 8.1% cases of resampling-based bagging ensembles. 47.4% of built models achieved ROC-AUC greater than 70% when resampling-based bagging ensembles were applied. Only one model based on the classic bagging technique has ROC-AUC greater than 70%. Log4j1.0 achieved a 71.17 ROC-AUC value with Bag. This statistic is increased to 76.72 for the same dataset with UBAG2.

Therefore, the result analysis supports the fact that the resampling-based bagging ensemble methods perform better than the classic bagging ensemble method in case of imbalanced data for predicting software defects. We deduced that UBAG2, UBAG, MSBAG, SMTBAG, and RUSB have comparable performances for both G-Mean and ROC-AUC and performed better than other methods. All classic ensembles have significantly lower performance than their performances. With ROC-AUC, SMTB also seems to be a promising approach for predicting defects.

5.4.2.2 RQ2b: Which resampling-based bagging ensemble method is the best classifier for predicting software defects using imbalanced data?

In this chapter, as mentioned earlier, we exploited the non-parametric Friedman test to uncover the rankings of classic bagging and resampling-based bagging methods. The null hypothesis and alternate hypothesis evaluated to get the answer of this RQ for Sensitivity are stated as:

- *Null Hypothesis (H_{01})*: There is no difference between predictive capabilities of classic ensemble methods and resampling-based boosting ensemble methods for SDP with imbalanced data in terms of Sensitivity.
- *Alternate Hypothesis (H_{a1})*: There is a difference between predictive capabilities of classic ensemble methods and resampling-based boosting ensemble methods for SDP with imbalanced data in terms of Sensitivity.

Similar hypotheses were made for the G-Mean, Balance, and ROC-AUC metric. These null hypotheses were tested using the Friedman test with a degree of freedom as nine at confidence level $\alpha = 0.05$ and results are noted for all performance measures in Table 5.10. It contains the mean ranks of Bag and resampling-based bagging ensemble methods with p-values. The higher the mean rank, the better are the predictions of the corresponding

model. Kendall C in Table 5.10 signifies Kendall's coefficient of concordance. The higher the Kendall C, the more the results are dependable.

Table 5.10: Friedman Rankings for SDP Models developed using Bagging based Ensembles

Rank	Sensitivity	G-Mean	Balance	ROC-AUC
Rank 1	UBAG (9.5)	UBAG2 (8.53)	UBAG2 (8.66)	UBAG2 (8.26)
Rank 2	UBAG2 (8.6)	MSBAG (7.76)	MSBAG (7.83)	MSBAG (7.86)
Rank 3	SMTBAG (7.5)	UBAG (7.6)	UBAG (7.8)	UBAG (7.33)
Rank 4	MSBAG (7.43)	SMTBAG (7.4)	SMTBAG (7.6)	SMTBAG (7.33)
Rank 5	UOBAG (6.1)	UOBAG (6.3)	UOBAG (6.3)	UOBAG (5.8)
Rank 6	ROSBAG (4.8)	ROSBAG (5.53)	ROSBAG (5.4)	ROSBAG (5.6)
Rank 7	OBAG2 (3.93)	OBAG2 (4.23)	OBAG2 (4.1)	OBAG2 (4.2)
Rank 8	IIVOT (3.3)	IIVOT (3.63)	IIVOT (3.43)	IIVOT (3.83)
Rank 9	OBAG (2.56)	OBAG (2.66)	OBAG (2.6)	OBAG (2.9)
Rank 10	Bag (1.26)	Bag (1.33)	Bag (1.26)	Bag (1.86)
p-value	**0.000**	**0.000**	**0.000**	**0.000**
Kendall C	0.832	0.648	0.7082	0.539

To refute the set hypotheses, the p-value needs to be less than 0.05. On performing the Friedman test for bagging-based ensembles, the detected p-value is again 0.000 for Sensitivity, G-Mean, Balance, and ROC-AUC, similar to that of boosting-based ensemble methods. Therefore, the stated null hypotheses are rejected and it is confirmed that there is a significant difference between performances of these ensemble methods based on Sensitivity, G-Mean, Balance, and ROC-AUC. Classic bagging- Bag has attained the last rank for all measures supporting the answer to RQ2a. UBAG2 seems to be a very promising approach with the first position and mean ranks of 8.53, 8.66, and 8.26 in G-Mean, Balance, and ROC-AUC respectively. UBAG has acquired the first position with a mean rank of 9.5 for Sensitivity. For Sensitivity, UBAG2 also performed better than others as it secured the second rank with a mean rank of 8.6. The top four rankers for predicting software defects are UBAG2, UBAG, MSBAG, and SMTBAG considering Sensitivity, G-Mean, Balance, and ROC-AUC.

5.4.2.3 Predicting best Software Defect Predictor by Statistical Analysis

This subsection provides an answer to the following RQ3:

RQ3: Which amongst addressed ensemble methods is the best for predicting software defects in an imbalanced data domain?

For a fair comparison, we constructed the bar graphs representing the median values of all the performance measures for the classic and hybrid ensemble methods. The median values take the skewness of data in their consideration. Figure 5.2, 5.3, 5.4, and 5.5 inspects the median-wise performances for different performance measures used in the chapter. This can be deduced from these four figures that the classic ensembles- AB, ABNC, ABM1, and Bag are the least performing.

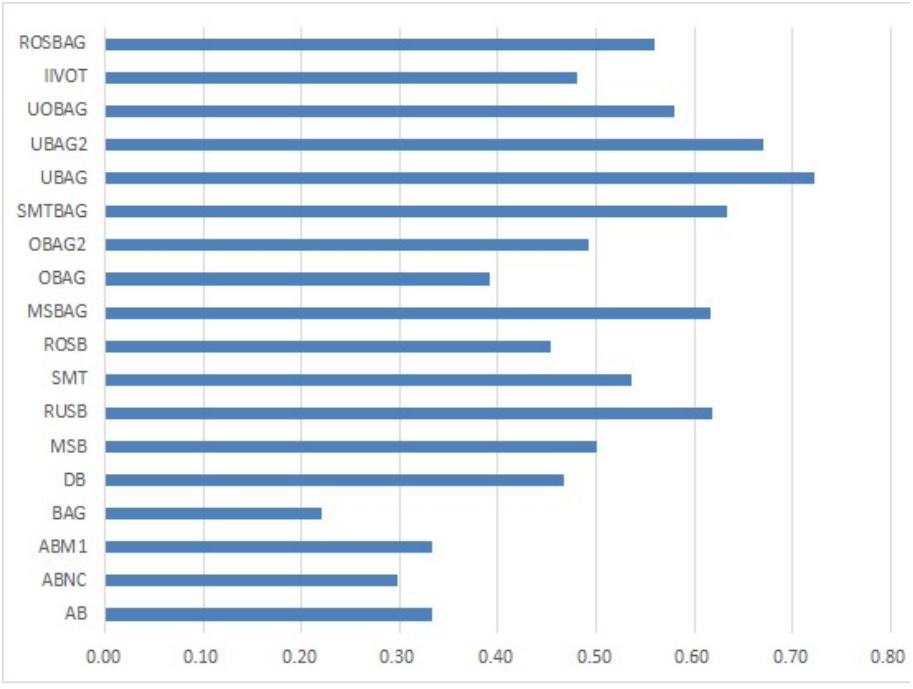

Figure 5.2: Median Sensitivity Values of Classic and hybrid Ensemble Methods

The median sensitive value is maximum for UBAG (0.72). UBAG2 illustrates the maximum median value for G-Mean (0.72), Balance (71.15), and ROC-AUC (0.72). Now, when

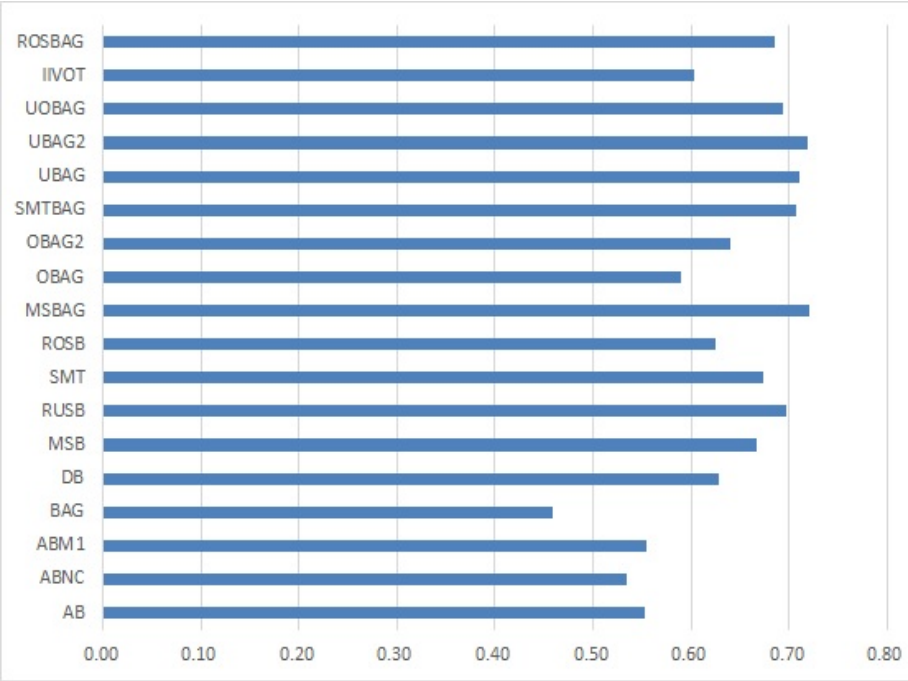

Figure 5.3: Median G-Mean Values of Classic and hybrid Ensemble Methods

we know the predictive capability of all the developed models, how to conclude which one is the best? The Friedman rankings of ensemble methods in RQ1 demonstrated the competitive prediction ability of RUSB in boosting-based ensemble methods and UBAG2, UBAG, MSBAG, and SMTBAG in bagging-based ensemble methods. Any existence of family-wise errors needs to be checked to confirm that the predictions are not by any chance.

For this, to find the cumulative impact first Friedman test is collectively applied on predictive results of seventeen ensemble methods for all addressed datasets. Considering the results for all datasets, we state the following hypotheses:

- *Null Hypothesis (H_{01})*: There is no difference between predictive capabilities of classic ensembles, resampling-based bagging ensembles, and resampling-based boosting ensembles for SDP with imbalanced data in terms of Sensitivity.

- *Alternate Hypothesis (H_{a1})*: There is a difference between predictive capabilities

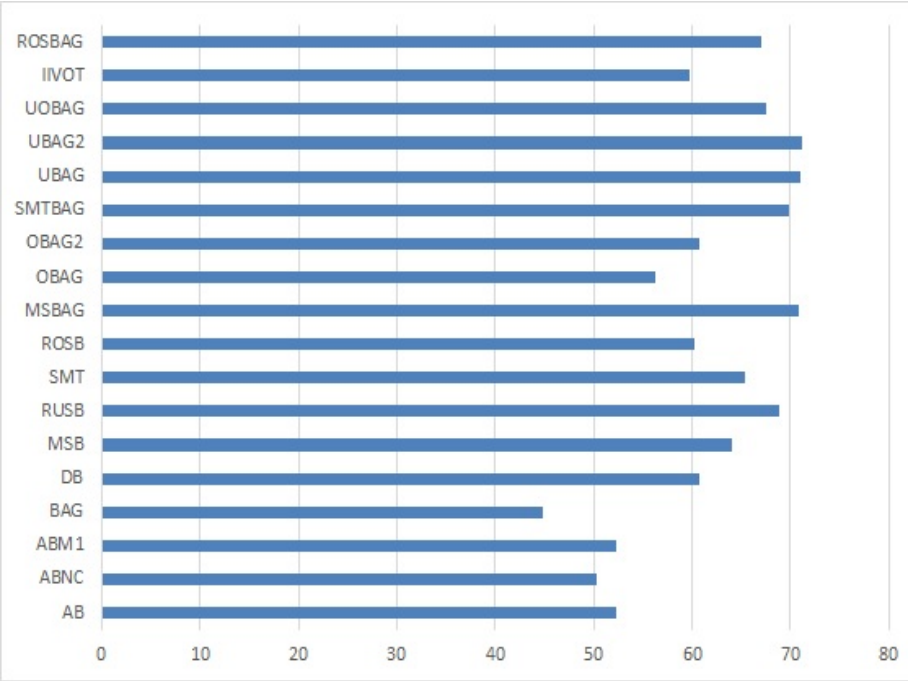

Figure 5.4: Median Balance Values of Classic and hybrid Ensemble Methods

of classic ensembles, resampling-based bagging ensembles, and resampling-based boosting ensembles for SDP with imbalanced data in terms of Sensitivity.

- *Null Hypothesis (H_{02}):* There is no difference between predictive capabilities of classic ensembles, resampling-based bagging ensembles, and resampling-based boosting ensembles for SDP with imbalanced data in terms of G-Mean.

- *Alternate Hypothesis (H_{a2})*: There is a difference between predictive capabilities of classic ensembles, resampling-based bagging ensembles, and resampling-based boosting ensembles for SDP with imbalanced data in terms of G-Mean.

- *Null Hypothesis (H_{03})*: There is no difference between predictive capabilities of classic ensembles, resampling-based bagging ensembles, and resampling-based boosting ensembles for SDP with imbalanced data in terms of Balance.

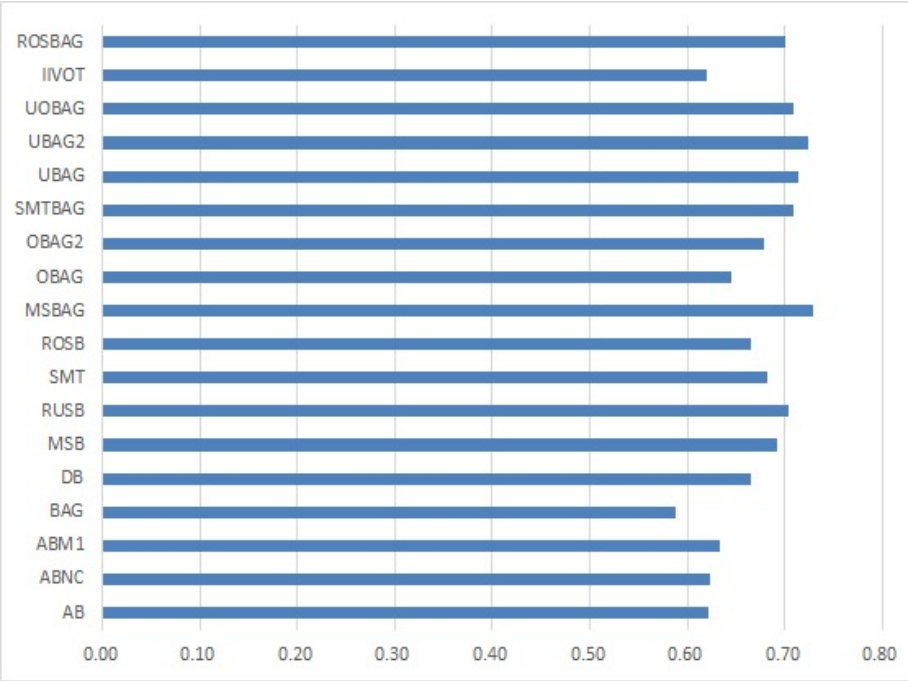

Figure 5.5: Median ROC-AUC Values of Classic and hybrid Ensemble Methods

- *Alternate Hypothesis (H_{a3})*: There is a difference between predictive capabilities of classic ensembles, resampling-based bagging ensembles, and resampling-based boosting ensembles for SDP with imbalanced data in terms of Balance.

- *Null Hypothesis (H_{04})*: There is no difference between predictive capabilities of classic ensembles, resampling-based bagging ensembles, and resampling-based boosting ensembles for SDP with imbalanced data in terms of ROC-AUC.

- *Alternate Hypothesis (H_{a4})*: There is a difference between predictive capabilities of classic ensembles, resampling-based bagging ensembles, and resampling-based boosting ensembles for SDP with imbalanced data in terms of ROC-AUC.

. Table 5.11 summarizes the Friedman rankings of all these 17 ensemble methods for Sensitivity, G-Mean, Balance, and ROC-AUC. The mean ranks of these methods are also noted in closed brackets. The p-value obtained against each test was 0.000. Therefore, for

each considered performance measure null hypothesis is refuted. Kendall C represents the agreement in the class classification as defective or non-defective in various datasets by the number of ensemble methods involved. The high value of Kendall C represents the similar behavior of predicting any class by different ensemble methods for a particular dataset. For example, if ensemble methods do not agree on classifying a particular class as defective, then we may not rely on the conclusions. Kendall C value achieved for Sensitivity, G-Mean, and Balance is 0.798, 0.631, and 0.702 respectively. Therefore, related rankings are reliable.

Table 5.11: Friedman Rankings for SDP Models developed using Bagging based Ensemble Methods

PM	Sensitivity	G-Mean	Balance	ROC-AUC
Rank 1	UBAG (17.36)	UBAG2 (15.93)	UBAG2 (16.13)	UBAG2 (15.33)
Rank 2	UBAG2 (16.23)	MSBAG (15.03)	MSBAG (15.16)	MSBAG (15.06)
Rank 3	SMTBAG (14.76)	UBAG (14.46)	UBAG (14.86)	SMTBAG (13.66)
Rank 4	MSBAG (14.73)	SMTBAG (14.26)	SMTBAG (14.76)	UBAG (13.6)
Rank 5	RUSB (14.36)	RUSB (13.1)	RUSB (13.56)	RUSB (11.9)
Rank 6	UOBAG (12.76)	UOBAG (12.63)	UOBAG (12.76)	UOBAG (11.53)
Rank 7	SMT (10.46)	ROSBAG (11.26)	ROSBAG (11.26)	ROSBAG (11.46)
Rank 8	ROSBAG (10.46)	SMT (10.8)	SMT (11)	SMT (11.26)
Rank 9	MSB (10.36)	MSB (10.3)	MSB (10.2)	MSB (10.23)
Rank 10	OBAG2 (8.9)	OBAG2 (9.53)	OBAG2 (9.36)	OBAG2 (9.13)
Rank 11	IIVOT (7.7)	ROSB (8.33)	ROSB (8.2)	ROSB (8.4)
Rank 12	ROSB (7.56)	IIVOT (8)	IIVOT (7.7)	IIVOT (8.13)
Rank 13	OBAG (5.8)	DB (6.1)	DB (5.9)	DB (6.23)
Rank 14	DB (5.56)	OBAG (5.63)	OBAG (5.7)	OBAG (6.13)
Rank 15	ABM1 (4.23)	ABM1 (4.6)	ABM1 (4.33)	ABM1 (5.46)
Rank 16	AB (4.2)	AB (4.4)	AB (4.26)	AB (5.33)
Rank 17	ABNC (3.4)	ABNC (4)	ABNC (3.53)	Bag (4.13)
Rank 18	Bag (2.1)	Bag (2.6)	Bag (2.26)	ABNC (3.96)
p-value	**0.000**	**0.000**	**0.000**	**0.000**
Kendall C	0.798	0.631	0.702	0.485

Classic ensemble methods got the lowest ranks from rank 15 to rank 18 for all the performance measures. Their performance is consistent and poor than any of the hybrid ensemble methods. The first five Friedman rankings are grabbed by four resampling+bagging models (UBAG2, UBAG, MSBAG, SMTBAG) and one resampling+boosting model (SMTB). UBAG2 scored the first rank for G-Mean, Balance, and ROC-AUC with a mean rank of 15.93, 16.13, and 15.33 respectively. Rank 2 for G-Mean, Balance, and ROC-AUC is grabbed by MSBAG with a mean rank of 15.03, 15.16, and 15.06 respectively. For Sensitivity, the first ranker was UBAG with a mean rank of 17.36 followed by UBAG2 (16.23). Only RUSB, SMT, and MSMTB are three boosting based ensemble methods that can make their position in the first nine rankers. The overall ranking gives the view that resampling-based bagging models are better predictors than boosting-based models. One of the reasons for their better performance is that C4.5 tends to have high variance and bagging helps in reducing this variance. Boosting-based ensemble methods can perform much better with machine learners that face high bias issues.

Significant results are obtained by the Friedman test. Hence, post-hoc analysis is conducted using the Wilcoxon signed-rank test. Wilcoxon signed-rank test is applied to compare the best ranker with other ensemble methods using SPSS. For Sensitivity, the Wilcoxon signed-rank test is performed with UBAG and for other measures, post-hoc analysis is carried out against UBAG2. We would evaluate the following hypotheses using the Wilcoxon signed-rank test at $\alpha = 0.05$.

- *Null Hypothesis (H_{01})*: UBAG has similar defect prediction capability than that of classic ensembles, resampling-based boosting ensembles, and other resampling-based bagging ensembles with imbalanced data in terms of Sensitivity.

- *Alternate Hypothesis (H_{a1})*: UBAG has better defect prediction capability than that of classic ensembles, resampling-based boosting ensembles, and other resampling-based bagging ensembles with imbalanced data in terms of Sensitivity.

- *Null Hypothesis (H_{02}):* UBAG2 has similar defect prediction capability than that of classic ensembles, resampling-based boosting ensembles, and other resampling-based bagging ensembles with imbalanced data in terms of G-Mean.

- *Alternate Hypothesis (H_{a2})*: UBAG2 has better defect prediction capability than that of classic ensembles, resampling-based boosting ensembles, and other resampling-based bagging ensembles with imbalanced data in terms of G-Mean.

- *Null Hypothesis (H_{03})*: UBAG2 has similar defect prediction capability than that of classic ensembles, resampling-based boosting ensembles, and other resampling-based bagging ensembles with imbalanced data in terms of Balance.

- *Alternate Hypothesis (H_{a3})*: UBAG2 has better defect prediction capability than that of classic ensembles, resampling-based boosting ensembles, and other resampling-based bagging ensembles with imbalanced data in terms of Balance.

- *Null Hypothesis (H_{04})*: UBAG2 has similar defect prediction capability than that of classic ensembles, resampling-based boosting ensembles, and other resampling-based bagging ensembles with imbalanced data in terms of ROC-AUC.

- *Alternate Hypothesis (H_{a4})*: UBAG2 has better defect prediction capability than that of classic ensembles, resampling-based boosting ensembles, and other resampling-based bagging ensembles with imbalanced data in terms of ROC-AUC.

The Sensitivity results are shown in Table 5.12. Pairs of ensemble methods that are significantly different from each other are typed in boldface. In pair (X, Y), technique X is significantly better than technique Y if p-value$>= 0.05$.

Table 5.12: Wilcoxon Signed-rank Test Results for Ensemble Methods using Sensitivity

Pair of Ensemble Methods	Sensitivity
UBAG - AB	**0.0007**
UBAG - ABNC	**0.0007**
UBAG - ABM1	**0.0007**
UBAG - DB	**0.0007**
UBAG - MSB	**0.0007**
UBAG - RUSB	**0.0023**
UBAG - SMT	**0.001**
UBAG - ROSB	**0.0007**
UBAG - Bag	**0.0007**
UBAG - MSBAG	**0.0024**
UBAG - OBAG	**0.0007**
UBAG - OBAG2	**0.0007**
UBAG - SMTBAG	**0.0015**
UBAG - UOBAG	**0.0007**
UBAG - IIVOT	**0.0007**
UBAG - ROSBAG	**0.0007**
UBAG - UBAG2	**0.0414**

From Table 5.12 this is clear that UBAG has significantly performed better than all other ensemble methods, whether classical, resampling-based boosting, or other resampling-based bagging ensembles.

Table 5.13 records the p-values for Wilcoxon signed-rank test with G-Mean, Balance, and ROC-AUC.

Table 5.13: Wilcoxon Signed-rank Test Results for Ensemble Methods using G-Mean, Balance and ROC-AUC

Pair of Ensemble Methods	G-Mean	Balance	ROC-AUC
UBAG2 - AB	**0.0007**	**0.0007**	**0.0008**
UBAG2 - ABNC	**0.0007**	**0.0007**	**0.0007**
UBAG2 - ABM1	**0.0007**	**0.0007**	**0.0008**
UBAG2 - DB	**0.0008**	**0.0008**	**0.001**
UBAG2 - MSB	**0.0012**	**0.0008**	**0.0031**
UBAG2 - RUSB	**0.0231**	**0.0268**	**0.0231**
UBAG2 - SMT	**0.0045**	**0.0022**	**0.0106**

Pair of Ensemble Methods	G-Mean	Balance	ROC-AUC
UBAG2 - ROSB	**0.0022**	**0.001**	**0.0054**
UBAG2 - Bag	**0.0007**	**0.0007**	**0.0007**
UBAG2 - MSBAG	0.1914	0.0691	0.5136
UBAG2 - OBAG	**0.0007**	**0.0007**	**0.0008**
UBAG2 - OBAG2	**0.0007**	**0.0007**	**0.0007**
UBAG2 - SMTBAG	**0.0356**	**0.0309**	0.0609
UBAG2 - UBAG	0.1556	0.1556	0.1728
UBAG2 - UOBAG	**0.0031**	**0.0022**	**0.0106**
UBAG2 - IIVOT	**0.0008**	**0.0007**	**0.0018**
UBAG2 - ROSBAG	**0.0045**	**0.0018**	**0.0106**

As depicted in Table 5.13 UBAG2 significantly outperforms classic ensembles, resampling-based boosting ensembles, and all other resampling-based bagging ensembles for performance measures G-Mean, Balance, and ROC-AUC over all the datasets except for MSBAG and UBAG. We conclude that UBAG2, UBAG, and MSBAG have comparable performance concerning G-Mean, Balance, and ROC-AUC as their p-value is greater than 0.05.

The results statistically authenticate the answers of RQ1 and RQ2 stating that resampling-based boosting and bagging ensemble methods significantly outperform the classic boosting and bagging ensemble models. Results also support the evidence of the supremacy of resampling-based bagging ensembles than the F boosting ensembles. Now, observing the post-hoc analysis outcome, it is clear that UBAG2, UBAG, and MSBAG helped to construct more effective defect prediction models. We accept the alternate hypotheses for G-Mean, Balance, and ROC-AUC for all ensemble methods except for MSBAG and UBAG. The null hypothesis is accepted for SMTBAG also in ROC-AUC. Therefore, SMTBAG also resulted in achieving good ROC-AUC scores for SDP models.

5.5 Discussion

In this chapter, several boosting-based and bagging-based ensemble methods were applied on highly imbalanced data and the results advocate incorporating resampling methods with ensemble methods for better and improved performance of SDP models. This research advocates the use of hybrid ensemble methods over the classic ensemble methods for dealing with imbalanced data problem in SDP.

The main contributions of this chapter are:

- Dimensionality reduction using CFS resulting in unbiased model building.
- Providing a hybrid framework for defect prediction in the imbalanced data domain.
- Statistically validating results using the Friedman and Wilcoxon signed-rank test.
- Use of stable performance measures –G-Mean, Balance, and ROC-AUC for better prediction of imbalanced data.
- Identification of preferable resampling methods to be integrated with boosting or bagging ensemble method.

Major observations in this experimental study are summarized below:

- Hybrid ensemble methods have statistically improved the results of defect prediction in both bagging and boosting ensemble methods.
- Preferable hybrid ensemble methods are UBAG2, UBAG, SMTBAG, and MSBAG concluding to effective and accurate SDP in imbalanced data.
- Bagging-based hybrid ensembles performed better than Boosting-based hybrid ensembles.

- If researchers or software practitioners are interested in developing boosting models, depending on their requirements, then they should incorporate the RUS resampling method to deal with imbalanced data. RUSB proved its potential in boosting ensemble methods.

UBAG2, UBAG, SMTBAG, and MSBAG seem promising hybrid ensemble methods. With the C4.5 decision tree as the base classifier, these hybrid ensemble methods can be used to uncover the unseen future defects. This knowledge can be exploited by researchers and software practitioners to build effective SDP models in an imbalanced data domain.

Next, we would like to tackle class imbalance problem at algorithmic level using meta-cost learners and analyze their efficacy in uncovering probable defects in software.

Chapter 6

Tackling Class Imbalance Problem at Algorithm Level: Cost-sensitive Learning

6.1 Introduction

As we have seen that with the growing advent of software and technology, software quality assurance is a very critical activity in today's world. An imbalanced data problem can be tackled at the data level or algorithm level. At the data level, resampling methods are employed to balance the data before model construction. Chapter 4 deals with alleviating imbalanced classification problem at the data level and this chapter will try to address the solution to this problem at the algorithm level by employing a cost-sensitive approach. In a cost-sensitive approach, we can employ different meta cost learners in which False Positives are penalized at different cost factors in comparison to False Negatives. For the SDP problem, as stated earlier, positives are defective classes and negatives are non-defective

classes.

A detailed empirical investigation of oversampling methods and meta cost learners to deal with imbalanced classification problem in the SDP domain is conducted by Malhotra and Kamal [89] over well known 12 NASA datasets. They explored the cost ratio of 10, 30, and 50 in meta cost learners and evaluated performance based on ROC-AUC, Sensitivity, and precision. The cost-sensitive learning has been primarily performed on NASA datasets. We want to test the suitability of the cost-sensitive learning on software datasets like Apache for SDP in imbalanced software data. Related work is performed primarily for NASA datasets and they didn't explore G-Mean and Balance metrics that are preferred for imbalanced data. This motivates us to explore cost-sensitive learning for software engineering oriented datasets like Apache datasets with stable and reliable performance measures.

This chapter aims at proposing effective SDP for imbalanced OO software using different meta cost learners. This research study addresses the following RQs to understand and exploit cost-sensitive learning for effective defect prediction:

- RQ1: What is the performance of ML techniques without involving cost-sensitive learning?
- RQ2: Do cost-sensitive learning improve the defect prediction capability of ML models built for imbalanced data?
- RQ3: Which ML technique outperforms in predicting KStarthe software defects?

This research exploits nine ML techniques-NB, MLP, IBk, KStar, ABM1, Bag, RSS, RT, and J48 to predict defects in imbalanced data. Correlation feature selection is used to remove irrelevant features and ten-fold cross-validation is used to train and construct the model. Model prediction capability is analyzed at different cost ratios (MC10, MC15, MC20, MC25, MC30) to find the appropriate cost settings. When data is imbalanced, traditional metrics like accuracy gives biased results. Instead, metrics that emphasize both

positive and negative classes must be considered. Therefore, the model performances are evaluated based on stable performance measures like G-Mean, Balance, and ROC-AUC. This study also compares the results based on Sensitivity. The empirical evaluation of results advocates the use of a cost-sensitive approach for better defect prediction.

The remainder chapter is organized as follows: Section 6.2 summarizes the experimental framework required in the execution of the study. Results and their analysis are detailed in Section 6.3. Section 6.4 concludes the study with the discussion.

The part of this work is published in [211].

6.2 Research Methodology

This section comprises datasets, independent and dependent variables, feature selection technique, ML techniques, cost-sensitive learning approach, performance measures, and validation techniques used in this study.

6.2.1 Datasets and Variables

These open-source JAVA projects contributed by Jureczko and Madeyski [145] are downloaded by the Promise library [192]. Datasets used are Tomcat6.0, Synapse1.0, Ivy2.0, Jedit4.2, Xerces1.3, Camel1.6, Ant1.7, Jedit4.0, Log4j1.0, Synapse1.1, Synapse1.2, and Log4j1.1. These datasets are the same as those used in Chapter 4. Description of these datasets can be referred from Section 3.6.

The independent variables of the study are 20 OO metrics identified by Jureczko and Madeyski [145] and the binary dependent variable is a defect. These variables are explained in Section 3.5.

6.2.2 Model Development

This chapter uses CFS [146] as the feature selector. It is one of the widely used methods for feature selection in ML applications as a feature selector. Features selected by CFS for the considered datasets are mentioned in section 3.7.2.1. The models built with help of these selected metrics will tend to classify defects appropriately and in less computation time. Model validation is carried out by ten-fold cross-validation. The use of ten-fold cross-validation reduces validation bias [212].

Now, cost-sensitive learning can be conducted in two ways; either by adding weights to samples or by using a cost matrix to penalize type-I and type-II errors. Meta cost learners were proposed by [157] in which penalization is done in cost matrix and training instance is relabeled based on the majority voting. These errors are false positives and false negatives in the model prediction. In this work, meta-cost classifiers are used and the cost penalization of wrongly predicted defective classes is done at different levels- 10,15, 20, 25, and 30 times the cost of wrongly predicted non-defective classes.

Models are developed using nine ML techniques. These techniques are NB, MLP, IBk, KStar, ABM1, Bag, RSS, RT, and J48. J48 is the base ML technique for ensembles used in this study. NB, MLP, IBk, KStar, ABM1, Bag, RSS, RT are identified as the good defect predictors inn their category in Chapter 4. Categories of ML techniques are discussed in 3.9. Details of these nine techniques and their parameter settings can be referred from Section 3.9.

6.2.3 Performance Measures and Statistical Validation

In this chapter, we used Sensitivity, G-Mean [196], Balance [196], and ROC-AUC for performance evaluation of models. These measures are explained in detail in Section 3.11.

Statistical tests are necessary to test whether there is a statistical difference in the perfor-

mance of investigated ML techniques, with and without cost-sensitive learning. Ten-fold cross-validation and nonparametric statistical tests are used in this study to alleviate the conclusion validation threat. To achieve this purpose Wilcoxon signed-rank test [209] is used for statistical analysis at a level of significance of $\alpha = 0.5$. The pairwise comparison is made between each pair of ML techniques for both cases.

We conducted Friedman test [213] to find out the answer to RQ3. In multiple comparison with the Friedman test, we determined if the performance of the different ML techniques is the same or different. This assists in ranking ML techniques according to their performances in various meta-cost learners.

These tests are explained in section 3.12.

6.3 Result and Analysis

This section summarizes the results and provides answers to the research questions raised in the Introduction section.

6.3.1 RQ1: What is the performance of used ML techniques without cost-sensitive learning?

To answer RQ1, ML models are built on 12 datasets, and performances of these models are compared on basis of Sensitivity, G-mean, Balance, and ROC-AUC. NB is the statistical technique while MLP is the only neural network that we have assessed in this thesis. Nearest neighbors and ensemble methods used in this chapter are ranked in the top six ML techniques in Chapter 4.

The performance of various ML techniques are recorded in Table 6.1, 6.2, 6.3, and 6.4. The maximum value of a particular performance measure is highlighted in bold typeface for each dataset.

Sensitivity Analysis:

Table 6.1 presents Sensitivity values attained by different ML techniques without using meta cost learners. On analysis of Table 6.1, the highest Sensitivity value of 0.70 is achieved by Log4j1.1 dataset with IBk technique. There is no other dataset that could attain a Sensitivity value greater than 0.70 with any ML technique. Only 24.07 % of models got a Sensitivity value greater than 0.5. Only five ML techniques- NB, IBk, Bag, J48, and RT were able to achieve a Sensitivity value greater than 0.5 in at least 25% of ML models. Not a single dataset got a Sensitivity value greater than 0.65 for MLP, KStar, ABM1, Bag, RSS, and J48. NB, MLP, and IBk attained a 0.65 value of Sensitivity only for the Log4j1.1 dataset.

Table 6.1: Sensitivity Results of ML Techniques without cost-sensitive learning

Datasets	NB	MLP	IBk	KStar	ABM1	Bag	RSS	J48	RT
Synapse1.2	0.56	**0.63**	0.55	0.55	0.58	0.58	0.49	0.6	0.55
Synapse1.1	0.5	0.47	**0.55**	0.48	0.48	0.47	0.32	0.43	**0.55**
Log4j1.0	0.53	0.38	0.44	0.32	0.44	**0.56**	0.35	0.44	0.41
Log4j1.1	0.68	0.65	**0.70**	0.54	0.62	0.54	0.57	0.59	0.57
Jedit4.0	0.31	0.41	**0.56**	0.43	0.45	0.48	0.41	0.48	0.48
Ant1.7	0.5	0.46	0.51	0.42	0.48	0.49	0.47	**0.57**	0.51
Camel1.6	0.21	0.13	0.34	0.23	0.32	0.22	0.03	0.19	**0.35**
Xerces1.3	0.39	0.35	**0.49**	0.43	0.42	0.39	0.26	0.36	0.43
Jedit4.2	**0.40**	0.31	0.38	0.29	0.38	0.31	0.13	0.23	0.29
Ivy2.0	**0.43**	0.18	0.4	0.28	0.3	0.25	0.05	0.13	0.38
Synapse1.0	**0.56**	0.31	0.31	0.25	0.19	0.19	0	0.06	0.25
Tomcat6.0	**0.38**	0.1	0.23	0.16	0.22	0.19	0.04	0.06	0.23

NB demonstrated the best classification based on Sensitivity for Jedit4.2, Ivy2.0, Synapse1.0, and Tomcat6.0. NB got value of 0.40, 0.43, 0.56, and 0.38 for Jedit4.2, Ivy2.0, Synapse1.0, and Tomcat6.0. Synapse1.1 achieved the highest Sensitivity value of 0.55 with IBk and RT. IBk also gave the highest value of 0.70, 0.56, and 0.49 for Log4j1.1, Jedit4.0, and Xerces1.3

respectively. Log4j1.0 got the highest value of 0.56 with Bag and Ant1.7 attained the highest Sensitivity value of 0.57 with J48. Camel1.6 dataset has a low range of Sensitivity from 0.03 to 0.35. A Sensitivity value of 0.35 is demonstrated by an ensemble technique (RT). One of the primary reasons for low values of Sensitivity is the imbalanced classification problem.

Range obtained for Synapse1.2, Synapse1.1, Log4j1.0, Log4j1.1, Jedit4.0, Ant1.7, Camel1.6, Xerces1.3, Jedit4.2, Ivy2.0, Synapse1.0, and Tomcat6.0 is 0.49-0.63, 0.32-0.55, 0.32-0.56, 0.54-0.70, 0.31-0.56, 0.42-0.57, 0.03-0.35, 0.26-0.49, 0.13-0.40, 0.05-0.43, 0.00-0.56, and 0.04-0.38 respectively with various ML techniques.

G-Mean Analysis:

The performance of the developed model can be considered acceptable if G-Mean is equal to greater than 50% owing to the skewness in the data. G-Mean values obtained for the models are depicted in Table 6.2. The median G-Mean value achieved is 0.61 for SDP models developed using ML techniques. The highest G-mean value achieved is 0.79. This value is obtained by NB and MLP for the Log4j1.1 dataset. Seven datasets achieved the highest G-Mean values on the application of the NB technique. With NB, Synapse1.1, Log4j1.0, Log4j1.1, Jedit4.2, Ivy2.0, Synapse1.0, and Tomcat6.0 attained highest G-Mean value of 0.66, 0.71, 0.79, 0.61, 0.62, 0.71, and 0.59 respectively. IBk got the highest G-Mean value of 0.66, 0.70, 0.54, and 0.62 for Synapse1.1, Jedit4.0, Camel1.6, and Xerces1.3 respectively. MLP has also attained the highest G-Mean value for three datasets each. MLP got G-Mean value of 0.71, 0.66, and 0.79 for Synapse1.2, Synapse1.1, and Log4j1.1 respectively.

Table 6.2: G-Mean Results of ML Techniques without cost-sensitive learning

Datasets	NB	MLP	IBk	KStar	ABM1	Bag	RSS	J48	RT
Synapse1.2	0.70	**0.71**	0.66	0.67	0.68	**0.71**	0.65	0.7	0.66
Synapse1.1	**0.66**	**0.66**	**0.66**	0.64	0.65	0.65	0.55	0.61	0.65
Log4j1.0	**0.71**	0.59	0.62	0.54	0.62	**0.71**	0.58	0.64	0.58

Datasets	NB	MLP	IBk	KStar	ABM1	Bag	RSS	J48	RT
Log4j1.1	**0.79**	**0.79**	0.75	0.68	0.71	0.69	0.74	0.73	0.67
Jedit4.0	0.53	0.62	**0.70**	0.61	0.64	0.67	0.63	0.65	0.64
Ant1.7	0.68	0.65	0.66	0.62	0.65	0.67	0.66	**0.71**	0.65
Camel1.6	0.44	0.35	**0.54**	0.45	0.53	0.46	0.18	0.42	**0.54**
Xerces1.3	0.6	0.57	**0.61**	0.65	0.63	0.61	0.51	0.59	0.64
Jedit4.2	**0.61**	0.55	0.59	0.52	0.59	0.55	0.35	0.47	0.51
Ivy2.0	**0.62**	0.41	0.61	0.5	0.53	0.49	0.22	0.35	0.57
Synapse1.0	**0.71**	0.55	0.53	0.48	0.42	0.43	0	0.24	0.47
Tomcat6.0	**0.59**	0.32	0.46	0.39	0.46	0.44	0.2	0.25	0.47

As evident from Table 6.2,considering the mean value achieved by ML techniques, NB, MLP, IBk, Bag, RT may have better performance for the datasets used in model development. In particular, RSS, and J48 depicted very low performance in terms of G-Mean. RSS is not able to classify defects in the Camel1.6, Jedit4.2, Ivy2.0, Synapse1.0, and Tomcat6.0.

Balance Analysis:

Table 6.3 scribes the Balance values obtained for the ML models for different datasets when FN and FP have an equal cost of 1.

Table 6.3: Balance Results of ML Techniques without cost-sensitive learning

Datasets	NB	MLP	IBk	KStar	ABM1	Bag	RSS	J48	RT
Synapse1.2	67.44	**70.13**	64.95	65.28	66.83	69.14	62.36	69.04	64.78
Synapse1.1	63.68	61.99	**65.25**	61.87	62.43	61.65	51.59	58.89	64.51
Log4j1.0	66.61	55.97	59.29	51.55	59.29	**68.17**	54.16	60.09	56.54
Log4j1.1	**76.55**	75.07	74.33	66.06	69.92	66.33	69.36	70.52	66.06
Jedit4.0	50.67	58.23	**67.65**	58.3	60.81	62.86	58.35	62.24	61.87
Ant1.7	64.16	61.32	63.66	58.52	62.07	63.32	62.13	**68.55**	63.27
Camel1.6	43.82	38.21	52.16	44.74	51.13	44.95	31.54	42.22	**52.48**
Xerces1.3	56.67	53.74	**63.66**	59.93	58.71	56.84	47.73	54.8	59.77
Jedit4.2	**57.05**	51.35	55.58	49.61	55.58	51.31	38.12	45.39	49.28
Ivy2.0	**58.88**	41.65	57.25	48.42	50.35	46.84	32.82	38.12	54.84

Datasets	NB	MLP	IBk	KStar	ABM1	Bag	RSS	J48	RT
Synapse1.0	**68.04**	51.26	50.95	46.78	42.33	42.49	29.29	33.62	46.5
Tomcat6.0	**55.60**	36.63	45.53	40.24	44.78	43.04	32.05	33.88	45.57

Only 39.81% of models have a Balance value greater than 60. No ML technique can secure a Balance value greater than 75 except for NB and MLP. In NB and MLP also, only one dataset, Log4j1.1, attained the Balance value of 76.55 and 75.07 respectively.

NB outperformed the other ML techniques by giving the maximum Balance value of 76.55, 57.05, 58.88, 68.04, and 55.60 for Log4j1.1, Jedit4.2, Ivy2.0, Synapse1.0, and Tomcat6.0 respectively. IBk gave the maximum value for Synapse1.1, Jedit4.0, Xerces1.3 of 65.25, 67.65, and 63.66 respectively. Bag, J48, and RT also achieved the highest Balance value for one dataset each.

The range obtained for Synapse1.2, Synapse1.1, Log4j1.0, Log4j1.1, Jedit4.0, Ant1.7, Camel1.6, Xerces1.3, Jedit4.2, Ivy2.0, Synapse1.0, and Tomcat6.0 is 62.36-70.13, 51.59-65.25, 51.55-68.17, 66.06-76.55, 50.67-67.65, 58.52-68.55, 31.54-52.48, 47.73-63.66, 38.12-57.05, 32.82-58.88, 29.29-68.04, and 32.05-55.60 respectively with various ML techniques.

ROC-AUC Analysis:

From Table 6.4 this can be observed that ROC-AUC values lie in the range of 0.58-0.86. The median ROC-AUC value attained for all the models is 0.74. It can be deduced from Table 6.4 that NB, MLP, and RSS performed better than other ML techniques. The minimum value of ROC-AUC is obtained by RT (by 0.58) for Synapse1.0, and Tomcat6.0 each.

Two-third of the models have ROC-AUC greater than 0.7. NB got the highest values of 0.83, 0.86, 0.74, and 0.78 for Log4j1.0, Log4j1.1, Synapse1.0, Tomcat6.0 respectively. MLP achieved the highest value of 0.79, 0.81, 0.74, and 0.78 for Jedit4.0, Ant1.7, Synapse1.0, and Tomcat6.0 respectively. Unlike threshold dependent metrics, ROC-AUC performed better for the majority of datasets for the RSS technique. RSS achieved good

value of ROC-AUC of 0.83, 0.83, 0.81, and 0.78 for Xerces1.3, Jedit4.2, Ivy2.0, and Tomcat6.0 respectively. Apart from these three ML techniques, classic ensemble methods-ABM1 and Bag also got the highest value for few datasets.

Table 6.4: ROC-AUC Results of ML Techniques without cost-sensitive learning

Datasets	NB	MLP	IBk	KStar	ABM1	Bag	RSS	J48	RT
Synapse1.2	0.78	0.76	0.67	0.71	0.75	**0.80**	0.76	0.74	0.67
Synapse1.1	0.75	0.77	0.7	0.72	**0.78**	0.77	0.75	0.66	0.66
Log4j1.0	**0.83**	0.73	0.64	0.72	0.71	0.77	0.78	0.65	0.62
Log4j1.1	**0.86**	0.82	0.75	0.8	0.76	0.82	0.85	0.72	0.68
Jedit4.0	0.76	**0.79**	0.72	0.76	0.78	0.74	0.78	0.67	0.67
Ant1.7	0.8	**0.81**	0.69	0.77	0.78	0.8	0.8	0.74	0.67
Camel1.6	0.68	0.69	0.64	0.65	0.71	**0.72**	0.69	0.59	0.59
Xerces1.3	0.78	0.72	0.77	0.78	0.76	0.8	**0.83**	0.6	0.68
Jedit4.2	0.83	0.82	0.66	0.75	0.75	0.82	**0.84**	0.69	0.59
Ivy2.0	0.79	0.73	0.68	0.72	0.73	**0.81**	**0.81**	0.72	0.62
Synapse1.0	**0.74**	**0.74**	0.66	0.72	0.64	0.59	0.59	0.59	0.58
Tomcat6.0	**0.78**	**0.78**	0.64	0.72	0.73	0.77	**0.78**	0.67	0.58

The range of ROC-AUC observed for Synapse1.2, Synapse1.1, Log4j1.0, Log4j1.1, Jedit4.0, Ant1.7, Camel1.6, Xerces1.3, Jedit4.2, Ivy2.0, Synapse1.0, and Tomcat6.0 is 0.67-0.80, 0.66-0.78, 0.62-0.83, 0.68-0.86, 0.67-0.79, 0.67-0.81, 0.59-0.72, 0.6-0.83, 0.59-0.84, 0.62-0.81, 0.58-0.74, and 0.58-0.78 respectively.

6.3.2 RQ2: Do cost-sensitive learning improve the defect prediction capability of ML models build for imbalanced data?

To answer this RQ, we first need to analyze the performance of ML techniques with cost-sensitive learning. The cost matrix of each classifier is penalized. Different cost factors of 10, 15, 20, 25, and 30 are tried and the program was designed to select the cost factor that

yields the best defect prediction. Table 6.5, 6.6, 6.7, 6.8 holds the performance statistics of ML models corresponding to the best cost factor settings for Sensitivity, G-Mean, Balance, and ROC-AUC. The results seem promising in the field of SDP for imbalanced data. Meta cost learners have handled the class imbalance issue with cost penalization.

Sensitivity Analysis: By analyzing Table 6.5, we found that the mean Sensitivity value becomes 0.66 and overall Sensitivity ranged between 0.19 and 0.97.

Table 6.5: Sensitivity Results of ML Techniques with cost-sensitive learning

Dataset	NB	MLP	IBk	KStar	ABM1	Bag	RSS	J48	RT
Synapse1.2	0.66	0.9	0.55	0.83	0.69	0.87	**0.97**	0.88	0.52
Synapse1.1	0.7	0.78	0.55	0.83	0.53	0.83	**0.95**	0.75	0.5
Log4j1.0	0.65	0.76	0.44	0.71	0.53	0.85	**0.91**	0.76	0.41
Log4j1.1	0.73	0.92	0.7	0.84	0.78	0.86	**0.97**	0.78	0.68
Jedit4.0	0.4	0.8	0.56	0.68	0.55	0.83	**0.93**	0.68	0.52
Ant1.7	0.6	**0.89**	0.51	0.73	0.54	0.77	0.87	0.83	0.5
Camel1.6	0.32	0.89	0.36	0.63	0.47	0.73	**0.93**	0.76	0.38
Xerces1.3	0.57	0.72	0.51	0.72	0.64	**0.83**	0.81	0.75	0.54
Jedit4.2	0.4	0.8	0.56	0.68	0.55	0.83	**0.93**	0.68	0.52
Ivy2.0	0.5	0.68	0.4	0.68	0.38	0.65	**0.73**	0.65	0.43
Synapse1.0	0.75	0.63	0.31	0.44	0.31	0.69	**0.88**	0.69	0.19
Tomcat6.0	0.69	**0.75**	0.23	0.49	0.29	0.65	**0.75**	0.61	0.3

After applying meta cost learners, performance of RSS improved the maximum for all the datasets. RSS gave maximum Sensitivity value for 83% of datasets. Synapse1.2, Synapse1.1, Log4j1.0, Log4j1.1, Jedit4.0, Camel1.6, Jedit4.2, Ivy2.0, Synapse1.0, and Tomcat6.0 got 0.97, 0.95, 0.91, 0.97, 0.93, 0.93, 0.93, 0.73, 0.88, and 0.75 respectively with RSS. Remaining two datasets- Ant1.7 and Tomcat6.0 got highest Sensitivity value of 0.89 and 0.75 with MLP.

Minimum Sensitivity increased from 0 to 0.19 and the maximum Sensitivity achieved by any ML technique experienced a minimum of 38.47% of increase. The improvement

in cost-sensitive models is marked by a huge increment of 71.22% in the mean Sensitivity value.

G-Mean Analysis:

The value of minimum G-Mean predicted by models amplified from 0 to 0.30 whereas the maximum G-Mean value raised by 5.43%. The range of G-Mean now becomes 0.30-0.84. The mean values for G-Mean boosted by 14.3%.

Bag achieved highest G-Mean value of 0.70, 0.72, 0.74, 0.84, and 0.72 for Synapse1.1, Jedit4.0, Ant11.7, Camel1.6, Jedit4.2 respectively. RSS performed the best for Xerces1.3, Ivy2.0, Synapse1.0, and Tomcat6.0 and got G-Mean value of 0.75, 0.75, 0.81, and 0.77 respectively. Log4j versions performed best with NB. Synapse1.2 and Xerces1.3 gained maximum G-Mean values of 0.74 and 0.75 respectively.

Comparison of these values with Table 6.2 gives a clear indication of improvement in model predictions with cost-sensitive learning.

Table 6.6: G-Mean Results of ML Techniques with cost-sensitive learning

Dataset	NB	MLP	IBk	KStar	ABM1	Bag	RSS	J48	RT
Synapse1.2	0.73	0.57	0.66	0.61	**0.74**	0.68	0.3	0.66	0.64
Synapse1.1	0.69	0.66	0.66	0.64	0.65	**0.70**	0.33	0.63	0.65
Log4j1.0	**0.77**	0.71	0.62	0.61	0.65	0.7	0.59	0.69	0.6
Log4j1.1	**0.81**	0.58	0.75	0.65	0.76	0.58	0.33	0.53	0.75
Jedit4.0	0.6	0.67	0.7	0.68	0.68	**0.72**	0.48	0.66	0.66
Ant1.7	0.71	0.65	0.66	0.69	0.68	**0.74**	0.7	0.7	0.65
Camel1.6	0.54	0.52	0.55	0.6	0.63	**0.84**	0.54	0.63	0.57
Xerces1.3	0.71	0.74	0.66	**0.75**	0.72	0.73	**0.75**	0.73	0.67
Jedit4.2	0.6	0.67	0.7	0.68	0.68	**0.72**	0.48	0.66	0.66
Ivy2.0	0.67	0.73	0.61	0.7	0.59	0.72	**0.75**	0.71	0.62
Synapse1.0	0.78	0.66	0.53	0.57	0.53	0.73	**0.81**	0.72	0.42
Tomcat6.0	0.71	0.73	0.46	0.63	0.52	0.73	**0.77**	0.7	0.53

Balance Analysis: On analysis of Table 6.7, Balance results seem similar to G-Mean

for the various ML techniques.

Table 6.7: Balance Results of ML Techniques with cost-sensitive learning

Dataset	NB	MLP	IBk	KStar	ABM1	Bag	RSS	J48	RT
Synapse1.2	72.06	54.47	64.95	59.4	**73.90**	65.11	35.9	62.89	62.77
Synapse1.1	**68.93**	65.04	65.07	62.32	63.82	68.87	37.48	62.41	63.13
Log4j1.0	**74.26**	70.38	59.29	60.48	63.62	68.15	55.45	68.03	57.27
Log4j1.1	**79.34**	54.46	74.33	62.83	75.88	55.75	37.12	52.31	74.22
Jedit4.0	57.04	65.44	67.47	68.2	66	**70.69**	46.84	65.98	64.32
Ant1.7	69.55	62.12	63.66	68.66	66.01	**73.38**	67.95	68.58	62.75
Camel1.6	51.43	49.93	53.6	60.27	60.99	**80.92**	51.48	62.47	55.05
Xerces1.3	68.38	74.31	63.91	**75.36**	71.24	72.26	74.82	73.28	65.51
Jedit4.2	57.04	65.44	67.47	68.2	66.00	**70.69**	46.84	65.98	64.32
Ivy2.0	63.72	72.95	57.25	69.87	55.6	71.54	**75.19**	70.85	58.7
Synapse1.0	78.03	65.5	50.95	55.9	51.02	72.69	**79.9**	71.78	42.33
Tomcat6.0	71.3	73.11	45.52	61.6	49.33	72.28	**77.2**	68.87	50.11

In particular, the predictive capability of Bag, NB, KStar, ABM1, and RSS for Balance are the same as with G-Mean. Bag got the highest Balance value for Jedit4.0, Ant1.7, Camel1.6, and Jedit4.2 of 70.69, 73.38, 80.92, and 70.69 respectively. Without cost-sensitive application, these values were 62.86, 63.32, 44.95, and 51.31. The range of Balance attained by cost-sensitive models is from 35.9 to 80.92. RSS and NB also secured the highest Balance value for three datasets each. RSS got the highest Balance value of 75.19, 79.9, and 77.2 for Ivy2.0, Synapse1.0, and Tomcat6.0 respectively. NB achieved the highest value for Synapse1.1, Log4j1.0, and Log4j1.1 of 68.93, 74.26, and 79.34 respectively. The mean value of Balance increased from 55.65 to 63.86 with a percentage increment of approximately 15% considering all the 108 models. Comparison of Table 6.3 with Table 6.7 exhibits the improvement in values of Balance for different ML models.

ROC-AUC Analysis:

Table 6.8 scribes the ROC-AUC values obtained by cost-sensitive models with the high-

est values in boldface.

Table 6.8: ROC-AUC Results of ML Techniques with cost-sensitive learning

Dataset	NB	MLP	IBk	KStar	ABM1	Bag	RSS	J48	RT
Synapse1.2	0.78	0.76	0.71	0.74	0.78	**0.79**	0.74	0.74	0.63
Synapse1.1	0.75	0.76	0.7	0.75	0.77	**0.80**	0.75	0.69	0.71
Log4j1.0	**0.83**	0.81	0.66	0.7	0.72	0.78	0.78	0.73	0.69
Log4j1.1	**0.86**	0.82	0.73	0.79	0.79	0.78	0.8	0.67	0.75
Jedit4.0	0.76	0.75	0.72	0.75	0.78	**0.81**	0.76	0.69	0.68
Ant1.7	0.8	0.8	0.69	0.77	0.8	**0.83**	0.82	0.75	0.67
Camel1.6	0.68	0.7	0.64	0.65	0.71	0.71	**0.73**	0.66	0.61
Xerces1.3	0.78	0.78	0.77	**0.84**	0.81	0.83	0.83	0.78	0.71
Jedit4.2	0.76	0.75	0.72	0.75	0.78	**0.81**	0.76	0.69	0.68
Ivy2.0	0.79	0.8	0.68	0.75	0.79	0.8	**0.82**	0.73	0.7
Synapse1.0	0.74	0.67	0.56	0.69	0.63	0.75	**0.78**	0.69	0.62
Tomcat6.0	0.78	0.79	0.59	0.74	0.78	0.82	**0.83**	0.72	0.59

The mean ROC-AUC value attained is 0.74. Bag and RSS rule the statistics represented in Table 6.8 but performances of ABM1, NB, and MLP are also noteworthy. It is evident on comparison of Table 6.4 and Table 6.8 that, for all the datasets, their ROC-AUC were quite similar. Though there is not much improvement in the performance of NB and MLP using meta cost learner, their comparative performance seems significant.

Bag got highest value for Synapse1.2, Synapse1.1, Jedit4.0, Ant1.7, and Jedit4.2 of 0.79, 0.80, 0.81, 0.83, and 0.81. For the datasets with a much lower percentage of defective classes, the ensemble technique-RSS performed better. It gave the highest values for Camel1.6 (0.73), Ivy2.0 (0.82), Synapse1.0 (0.78), and Tomcat6.0 (0.83). For Xerces1.3, KStar gained the maximum ROC-AUC value of 0.84 which is comparable to the value achieved by Bag and RSS. Bag and RSS both got 0.83 ROC-AUC value for Xerces1.3.

Statistical Validation of RQ2:

The above analysis exhibits the improvement in values of all the performance mea-

sures. Still, these results are required to be statistically validated. We perform a statistical test to compare the cumulative performance of ML techniques without cost-sensitive learning (NotCS) and the cumulative performance of ML techniques with cost-sensitive learning (CS). This requires comparing two scenarios, therefore we employed the Wilcoxon-signed rank test and found considerable statistical improvement in SDP models using cost-sensitive learning for all the performance measures.

Wilcoxon signed-ranks for the pairs are scribed in Table 6.9 with their p-values for different performance measures. Significant p-values are boldfaced. A p-value of less than 0.05 indicates that results are 95% statistically significant. S+ signifies the significant increment in the value of a particular performance measure when the ML model incorporates cost-sensitive learning.

Table 6.9: Wilcoxon signed-rank results for various performance measures

Dataset	ROC-AUC	G-Mean	Balance	Sensitivity
CS vs NoCS	S+	S+	S+	S+
p-value	**0.015**	**0.008**	**0.008**	**0.008**

The p-value for ROC-AUC is 0.015. For G-Mean, Balance, and Sensitivity this value is 0.008. As these values are less than 0.05, therefore we reject the null hypothesis that there is no difference between the performance of ML models and their cost-sensitive versions. Therefore, it is concluded that cost-sensitive learning enhances the defect prediction capability of the ML models built for imbalanced data.

6.3.3 RQ3: Which ML technique outperforms in predicting software defects?

The nature of ML techniques is analyzed for both the cases - without and with cost-sensitive learning. To observe whether there is a statistical change in performances of different

classifiers, the Friedman test is performed at the $\alpha = 0.05$ level of significance and their ranks for various performance measures corresponding to cost-sensitive are recorded in Table 6.10 and Table 6.11. The performance of ML techniques is ranked with help of the Friedman test based on ROC-AUC, Balance, G-Mean, and Sensitivity. Mean ranks are displayed in brackets for each ML technique. Higher the mean rank, the better the performance of the ML technique. Kendall's coefficient of concordance (Kendall C) is also reported in Table 6.10 and Table 6.11 to gauge the effect.

ML Performance without cost-sensitive learning:

The performance of various ML techniques without exploiting cost-Sensitivity are recorded in Table 6.1, 6.2, 6.3, and 6.4 and analyzed in subsection 6.3.1.

Different hypothesis for different performance measure is set. The hypotheses formed to achieve the rankings of ML techniques are stated as:

- H_{10} (Null Hypothesis): There is no significant statistical difference between the performance of any of the SDP models developed using ML techniques in terms of Sensitivity.
- H_{1a} (Alternate Hypothesis): There is a significant statistical difference between the performance of SDP models developed using ML techniques in terms of Sensitivity.
- H_{20} (Null Hypothesis): There is no significant statistical difference between the performance of any of the SDP models developed using ML techniques in terms of G-Mean.
- H_{2a} (Alternate Hypothesis): There is a significant statistical difference between the performance of SDP models developed using ML techniques in terms of G-Mean.
- H_{30} (Null Hypothesis): There is no significant statistical difference between the performance of any of the SDP models developed using ML techniques in terms of Balance.

- H_{3a} (Alternate Hypothesis): There is a significant statistical difference between the performance of SDP models developed using ML techniques in terms of Balance.

- H_{40} (Null Hypothesis): There is no significant statistical difference between the performance of any of the SDP models developed using ML techniques in terms of ROC-AUC.

- H_{4a} (Alternate Hypothesis): There is a significant statistical difference between the performance of SDP models developed using ML techniques in terms of ROC-AUC.

Friedman rankings in Table 6.10 for threshold-dependent performance measures have demonstrated that Ibk and NB have the best predictive capability for software defects. This is analogous to our observations in subsection 6.3.1. Their performances are not affected by the imbalanced nature of data. IBk is the nearest neighbor ML technique and classifies a class as defective or non-defective based on its nearest neighbor class. NB achieved the first rank with ROC-AUC as well. NB is the robust ML technique and performs probability-based classification.

Table 6.10: Friedman Results for ML Techniques without Cost-Sensitive Learning

Rank	Sensitivity	G-Mean	Balance	ROC-AUC
Rank 1	IBk (7.54)	IBk (7.16)	IBk (7.25)	NB (7.37)
Rank 2	NB (6.62)	NB (6.91)	NB (6.91)	RSS (7.00)
Rank 3	RT (6.12)	Bag (5.75)	Bag (5.5)	MLP (6.87)
Rank 4	ABM1 (5.75)	ABM1 (5.25)	RT (5.45)	Bag (6.75)
Rank 5	Bag (5.00)	RT (4.91)	ABM1 (5.41)	ABM1 (5.66)
Rank 6	J48 (4.25)	MLP (4.83)	MLP (4.41)	KStar (4.58)
Rank 7	MLP (4.16)	J48 (4.25)	J48 (4.33)	IBk (2.83)
Rank 8	KStar (3.95)	KStar (3.75)	KStar (3.87)	J48 (2.62)
Rank 9	RSS (1.58)	RSS (2.16)	RSS (1.83)	RT (1.29)
p-value	**0.000**	**0.000**	**0.000**	**0.000**
Kendall C	0.424	0.320	0.358	0.683

The ensemble methods Bag, ABM1, and RT grabbed the position in the first five rankers for Sensitivity, G-Mean, and Balance. RSS secured a second position with ROC-AUC. It performed the worst at a threshold of 0.5 in terms of Sensitivity, G-Mean, and Balance. MLP, though achieved the highest performance value in few datasets for the considered performance measures, failed to secure any good rank because its predictive capability decreased with the drop in the percentage of defective classes in the software.

The p-value for Sensitivity, G-Mean, Balance, and ROC-AUC is 0.000. This signifies that the rankings are highly significant at $\alpha = 0.05$. Therefore, we refute the null hypotheses and accept alternate hypotheses H_{1a}, H_{2a}, H_{3a}, and H_{4a}. Kendall C ranges from 0.32 to 0.683 and supports the dissimilarity amongst the ML techniques.

ML Performance with Cost-sensitive Learning:

Subsection 6.3.2 has proved statistically that ML models generated with cost-sensitive learning gave better defect predictions that the original ones. Friedman test is executed at $\alpha = 0.5$ to find the rankings of ML techniques. Performance of ML models with the best classification ratio is considered for each performance measure on 12 datasets for Sensitivity, G-Mean, Balance, and ROC-AUC.

The performance of various ML techniques with cost-sensitive learning are presented in recorded in Table 6.5, 6.6, 6.7, and 6.8. Detailed analysis is provided earlier in subsection 6.3.2.

The hypotheses set to find the competency of ML techniques for SDP are:

- H_{10} (Null Hypothesis): There is no significant statistical difference between the performance of any of the SDP models developed using ML techniques with cost-sensitive learning in terms of Sensitivity.

- H_{1a} (Alternate Hypothesis): There is a significant statistical difference between the performance of SDP models developed using ML techniques with cost-sensitive learning in terms of Sensitivity.

- H_{20} (Null Hypothesis): There is no significant statistical difference between the performance of any of the SDP models developed using ML techniques with cost-sensitive learning in terms of G-Mean.
- H_{2a} (Alternate Hypothesis): There is a significant statistical difference between the performance of SDP models developed using ML techniques with cost-sensitive learning in terms of G-Mean.
- H_{30} (Null Hypothesis): There is no significant statistical difference between the performance of any of the SDP models developed using ML techniques with cost-sensitive learning in terms of Balance.
- H_{3a} (Alternate Hypothesis): There is a significant statistical difference between the performance of SDP models developed using ML techniques with cost-sensitive learning in terms of Balance.
- H_{40} (Null Hypothesis): There is no significant statistical difference between the performance of any of the SDP models developed using ML techniques with cost-sensitive learning in terms of ROC-AUC.
- H_{4a} (Alternate Hypothesis): There is a significant statistical difference between the performance of SDP models developed using ML techniques with cost-sensitive learning in terms of ROC-AUC.

Friedman rankings in Table 6.11 unanimously declare Bag as the best ML technique with stable performance measures (G-Mean, Balance, and ROC-AUC). Bag secured the third position with a mean rank of 6.95 with Sensitivity as the performance measure. Analysis of Table 6.11 enlightens the fact that after penalizing the wrongly predicted defective classes, models based on Bag, NB, and ABM1 are consistently better than the others for stable metrics.

Table 6.11: Friedman Results for ML Techniques with cost-sensitive learning

Rank	Sensitivity	G-Mean	Balance	ROC-AUC
Rank 1	RSS (8.79)	Bag (7.50)	Bag (7.33)	Bag (7.91)
Rank 2	MLP (7.16)	NB (5.75)	NB (5.83)	RSS (7.16)
Rank 3	Bag (6.95)	KStar (5.08)	KStar (5.41)	NB (6.41)
Rank 4	J48 (6.00)	MLP (4.91)	J48 (5.25)	ABM1 (6.12)
Rank 5	KStar (5.45)	ABM1 (4.91)	ABM1 (5)	MLP (5.87)
Rank 6	NB (3.58)	J48 (4.83)	MLP (4.75)	KStar (4.83)
Rank 7	ABM1 (3.00)	IBk (4.33)	IBk (4.16)	J48 (3.16)
Rank 8	IBk (2.29)	RSS (4.25)	RSS (4.08)	IBk (1.83)
Rank 9	RT (1.75)	RT (3.41)	RT (3.16)	RT (1.66)
p-value	**0.000**	**0.035**	**0.020**	**0.000**
Kendall C	0.807	0.173	0.190	0.697

Sensitivity values are affected more by penalization. According to Sensitivity, RSS has the highest mean rank of 8.79. RSS secured the second position with a mean rank of 7.16 with ROC-AUC measure. The p-value obtained for Sensitivity and ROC-AUC is 0.000. For G-Mean and Balance, the p-value corresponding to the Friedman test is 0.035 and 0.02 respectively. All the four p-values are less than 0.05, therefore, we reject the null hypothesis and conclude that the performances of ML techniques are statistically different. Kendall C for Sensitivity and ROC-AUC is 0.807 and 0.697 respectively. This high value of Kendall C signifies the agreement in their performance with respect to datasets. The behavior of ML techniques is different in different datasets for G-Mean and Balance. Kendall C is small for these performance measures. This value is 0.173 and 0.19 for G-Mean and Balance respectively.

As concluded earlier models employing cost-sensitive learning have improved the performance of ML techniques for SDP, we further conducted the Wilcoxon signed-rank test for ML techniques with cost-sensitive learning to explore whether the best ranker is statistically better than all other ML techniques. According to Table 6.11, Bag is the first ranker

with the stable metrics (G-Mean, Balance, and ROC-AUC). Therefore, we conducted a Wilcoxon signed-rank test to perform the pair-wise comparison of ML techniques with Bag. Table 6.12 shows the pair-wise comparison of Bag with other ML techniques.

Table 6.12: Wilcoxon Signed-Rank Results for ML Techniques with Cost-Sensitive Learning

Pair	G-Mean	Balance	ROC-AUC
Bag-NB	0.480	0.480	0.117
Bag-MLP	**0.034**	**0.041**	**0.033**
Bag-IBk	**0.019**	**0.019**	**0.002**
Bag-KStar	**0.023**	**0.034**	**0.005**
Bag-ABM1	0.071	0.084	**0.008**
Bag-RSS	**0.034**	**0.028**	0.480
Bag-J48	**0.003**	**0.006**	**0.002**
Bag-RT	**0.019**	**0.019**	**0.002**

Performance of Bag, when measured in terms of G-Mean, Balance, and ROC-AUC, is statistically better than MLP, IBk, KStar, J48, and RT. The corresponding p-values are less than 0.05 and hence the difference in predictive capability is statistically significant. The statistically significant p-values are bold-faced in Table 6.12.

The performance of the Bag is similar to NB for all three robust performance measures. Bag also performed comparable to ABM1 for G-Mean and Balance. With threshold independent ROC-AUC, RSS also performed similar to Bag as the p-value obtained is 0.480. Therefore, Bag, NB, ABM1, and RSS are good ML techniques to build SDP models with imbalanced data.

6.4 Discussion

This chapter determines the usage of the penalized cost matrix for effective defect prediction in imbalanced software data. MC10, MC15, MC20, MC25, and MC30 were employed to solve the purpose. The study incorporates model construction with ten-fold validation on nine ML methods and twelve datasets of the imbalanced nature. Important features were retained by employing the CFS technique. The empirical evaluation was executed with the help of ROC-AUC, G-Mean, Balance, and Sensitivity. The key results of this chapter are as follows:

- It is proved statistically with the help of the Wilcoxon-signed rank test that cost-sensitive models predict defects better than ML models without modifying cost ratios of FN and FP.
- With cost-sensitive learning, Bag, and NB statistically provided better predictions than other ML techniques in terms of G-Mean, Balance, and ROC-AUC. ABM1 results of ML models were statistically comparable with Bag and NB for G-Mean and Balance. RSS results were comparable with Bag and NB results for threshold independent ROC-AUC metric.
- The accuracy of cost-sensitive models in terms of Sensitivity are statistically better than that of the base ML models. The Sensitivity values of Synapse1.2, Synapse1.1, Log4j1.0, Log4j1.1, Jedit4.0, Ant1.7, Camel1.6, Xerces1.3, Jedit4.2, Ivy2.0, Synapse1.0, and Tomcat6.0 with Bag has improved by 50%, 78.55%, 52.63%, 60.02%, 72.23%, 58.01%, 228.55%, 111.12%, 164.54%, 160%, 266.67%, and 233.37% respectively.
- The cost-sensitive based ML models were found statistical significant than ML models in terms of G-Mean. Synapse1.1, Jedit4.0, Ant1.7, Camel1.6, Xerces1.3, Jedit4.2, Ivy2.0, Synapse1.0, and Tomcat6.0 with Bag experienced 8.4%, 7.7%, 9.9%, 82.24%,

19.69%, 30.9%, 48.19%, 71.4%, and 67.65% increase with the involvement of meta cost learners.

- Cost-sensitive learning statistically increased the Balance values of ML models. The Balance values of Synapse1.1, Jedit4.0, Ant1.7, Camel1.6, Xerces1.3, Jedit4.2, Ivy2.0, Synapse1.0, and Tomcat6.0 increased by 11.7%, 12.46%, 15.89%, 80.02%, 27.13%, 37.77%, 52.73%, 71.08%, and 67.93% with the Bag ensemble.

- ROC-AUC is stable and threshold independent measure. Therefore, it can capture the imbalanced nature of the data. The performances of ML techniques were statistically improvised with the incorporation of cost-Sensitivity in terms of ROC-AUC also for the majority of datasets. Synapse1.1, Log4j1.0, Jedit4.0, Ant1.7, Xerces1.3, Synapse1.0, and Tomcat6.0 shows 3.9%, 1.29%, 10.16%, 4.02%, 3.87%, 27.63%, and 7.03% of improvement in ROC-AUC values.

The main contribution of this chapter is

- to develop a useful and efficient SDP model for imbalanced data using cost-sensitive learning

- to use stable performance measures (G-Mean, Balance, and ROC-AUC) for model evaluation

- to statistically validate the results using nonparametric tests

- To incorporate CFS in defect prediction models to provide better and unbiased results.

Therefore, this chapter aids in understanding the role of cost-sensitive learning and building better SDP models with imbalanced software data.

Till now, various SDP models are empirically validated using ML techniques in the current thesis work. Next, we would like to explore the defect prediction framework with

different search-based techniques and assess their predictive capabilities in defect prediction.

Chapter 7

Software Defect Prediction Using Search-based Techniques

7.1 Introduction

Efficient defect prediction helps in the timely identification of areas in software which can lead to defects in software owing to better resource utilization [1]. More resources need to be allocated to defect prone areas in software development resulting in better quality software. Apart from ML techniques, now evolutionary computing is getting the attention of researchers to solve many complex problems that require exploring a large search space.

With many advantages like easy adaptations and computational efficiency, Harman [59] advocated analyzing the suitability and applicability of evolutionary techniques in the field of SDP. With the growing interest of researchers in these SBTs, this study tends to explore various SBTs and provide researchers and developers the set of SBTs that are effective in predicting probable defects in OO software. The SBTs that have low defect prediction capabilities are identified and their usage may not lead to desired results in software engineering problems. Though in 2018, Malhotra [4] has extensively analyzed the SBTs and

their hybrid versions for applicability in defect prediction domain, still very few studies exist in literature where SBTs are used for model development. This motivates us to employ SBTs in constructing models and assess their effectiveness in the field of SDP in this chapter.

The objective of this study is threefold;

1. to ensure the efficacy of SBTs in envisaging the defects in software
2. to explore the predictive potential of the addressed SBTs by employing stable performance measures like G-Mean, Balance, and ROC-AUC, and
3. to contribute a strong empirical study by validating results statistically.

The evaluation measures that are used in this study assist in the unbiased classification of imbalanced software data.

In this chapter, the performance of sixteen SBTs is examined and the following research questions are formed to achieve the objective:

- RQ1: What is the relative performance of addressed SBTs for predicting software defects using OO metrics when sensitivity, G-Mean, Balance, and ROC-AUC are considered?
- RQ2: Which SBTs are statistically good or bad defect predictors for OO projects when sensitivity, G-Mean, Balance, and ROC-AUC are considered?

Each SBT is run thirty times to incorporate its inherent stochastic nature. Ten-fold with-in project cross-validation is done. Further, results are statistically validated by using the nonparametric Friedman test. In addition to this, post-hoc analysis is carried out by Wilcoxon signed-rank test to eradicate family-wise errors. Nonparametric tests do not require to follow any assumptions for data distribution. Therefore, the Friedman test and

Wilcoxon signed-rank test are used considering the nonnormal behavior of predictors of the study.

There is a lack of studies in the literature that uses SBTs for defect classification. In 2014, Malhotra [214] found only eight SDP studies that used SBTs. Li et al. [63] had summarized the progress on different strategies in SDP until 2017 and recorded a very fewer number of studies addressing the use of SBTs. The nature of the SBTs makes them more pertinent to problems in the software engineering domain. The focus of the study is to look into SBTs with the perspective of model evaluation for defect prediction. With very few related studies, the research community needs such studies to ensure their practical applicability and usability for software practitioners in early defect detection. This motivates us to conduct the empirical assessment of SBTs for SDP with more software engineering datasets that are less explored in literature. Therefore, this study is important to be considered despite the existence of Malhotra's work [4] which is the closest to this work. This study is different from [4] because of the following reasons:

- Unlike [4] this study uses threshold independent metric: ROC-AUC [15]. ROC-AUC is used to mitigate the problem of imbalanced data in this study [40, 41].
- We have used software engineering datasets that are less explored in the literature to ascertain the usability of these SBTs. The datasets used in this study are not used in [4].

Results support the usage of SBTs like UCS, XCS, SGA, MPLCS, BIOHEL, GA_ADI, GA_INT, CHC, and PBIL for building good SDP models. SBTs can explore a larger search space for potential solutions and require less domain-specific knowledge while building the model. Most of the genetic-based SBTs performed better than PSO based SBTs. Genetic algorithms (GA) are advantageous because they do parallel computations and converge to the global optimal solution quickly. GA based SBTs performed comparatively better than PSO variants because defect prediction is a binary problem. The prediction can be either

defective or non-defective. GA performs better in discrete binary problems whereas PSO is effective in continuous domain problems.

Further, we will also like to explore the consequences of resampling methods with search-based techniques for the classification of software defects in next chapter.

The remainder of this chapter is divided into sections as follows: Section 7.2 expounds research framework designed to accomplish objectives set in this chapter. Experimental results are documented and analyzed in Section 7.3. Section 7.4 provides a discussion of the overall findings of the chapter.

The results of the chapter are published in [215].

7.2 Reaseach Framework

The subject of this study is finding the worth of applying SBTs for developing machine learning models. This section describes the components involved in this empirical study.

7.2.1 Research Variables and Datasets Details

OO metrics are selected as independent variables in this study owing to their wide acceptance in defect prediction literature that deftly captures the quality attributes of software [2, 3]. Independent variable of the study is discrete binary variable named 'Defect'. These research variables are explained in section 3.5.

17 datasets of JAVA projects are collected from promise repository [208]. Datasets used are Tomcat6.0, Synapse1.0, Ivy2.0, Jedit4.2, Xalan2.4, Xerces1.3, Xerces1.2, Camel1.6, Ant1.7, Jedit4.0, Log4j1.0, Jedit4.1, Synapse1.1, Synapse1.2, Log4j1.1, Xalan2.6,and Xalan2.4. Description of these datasets is provided in section 3.6. All datasets have less than 34% of defective classes except for Xalan2.6 and Xalan2.5. 85% of datasets are imbalanced with the % age of #D ranging from 8.97 to 33.59. Different datasets of varied size and nature

are used to train and validate the performance of 16 common SBTs.

7.2.2 Search-based Techniques

16 SBTs are selected for the performance evaluation so that fair comparison can be made to assess the effectiveness of SBTs in the software engineering domain. Addressed SBTs are introduced in section 3.9.2. Experiments are executed in the KEEL environment [205] with default parameter settings of SBTs to support the repeatability of the study. The default parameter settings are used in the chapter because according to Arcuri and Fraser [216] parameter tuning in SBTs is an expensive process. Several SBTs are used in this study; therefore, default parameter settings provide ease of reproducing it and strengthens conclusion validity.

7.2.3 Performance Measures and Statistical Validation

Instead of traditional measures like accuracy, we have used sensitivity, G-Mean [196], Balance [196], and ROC-AUC for performance evaluation. These measures are explained in detail in Section 3.11.

Ten-fold with-in the project cross-validation method elucidated in subsection 3.10.2 is used to reduce the validation bias. Apart from this, owing to the stochastic nature of SBTs, each technique is run 30 times as supported in the literature [201], and averaged results are presented. For statistical analysis of observed results, the nonparametric Friedman test [197] is used for sensitivity, G-Mean, Balance, and ROC-AUC. The post-hoc analysis is carried out by the Wilcoxon signed-rank test [209]. Details of these tests can be referred to from Section 3.12.

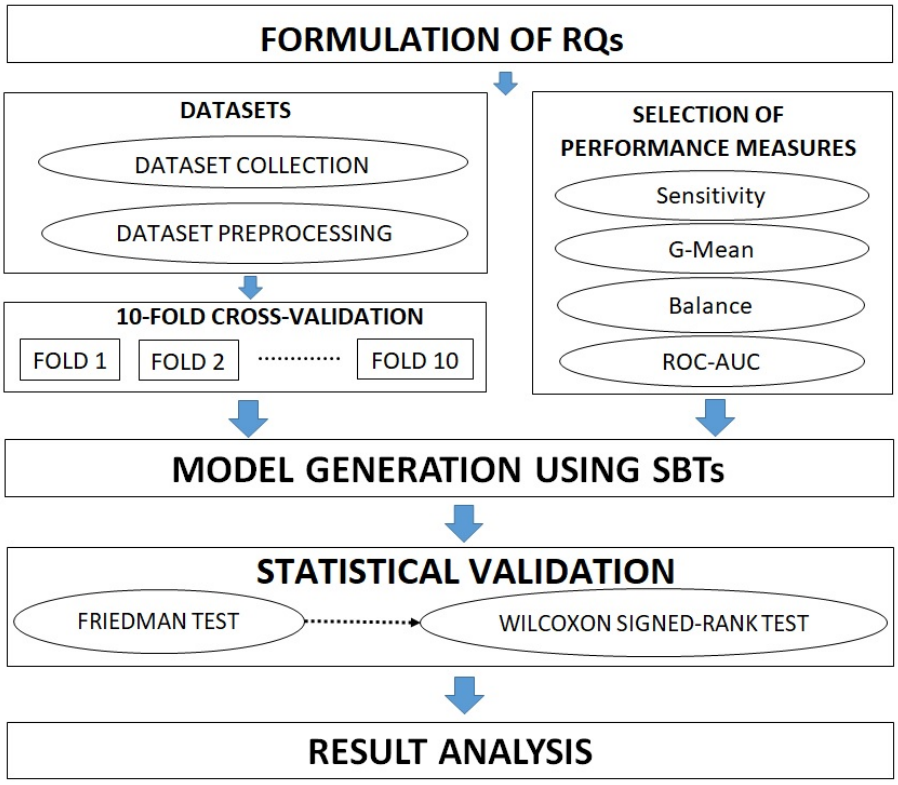

Figure 7.1: Experimental Setup

7.2.4 Experimental Setup

Figure 7.1 explains the experimental setup designed for the study. Once RQs are formulated keeping in mind the objective of the study, the datasets are collected and preprocessed to form dependent and independent variables. The research variables, datasets, SBTs and performance measures are summarized in Chapter 3.

16 SBTs are exploited to build models classifying the defects using ten-fold cross-validation. The performances of models are assessed based on selected performance measures and results are validated with the Friedman Test and Wilcoxon signed-rank test. The pseudocode for search-based classification is provided in Figure 7.2.

Loop 9-15 trains the models with ten-fold cross-validation and record the average of 10

Algorithm 1: Search-based Classification

```
Input: Datasets (DS), Search-based Techniques (SBT)
Output: Sensitivity, G-Mean, Balance, ROC-AUC
1 DS← {Tomcat6.0, Synapse1.0, Ivy2.0, Jedit4.2, Xalan2.4, Xerces1.3, Xerces1.2, Camel1.6,
 Ant1.7, Jedit4.0, Log4j1,0, Jedit4.1, Synapse1.1, Synapse1.2, Log4j1.1, Xalan2.6, Xalan2.5}
2 SBT← {BIOHEL, CHC, CORE, CPSO, GA_ADI, GA_INT, GGA, ILGA, LDWPSO,
 MPLCS, PBIL, REPSO, SGA, SIA, UCS, XCS}
3  Initialize Sensitivity, G-Mean, Balance, ROC-AUC with empty arrays {}
4  For each DS_n contained in symbol DS do
5        Generate 10 folds for cross-validation
6  For each DS_n contained in symbol DS do
7   For each SBT_i contained in symbol SBT do
8    For j = 1 to 30 do
9     For k = 1 to 10 do
10     Take fold 'k' as test fold
11          SBT_i← Train(SBT_i, XTrain, XTrainLabels)
12          Sensitivity_k← EvalSensitivity(SBT_i, XTest, XTestLabels)
13          G-Mean_k← EvalG-Mean(SBT_i, XTest, XTestLabels)
14          Balance_k← EvalBalance(SBT_i, XTest, XTestLabels)
15          ROC-AUC_k← EvalROC-AUC(SBT_i, XTest, XTestLabels)
16    end
17     Sensitivity_j← EvalAverage(Sensitivity_k)
18     G-Mean_j← EvalAverage(G-Mean_k)
19     Balance_j← EvalAverage(Balance_k)
20     ROC-AUC_j← EvalAverage(ROC-AUC_k)
21   end
22   AvgSensitivity_(n, i)← EvalAverage(Sensitivity_j)
23   AvgG-Mean_(n, i)← EvalAverage(G-Mean_j)
24   AvgBalance_(n, i)← EvalAverage(Balance_j)
25   AvgROC-AUC_(n, i)← EvalAverage(ROC-AUC_j)
26  End
27 End
28 Calculate the Friedman Rankings for SBTs based on Sensitivity using AvgSensitivity_(n, i) values.
29 Calculate the Friedman Rankings for SBTs based on G-Mean using AvgG-Mean_(n, i) values.
30 Calculate the Friedman Rankings for SBTs based on Balance using AvgBalance_(n, i) values.
31 Calculate the Friedman Rankings for SBTs based on ROC-AUC using AvgROC-AUC_(n, i) values.
32 For Sensitivity, G-Mean, Balance, and AUC
33      Perform a Wilcoxon signed-rank test for the best ranker SBT.
```

Figure 7.2: Pseudocode for SDP model development using SBTs

runs. Loop 8-19 deals with the stochastic behavior of SBTs by running them for 30 times. Lines 25-27 performs statistical validation of averaged results by Friedman test and Line 28-29 conducts the post-hoc Wilcoxon signed-rank test to compare pairs of SBTs.

7.3 Experimental Results and Analysis

This section describes the results of the chapter.

7.3.1 Answer to RQ1

RQ1: What is the relative performance of addressed SBTs for predicting software defects using OO metrics when Sensitivity, G-Mean, Balance, and ROC-AUC are considered?

Sensitivity, G-Mean, Balance, and ROC-AUC values obtained for 16 SBTs for 17 datasets are recorded in Table 7.1, 7.2, 7.3 and 7.4 respectively and the maximum values achieved for each dataset are highlighted in bold typeface.

7.3.1.1 Sensitivity Analysis

Dataset-wise performance of SBTs with their sensitivity values are presented in Table 7.1. The median sensitivity value achieved by all SBTs for Tomcat6.0, Synapse1.0, Ivy2.0, Jedit4.2, Xalan2.4, Xerces1.3, Xerces1.2, Camel1.6, Ant1.7, Jedit4.0, Log4j1.0, Jedit4.1, Synapse1.1, Synapse1.2, Log4j1.1, Xalan2.6 and Xalan2.5 are 0.12, 0.19, 0.80, 0.44, 0.19, 0.31, 0.08, 0.10, 0.41, 0.43, 0.38, 0.44, 0.42, 0.48, 0.58, 0.63, and 0.57 respectively. The sensitivity value ranges from 0.01 to 0.19 for Tomcat6.0. An analysis of Table 7.1 depicts sensitivity values in the range 0.06-0.38 on Synapse1.0, 0.03-1.00 on Ivy2.0, 0.08-1 on Jedit4.2, 0.02-0.32 on Xalan2.4, 0.03-0.36 Xerces1.3, 0.01-0.45 on Xerces1.2, 0.01-0.30 on Camel1.6, 0.02-0.52 on Ant1.7, 0.04-0.52 on Jedit4.0, 0.09-0.59 on Log4j1.0, 0.01-0.51 on Jedit4.1, 0.03-0.53 on Synapse1.1, 0.01-0.55 on Synapse1.2, 0.14-0.76 on Log4j1.1, 0.16-0.70 on Xalan2.6, and 0.02-0.65 on Xalan2.5.

The models developed by GA_INT, BIOHEL, GA_ADI, UCS, XCS, and MPLCS are superior than other SBTs considering the highest sensitivity value achieved for datasets. An

analysis of Table 7.1 depicts that BIOHEL obtained the best sensitivity values of 1.00, 1.00, 0.45, 0.30, 0.52, and 0.70 for Ivy2.0, Jedit4.2, Xerces1.2, Camel1.6, Jedit4.0, and Xalan2.6 respectively. LDWPSO has highest sensitivity value for Synapse1.0 (0.38), Xerces1.3 (0.36), Ant1.7 (0.52), Log4j1.0 (0.59), and Synapse1.2 (0.55). But, LDWPSO showed poor performance for other datasets.

Therefore, instead of considering the highest value, we evaluated the cumulative performance of SBTs based on the mean values attained by them. Fig. 7.3 presents the mean values of sensitivity measure for 30 runs on each dataset investigated in the chapter. BIOHEL has shown considerable performance in approximately all datasets and thus has highest mean sensitivity value of 0.48. XCS and UCS have given it close competition with mean sensitivity value of 0.46 followed by MPLCS (0.45). GA_ADI and GA_INT also seem to have comparable performance with mean sensitivity value of 0.44. REPSO and CORE exhibited the worst performance with mean sensitivity value of 0.11. Low sensitivity signifies low representation of correct predictions of defective classes.

Figure 7.3: Comparison of mean values of Sensitivity Performance of SBTs

Table 7.1: Sensitivity Results for SDP Models developed using SBTs

Sensitivity	CORE	REPSO	CPSO	LDWPSO	ILGA	SIA	GGA	CHC	PBIL	SGA	BIOHEL	GA_ADI	XCS	GA_INT	UCS	MPLCS
Tomcat6.0	0.04	0.01	0.12	0.05	0.07	0.13	0.10	0.04	0.12	**0.19**	0.18	0.13	0.11	0.12	0.18	0.17
Synapse1.0	0.13	0.06	0.25	**0.38**	0.08	0.19	0.19	0.06	0.06	0.19	0.13	0.19	0.13	0.31	0.25	0.25
Ivy2.0	0.03	0.73	0.08	0.03	0.59	0.40	0.83	0.88	0.9	0.63	**1.00**	**1.00**	0.97	0.98	0.78	0.98
Jedit4.2	0.29	0.40	0.10	0.08	0.56	0.42	0.44	0.44	0.31	0.44	**1.00**	**1.00**	0.96	**1.00**	0.88	**1.00**
Xalan2.4	0.05	0.03	0.25	0.30	0.02	0.18	0.17	0.09	0.21	0.18	0.23	0.19	0.23	0.16	**0.32**	0.23
Xerces1.3	0.07	0.03	0.30	**0.36**	0.07	0.25	0.33	0.32	0.33	0.32	0.3	0.29	0.31	0.32	0.32	0.29
Xerces1.2	0.03	0.01	0.10	0.01	0.02	0.27	0.01	0.01	0.03	0.08	**0.45**	0.10	0.14	0.08	0.23	0.11
Camel1.6	0.02	0.01	0.09	0.06	0.01	0.14	0.09	0.09	0.12	0.12	**0.30**	0.13	0.18	0.07	0.27	0.12
Ant1.7	0.17	0.02	0.42	**0.52**	0.30	0.29	0.35	0.37	0.41	0.36	0.50	0.41	0.41	0.4	0.45	0.47
Jedit4.0	0.07	0.04	0.23	0.16	0.23	0.21	0.47	0.43	0.44	0.48	**0.52**	0.43	0.48	0.45	0.48	0.51
Log4j1.0	0.24	0.09	0.44	**0.59**	0.39	0.26	0.38	0.38	0.35	0.38	0.38	0.44	0.56	0.5	0.35	0.47
Jedit4.1	0.04	0.01	0.15	0.43	0.4	0.33	0.49	0.46	0.49	**0.51**	0.47	0.46	0.49	0.46	0.38	0.37
Synapse1.1	0.03	0.05	0.48	0.23	0.27	0.42	0.38	0.40	0.42	0.45	0.43	0.42	0.44	0.48	**0.53**	0.42
Synapse1.2	0.07	0.01	0.53	**0.55**	0.37	0.49	0.47	0.47	0.50	0.48	0.49	0.48	0.52	0.42	0.43	0.52
Log4j1.1	0.27	0.14	**0.76**	0.43	0.53	0.57	0.57	0.62	0.62	0.62	0.57	0.62	0.58	0.51	0.62	0.59
Xalan2.6	0.24	0.16	0.55	0.50	0.54	0.57	0.64	0.64	0.60	0.66	**0.70**	0.64	0.67	0.63	0.67	0.63
Xalan2.5	0.09	0.02	0.34	0.45	0.46	0.60	0.57	0.56	0.61	0.64	0.59	0.57	**0.65**	0.59	0.62	0.50
Mean	0.11	0.11	0.31	0.30	0.29	0.34	0.38	0.37	0.38	0.40	**0.48**	0.44	0.46	0.44	0.46	0.45

7.3.1.2 G-Mean Analysis

In Table 7.2, it can be seen from the results that the maximum value of G-Mean measure for various datasets is scored by BIOHEL (in four datasets), MPLCS (in four datasets) LDWPSO (in three datasets). Moreover, SGA predicted the maximum G-Mean value for Tomcat6.0 and Jedit4.1. The maximum G-Mean value achieved for Ivy2.0 is 1.00 which is depicted by BIOHEL and GA_ADI followed by GA_INT and MPLCS with a remarkable value of 0.99. 76.5% of datasets have a G-Mean value greater than 0.5 for BIOHEL, GA_INT, and UCS. The range of BIOHEL lies from 0.34 to 1.00. CORE and REPSO performed poorly on defect prediction as only 15.4% datasets exhibit G-Mean value greater than 0.5. An analysis of Table 7.2 discloses that CORE, CPSO, LDWPSO, and SIA are not able to achieve G-Mean greater than 0.7 for a single dataset. REPSO achieved G-Mean value of 0.85 for Ivy2.0. For all other datasets, its G-Mean values were less than 65.0.

The overall effectiveness of these SBTs can be assessed by taking averaged G-Mean values. The bar values in Figure 7.4 represents the values of G-Mean averaged over all the datasets in increasing order. The bar in Figure 7.4 represents the value of performance measure for each specific SBT. BIOHEL and UCS have the maximum averaged G-Mean value of 0.63 followed by MPLCS (0.62), GA_INT (0.60), GA_ADI (0.60), and XCS (0.61). BIOHEL, GA_ADI, GA_INT, XCS, and MPLCS are able to predict defects with G-Mean value greater than 0.9 for 11.8% of datasets.

Table 7.2: G-Mean Results for SDP Models developed using SBTs

Dataset	CORE	REPSO	CPSO	LDWPSO	ILGA	SIA	GGA	CHC	PBIL	SGA	BIOHEL	GA_ADI	XCS	GA_INT	UCS	MPLCS
Tomcat6.0	0.20	0.11	0.34	0.23	0.26	0.35	0.32	0.20	0.34	**0.44**	0.41	0.36	0.32	0.34	0.42	0.41
Synapse1.0	0.35	0.25	0.47	**0.56**	0.28	0.41	0.43	0.25	0.25	0.43	0.34	0.42	0.35	0.53	0.49	0.49
Ivy2.0	0.16	0.85	0.25	0.16	0.77	0.62	0.9	0.93	0.95	0.78	**1.00**	**1.00**	0.98	0.99	0.88	0.99
Jedit4.2	0.54	0.63	0.32	0.29	0.75	0.64	0.66	0.66	0.55	0.65	**1.00**	**1.00**	0.98	**1.00**	0.94	**1.00**
Xalan2.4	0.21	0.17	0.47	0.52	0.14	0.41	0.41	0.30	0.45	0.42	0.45	0.43	0.46	0.4	**0.55**	0.46
Xerces1.3	0.27	0.17	0.52	**0.58**	0.27	0.48	0.57	0.56	0.57	0.56	0.53	0.53	0.54	0.56	0.55	0.53
Xerces1.2	0.17	0.12	0.31	0.12	0.12	0.51	0.12	0.12	0.17	0.28	**0.64**	0.31	0.37	0.29	0.46	0.33
Camel1.6	0.13	0.07	0.29	0.25	0.1	0.36	0.29	0.29	0.33	0.34	0.49	0.36	0.42	0.27	**0.50**	0.34
Ant1.7	0.40	0.15	0.58	**0.66**	0.53	0.52	0.57	0.59	0.62	0.57	0.64	0.62	0.62	0.61	0.64	**0.66**
Jedit4.0	0.26	0.20	0.45	0.38	0.46	0.45	0.66	0.63	0.63	0.66	0.67	0.62	0.66	0.64	0.65	**0.68**
Log4j1.0	0.48	0.30	0.55	**0.69**	0.61	0.49	0.59	0.59	0.57	0.6	0.56	0.62	0.69	0.63	0.58	0.65
Jedit4.1	0.19	0.11	0.38	0.57	0.62	0.55	0.67	0.65	0.67	**0.68**	0.63	0.65	0.67	0.64	0.6	0.59
Synapse1.1	0.18	0.22	0.60	0.43	0.51	0.60	0.58	0.61	0.61	0.64	0.59	0.62	0.63	0.64	**0.70**	0.60
Synapse1.2	0.26	0.10	0.53	0.60	0.58	0.64	0.62	0.64	0.67	0.64	0.62	0.62	0.65	0.61	0.60	**0.68**
Log4j1.1	0.52	0.37	0.66	0.59	0.72	0.69	0.70	**0.76**	0.74	0.73	0.68	0.73	0.71	0.68	0.75	0.72
Xalan2.6	0.47	0.40	0.63	0.62	0.68	0.67	0.73	0.73	0.70	0.73	**0.74**	0.72	**0.74**	0.73	0.73	0.73
Xalan2.5	0.30	0.13	0.50	0.56	0.58	0.64	0.6	0.59	0.62	0.64	0.65	0.64	**0.67**	0.60	0.65	0.63
Mean	0.3	0.26	0.46	0.46	0.47	0.53	0.55	0.54	0.55	0.58	**0.63**	0.60	0.61	0.60	**0.63**	0.62

CORE and REPSO have an average G-Mean value of 0.29 and 0.28 respectively. Such low values indicate their failure in the detection of software defects. G-Mean is the geometric mean of sensitivity and specificity. If any classifier is not capable of recognizing minority classes (defective classes), it results in low sensitivity. As seen in sensitivity analysis, CORE and REPSO failed to give correct predictions. Their low sensitivity values decrease the corresponding G-Mean values. Therefore, low G-Mean values of CORE and REPSO demonstrate their vulnerability toward class imbalance. In CORE, 15 co-populations works in parallel to coevolve rules with 0.1 probability of mutation and 0.5 probability of regenerations. The fitness function of CORE is defined by [27]. In the fitness function of CORE, more weightage is given to non-defective classes (TN) and the penalty is imposed for the non-defective classes that are wrongly predicted. Therefore, this technique did not perform well for the SDP application where the classification of TP is more important. In REPSO only one rule can be extracted in a single iteration. Furthermore, the fitness function of REPSO is dependent on the accuracy. Accuracy is not considered a good metric for imbalanced data, therefore REPSO results in low predictive capability for defects.

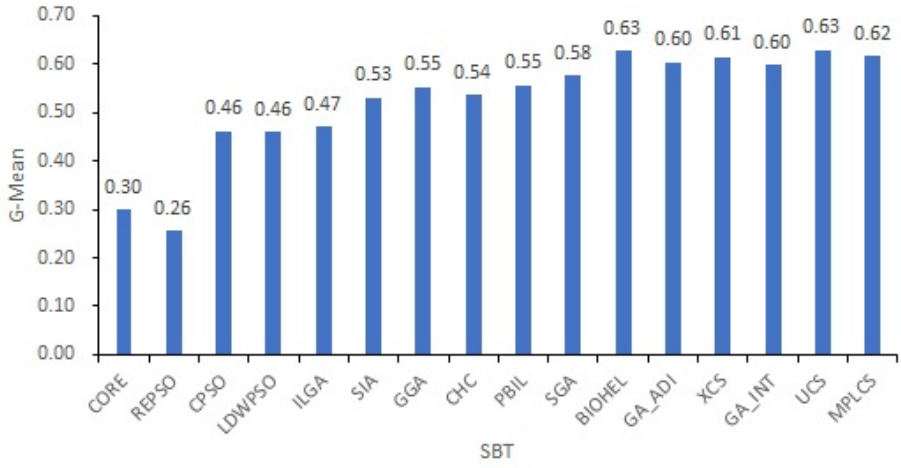

Figure 7.4: Comparison of mean values of G-Mean Performance of SBTs

7.3.1.3 Balance Analysis

Balance values for datasets ranged from 29.66 to the remarkable 100. The maximum value of 100 is achieved by BIOHEL, GA_ADI, GA_INT, and MPLCS for at least one dataset. These techniques performed fairly well in other datasets too. For BIOHEL, Balance value varies from 37.93 to 100 and it predicted the maximum Balance for six datasets. LDW-PSO predicted maximum Balance value for four datasets- Synapse1.0 (54.57), Xerces1.3 (54.52), Ant1.7 (63.88), and Log4j1.0 (67.99). MPLCS, UCS, and SGA achieved the highest Balance value in two datasets each. The range for GA_ADI is from 38.46 to 100 and a similar range is obtained for GA_INT (35.23-100) and MPLCS (37.22-100). For BIOHEL, 58.8% of datasets gave a Balance value greater than 60. Models developed using SGA, XCS, and MPLCS exhibited Balance value greater than 70. BIOHEL, GA_ADI, GA_INT, MPLCS, and XCS have a value greater than 90 for at least 15.4% of datasets.

Table 7.3: Balance Results for SDP Models developed using SBTs

Dataset	CORE	REPSO	CPSO	LDWPSO	ILGA	SIA	GGA	CHC	PBIL	SGA	BIOHEL	GA_ADI	XCS	GA_INT	UCS	MPLCS
Tomcat6.0	32.05	30.21	37.54	32.94	34.07	38.39	36.62	32.05	37.53	**43.04**	41.9	38.46	36.7	37.56	42.1	41.22
Synapse1.0	38.1	33.71	46.29	**54.57**	34.73	42.06	42.51	33.71	33.71	42.54	37.93	42.44	38.08	51.02	46.91	46.82
Ivy2.0	31.05	80.55	33.58	31.06	70.98	57.47	87.52	91.15	92.91	73.34	**100.00**	**100.00**	98.08	98.23	84.09	98.22
Jedit4.2	49.92	57.28	36.66	35.18	68.58	58.73	60.2	60.21	51.34	60.18	**100.00**	99.78	96.86	**100.00**	91.16	**100.00**
Xalan2.4	32.51	31.22	46.09	49.9	30.66	42.01	41.42	35.7	43.98	42.08	44.93	42.76	45.2	40.84	**51.60**	45.2
Xerces1.3	34.41	31.33	50.23	**54.52**	34.56	46.61	52.82	51.8	52.77	51.78	50.39	49.71	51.09	51.81	51.65	49.68
Xerces1.2	31.28	30.29	36.2	30.26	30.39	48.12	30.26	30.29	31.27	35.19	**60.52**	36.25	39.45	35.23	45.08	37.22
Camel1.6	30.42	29.66	35.57	33.74	30.07	38.83	35.26	35.28	37.49	37.83	**48.77**	38.68	42.11	34.53	47.87	37.87
Ant1.7	41.18	30.99	56.92	**63.88**	50.12	49.45	53.77	55.53	57.95	54.03	62.6	57.94	58.24	57.57	60.34	62.06
Jedit4.0	34	32.12	44.87	40.15	45.15	44.16	62.05	59.24	59.93	62.67	**64.92**	58.91	62.53	60.81	62.44	64.58
Log4j1.0	45.91	35.52	54.58	**67.99**	56.97	47.63	55.77	55.97	53.81	56.13	54.35	59.6	67.26	61.97	54.05	61.78
Jedit4.1	31.98	30.19	39.92	56.4	57.31	52.25	63.74	61.21	63.69	**64.62**	60.94	61.17	63.38	60.94	55.99	55.16
Synapse1.1	31.63	32.81	58.93	44.02	48.2	57.45	55.35	57.2	58.09	60.56	57.7	58.37	59.74	61.99	**66.52**	57.84
Synapse1.2	34.12	29.93	53.21	59.87	54.84	62	60.04	61.27	63.77	61.56	61	60.81	63.68	57.97	57.82	**65.28**
Log4j1.1	48.39	38.83	65.88	57.58	66.71	67.57	68.17	**72.80**	72.11	71.5	66.86	71.14	68.94	64.71	72.6	69.7
Xalan2.6	46.09	40.62	62.19	61.25	65.98	66.06	71.69	72.25	69.22	72.57	**74.08**	71.11	73.06	71.56	72.49	71.56
Xalan2.5	35.79	30.56	49.71	55.33	57.26	63.62	59.77	58.8	61.86	64.46	64.3	63.98	**66.99**	59.63	64.59	61.87
Mean	36.99	36.81	47.55	48.74	49.21	51.91	55.12	54.38	55.38	56.12	**61.83**	59.48	60.67	59.2	60.43	60.36

This can be construed from Table 7.3 that CORE, REPSO, CPSO, LDWPSO, and ILGA did not perform that well and exhibited comparatively lower Balance values than other SBTs. CPSO and LDWPSO are continuous representations of PSO and lacks global searchability. Both techniques were stuck in local minima. Though in LDWPSO the parameter 'inertia weight' is used to balance local and global search, parameter tuning is required. Therefore these techniques didn't perform well. The Balance values of all datasets are averaged to assess the overall capability of SBTs. Mean values of Balance values in ascending order are illustrated by the bar graph in Figure 7.5. Figure 7.5 also supports the conclusions derived from Table 7.3 BIOHEL, GA_ADI, GA_INT, MPLCS, UCS, and XCS have a mean Balance value greater than 60 showing their superior predictive capability for defect prediction. The maximum value is gained by BIOHEL (61.83) followed by XCS (60.67). CORE and REPSO underperformed in terms of Balance and exhibited the minimum mean Balance value of 36.99 and 36.81 respectively. The underlying reason lies in the inadequacy of these SBTs to classify defective classes resulting in low sensitivity. Low sensitivity values are responsible for low Balance value.

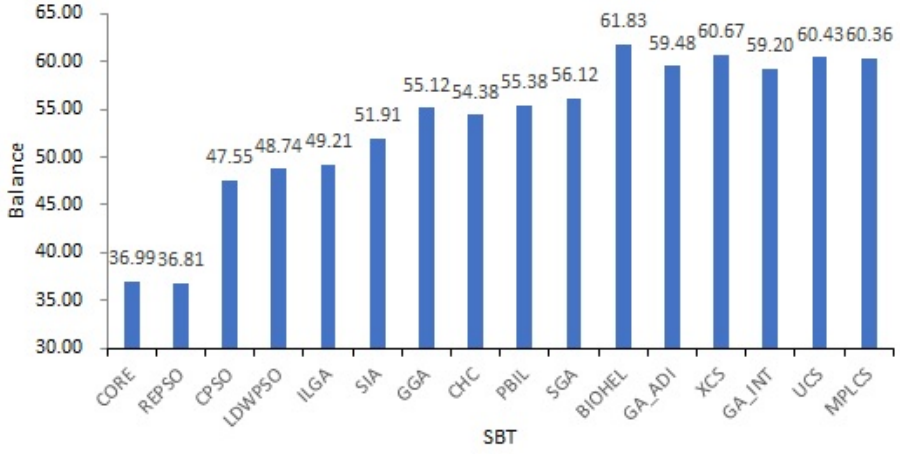

Figure 7.5: Comparison of mean values of Balance performance of SBTs

7.3.1.4 ROC-AUC Analysis

The boldfaced values in Table 7.4 depict that BIOHEL and MPLCS got the highest ROC-AUC values for four datasets each followed by UCS and SGA having the highest ROC-AUC values for three datasets each. GA_INT, GGA, CHC, GA_ADI, and XCS, also exhibited best predictive capability for defect prediction in atleast one dataset. Except for CPSO, datasets have experienced ROC-AUC greater than 0.6 in some or other cases. BIOHEL, GA_ADI, GA_INT, GGA, MPLCS, UCS, and XCS have demonstrated the remarkable ROC-AUC values ranging from 0.9 to 1.00 for a few datasets. The results support the selection of performance measures in handling the imbalanced nature of software engineering data and aid the correct prediction of developed models. Figure 7.6 depicts the mean value of ROC-AUC obtained for sixteen SBTs. The mean ROC-AUC value of UCS, XCS, and MPLCS is 0.69. GA_ADI, BIOHEL, and GA_INT have comparable mean ROC-AUC values with a value of 0.69. The lowest mean ROC-AUC values are obtained for CORE, REPSO, LDWPSO, and CPSO which is in the range 0.55 to 0.58. They are not capable of separating the defective and non-defective classes properly and therefore should be avoided to build SDP models. No single dataset was able to attain ROC-AUC greater than 0.8 with CORE, CPSO, LDWPSO, ILGA, and SIA.

Table 7.4: ROC-AUC Results for SDP Models developed using SBTs

Dataset	CORE	REPSO	CPSO	LDWPSO	ILGA	SIA	GGA	CHC	PBIL	SGA	BIOHEL	GA_ADI	XCS	GA_INT	UCS	MPLCS
Tomcat6.0	0.51	0.51	0.55	0.51	0.53	0.54	0.54	0.51	0.55	**0.59**	0.55	0.55	0.54	0.56	0.58	0.58
Synapse1.0	0.56	0.5	0.54	0.65	0.45	0.55	0.56	0.52	0.51	0.59	0.54	0.55	0.56	**0.66**	0.63	0.62
Ivy2.0	0.49	0.86	0.46	0.50	0.79	0.68	0.90	0.93	0.95	0.79	**1.00**	**1.00**	0.98	0.99	0.89	0.99
Jedit4.2	0.64	0.7	0.55	0.52	0.77	0.70	0.71	0.71	0.64	0.71	**1.00**	**1.00**	0.98	**1.00**	0.94	**1.00**
Xalan2.4	0.52	0.51	0.57	0.60	0.51	0.56	0.57	0.53	0.58	0.57	0.56	0.58	0.58	0.57	**0.63**	0.58
Xerces1.3	0.53	0.51	0.60	0.64	0.53	0.60	**0.66**	0.65	0.65	0.65	0.61	0.63	0.62	0.65	0.63	0.62
Xerces1.2	0.51	0.50	0.53	0.49	0.50	0.61	0.49	0.50	0.50	0.52	**0.68**	0.54	0.55	0.53	0.58	0.54
Camel1.6	0.50	0.50	0.52	0.51	0.50	0.53	0.53	0.53	0.54	0.54	0.56	0.56	0.57	0.53	**0.60**	0.54
Ant1.7	0.57	0.51	0.62	0.68	0.63	0.61	0.64	0.66	0.67	0.64	0.66	0.67	0.67	0.67	0.68	**0.69**
Jedit4.0	0.53	0.52	0.56	0.53	0.59	0.57	0.70	0.68	0.68	0.69	0.70	0.67	0.69	0.68	0.69	**0.71**
Log4j1.0	0.79	0.76	0.62	0.76	0.75	0.75	0.77	0.79	0.77	**0.80**	0.70	0.77	0.76	0.73	0.79	0.79
Jedit4.1	0.52	0.51	0.55	0.60	0.68	0.63	0.70	0.69	0.70	**0.71**	0.66	0.69	**0.71**	0.68	0.66	0.66
Synapse1.1	0.51	0.52	0.61	0.52	0.60	0.63	0.62	0.66	0.66	0.68	0.62	0.67	0.67	0.67	**0.73**	0.65
Synapse1.2	0.51	0.47	0.53	0.60	0.63	0.66	0.64	0.67	0.69	0.66	0.64	0.65	0.67	0.65	0.63	**0.70**
Log4j1.1	0.63	0.56	0.68	0.63	0.76	0.72	0.72	**0.78**	0.76	0.74	0.70	0.74	0.73	0.70	0.77	0.73
Xalan2.6	0.58	0.57	0.63	0.64	0.71	0.68	**0.74**	**0.74**	0.71	**0.74**	**0.74**	0.73	**0.74**	**0.74**	0.73	**0.74**
Xalan2.5	0.53	0.50	0.53	0.57	0.59	0.64	0.60	0.59	0.62	0.64	0.65	0.65	**0.67**	0.60	0.65	0.65
Mean	0.55	0.56	0.57	0.58	0.62	0.63	0.65	0.66	0.66	0.66	0.68	0.68	**0.69**	0.68	**0.69**	**0.69**

Figure 7.6: Comparison of mean values of ROC-AUC performance of SBTs

Collectively, this can be concluded with the help of Figure 7.3, 7.4, 7.5 and 7.6 that BIOHEL, GA_ADI, GA_INT, MPLCS, UCS, and XCS performed well for the SDP in terms of sensitivity, G-Mean, Balance, and ROC-AUC and depicted similar performances.

7.3.1.5 Comparison with the closest studies

Since very few studies are conducted in the SDP area with SBT trained classifiers, it may be not possible to conduct a fair comparison between them and this study. Chatterjee et al. [72] used MSE as a performance measure, they used four-fold cross-validation. Like many other studies, [73] assessed the predictive power of the SDP model by traditional metrics like accuracy, precision, sensitivity, specificity, and f-score. This study, on contrary, has employed more stable and reliable performance measures as G-Mean, Balance, and ROC-AUC. The studies conducted by Chatterjee et al. [72] as well as Manjula and Florence [73] are weak studies because results are not statistically validated. Further, authors in [73] have not considered the stochastic behavior of GA. The results change with each run of any of SBTs. Moreover, in [72] experiments are repeated only 13 times. In this study, each experiment is run 30 times with ten-fold cross–validation. The results based on stable

performance measures are further validated by the nonparametric Friedman test. Friedman test is equivalent to a parametric two way ANOVA test. All these differences make this work a strong research study that is useful to developers and software practitioners. In [73], five C/C++ projects are used whereas this study explored JAVA projects. Except for [4], no other study has extensively analyzed the impact of SBTs for defect prediction like this study. This study focuses on analyzing predictive capabilities of a wide range of SBTs with open source software datasets and to find their suitability to predict software defects. This study is different from [4] as it considers ROC-AUC for comparing the SDP models created using SBTs. ROC-AUC is a widely accepted evaluation metric because it is threshold independent [194] and is not influenced by the class imbalance problem [74, 195]. In [4] G-Mean and Balance are used to evaluate prediction models; prediction results vary with threshold values in both cases. Additionally, we explored more software engineering datasets that are less explored in literature. The datasets used in this study are not used in [4]. Ozakinci and Tarhan [217] in their recent review have proved that 75%of empirical studies conducted for SDP are weak. Unlike these studies, this chapter presents a strong empirical study by statistically validating the results using the Friedman and Wilcoxon signed–rank test.

The closest existing study to this work is [4]. Though in [4] many SBTs are explored with JAVA projects but performance measures of that study are dependent on the predictor's threshold. The results of [4] and this chapter are quantitatively compared based on the averaged value of G-Mean and Balance. Table 7.5 presents the averaged G-Mean values and Balance values for the sixteen SBTs. As per the authors' knowledge, the most widely accepted and threshold independent metric AUC is used only in this study for classifying defects using SBTs. The study with a larger value of averaged G-Mean or Balance value is boldfaced in Table 7.5 to provide an easy comparison. The performance of LDWPSO decreased in terms of G-Mean and performance of REPSO reduced for Balance. CORE, CPSO, GA_INT, ILGA, SIA show similar mean values in terms of G-Mean and Balance

performance in and this study.

Table 7.5: Result comparison with [4]

	G-Mean		Balance	
	[4]	This study	[4]	This study
BIOHEL	0.47	**0.63**	47.45	**61.83**
CHC	0.44	**0.54**	43.15	**54.38**
CORE	**0.34**	0.30	**41.52**	36.99
CPSO	0.51	**0.46**	44.82	**47.55**
GA_ADI	0.50	**0.60**	54.6	**59.48**
GA_INT	0.54	**0.60**	55.73	**59.20**
GGA	0.45	**0.55**	47.34	**55.12**
ILGA	0.43	**0.47**	**50.92**	49.21
LDWPSO	**0.55**	0.46	**48.96**	48.74
MPLCS	0.54	**0.62**	55.48	**60.36**
PBIL	0.46	**0.55**	45.24	**55.38**
REPSO	**0.29**	0.26	**48.36**	36.81
SGA	0.45	**0.58**	47.13	**56.12**
SIA	0.51	**0.53**	**52.9**	51.91
UCS	0.53	**0.63**	53.95	**60.43**
XCS	0.49	**0.61**	48.74	**60.67**

Mean values of G-Mean and Balance values have quantitatively improved in this study for nine SBTs namely BIOHEL, CHC, GA_ADI, GGA, MPLCS, PBIL, SGA, UCS, and XCS.

7.3.2 Answer to RQ2

RQ2: Which SBTs are statistically good or bad defect predictors for OO projects when G-Mean, Balance, and ROC-AUC are considered?

To find the answer to this question, the nonparametric Friedman test is executed to find the mean rankings of all the SBTs that were used to develop SDP models. Three null hypotheses and three alternate hypotheses were formed for three performance measures. Null hypotheses and alternate hypotheses are framed as:

- H_{10} (Null Hypothesis): There is no significant statistical difference between the predictive performance of SBTs when G-Mean is considered.
- H_{1a} (Alternate Hypothesis): There is a significant statistical difference between the predictive performance of SBTs when G-Mean is considered.
- H_{20} (Null Hypothesis): There is no significant statistical difference between the predictive performance of SBTs when Balance is considered.
- H_{2a} (Alternate Hypothesis): There is a significant statistical difference between the predictive performance of SBTs when Balance is considered.
- H_{30} (Null Hypothesis): There is no significant statistical difference between the predictive performance of SBTs when ROC-AUC is considered.
- H_{3a} (Alternate Hypothesis): There is a significant statistical difference between the predictive performance of SBTs when ROC-AUC is considered.

The results of Friedman testing are scribed in Table 6. Technique receiving rank 1 is considered to be the best SBT and is boldfaced in Table 7.6. The mean rank obtained by each technique in the Friedman test is inscribed in brackets. It can be noted from Table 7.6 that p-value for G-Mean, Balance, and ROC-AUC for Friedman ranking evaluation comes to be 0.000. The null hypothesis designed for each performance measure is rejected because the p–value is less than 0.05. Therefore, the performances of addressed SBTs are found to be significantly different from each other. The Friedman results for G-Mean and ROC-AUC ascertain UCS, MPLCS, and SGA as top three rankers followed by XCS, BIOHEL, GA_ADI, and GA_INT. The results announce UCS and MPLCS as the best SBTs when G-Mean and ROC-AUC values are considered respectively. The G-Mean Friedman results are analogous to that of Malhotra [4] except for SGA and SIA. Kendall C is noted as 0.443, 0.439, and 0.482 for G-Mean, Balance, and ROC-AUC. These values suggest the

effect size be moderately good. Kendall's coefficient for G-Mean in [4] was 0.308 and for Balance was 0.166 only.

Table 7.6: Friedman Rankings

	G-Mean	Balance	ROC-AUC
MPLCS	4 (11.27)	5 (11.08)	**1 (12.08)**
UCS	**1 (12.15)**	**1 (12.15)**	2 (11.69)
SGA	3 (11.31)	3 (11.54)	3 (11.62)
GA_INT	7 (10.42)	7 (10.15)	4 (11.46)
XCS	2 (11.69)	2 (11.73)	4 (11.46)
GA_ADI	6 (10.96)	6 (10.65)	6 (11.42)
BIOHEL	5 (11.04)	4 (11.38)	7 (10.12)
PBIL	8 (9.62)	8 (9.54)	8 (9.62)
CHC	9 (8.73)	10 (8.15)	9 (9.38)
GGA	10 (8.35)	9 (8.58)	10 (8.65)
SIA	11 (7.81)	11 (8)	11 (7.23)
LDWPSO	12 (6.12)	13 (6.35)	12 (5.58)
ILGA	14 (5.46)	14 (5)	13 (5.54)
CPSO	12 (6.12)	12 (6.69)	14 (4.69)
CORE	15 (2.77)	15 (2.81)	15 (3.23)
REPSO	16 (2.19)	16 (2.19)	16 (2.23)

Malhotra [4] considered ML techniques and hybrid versions of SBTs in addition to the SBTs when performing Friedman ranking. In this study, experimentation is restricted to SBTs only to reveal their true predictive capabilities for SDP in comparison to each other. REPSO projected similar behavior as in [4] and is proclaimed as the worst performer for all the three stable performance measures. Since the null hypotheses are rejected for all the three performance measures, post-hoc analysis using the Wilcoxon signed-rank test is conducted to find a pairwise statistical difference between top ranker and other SBTs. The statistical significance is checked for the 95% level of confidence. UCS is ranked as the best SBT for both the G-Mean and Balance with a mean rank of 12.15. Therefore, the Wilcoxon signed–rank test is done for UCS for G-Mean and Balance. MPLCS is ranked as

the number 1 SBT for ROC-AUC with a mean rank of 12.08 and therefore, pairwise comparison of MPLCS is performed with all other SBTs. Results of the Wilcoxon signed-rank test for G-Mean, Balance, and ROC-AUC are recorded in Table 7.7 with their p–values. Statistically significant pairs with their p–values are boldfaced. For G-Mean and Balance, the performance of UCS is found to be statistically better than CORE, CPSO, ILGA, LDWPSO, SIA, and REPSO. UCS is statistically better than GGA in the case of G-Mean but nearly missed for Balance. Similar observations can be drawn for ROC-AUC. These results indicate the superiority of GA based classifiers over the PSO variants. UCS, BIOHEL, CHC, GA_ADI, GA_INT, MPLCS, PBIL, and SGA projected comparable performance in the case of G-Mean, Balance as well as ROC-AUC.

Table 7.7: Wilcoxon signed-rank results of G-Mean, Balance and ROC-AUC values

G-Mean	p-value	Balance	p-value	ROC-AUC	p-value
UCS - BIOHEL	0.972	UCS - BIOHEL	0.753	MPLCS-BIOHEL	0.136
UCS - CHC	0.087	UCS - CHC	0.064	MPLCS-CHC	0.064
UCS - CORE	**0.001**	**UCS - CORE**	**0.001**	**MPLCS-CORE**	**0.001**
UCS - CPSO	**0.001**	**UCS - CPSO**	**0.001**	**MPLCS-CPSO**	**0.001**
UCS - GA_ADI	0.279	UCS - GA_ADI	0.507	MPLCS-GA_ADI	0.753
UCS - GA_INT	0.382	UCS - GA_INT	0.552	MPLCS-GA_INT	0.638
UCS - GGA	**0.046**	UCS - GGA	0.055	**MPLCS-GGA**	**0.023**
UCS - ILGA	**0.002**	**UCS - ILGA**	**0.002**	**MPLCS-ILGA**	**0.003**
UCS - LDWPSO	**0.011**	**UCS - LDWPSO**	**0.023**	**MPLCS-LDWPSO**	**0.006**
UCS - MPLCS	0.249	UCS - MPLCS	0.422	MPLCS-UCS	0.861

G-Mean	p-value	Balance	p-value	ROC-AUC	p-value
UCS - PBIL	0.075	UCS - PBIL	0.173	MPLCS-PBIL	0.133
UCS - REPSO	**0.001**	**UCS - REPSO**	**0.001**	**MPLCS-REPSO**	**0.001**
UCS - SGA	0.133	UCS - SGA	0.108	MPLCS-SGA	0.507
UCS - SIA	**0.005**	**UCS - SIA**	**0.007**	**MPLCS-SIA**	**0.013**
UCS - XCS	0.402	UCS - XCS	0.917	MPLCS-XCS	0.552

The results corroborated the findings of subsection 7.3.1 and statistically conclude that UCS, BIOHEL, CHC, GA_ADI, GA_INT, MPLCS, PBIL, and SGA performed better for defect prediction than the remaining techniques. This offers guidelines to software practitioners to choose from a wide set of SBTs which they should opt for and which they should avoid when considering similar software.

7.4 Discussion

Replication of existing studies is necessary to build an adequate body of knowledge in order to draw more strong conclusions leading to widely accepted and well-formed theories. Our primary goal is to provide empirical evidence that will help in strengthening the results from existing studies and also report differences across studies. It is the researchers' responsibility to provide the software community the efficient and reliable SDP models so that better quality software be developed and delivered in time. In this study, empirical research is accomplished to validate the applicability of SBTs in SDP.

The main contribution of this chapter is to

- Authenticate the usage of SBTs being the efficient defect predictors with more software engineering datasets since these are less explored set of techniques in the liter-

ature

- Provide statistical validation for SBTs and
- Use stable performance evaluation metrics alleviating the imbalanced data problem of datasets.

Except for SIA, performances of developed models are nearly similar to models trained by Malhotra in [4] in terms of G-Mean and Balance.

SBTs that are good defect predictors (UCS, BIOHEL, GA_ADI, GA_INT, MPLCS, XCS, CHC, PBIL) are identified with UCS and MPLCS as top rankers. Authors encourage developers to exercise these techniques during software development. Additionally, SBTs that underperformed are recognized (CORE, CPSO, GGA, ILGA, LDWPSO, and REPSO). Developers should avoid using these techniques for predicting defect prone classes in undergoing projects. The practical applicability of this research is to help developers building better SDP models by detecting defect prone areas in the design phase and plan the resource allocation effectively using SBTs. Software practitioners and developers are recommended to incorporate SBTs for revealing defect prone areas in software while advancing for software development. SBTs are practically usable since they can explore larger search space for potential solutions.

Therefore, next, we would prefer to enhance their defect prediction capabilities by employing resampling methods to alleviate imbalanced data problem.

Chapter 8

Predicting Software Defects using Resampling Methods with Search-based Techniques

8.1 Introduction

In recent years, though various ML techniques are used for predicting software defects [51], many SBTs have now gained attention of researchers to attain this purpose. Harman [218] and Harman and Jones [219] inspired us to instigate the application of SBTs in the software engineering predictive modeling domain as these techniques are effective in handling constraints and conflicts. Alos, in last chapter we have assessed the suitability of SBTs for predicting defects. Imbalanced classification problem is one of the major apprehensions in data mining. Software face imbalanced classification problem because only minority percentage of software modules is responsible for majority of defects in complete software [220]. It is imperative to address imbalanced classification problem for effective and correct

training of any software quality prediction model. With existence of number of imbalanced learning methods in literature to deal with this issue [28], this chapter focuses only on data resampling methods that comprises of oversampling and undersampling methods to tackle imbalanced classification problem in SBT. From the previous chapters, we may conclude that the performance of ML techniques were best achieved when resampling methods were applied. Therefore, different resampling methods are used to evaluate and enhance the effectiveness of SBTs for SDP in this chapter. We get motivated to perform this study with the positive results obtained in previous Chapter 4 and Chapter 7.

Researchers in [69, 71] aimed at handling class imbalance issue but they did not address resampling solutions and they also used NASA datasets. Details of SBT studies related to defect prediction can be explored in systematic review done by Malhotra et al. [200]. Recently, Cai et al. [67] have proposed the hybrid undersampled approach for solving imbalanced classification problem. They exploited multi-objective cuckoo search to optimize support vector machine, but resampling methods are not explored with any of SBT alone for defect prediction yet. To the best of our knowledge, no detailed study has been found in existing literature with focus on effectiveness of resampling methods in developing SBT based SDP prediction models.

The aim of the study is to apply resampling methods on open-source software projects and perform comparative analysis of predictive capabilities of defect prediction models built using SBT. This study seeks answers to following RQs:

- RQ1: What is comparative performance of SDP models built using SBTs?
- RQ2: Which SBT outperforms the other SBTs in terms of sensitivity, G-Mean, Balance and ROC-AUC for defect prediction?
- RQ3: Do the usage of resampling methods facilitate in building improved SDP model with SBTs?

- RQ4: Which resampling method adds the most toward the improvement of models developed using SBTs for defect prediction?

In this research SDP models are developed using eight SBTs (GA_INT, MPLCS, UCS, BIOHEL, GA_ADI, XCS, CPSO , and LDWPSO) adopting stable performance evaluators for imbalanced data. Seven oversampling and four undersampling methods are applied on five OO projects. Models are based on static quality attributes defined by recognized set of OO metrics. Stochastic nature of SBTs is well handled by repeating execution of each experiment 30 times. This aids in providing unbiased predictions. Validation of datasets is carried out using ten-fold cross-validation and statistical validation of results is done by Friedman test followed by Wilcoxon signed-rank test.

The chapter is organized as follows: Section 8.2 describes the pursued research methodology and experimental setup laid down for this research. Section 8.4 explicates the results of this chapter and Section 8.5 is the closing section that acquaints interested researchers and developers with discussion on results.

The part of this work is published in [221].

8.2 Elements of Experimental Design

This section briefly describes the datasets used, independent and dependent variables of the study and performance evaluators used for comparing performance of various models developed for defect prediction.

8.2.1 Dataset Collection

In this chapter, we used five datasets corresponding to JAVA projects. Five considered projects are Ant1.7, Camel1.6, Xerces1.3, Synapse1.0, and Tomcat6.0. Selection is based considering their varied ratio of defective classes to encompass different conditions of im-

balanced classification and these datasets are already used in Chapter 4, Chapter 5, Chapter 7, and Chapter 7.

8.2.2 Independent and Dependent Variables

Object-oriented (OO) metrics are well established defect predictors in literature (Singh et al. 2010). Independent variables are 20 OO metrics and dependent variable is discrete and binary in nature predicting whether the class is defective or not. In this chapter independent and dependent variables used are the same as that used in Chapter 4, Chapter 5, Chapter 7, and Chapter 7.

8.2.3 Resampling Methods

imbalanced classification problem can be resolved effectually by using resampling methods that comprise of oversampling methods and undersampling methods [222]. The resampling methods investigated in Chapter 4 are considered in this chapter for balancing the datasets. These methods are SMT, SLSMT, ADSYN, SPD, ROS, AHC, CNNTL, RUS, NCL, and OSS. Details of these methods are provided in subsection 3.8.1.

8.2.4 Performance Evaluators and Statistical Validation

When data is imbalanced, it is wiser to use evaluation measures that considers FP along with TP [150]. Therefore, we have used stable performance evaluation metrics- Balance, G-Mean and ROC-AUC along with sensitivity. These metrics are explained in Section 3.11. We used ten-fold cross-validation for model development and nonparametric tests for statistical validation of results. Friedman test followed by Wilcoxon post-hoc analysis is executed. Section 3.12 summarizes these tests.

8.3 Experimental Setup

Experimental setup is described in Figure 8.1.

Ten-fold cross-validation portrayed in subsection 3.10.2 is executed on datasets to provide unbiased evaluation of model. Then, resampling methods comprising of oversampling and undersampling methods are used to tackle imbalanced nature of data. Models are developed using eight SBTs. Parameter tuning in SBT is very expensive and may not achieve desired improvement [216]. Therefore, default parameter settings for SBTs and resampling methods in KEEL environment have been used in the chapter. Default settings promote replicability of the study. It is important to perform multiple iterations to effectively handle the nondeterministic nature of SBTs [223]. Therefore, each experiment is run thirty times to strengthen conclusion validity.

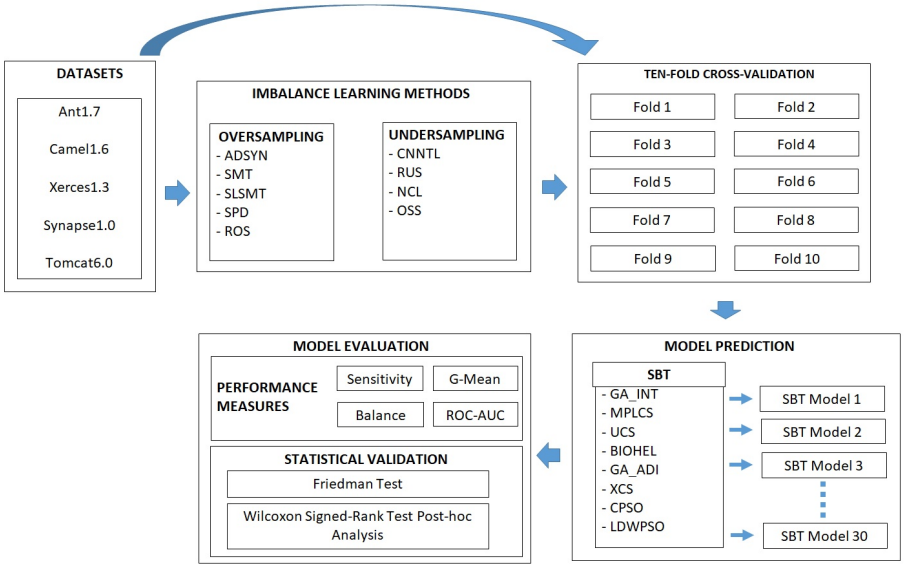

Figure 8.1: Experimental Design

We selected GA_INT, MPLCS, UCS, BIOHEL, GA_ADI, and XCS based on their good prediction capabililites in previous chapter. Despite the incompetency in predicting defects correctly, CPSO and LDWPSO are chosen to analyze their performance with resampling

methods. The primary reason for their inclusion was to explore the efficacy of PSO-based SBTs on the data that is rebalanced. Will they perform in similar fashion as in Chapter 7 or better? Stable evaluation metrics like Balance, G-Mean and ROC-AUC are used for performance comparison of these developed models along with sensitivity. Lastly, to ascertain the validity of results, as discussed, Friedman test is also applied at significant level of α = 0.05 followed by post-hoc analysis.

8.4 Results and Analysis

This section presents experimental results and attempts to find answers to addressed RQs. Table 8.1- 8.15 presents the dataset-wize values of Balance, G-Mean and ROC-AUC. NS is no sampling scenario, i.e., when no resampling method is used for balancing of defective and nondefective classes. For a particular dataset, model that predicts the highest value of performance evaluator is highlighted in bold typeface.

8.4.1 RQ1: What is comparative performance of SDP models built using SBTs?

In this study eight SBTs are exploited to envisage the software defects and to answer this research question we analyzed performance using G-Mean with help of Tables 8.1, 8.2, 8.3, 8.4, and 8.5. Balance analysis is scribed in Tables 8.6, 8.7, 8.8, 8.9, and 8.10. Tables 8.11, 8.12, 8.13, 8.14, and 8.15 presents the ROC-AUC values for developed models. Monitoring the NS scenario of all datasets for Balance, G-Mean and ROC-AUC gives us an idea that in the majority of cases LDWPSO gave better results followed by UCS and GA_INT.

8.4.1.1 G-Mean Analysis

All SBT models that are developed using resampling methods have shown improvement as compared to NS scenario except for UCS and LDWPSO. It is worth noting that on usage of resampling methods, GA_INT and GA_ADI best classified atleast 40% of datasets each. Maximum value for Ant1.7 and Camel1.6 is 0.76 with GA_ADI when resampling methods are used. For NS scenario, maximum G-Mean value is for Ant1.7 is 0.66 with MPLCS and for Camel1.6 maximum value is 0.5 with UCS. Similarly, for Synapse1.0 and Tomcat6.0 GA_INT depicts maximum G-Mean value of 0.78 and 0.76 with SMT and SLSMT. Considering all datasets and all scenarios, GA_INT has maximum mean G-Mean value of 0.65 closely followed by GA_ADI and MPLCS with mean G-Mean value of 0.64 and 0.63 respectively.

Table 8.1: G-Mean Results of search-based models for No Sampling and Resampling Methods on Ant1.7

ANT1.7	GA_INT	MPLCS	UCS	BIOHEL	GA_ADI	XCS	CPSO	LDWPSO
NS	0.61	0.66	0.64	0.64	0.62	0.64	0.58	0.66
ADSYN	0.72	0.74	0.64	0.72	0.74	0.72	0.67	0.66
SMT	0.73	0.75	0.64	0.64	0.73	0.71	0.65	0.62
SLSMT	0.74	0.72	0.64	0.70	**0.76**	0.75	0.66	0.68
SPD	0.73	0.75	0.64	0.71	0.72	0.71	0.70	0.63
SPD2	0.71	0.74	0.64	0.72	0.73	0.73	0.69	0.64
ROS	0.74	0.73	0.64	0.62	0.75	0.68	0.70	0.66
AHC	0.74	0.72	0.64	0.67	0.73	0.74	0.71	0.67
CNNTL	0.65	0.63	0.64	0.61	0.63	0.62	0.57	0.60
RUS	0.73	0.7	0.64	0.68	0.72	0.69	0.67	0.67
NCL	0.73	0.74	0.64	0.70	0.74	0.76	0.45	0.67
OSS	0.74	0.73	0.64	0.67	0.74	0.70	0.62	0.67

Table 8.2: G-Mean Results of search-based models for No Sampling and Resampling Methods on Camel1.6

CAMEL1.6	GA_INT	MPLCS	UCS	BIOHEL	GA_ADI	XCS	CPSO	LDWPSO
NS	0.27	0.34	0.50	0.49	0.36	0.42	0.29	0.25
ADSYN	0.62	0.64	0.44	0.61	0.64	0.65	0.5	0.52
SMT	0.60	0.58	0.44	0.59	0.61	0.59	0.54	0.49
SLSMT	0.57	0.57	0.44	0.59	**0.76**	0.63	0.55	0.53
SPD	0.56	0.58	0.50	0.58	0.57	0.58	0.53	0.45
SPD2	0.63	0.58	0.50	0.59	0.64	0.54	0.53	0.55
ROS	0.64	0.59	0.44	0.59	0.62	0.54	0.48	0.55
AHC	0.61	0.63	0.50	0.61	0.66	0.6	0.53	0.47
CNNTL	0.53	0.60	0.44	0.57	0.59	0.60	0.52	0.38
RUS	0.63	0.62	0.44	0.60	0.61	0.65	0.52	0.50
NCL	0.58	0.62	0.50	0.65	0.58	0.67	0.37	0.54
OSS	0.58	0.59	0.44	0.63	0.63	0.63	0.49	0.47

Table 8.3: G-Mean Results of search-based models for No Sampling and Resampling Methods on Xerces1.3

XERCES1.3	GA_INT	MPLCS	UCS	BIOHEL	GA_ADI	XCS	CPSO	LDWPSO
NS	0.56	0.53	0.55	0.53	0.53	0.54	0.52	0.58
ADSYN	0.73	0.65	0.60	0.69	0.69	0.73	0.6	0.64
SMT	0.76	0.72	0.60	0.67	0.71	0.67	0.61	0.65
SLSMT	0.77	0.74	0.60	0.74	0.75	0.70	0.62	0.70
SPD	0.66	0.62	0.55	0.63	0.64	0.62	0.50	0.66
SPD2	0.68	0.68	0.55	0.65	0.65	0.65	0.52	0.67
ROS	0.70	0.59	0.60	0.67	0.67	0.69	0.42	0.68
AHC	0.73	0.66	0.55	0.70	0.64	0.60	0.58	0.71
CNNTL	0.63	0.65	0.60	0.65	0.68	0.72	0.66	0.56
RUS	0.69	0.71	0.60	0.69	0.65	**0.77**	0.50	0.66
NCL	0.68	0.68	0.55	0.62	0.62	0.74	0.52	0.62
OSS	0.63	0.63	0.60	0.64	0.71	0.72	0.58	0.54

Table 8.4: G-Mean Results of search-based models for No Sampling and Resampling Methods on Synapse1.0

SYNAPSE1.0	GA_INT	MPLCS	UCS	BIOHEL	GA_ADI	XCS	CPSO	LDWPSO
NS	0.53	0.49	0.49	0.34	0.42	0.35	0.47	0.56
ADSYN	0.67	0.56	0.49	0.55	0.67	0.69	0.65	0.60

SYNAPSE1.0	GA_INT	MPLCS	UCS	BIOHEL	GA_ADI	XCS	CPSO	LDWPSO
SMT	**0.78**	0.69	0.49	0.51	0.56	0.64	0.58	0.63
SLSMT	0.71	0.68	0.49	0.66	0.68	0.67	0.65	0.54
SPD	0.66	0.23	0.49	0.52	0.33	0.49	0.60	0.71
SPD2	0.57	0.52	0.49	0.52	0.47	0.48	0.47	0.62
ROS	0.63	0.46	0.49	0.62	0.42	0.50	0.67	0.65
AHC	0.64	0.52	0.49	0.52	0.56	0.53	0.54	0.56
CNNTL	0.58	0.57	0.49	0.59	0.55	0.53	0.61	0.54
RUS	0.72	0.71	0.49	0.62	0.71	0.67	0.42	0.60
NCL	0.67	0.72	0.49	0.68	0.65	0.65	0.41	0.68
OSS	0.59	0.51	0.49	0.58	0.52	0.62	0.6	0.63

Table 8.5: G-Mean Results of search-based models for No Sampling and Resampling Methods on Tomcat6.0

TOMCAT6.0	GA_INT	MPLCS	UCS	BIOHEL	GA_ADI	XCS	CPSO	LDWPSO
NS	0.34	0.41	0.42	0.41	0.36	0.32	0.34	0.23
ADSYN	0.70	0.70	0.45	0.67	0.73	0.68	0.65	0.65
SMT	0.74	0.75	0.45	0.67	0.74	0.72	0.62	0.69
SLSMT	**0.76**	0.76	0.45	0.69	0.74	0.70	0.62	0.63
SPD	0.62	0.55	0.42	0.53	0.63	0.43	0.61	0.62
SPD2	0.69	0.69	0.42	0.55	0.61	0.52	0.66	0.57
ROS	0.69	0.64	0.45	0.46	0.71	0.51	0.65	0.71
AHC	0.75	0.69	0.42	0.51	0.72	0.6	0.71	0.62
CNNTL	0.66	0.69	0.45	0.59	0.67	0.62	0.63	0.55
RUS	0.70	0.73	0.45	0.71	0.71	0.73	0.44	0.65
NCL	0.61	0.58	0.42	0.6	0.59	0.62	0.55	0.47
OSS	0.62	0.58	0.45	0.61	0.63	0.59	0.58	0.46

8.4.1.2 Balance Analysis

Similar trend is observed in mean value of Balance for SBTs when all datasets and scenarios are considered together. Mean value of Balance is maximum for GA_INT with value 64.67% followed by GA_ADI (63.09%) and MPLCS (62.29%). To understand dataset wise model prediction, let us take an example of Ant1.7 dataset. Though in NS situation maximum value of Balance is shown by LDWPSO for Ant1.7 with challenging value of

63.88% but when resampling methods were used along with SBTs then for the same dataset SLSMT+GA_ADI gave the best Balance value with approximately 20% increment.

Table 8.6: Balance Results of search-based models for No Sampling and Resampling Methods on Ant1.7

Resampling	GA_INT	MPLCS	UCS	BIOHEL	GA_ADI	XCS	CPSO	LDWPSO
NS	57.57	62.06	60.34	62.6	57.94	61.02	56.92	63.88
ADSYN	71.94	73.58	60.34	71.56	73.84	71.36	66.32	65.48
SMT	73.43	75.16	60.34	62.6	72.53	69.7	64.88	60.36
SLSMT	74.35	71.91	60.34	69.87	**76.08**	74.88	65.26	67.78
SPD	71.79	73.96	60.34	70.31	70.30	70.11	69.43	61.76
SPD2	71.29	73.63	60.34	72.12	72.53	72.14	68.53	63.29
ROS	73.81	72.17	60.34	60.05	74.71	65.76	70.24	65.62
AHC	73.83	71.30	60.34	66.29	72.33	73.10	70.68	66.31
CNNTL	63.26	62.21	60.34	59.01	60.79	60.65	55.93	57.85
RUS	73.36	70.26	60.34	67.96	71.82	69.33	66.73	66.58
NCL	73.17	73.55	60.34	70.07	73.36	75.45	45.44	67.31
OSS	74.13	72.97	60.34	67.39	73.66	69.45	61.01	66.89

Table 8.7: Balance Results of search-based models for No Sampling and Resampling Methods on Camel1.6

Resampling	GA_INT	MPLCS	UCS	BIOHEL	GA_ADI	XCS	CPSO	LDWPSO
NS	34.53	37.87	47.87	48.77	38.68	42.11	35.57	33.74
ADSYN	61.39	63.76	43.97	60.76	63.97	65.06	50.48	51.2
SMT	59.55	57.07	43.97	57.48	61.21	58.39	54.05	49.18
SLSMT	57.47	56.87	43.97	58.97	**72.36**	63.04	54.75	52.48
SPD	54.63	56.65	47.87	56.92	55.03	55.93	52.68	45.28
SPD2	62.77	56.72	47.87	58.67	63.67	53.11	52.62	54.32
ROS	63.47	58.41	43.97	57.59	62.12	52.34	48.48	54.52
AHC	60.51	62.56	47.87	59.67	65.60	58.69	53.02	47.47
CNNTL	50.86	59.61	43.97	56.81	56.92	59.09	52.07	40.01
RUS	62.62	61.67	43.97	59.78	60.51	64.7	51.92	49.55
NCL	55.66	61.53	47.87	65.15	56.76	67.16	39.74	53.56
OSS	57.38	58.91	43.97	62.54	62.66	62.89	48.8	47.27

Table 8.8: Balance Results of search-based models for No Sampling and Resampling Methods on Xerces1.3

XERCES1.3	GA_INT	MPLCS	UCS	BIOHEL	GA_ADI	XCS	CPSO	LDWPSO
NS	51.81	49.68	51.65	50.39	49.71	51.09	50.23	54.52
ADSYN	73.08	63.66	56.71	67.82	68.76	71.69	58.77	63.18
SMT	75.47	70.94	56.71	64.96	70.36	64.52	58.68	64.25
SLSMT	76.31	74.11	56.71	74.07	74.51	69.35	62.03	69.94
SPD	62.58	58.59	51.65	60.94	61.34	57.9	48.36	64.75
SPD2	67.07	66.48	51.65	62.53	63.21	62.25	50.85	67.07
ROS	68.99	56.36	56.71	63.42	66.30	64.90	43.21	67.61
AHC	72.93	63.81	51.65	67.63	62.58	57.87	57.48	70.34
CNNTL	62.60	65.28	56.71	64.22	67.70	71.29	66.14	55.88
RUS	68.60	71.07	56.71	68.94	64.45	**77.15**	48.78	64.66
NCL	66.36	65.57	51.65	59.95	59.90	73.80	50.02	61.41
OSS	61.04	61.87	56.71	63.77	70.72	71.34	57.58	54.15

Table 8.9: Balance Results of search-based models for No Sampling and Resampling Methods on Synapse1.0

SYNAPSE1.0	GA_INT	MPLCS	UCS	BIOHEL	GA_ADI	XCS	CPSO	LDWPSO
NS	51.02	46.82	46.91	37.93	42.44	38.08	46.29	54.57
ADSYN	66.03	54.06	46.82	53.63	66.23	67.41	65.05	60.27
SMT	**77.42**	67.32	46.82	50.04	54.33	62.60	57.98	62.29
SLSMT	71.15	67.64	46.82	65.15	67.92	65.73	65.05	53.90
SPD	63.38	33.16	46.91	50.64	37.47	47.79	59.6	70.54
SPD2	54.89	50.55	46.91	50.55	46.44	46.56	47.41	61.71
ROS	59.52	45.83	46.82	59.32	42.28	47.85	66.39	64.18
AHC	62.49	50.36	46.91	50.55	54.57	51.00	53.77	56.36
CNNTL	57.67	56.85	46.82	59.14	55.16	52.85	61.37	54.22
RUS	72.11	70.54	46.82	61.14	69.85	67.03	43.30	59.9
NCL	65.82	70.67	46.91	66.81	63.11	62.89	42.12	67.64
OSS	58.59	50.60	46.82	57.79	51.47	61.75	59.84	62.57

Table 8.10: Balance Results of search-based models for No Sampling and Resampling Methods on Tomcat6.0

TOMCAT6.0	GA_INT	MPLCS	UCS	BIOHEL	GA_ADI	XCS	CPSO	LDWPSO
NS	37.56	41.22	42.10	41.9	38.46	36.70	37.54	32.94
ADSYN	70.25	69.89	43.91	65.76	72.26	65.59	65.25	65.25

TOMCAT6.0	GA_INT	MPLCS	UCS	BIOHEL	GA_ADI	XCS	CPSO	LDWPSO
SMT	73.70	74.21	43.91	65.18	73.97	70.79	62.29	68.40
SLSMT	**76.31**	75.86	43.91	68.78	74.02	68.37	61.70	62.86
SPD	59.10	51.98	42.10	50.92	59.19	43.28	59.91	60.55
SPD2	67.23	66.08	42.10	53.21	58.00	49.55	66.16	54.19
ROS	68.1	61.92	43.91	45.41	69.19	49.12	64.85	70.53
AHC	74.94	67.89	42.10	49.72	71.73	56.67	70.79	61.85
CNNTL	66.17	69.37	43.91	58.79	66.46	62.10	63.41	54.16
RUS	70.11	72.76	43.91	70.85	70.87	73.21	44.21	64.29
NCL	57.31	54.56	42.10	58.14	55.60	58.87	53.66	45.65
OSS	59.53	55.16	43.91	59.59	61.10	57.11	55.56	45.73

8.4.1.3 ROC-AUC Analysis

ROC-AUC results for SBTs were also no different. GA_INT has maximum mean value of 68.42 for ROC-AUC considering all cases, having close competition with GA_ADI, MPLCS and XCS. Maximum ROC-AUC value is 0.80 noted for Synapse1.0 in SMT+GA_INT case. We can see little improvement in ROC-AUC values of LDWPSO models but UCS remained unaffected of resampling methods. Many models built using GA_INT, MPLCS, GA_ADI and XCS with resampling methods have ROC-AUC greater than 0.7, except for NS cases. Thus, though mean values of stable performance measures are greatest for GA_INT, still MPLCS, GA_ADI and XCS have also shown remarkable performance in terms of SDP.

Table 8.11: ROC-AUC Results of search-based models for No Sampling and Resampling Methods on Ant1.7

ANT1.7	GA_INT	MPLCS	UCS	BIOHEL	GA_ADI	XCS	CPSO	LDWPSO
NS	0.67	0.69	0.68	0.66	0.67	0.68	0.62	0.68
ADSYN	0.72	0.74	0.68	0.72	0.74	0.72	0.67	0.66
SMT	0.73	0.75	0.68	0.65	0.73	0.71	0.66	0.63
SLSMT	0.74	0.72	0.68	0.70	**0.76**	0.76	0.66	0.68
SPD	0.73	0.75	0.68	0.72	0.73	0.72	0.70	0.64
SPD2	0.71	0.74	0.68	0.73	0.73	0.73	0.69	0.65
ROS	0.74	0.73	0.68	0.65	0.75	0.69	0.70	0.66

ANT1.7	GA_INT	MPLCS	UCS	BIOHEL	GA_ADI	XCS	CPSO	LDWPSO
AHC	0.74	0.73	0.68	0.68	0.74	0.75	0.71	0.67
CNNTL	0.68	0.64	0.68	0.64	0.66	0.64	0.60	0.63
RUS	0.73	0.70	0.68	0.68	0.72	0.69	0.67	0.67
NCL	0.73	0.74	0.68	0.70	0.74	0.75	0.53	0.67
OSS	0.74	0.73	0.68	0.68	0.74	0.70	0.64	0.67

Table 8.12: ROC-AUC Results of search-based models for No Sampling and Resampling Methods on Camel1.6

CAMEL1.6	GA_INT	MPLCS	UCS	BIOHEL	GA_ADI	XCS	CPSO	LDWPSO
NS	0.53	0.54	0.60	0.56	0.56	0.57	0.52	0.51
ADSYN	0.62	0.64	0.56	0.62	0.64	0.66	0.51	0.55
SMT	0.61	0.61	0.56	0.61	0.62	0.61	0.55	0.52
SLSMT	0.58	0.58	0.56	0.59	0.60	0.63	0.55	0.56
SPD	0.60	0.62	0.60	0.61	0.62	0.63	0.56	0.54
SPD2	0.63	0.60	0.60	0.61	0.64	0.58	0.54	0.55
ROS	0.64	0.61	0.56	0.61	0.62	0.59	0.49	0.55
AHC	0.61	0.63	0.60	0.63	0.66	0.63	0.54	0.52
CNNTL	0.59	0.61	0.56	0.58	0.62	0.61	0.55	0.53
RUS	0.63	0.62	0.56	0.60	0.61	0.65	0.53	0.54
NCL	0.62	0.62	0.60	0.65	0.60	**0.67**	0.51	0.57
OSS	0.60	0.60	0.56	0.63	0.63	0.63	0.50	0.54

Table 8.13: ROC-AUC Results of search-based models for No Sampling and Resampling Methods on Xerces1.3

XERCES1.3	GA_INT	MPLCS	UCS	BIOHEL	GA_ADI	XCS	CPSO	LDWPSO
NS	0.65	0.62	0.63	0.61	0.63	0.62	0.6	0.64
ADSYN	0.74	0.66	0.66	0.70	0.69	0.73	0.62	0.64
SMT	0.76	0.73	0.66	0.68	0.71	0.69	0.65	0.66
SLSMT	**0.78**	0.74	0.66	0.74	0.75	0.7	0.62	0.7
SPD	0.70	0.67	0.63	0.67	0.68	0.67	0.59	0.67
SPD2	0.69	0.70	0.63	0.67	0.67	0.68	0.59	0.68
ROS	0.71	0.64	0.66	0.70	0.69	0.72	0.46	0.68
AHC	0.73	0.69	0.63	0.71	0.66	0.64	0.59	0.71
CNNTL	0.63	0.65	0.66	0.67	0.68	0.73	0.66	0.58
RUS	0.69	0.71	0.66	0.69	0.65	0.77	0.58	0.67
NCL	0.7	0.70	0.63	0.64	0.66	0.75	0.59	0.64

XERCES1.3	GA_INT	MPLCS	UCS	BIOHEL	GA_ADI	XCS	CPSO	LDWPSO
OSS	0.65	0.65	0.66	0.64	0.72	0.72	0.59	0.55

Table 8.14: ROC-AUC Results of search-based models for No Sampling and Resampling Methods on Synapse1.0

SYNAPSE1.0	GA_INT	MPLCS	UCS	BIOHEL	GA_ADI	XCS	CPSO	LDWPSO
NS	0.66	0.62	0.63	0.54	0.55	0.56	0.54	0.65
ADSYN	0.73	0.64	0.62	0.6	0.71	0.75	0.66	0.64
SMT	**0.80**	0.75	0.62	0.59	0.64	0.71	0.59	0.62
SLSMT	0.74	0.72	0.62	0.71	0.72	0.72	0.66	0.51
SPD	0.71	0.49	0.63	0.59	0.51	0.61	0.61	0.73
SPD2	0.66	0.61	0.63	0.64	0.60	0.60	0.53	0.64
ROS	0.70	0.55	0.62	0.69	0.59	0.64	0.70	0.65
AHC	0.71	0.63	0.63	0.59	0.63	0.63	0.56	0.57
CNNTL	0.62	0.59	0.62	0.61	0.56	0.55	0.60	0.59
RUS	0.75	0.73	0.62	0.66	0.73	0.71	0.51	0.61
NCL	0.70	0.76	0.63	0.74	0.70	0.72	0.58	0.69
OSS	0.65	0.57	0.62	0.64	0.61	0.69	0.62	0.68

Table 8.15: ROC-AUC Results of search-based models for No Sampling and Resampling Methods on Tomcat6.0

TOMCAT6.0	GA_INT	MPLCS	UCS	BIOHEL	GA_ADI	XCS	CPSO	LDWPSO
NS	0.56	0.58	0.58	0.55	0.55	0.54	0.55	0.51
ADSYN	0.71	0.71	0.58	0.68	0.73	0.70	0.66	0.66
SMT	0.74	0.75	0.58	0.69	0.75	0.74	0.62	0.69
SLSMT	**0.76**	0.76	0.58	0.70	0.75	0.71	0.62	0.63
SPD	0.67	0.63	0.58	0.61	0.67	0.57	0.63	0.65
SPD2	0.71	0.71	0.58	0.61	0.66	0.61	0.66	0.62
ROS	0.71	0.67	0.58	0.57	0.72	0.61	0.66	0.72
AHC	0.76	0.70	0.58	0.59	0.73	0.65	0.71	0.62
CNNTL	0.67	0.70	0.58	0.59	0.67	0.62	0.63	0.56
RUS	0.70	0.73	0.58	0.71	0.72	0.73	0.55	0.67
NCL	0.66	0.64	0.58	0.64	0.65	0.66	0.60	0.59
OSS	0.66	0.63	0.58	0.62	0.67	0.62	0.62	0.55

8.4.2 RQ2: Which SBT outperforms the other SBTs in terms of G-Mean, Balance and ROC-AUC for defect prediction?

After careful investigation of behavior of eight SBTs, we concluded that GA_INT, MPLCS, GA_ADI and XCS performs comparatively well than other SBTs and GA_INT has highest mean values for G-Mean, Balance and ROC-AUC. Now, we need to find that do there really exist one such technique that outperforms other SBTs? Is GA_INT statistically better than other SBTs? To answer these questions, Friedman test at 0.05 significance level was accomplished for G-Mean, Balance and ROC-AUC values.

Following hypotheses were formed and tested:

- *Null Hypothesis (H_{01})*: There is no significant difference between the predictive capabilities of eight SBTs in terms of G-Mean for SDP.
- *Alternate Hypothesis (H_{a1})*: There is significant difference between the predictive capabilities of eight SBTs in terms of G-Mean for SDP.
- *Null Hypothesis (H_{02})*: There is no significant difference between the predictive capabilities of eight SBTs in terms of Balance for SDP.
- *Alternate Hypothesis (H_{a2})*: There is significant difference between the predictive capabilities of eight SBTs in terms of Balance for SDP.
- *Null Hypothesis (H_{03})*: There is no significant difference between the predictive capabilities of eight SBTs in terms of ROC-AUC for SDP.
- *Alternate Hypothesis (H_{a3})*: There is significant difference between the predictive capabilities of eight SBTs in terms of ROC-AUC for SDP.

Ranks achieved by eight SBTs are listed in Table 8.16 with their mean ranks in brackets. P-values are also mentioned to prove their statistical significance.

Table 8.16: Friedman Rankings of SBTs with respect to G-Mean, Balance and ROC-AUC

Rank	G-Mean	Balance	ROC-AUC
Rank 1	GA_INT (6.27)	GA_INT (6.23)	GA_INT (6.37)
Rank 2	GA_ADI (5.81)	GA_ADI (5.7)	GA_ADI (5.82)
Rank 3	MPLCS (5.55)	MPLCS (5.44)	MPLCS (5.55)
Rank 4	XCS (5.23)	XCS (5.10)	XCS (5.50)
Rank 5	BIOHEL (4.44)	BIOHEL (4.59)	BIOHEL (4.18)
Rank 6	LDWPSO (3.68)	LDWPSO (3.88)	UCS (3.30)
Rank 7	CPSO (2.99)	CPSO (3.21)	LDWPSO (2.93)
Rank 8	UCS (2.03)	UCS (1.85)	CPSO (2.35)
p-value	**0.000**	**0.000**	**0.000**

As expected, first four rankers are those SBTs which were identified as better software defect predictors in Chapter 7 with GA_INT as topper. G-Mean, Balance and ROC-AUC demonstrates exactly same rankings for first five ranks. As p-value for all three measures is 0.000, therefore, we refute the null hypothesis that there is no difference between predictive capabilities of all addressed SBTs. Even after balancing the datasets, PSO-based SBTs were not able to prove their potential for reliable SDP.

Now to check whether GA_INT, which has highest mean rank for all three performance evaluators, is statistically different from all other SBTs, post-hoc analysis is required to do pair-wise comparison to eradicate family-wize error. For this purpose, we used Wilcoxon signed-rank test at significance level α = 0.05. It looks to the difference in predictive capability of SBTs that are considered in single pair. Therefore, to remove family-wise error, GA_INT is paired with remaining seven SBTs and their p-values calculated in Wilcoxon results corresponding to Balance, G-Mean and ROC-AUC are presented in Table 8.17. Significant p-values are in bold typeface.

Table 8.17: Results of Wilcoxon Signed-rank test for SBTs with respect to G-Mean, Balance, and ROC-AUC

	Balance	G-Mean	ROC-AUC
GA_INT - CPSO	**0.0000**	**0.0000**	**0.0000**
GA_INT - UCS	**0.0000**	**0.0000**	**0.0000**
GA_INT - LDWPSO	**0.0000**	**0.0000**	**0.0000**
GA_INT - BIOHEL	**0.0010**	**0.0006**	**0.0000**
GA_INT - XCS	0.1172	0.2677	0.7477
GA_INT - MPLCS	0.2086	0.3106	0.1347
GA_INT - GA_ADI	0.7546	1.278	0.6543

Table 8.17 depicts that GA_INT is statistically better than CPSO, UCS, LDWPSO and BIOHEL. Even though mean values of G-Mean, Balance and ROC-AUC of GA_INT are somewhat greater than XCS, MPLCS and GA_ADI, but statistically there is no difference in their performance. These four techniques have comparable performance considering any of three performance evaluators. GA_INT, GA_ADI, MPLCS, and XCS are concluded as best defect predictors.

8.4.3 RQ3: Do the usage of resampling methods facilitate in building improved search-based SDP model?

8.4.3.1 G-Mean Analysis

When no resampling method is employed, value of G-Mean ranges from 0.23 to 0.66. Application of different resampling methods increases G-Mean to maximum of 0.78 which is shown by SMT over GA_INT SBT for Synapse1.0 dataset. Comparisons based on performance of GA_INT are considered as it has been proved statistically better than other SBTs in answer to RQ2. There is not a single model of NS scenario that has G-Mean greater than 66%. Only 18% models of NS scenario have G-Mean greater than 60% where as on application of resampling methods this count increases to 64% from meager 18%. Mean value

of G-Mean in NS when no resampling method is used is 0.46 where as mean value of SMT + GA_INT is 0.72 which is maximum mean of all the resampling techniques employed in the study. Mean values of G-Mean for GA_INT are given in Figure 8.2. Approximately 33% increment is observed in mean value of G-Mean of each resampling method used as compared to NS scenario.

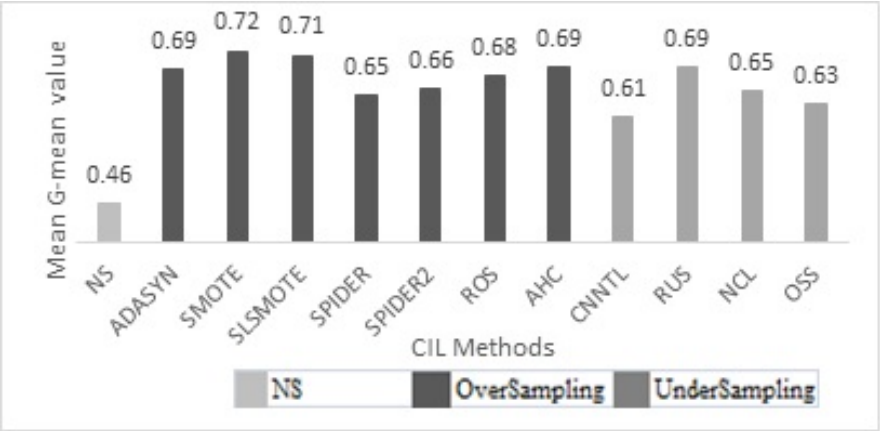

Figure 8.2: Mean G-Mean values for GA_INT

8.4.3.2 Balance Analysis

Value of Balance ranges from 32.94 to 63.88 in case of NS scenario for all datasets. Application of different resampling methods changed this range to minimum of 33.16 and maximum of 77.42. Maximum value is again shown by SMT over GA_INT SBT for Synapse1.0 dataset. 39% of models that are built using resampling methods have Balance greater than 64% where as not a single model of NS scenario has shown Balance greater than 64%. 88% of NS models have Balance less than 60% and this value got reduced to 44% models with employment of resampling methods. Mean Balance value of NS is 47.08 and it is too low as compared to mean of resampling +SBT models, which is 60.75. 60% of datasets have depicted maximum Balance value when SLSMT is applied to them with SBTs. 84% of resampling+GA_INT models have Balance greater than 60%. Mean values of Balance

are summarized in Figure 8.3 for GA-INT SBT. On analysis, we observed 42% increment in mean value of Balance for resampling+GA_INT as compared to that of NS.

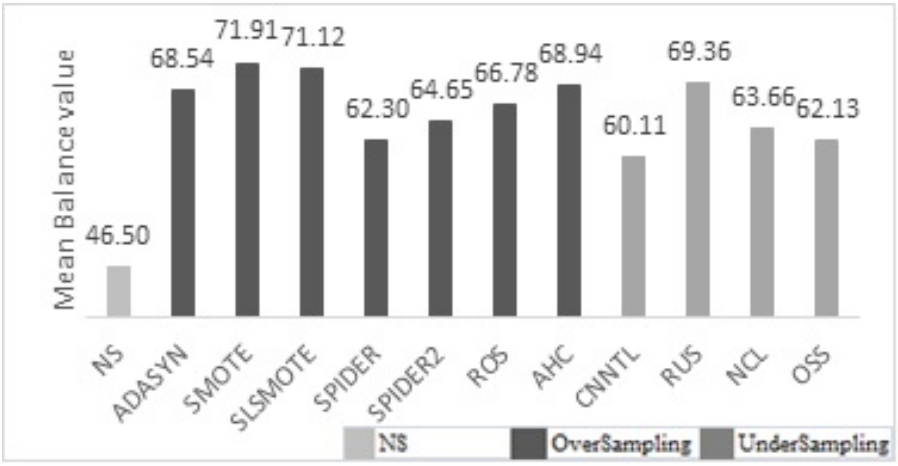

Figure 8.3: Mean Balance values for GA_INT

8.4.3.3 ROC-AUC Analysis

Range of ROC-AUC for models developed using SBTs under NS scenario is from 0.51 to 0.69. 50% of NS models have ROC-AUC values less than 60%. This statistic is decreased to 21% when we consider all the models developed using resampling methods. Figure 8.4 demonstrates the mean ROC-AUC values for NS and applied resampling methods with the GA_INT technique. Mean ROC-AUC value for NS when GA_INT is applied is 0.61which increases to 0.69 for resampling methods. If we talk about GA_INT alone, maximum value of ROC-AUC is 0.8010 with SMT combination. GA_INT+SMT and GA_INT+SLSMT has mean ROC-AUC value of 0.73 and 0.72 respectively. Therefore, we can surely say that usage of resampling methods facilitate in building improved search-based SDP models in term of G-Mean, Balance and ROC-AUC.

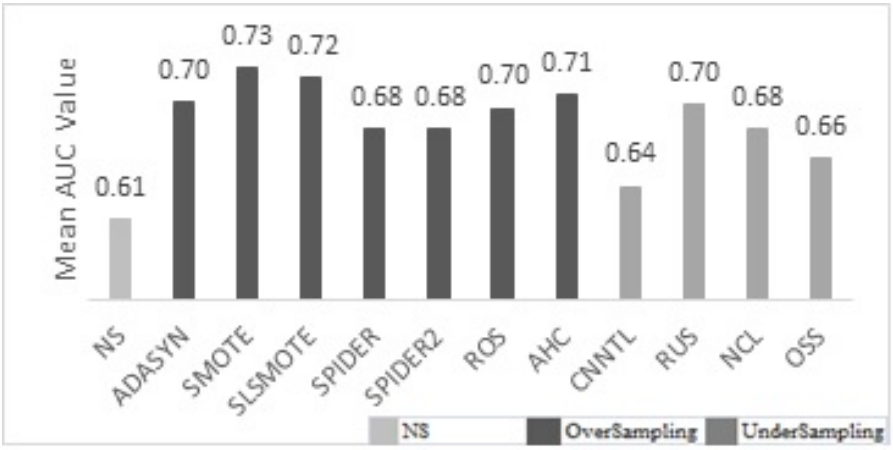

Figure 8.4: Mean ROC-AUC values for GA_INT

8.4.4 RQ4: Which resampling method adds the most toward the improvement of models developed using SBTs for defect prediction?

To answer this question, we will conduct Friedman test to find rankings of resampling methods with respect to G-Mean, Balance and ROC-AUC. Following hypotheses is structured to execute Friedman test:

- *Null Hypothesis ($H2_{01}$)*: There is no significant difference between the predictive capabilities of SDP models developed using eight SBTs when no resampling method is used as compared to search-based SDP models when any of eleven resampling methods are applied in terms of G-Mean.

- *Alternate Hypothesis ($H2_{a1}$)*: There is significant difference between the predictive capabilities of SDP models developed using eight SBTs when no resampling method is used as compared to search-based SDP models when any of eleven resampling methods are applied in terms of G-Mean.

- *Null Hypothesis ($H2_{02}$)*: There is no significant difference between the predictive capabilities of SDP models developed using eight SBTs when no resampling method is used as compared to search-based SDP models when any of eleven resampling methods are applied in terms of Balance.

- *Alternate Hypothesis ($H2_{a2}$)*: There is significant difference between the predictive capabilities of SDP models developed using eight SBTs when no resampling method is used as compared to search-based SDP models when any of eleven resampling methods are applied in terms of Balance.

- *Null Hypothesis ($H2_{03}$)*: There is no significant difference between the predictive capabilities of SDP models developed using eight SBTs when no resampling method is used as compared to search-based SDP models when any of eleven resampling methods are applied in terms of ROC-AUC.

- *Alternate Hypothesis ($H2_{a3}$)*: There is significant difference between the predictive capabilities of SDP models developed using eight SBTs when no resampling method is used as compared to search-based SDP models when any of eleven resampling methods are applied in terms of ROC-AUC.

Friedman test is carried out at α=0.05 confidence level with 11 as degree of freedom. Table 8 illustrates the Friedman rankings of all addressed resampling methods for G-Mean, Balance and ROC-AUC with their mean ranks. Higher the rank attained by resampling methods, better is its predictive capability for uncovering unseen software defects. For all three performance evaluators, as can be seen in Table 8.18, NS scenario has the worst ranking, i.e., rank 12. Rank 1 is awarded to SLSMT followed by ADSYN, RUS and SMT. Mean rank of NS is 2.33 for G-Mean and Balance where as mean rank of highest ranker SLSMT is 9.08 in both cases. Similarly, huge variation can be seen in mean ranks of NS (3.01) and SLSMT (8.61) for ROC-AUC metric.

Table 8.18: Friedman Rankings of SBTs with respect to G-Mean, Balance and ROC-AUC

PM	G-Mean	Balance	ROC-AUC
Rank 1	SLSMT (9.08)	SLSMT (9.08)	SLSMT (8.61)
Rank 2	ADSYN (8.38)	ADSYN (8.38)	ADSYN (8.14)
Rank 3	RUS (7.54)	RUS (7.54)	SMT (7.51)
Rank 4	SMT (7.53)	SMT (7.53)	NCL (7.24)
Rank 5	AHC (7.25)	AHC (7.25)	AHC (7.09)
Rank 6	NCL (6.81)	NCL (6.81)	RUS (6.94)
Rank 7	ROS (6.56)	ROS (6.56)	ROS (6.66)
Rank 8	SPD2 (6.21)	SPD2 (6.21)	SPD2 (6.46)
Rank 9	OSS (6.14)	OSS (6.14)	SPD (6.09)
Rank 10	CNNTL (5.11)	CNNTL (5.11)	OSS (5.71)
Rank 11	SPD (5.08)	SPD (5.08)	CNNTL (4.54)
Rank 12	NS (2.33)	NS (2.33)	NS (3.01)
p-value	**0.000**	**0.000**	**0.000**

Also, p-value is 0.000 which indicate towards the significant difference among their performance as it is less than 0.05. These values are bold-faced in Table 8.18. Therefore, null hypothesis is rejected and we conclude that there is significant improvement in SDP models when resampling methods are incorporated with SBTs.

Motivated by results analyzed in Table 8.18, we further conducted post-hoc analysis to ascertain whether, in particular, SLSMT is statistically better than other resampling methods and NS scenario. This objective is achieved by carrying out Wilcoxon signed-rank test at α = 0.05 for SLSMT with respect to Balance, G-Mean and ROC-AUC. Results are summarized in Table 8.19.

Pair-wise comparison of SLSMT with all other resampling methods and NS is done and corresponding p-values with bonferroni correction for all performance measures are noted in Table 8.19. Significant p-values are highlighted in bold typeface. P-value of 0.0000

for Balance, G-Mean, and p-value of 0.0001 for ROC-AUC in case of NS reconfirms the suitability of resampling methods for enhancing the predictive capability of SDP model.

Table 8.19: Results of Wilcoxon Signed-rank test for SLSMT with respect to G-Mean, Balance and ROC-AUC

	Balance	G-Mean	ROC-AUC
SLSMT - NS	**0.000**	**0.000**	**0.0001**
SLSMT - ADSYN	0.9564	2.1974	10.8144
SLSMT - SMT	0.1528	0.7005	10.8144
SLSMT - SPD	**0.0004**	**0.0019**	0.1412
SLSMT - SPD2	**0.0032**	**0.0058**	0.148
SLSMT - ROS	**0.0235**	0.0656	0.7693
SLSMT - AHC	**0.0178**	**0.0374**	0.927
SLSMT - CNNTL	**0.0001**	**0.0001**	**0.0016**
SLSMT - RUS	2.7887	2.2842	3.5157
SLSMT - NCL	0.0591	0.1623	2.7461
SLSMT - OSS	**0.0009**	**0.0012**	**0.0235**

Table 8.19 also points to the fact that though applying resampling methods are superior to the case when no such method is used, but SLSMT, ADSYN, RUS, SMT and NCL have comparable performance in terms of G-Mean, Balance and ROC-AUC. This concludes in identification of effective subset of resampling methods that can be used to build better search-based SDP models.

8.5 Discussion

Investigation of SBTs in light of resampling methods with more projects is required as our literature lack such studies. Possible contribution of this chapter is twofold. This study not only promotes developing efficient SDP models with the help of search-based techniques but also provides solution for imbalanced classification problem which is very common in real life datasets. Eight SBTs are explored with the help of five Apache datasets to validate their significance in predicting software defects. Further, eleven resampling methods, comprising of oversampling and undersampling methods, are applied to alleviate the imbalanced issue in software. In total, 14,400 experiments were created. Study includes statistical validation of results strengthening its findings.

On application of SLSMT with GA_INT, G-Mean values of Ant1.7, Camel1.6, Xerces1.3, Synapse1.0, and Tomcat6.0 datasets are improved by 21.1%, 113.17%, 38.27%, 33.18%, and 123.48% respectively. Mean value of G-Mean with GA_INT is increased by 53.81%. Similarly, Balance values are also improved with 52.95% increase in mean value of Balance. Percentage increment of 29.15%, 66.43%, 47.29%, 39.46%, and 103.17% is observed in Ant1.7, Camel1.6, Xerces1.3, Synapse1.0, and Tomcat6.0 respectively. ROC-AUC values have also increased by 11.18%, 9.57%, 19.52%, 13.18%, and 36.9% in Ant1.7, Camel1.6, Xerces1.3, Synapse1.0, and Tomcat6.0 respectively. Mean value of ROC-AUC is alleviated by 17.79% on application of SLSMT when GA_INT is used for model development. Other resampling methods have also shown statistical increase in the values of performance measures as compared to NS scenario.

On basis of the Friedman test, GA_INT emerged as the best SBT and SLSMT is proved to be the best resampling method for SDP when performances were evaluated using G-Mean, Balance and ROC-AUC. Four SBTs viz. GA_INT, MPLCS, GA_ADI and XCS were statistically proven better SBTs as compared to others during post-hoc analysis. Sta-

tistically better subset of resampling methods using G-Mean, Balance and ROC-AUC for defect prediction includes SLSMT, ADSYN, RUS, SMT and NCL.

Results of study advocate that proper selection of performance metrics and resampling methods plays vital role in predictive capability of SDP models generated using SBTs. Hence, the empirical experimentation conducted in this chapter favors the predictive modeling using resampling methods with SBTs for the effective identification of defective classes in OO software.

The results show the good performance of SBTs, especially GA-based, in constructing classification models. Hence, we are intrigued to analyze their efficacy in feature selection as well. The next work addresses the GA-based evolutionary feature selection.

Chapter 9

Empirical Validation of Evolutionary Feature Selection Techniques for SDP

9.1 Introduction

SDP is an important research problem in the software engineering domain and we have seen many solutions to deal with imbalanced data problem in previous chapters. Various machine learning techniques, search-based techniques, and resampling methods were explored in the development of SDP models for the early detection of defects. Defect prone areas uncovered in the design phase prevents defects to propagate and amplify in further stages of the development of software [1, 54]. The software metrics capture quality attributes like cohesion, encapsulation, coupling, complexity, inheritance, etc, and provides useful insights into the software. We have exploited OO metrics in this thesis work because they are widely used for building SDP models in literature [2, 3]. Many filter-based and wrapper based FS techniques are assessed for the construction of effective SDP models [42–45, 48]. The studies did not include the genetic algorithm or any other evolutionary technique for FS. Many of the researchers agreed for CFS to be one of the effective fea-

ture selectors. Therefore, we constructed ML models based on important features extracted using CFS in the preceding chapters.

On the recommendation of Harman [59] we explored search-based techniques for model development in the SDP field. Looking toward the positive outcome in this direction (based on Ch 7 and Ch 8), we were intrigued to find their impact on feature selection. The number of research studies in FS using search-based techniques is very less. Wahono and Herman [117] combined the genetic algorithm (GA) and bagging for FS over nine NASA datasets to develop the SDP model using 10 ML techniques with ROC-AUC as the performance measure. Their study shows significant improvement in ROC-AUC of constructed models when GA was used for FS than when all features are used. Xiang et al. [63] tried to reduce the curse of dimensionality by implementing the genetic algorithm and handled the class imbalance problem by increasing the number of defective classes using SMT in NASA datasets. They considered both forward and backward evolutionary search methods and proved the effectiveness of methods in making models with decision tree followed by NaÃ¯ve Bayes. Both of these studies used NASA datasets.

According to a survey on SEPM [200], more studies need to be conducted to explore the effectiveness of evolutionary techniques. This motivates us to analyze the effect of genetic-based FS for SDP. Genetic algorithms are preferred because it is a population-based algorithm with parallel processing and global searching capabilities. Thus, this chapter tends to explore the two variations of the genetic algorithm–generational genetic algorithm (GGA) and steady-state genetic algorithm (SGA) along with the most commonly used CFS technique to answer the following research questions:

1. RQ1: Which features are frequently selected by the FS techniques?

2. RQ2: What is the comparative performance of SDP models developed when features are minimized using evolutionary FS techniques and CFS in terms of Sensitivity, ROC-AUC, Balance, and G-Mean?

3. RQ3: Which FS technique can be categorized as the best amongst all for SDP?
4. RQ4: Which ML technique performs the best with the best FS technique for the classification of software defects?

The objectives set for this chapter are:

- to validate the evolutionary genetic-based FS techniques for SDP,
- to perform a comparative analysis of constructed models with different ML techniques when features are selected using CFS and evolutionary FS techniques,
- to use stable performance measures like ROC-AUC, Balance, and G-Mean to perform a fair comparison, and
- to employ statistical tests to validate the result observations.

Experiments are executed with 15 different ML techniques on open-source JAVA projects. We have considered 13 projects in this study and are categorized as large, mid-sized, and small projects based on their number of classes. Empirical findings of the study are secured by using the Friedman test and Wilcoxon signed-rank tests. Both of these tests are nonparametric tests.

The rest of the chapter organization is as follows: Section 9.2 describes the research methodology and experimental design followed in this chapter is explained in Section 9.3. Section 9.4 emphasizes the obtained results and provides a detailed analysis of empirical results. Last, but not least, the discussion is presented in Section 9.5.

The results of the chapter are published in [224].

9.2 Research Methodology

This section summarizes the datasets collected, FS techniques, ML techniques, and various performance measures used in this study. It also addresses the statistical tests carried out

for result validation. The selection of appropriate FS techniques, statistical tests, and cross-validation techniques results in validating conclusions and hence mitigating conclusion validity threat.

9.2.1 Dataset Collection

13 datasets of JAVA open-source projects are considered for validation of this chapter. Data collection and project details can be referred to from Section 3.6. Table 9.1 describes the addressed datasets with their statistics. #classes represents the total number of classes, #DefClasses corresponds to the number of defective classes in the particular dataset. #%ageDefects symbolizes the percentage of defective classes in that project. We categorized these datasets further in terms of size based on the number of classes in them. Projects having (100-300) classes, (300-600) classes, and (600-900) classes are identified as small projects, mid-sized projects, and large projects in this chapter. Out of 13 projects, five are small projects, five are mid-sized and three are large projects. Revisiting #%ageDefects assist in understanding the imbalanced data problem prominent in real-world software projects. #ageDefects for 12 datasets range from 11.4% to 33.9% which is very less as compared to the percentage of non-defective classes in the projects. These datasets are therefore considered as imbalanced datasets. Only one dataset, Xalan2.6, has 46.4% of defective classes. As the number of both types of classes is approximately equal, therefore, it is considered a balanced dataset.

Table 9.1: Dataset Description according to their Size

Size	Dataset	#Classes	#DefClasses	#%ageDefects
Large	Ant1.7	745	166	22.3
	Xalan2.6	885	411	46.4
	Xalan2.4	723	110	15.2
Mid-sized	Jedit4.1	312	79	25.3
	Xerces1.3	453	69	15.2
	Jedit4.0	306	75	24.5
	Ivy2.0	352	40	11.4

Size	Dataset	#Classes	#DefClasses	#%ageDefects
	Jedit4.2	367	48	13.1
Small	Log4j1.0	135	34	25.2
	Log4j1.1	109	37	33.9
	Synapse1.1	222	60	27
	Synapse1.2	256	86	33.6
	Jedit3.2	272	90	33.1

OO metrics include Response For a Class (RFC), Lack of Cohesion in Methods (LCOM), Coupling Between Objects (CBO), Number of Children (NOC), Depth of Inheritance Tree (DIT), Weighted Methods of a Class (WMC), Method of Functional Abstraction (MFA), Data Access Metric (DAM), Number of Public Methods (NPM), Cohesion among Methods of a class (CAM) and Measure of Aggression (MOA), Efferent Coupling (Ce), Afferent Coupling (Ca), Inheritance Coupling (IC), Coupling Between Methods of a Class (CBM), Average Method Complexity (AMC), a variant of LCOM (LCOM3), the maximum Cyclomatic complexity (Max_CC), average cyclomatic complexity (Avg_CC)) and Lines of Code (LOC) [18, 36, 37, 142] . The details of metrics can be referred from Section 3.3.

9.2.2 FS Techniques

FS techniques can be organized into three categories- filter methods, wrapper methods, and embedded methods. The focal point of this study is evolutionary-based wrapper methods and results are compared with one of the filter methods - the CFS technique. Details of these techniques with their parameter settings are discussed in Section 3.7.2.

9.2.3 Machine Learning Techniques

This study employs 15 ML techniques that can be categorized as statistical techniques, neural networks, nearest neighbor methods, ensemble methods, and decision trees. Parameter settings and details of ML techniques used in this chapter are mentioned in Table 3.6 and

Section 3.9.1.

9.2.4 Performance Measures

The results of defect prediction models developed by various ML techniques with different FS methods are evaluated using Sensitivity, G-Mean, Balance, and ROC-AUC performance measures. These measures are explained in Section 3.11.

9.2.5 Statistical Tests

Statistical validation empowers the findings of the study. Therefore, as done in all other chapters, here also we used a nonparametric Friedman test [213] followed by Wilcoxon signed-rank test [209] to perform the pairwise comparison. These tests are expounded in Section 3.12.

9.3 Experimental Framework

This study tends to find the features of datasets that are frequently selected by GGA, SGA, and CFS to answer RQ1. GGA and SGA are variants of GA which is one of the most popular evolutionary techniques. To answer RQ2 and RQ3, this study accomplishes ten-fold cross-validation within project defect prediction illustrated in subsection 3.10.2. Figure 9.1 illustrates the experimental setup followed in the study to get the solutions for the set of RQs.

In the SDP process, software repositories need to be mined to extract useful attributes called software metrics or features. After data preprocessing FS techniques are applied to datasets to extract useful features. GGA and SGA are wrapper methods; therefore, selecting features involves training on the corresponding GA variant. Different features are selected in each fold of cross-validation after applying the genetic algorithm. Then models are

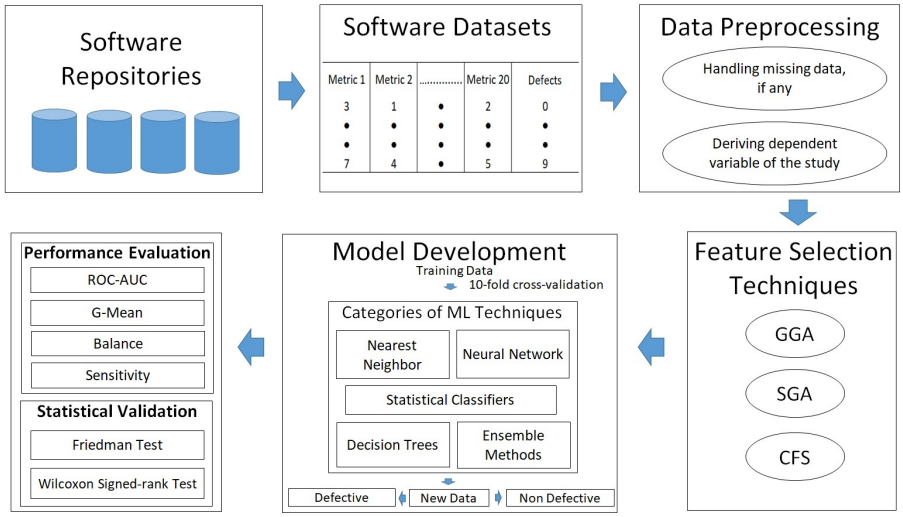

Figure 9.1: Experimental Setup

constructed using selected features for 15 ML techniques and tested with unseen data. To empirically compare their performance, ROC-AUC, Balance, and G-Mean and Sensitivity values are calculated and statistically compared using the Friedman test. If by Friedman testing, results come to be significant, further post-hoc analysis is conducted to evaluate pairwise comparison of techniques.

Features are selected using KEEL algorithms, models are build using Waikato Environment for Knowledge Analysis (WEKA) version 3.9 (www.cs.waikato.ac.nz/ml/weka/) for ML techniques, and statistical tests are carried out in SPSS. To reduce internal validity threats, the parameter settings for evolutionary FS techniques, CFS, and ML techniques incorporated in this chapter are mentioned.

9.4 Results and Analysis

9.4.1 RQ1: Which features are frequently selected by evolutionary FS techniques and CFS?

FS techniques are believed to reduce the curse of dimensionality resulting in improved performance of ML techniques. This fact is well proven for CFS [42, 51].

This chapter extends this concept for GA-based feature selection. In the direction of exploration of the impact of SBTs, this RQ addresses the identification of features of software engineering related datasets that are important and play a crucial role in predicting software defects using GA variants. The structure of the software can be evaluated using its IQAs. This RQ also aims to find the relationship between these IQAs and FS techniques. To serve the purpose, GGA, SGA, and CFS are applied on 12 datasets in this chapter.

Features selected by Evolutionary FS Techniques

As discussed, subsets of features are validated for ten folds of genetically generated subsets. So, when GGA or SGA is applied, then different features are selected in each fold. Table 9.2 and Table 9.3 scribe the features for each dataset that were selected in at least five partitions during ten-fold validation for GGA and SGA.

Table 9.2: Frequently selected features by GGA

Dataset	GGA-selected Features
Ant1.7	DIT, NOC, RFC, LCOM3, LOC, MFA, AMC, Max_CC
Xalan2.6	WMC, CBO, RFC, LCOM, Ce, MFA, CBM, AMC
Xalan2.4	DIT, NOC, RFC, Ce, NPM, LOC, MOA, CAM, CBM
Jedit4.1	CBO, Ce, LCOM3, LOC, IC, CBM, Avg_CC
Xerces1.3	NOC, CBO, Ce, NPM, MFA, CBM, Avg_CC
Jedit4.0	WMC, DIT, CBO, Ca, Ce, NPM, DAM, MOA, MFA, CBM, Max_CC
Ivy2.0	WMC, DIT, CBO, RFC, LCOM3, MOA, CAM, AMC, Max_CC

Dataset	Frequently selected features by GGA
Jedit4.2	CBO, NPM, LCOM3, LOC, CAM, CBM, Avg_CC
Log4j1.0	WMC, NOC, Ce, NPM, CAM, Max_CC
Log4j1.1	CBO, RFC, DAM, MOA, MFA, CAM, AMC
Synapse1.1	WMC, DIT, CBO, RFC, Ce, NPM, MOA, CBM, AMC, Max_CC
Synapse1.2	RFC, Ce, LCOM3, LOC, MFA, CAM, Max_CC, Avg_CC
Jedit3.2	DIT, RFC, Ce, MOA, MFA, CAM, CBM, AMC, Max_CC, Avg_CC

Table 9.3: Frequently selected features by SGA

Dataset	Frequently selected features by SGA
Ant1.7	WMC, DIT, RFC, LCOM, NPM, LOC, MOA, MFA, Max_CC
Xalan2.6	CBO, RFC, LCOM, Ca, Ce, LOC, MFA, CAM, CBM, AMC
Xalan2.4	DIT, CBO, Ca, NPM, LOC, MOA, CAM, CBM, AMC, Max_CC
Jedit4.1	WMC, NOC, CBO, RFC, LCOM, Ca, Ce, NPM, LOC, MOA, IC, CBM, Max_CC
Xerces1.3	WMC, NOC, CBO, RFC, CA, NPM, MOA, MFA, IC
Jedit4.0	DIT, NOC, CBO, LCOM, Ce, NPM, DAM, MOA, MFA, CBM, Max_CC
Ivy2.0	DIT, LCOM, Ca, LCOM3, MFA, CAM, IC, AMC
Jedit4.2	WMC, LCOM, NPM, LCOM3, AMC
Log4j1.0	DIT, NOC, CBO, LCOM, Ca, Ce, NPM, LOC, CBM, Max_CC
Log4j1.1	CBO, RFC, NPM, DAM, MOA, MFA, CBM, AMC, Max_CC
Synapse1.1	DIT, CBO, RFC, LCOM, Ca, Ce, NPM, LCOM3, LOC, Max_CC
Synapse1.2	WMC, DIT, NOC, CBO, LCOM, Ca, NPM, LOC, DAM, MFA, Max_CC, Avg_CC
Jedit3.2	DIT, CBO, Ce, NPM, LOC, MFA, CAM, IC, CBM, AMC, Max_CC, Avg_CC

20 OO metrics can be categorized into different IQAs of the software like coupling, cohesion, etc. These metrics are grouped into coupling metrics, inheritance metrics, size metrics, cohesion metrics, composition metrics, encapsulation metrics, and complexity metrics. In Table 9.4, 'IQA' represents internal quality attributes and 'OO Metrics' rep-

resents the features related to that IQA. When analyzing the frequently selected features by GGA from Table 9.2, it was recognized that CBO, RFC, and CBM are selected in 61.5% of datasets. These three metrics fit directly in the coupling category and this clearly illustrates the relevance of coupling metrics in building better defect prediction models with GGA. 53.8% of datasets use Ce, MFA, CAM, and max_CC. Considering SGA, NPM (size metric) is selected by 84.6% of datasets and is pursued by CBO (coupling metric) with 76.9% coverage of datasets. The importance of OO metrics can be estimated in terms of IQAs by determining the proportionate selection of these features. Table 9.4 summarizes the proportion selection of IQAs for GGA, SGA, and CFS. As is evident from Table 9.4, when GGA is exploited to decide on features, the proportion selection of composition and complexity metrics is 0.462. Coupling metrics also have proven their potential in model development with a proportion selection of 0.449. Similarly, relevant IQAs can be targeted for SGA. Preferable IQAs when using SGA are size, inheritance, and coupling with the proportion selection of 0.577, 0.538, and 0.526 respectively. The encapsulation metric that is represented by the DAM was the least selected in both GGA and SGA with a proportion selection of 0.154 and 0.231 respectively.

Table 9.4: Proportion Selection of IQAs in GGA, SGA, and CFS

IQA	OO Metrics	Proportion selection		
		GGA	SGA	CFS
Coupling	Ca, CBO, Ce, RFC, CBM, IC	0.449	0.526	0.564
Inheritance	NOC, DIT, MFCA	0.436	0.538	0.154
Size	WMC, NPM, LOC, AMC	0.423	**0.577**	0.635
Cohesion	LCOM, CAM, LCOM3	0.333	0.41	0.641
Composition	MOA	**0.462**	0.462	**0.692**
Encapsulation	DAM	0.154	0.231	0.462
Complexity	Avg_CC, Max_CC	**0.462**	0.423	0.5

Features selected by CFS Technique

Selected features for datasets used in this chapter can be referred from Table 3.5. CFS is

applied using the bestFirst search. LCOM, Ce, and RFC are selected by 84.6% of datasets followed by LOC. The LOC is preferred by 76.9% of datasets. CBO and MOA emerged as good predictors as they are selected by 69.2% of datasets. According to the IQAs' analysis from Table 9.4, CFS features that favor the construction of effective models are composition, cohesion, and size metrics. The proportion selection of these metrics is 0.692, 0.642, and 0.635 respectively. Inheritance and encapsulation metrics depict somewhat opposite proportion selection in evolutionary FS and CFS. While inheritance metrics are the preferable metrics with GGA and SGA, these are the least selected features by CFS with a proportion selection of 0.154. Similarly, encapsulation metrics that are the least preferred in GGA and SGA are proved to be good predictors in CFS with the proportion selection of 0.462.

9.4.2 RQ2: What is the comparative performance of SDP models developed using evolutionary FS techniques and CFS in terms of ROC-AUC, Balance, G-Mean, and Sensitivity?

To answer this research question, models are developed using features selected by GGA, SGA, and CFS. The performances of these models are compared with those of BASE models. BASE models use the original feature set to predict defects. This means that no FS technique is used in developing these models. Constructed models can be divided into four cases based on the FS technique involved:

- Case 1: When the original feature set is used, that is, all the 20 metrics are used to build a model. It is denoted by 'BASE'. In this case, no FS technique is involved.
- Case 2: When GGA is exercised for selecting relevant features.
- Case 3: When SGA is exercised for selecting relevant features.

- Case 4: When CFS is exercised for selecting relevant features.

13 projects that are analyzed in this study can also be categorized according to size as:

- Large projects: number of classes is in between 600 and 900 (Ant1.7, Xalan2.6, Xalan2.4)
- Mid-sized projects: number of classes is in between 300 and 600 (Jedit4.1, Xerces1.3, Jedit4.0, Ivy2.0, Jedit4.2)
- Small projects: number of classes is in between 100 and 300 (Log4j1.0, Log4j1.1, Synapse1.1, Synapse1.2, Jedit3.2)

The four cases of FS are examined using ROC-AUC, Balance, and G-Mean cumulatively for all ML techniques.

ROC-AUC Analysis:

The ROC-AUC values for developed models are depicted in Tables 9.5, 9.6 and 9.7 for large, mid-sized, and small projects with case-specific mean and median values. The highest ROC-AUC values achieved in all four FS cases is boldface typed for each ML technique in the Tables 9.5, 9.6 and 9.7.

The models constructed using GGA exhibits the highest ROC-AUC value for Ant1.7, Xalan2.6, Xalan2.4, Jedit4.1, Jedit4.0, Jedit4.2, Ivy2.0, Log4j1.1, and Synapse1.2. Models developed using GGA demonstrated the best ROC-AUC values of 0.87 in the Log4j1.1 dataset. Another evolutionary FS technique- SGA also yields the maximum ROC-AUC value of 0.85 and 0.84 for two datasets- Jedit3.2 and Xerces1.3 respectively. In Xerces1.3, BASE also got an ROC-AUC value of 0.84 with RSS, indicating their similar performances.

Detailed Ant1.7 exhibits the highest ROC-AUC values for remarkably 13 ML techniques on the use of the evolutionary FS technique. The highest value attained for Ant1.7 is 0.83 gained by LR, SL, and LMT with GGA. The use of CFS gave the maximum ROC-AUC value only for KStar in Ant1.7. GGA was able to get an ROC-AUC value of more

than 0.8 for 9 datasets as compared to only 5 datasets when no FS was done for Ant1.7. Similarly, ROC-AUC results for GGA and SGA seem better for the remaining two large datasets. Xalan2.6 depicted 0.84 as the highest ROC-AUC value when RSS is applied to train and build model and 10 ML techniques were able to determine ROC-AUC value greater than 0.72 as compared to only four ML techniques in the BASE. Xalan2.4 achieved a maximum of 0.80 ROC-AUC value for LB with GGA selection of features.

Table 9.5: ROC-AUC Performance of ML Models for Large Projects

Dataset	Case	NB	LR	SL	LB	MLP	IBk	KStar	ABM1	Bag	ICO	LMT	RT	RSS	PART	J48	Mean	Median
Ant1.7	BASE	0.81	0.81	0.83	0.80	0.76	0.66	0.75	**0.79**	0.80	0.78	0.83	0.66	0.81	0.73	0.67	0.76	0.79
	GGA	**0.81**	**0.83**	**0.83**	**0.82**	**0.80**	0.70	0.76	0.78	0.81	**0.81**	**0.83**	0.66	**0.82**	0.77	0.74	0.78	**0.81**
	SGA	0.81	0.83	0.83	0.81	0.80	**0.72**	0.76	0.78	**0.82**	0.80	0.83	**0.69**	0.81	**0.79**	**0.74**	**0.79**	0.80
	CFS	0.81	0.82	0.83	0.80	0.79	0.69	**0.78**	0.78	0.80	0.80	0.83	0.67	0.81	0.76	0.73	0.78	0.80
Xalan2.6	BASE	0.79	0.80	**0.81**	0.79	0.80	0.75	0.77	**0.81**	0.84	**0.79**	**0.81**	0.68	0.84	**0.80**	0.76	**0.79**	**0.80**
	GGA	0.76	0.80	0.80	**0.80**	0.80	**0.77**	**0.81**	0.80	0.84	**0.79**	0.78	**0.71**	**0.84**	0.79	0.75	0.79	0.80
	SGA	0.76	0.79	0.79	0.77	**0.81**	**0.77**	0.80	0.79	**0.84**	0.77	0.80	0.70	0.83	0.79	**0.77**	0.78	0.79
	CFS	**0.79**	**0.81**	0.81	0.79	0.80	0.75	0.77	0.78	0.80	0.78	0.81	0.66	0.81	0.77	0.72	0.78	0.79
Xalan2.4	BASE	0.74	0.76	0.71	0.77	0.72	0.61	0.72	0.77	0.77	0.76	0.7	0.61	0.76	0.68	0.55	0.71	0.72
	GGA	**0.75**	0.76	**0.76**	**0.80**	0.73	0.65	**0.78**	**0.77**	**0.81**	**0.79**	**0.76**	0.61	**0.79**	0.71	0.69	**0.74**	**0.76**
	SGA	0.74	0.75	**0.76**	0.77	0.75	**0.65**	0.78	0.73	0.76	0.75	**0.76**	**0.64**	0.78	0.68	0.62	0.73	0.75
	CFS	**0.75**	**0.77**	0.68	0.79	**0.76**	0.61	0.71	0.73	0.78	0.77	0.67	0.57	0.78	**0.72**	**0.70**	0.72	0.73

Table 9.6: ROC-AUC Performance of ML Models for Mid-sized Projects

Dataset	Case	NB	LR	SL	LB	MLP	IBk	KStar	ABM1	Bag	ICO	LMT	RT	RSS	PART	J48	Mean	Median
Jedit4.1	BASE	0.77	0.82	0.82	0.75	0.73	0.66	**0.83**	**0.79**	0.80	**0.76**	0.82	0.68	**0.79**	0.72	0.67	0.76	0.77
	GGA	0.79	**0.84**	**0.82**	**0.76**	0.79	0.70	0.80	0.77	0.78	0.75	0.81	0.63	0.78	**0.73**	0.63	0.76	0.78
	SGA	0.74	0.81	0.81	0.75	**0.80**	**0.71**	0.81	0.76	0.80	0.75	0.79	**0.69**	0.79	0.71	**0.71**	0.76	0.76
	CFS	**0.80**	0.82	0.82	0.73	0.79	0.66	0.81	0.76	**0.81**	0.75	**0.82**	0.67	0.78	0.70	0.68	0.76	**0.78**
Xerces1.3	BASE	0.78	**0.78**	**0.80**	0.80	**0.77**	0.77	0.79	0.79	0.81	0.79	**0.80**	0.65	**0.84**	**0.77**	**0.67**	**0.77**	**0.79**
	GGA	0.65	0.68	0.64	0.76	0.71	0.82	0.83	0.76	**0.82**	0.76	0.67	0.68	0.77	0.71	0.59	0.72	0.71
	SGA	0.72	0.74	0.75	0.78	0.74	**0.82**	**0.84**	**0.81**	0.78	0.74	0.74	**0.71**	0.79	0.71	0.63	0.75	0.74
	CFS	**0.79**	0.76	0.67	**0.80**	0.75	0.77	0.81	0.77	0.81	**0.79**	0.74	0.69	0.83	0.70	0.61	0.75	0.77
Jedit4.0	BASE	0.74	0.78	0.77	0.78	0.81	0.70	0.75	0.78	0.75	0.76	0.77	**0.68**	0.77	0.64	0.65	0.74	0.76
	GGA	0.74	0.78	0.78	**0.78**	**0.82**	0.74	0.75	**0.79**	0.73	0.78	**0.79**	0.66	**0.78**	0.63	0.65	0.75	**0.78**
	SGA	0.76	0.79	**0.78**	0.77	0.80	**0.76**	**0.78**	0.77	**0.76**	0.76	0.77	0.65	0.77	**0.74**	0.65	**0.75**	0.77
	CFS	**0.76**	**0.79**	0.77	**0.78**	0.77	0.68	0.69	0.72	0.70	**0.78**	0.78	0.61	**0.78**	0.69	**0.68**	0.73	0.76
Ivy2.0	BASE	0.77	0.76	0.76	0.77	0.70	0.62	0.68	**0.73**	0.73	0.71	0.76	0.61	0.77	**0.74**	0.68	0.72	0.73
	GGA	0.79	0.77	0.74	0.77	**0.76**	**0.70**	0.72	0.69	0.74	**0.76**	0.74	0.63	**0.81**	0.70	**0.73**	**0.74**	0.74
	SGA	0.74	0.72	0.71	0.76	0.71	0.60	0.71	0.64	0.77	0.72	0.67	0.59	0.76	0.72	0.63	0.70	0.71
	CFS	**0.80**	**0.78**	**0.76**	**0.78**	0.74	0.61	**0.72**	0.70	**0.77**	0.74	**0.76**	**0.64**	0.78	0.72	0.72	0.73	**0.74**
Jedit4.2	BASE	0.84	0.81	0.82	0.82	0.75	0.66	0.77	0.76	**0.83**	0.81	0.82	0.61	0.84	0.59	0.71	0.76	0.81
	GGA	0.84	**0.82**	0.82	**0.85**	0.81	**0.71**	**0.79**	0.79	0.80	**0.85**	0.80	0.66	**0.85**	0.79	0.76	**0.80**	0.80
	SGA	0.76	0.81	0.78	0.83	0.81	0.68	0.74	0.75	0.80	0.84	0.78	**0.66**	0.80	0.70	0.69	0.76	0.78
	CFS	**0.84**	0.81	**0.84**	0.81	**0.82**	0.65	0.78	**0.81**	0.81	0.80	**0.84**	0.64	0.84	**0.79**	**0.76**	0.79	**0.81**

Table 9.7: ROC-AUC Performance of ML Models for Small Projects

Dataset	Case	NB	LR	SL	LB	MLP	IBk	KStar	ABM1	Bag	ICO	LMT	RT	RSS	PART	J48	Mean	Median
Log4j1.0	BASE	**0.84**	0.74	0.81	0.76	0.71	0.68	0.73	0.73	**0.8**	0.73	0.81	0.58	0.77	0.70	0.70	0.74	0.73
	GGA	**0.84**	0.81	0.84	0.78	0.80	**0.77**	**0.78**	0.75	0.79	0.8	0.84	**0.71**	**0.78**	**0.77**	0.60	**0.78**	**0.78**
	SGA	0.81	0.82	0.82	**0.82**	**0.81**	0.70	0.75	**0.77**	0.77	**0.81**	0.82	0.66	0.76	0.76	**0.74**	0.77	0.77
	CFS	0.83	**0.82**	**0.85**	0.78	0.73	0.64	0.73	0.71	0.77	0.76	**0.85**	0.62	0.78	0.73	0.65	0.75	0.76
Log4j1.1	BASE	0.82	0.84	0.84	0.79	0.80	0.73	0.76	0.76	0.52	0.79	0.84	0.71	0.82	0.75	**0.72**	0.77	0.79
	GGA	0.84	**0.87**	**0.85**	**0.85**	0.83	**0.85**	**0.84**	0.79	**0.83**	0.81	**0.85**	**0.73**	0.82	0.74	0.69	**0.81**	**0.83**
	SGA	0.83	0.84	0.85	0.81	**0.83**	0.77	0.8	**0.79**	0.8	0.8	0.85	**0.73**	0.83	0.75	0.72	0.8	0.8
	CFS	**0.86**	0.84	0.83	0.83	0.82	0.75	0.82	0.76	0.82	**0.81**	0.81	0.68	**0.85**	**0.80**	0.72	0.8	0.82
Synapse1.1	BASE	0.72	0.72	0.74	**0.74**	0.71	0.70	0.73	0.76	0.76	0.72	0.74	0.64	0.75	0.68	**0.67**	0.72	0.72
	GGA	0.76	**0.75**	0.74	0.72	0.72	**0.76**	0.78	**0.77**	0.74	0.72	0.73	**0.69**	0.72	0.65	0.65	0.73	0.73
	SGA	0.72	0.73	0.75	0.69	0.70	0.73	0.75	0.77	**0.79**	0.68	0.75	0.66	0.70	**0.70**	0.63	0.71	0.72
	CFS	**0.76**	0.74	**0.76**	0.73	**0.78**	0.67	**0.80**	0.73	0.77	**0.73**	**0.76**	0.67	**0.77**	0.68	0.67	**0.73**	**0.74**
Synapse1.2	BASE	0.76	0.75	0.76	0.79	0.72	0.72	0.81	0.79	**0.81**	0.78	0.75	0.68	0.79	0.74	0.69	0.75	0.76
	GGA	0.73	0.73	0.74	**0.81**	0.77	**0.74**	**0.82**	0.81	0.78	**0.81**	0.74	**0.69**	**0.80**	**0.75**	0.73	**0.76**	0.75
	SGA	0.71	0.74	0.76	**0.79**	0.79	0.72	**0.81**	0.81	0.8	0.77	0.78	0.66	0.79	0.71	0.71	0.76	**0.77**
	CFS	**0.78**	**0.80**	**0.79**	0.78	0.76	0.67	0.77	0.75	0.80	0.76	**0.82**	0.67	0.76	0.73	0.74	0.76	0.76
Jedit3.2	BASE	0.79	**0.85**	**0.84**	0.82	0.77	0.72	0.80	0.80	0.82	0.80	**0.84**	**0.69**	0.81	0.72	0.67	0.78	0.80
	GGA	**0.82**	0.84	0.82	0.82	0.8	**0.74**	0.79	**0.80**	0.81	0.78	0.77	**0.69**	**0.83**	**0.80**	**0.73**	0.79	0.80
	SGA	0.78	0.84	0.81	**0.85**	**0.82**	0.72	**0.82**	0.79	**0.83**	**0.83**	0.81	**0.69**	0.83	0.76	0.71	**0.79**	**0.81**
	CFS	0.81	0.83	0.82	0.82	0.82	0.73	0.77	0.80	0.81	0.79	0.79	0.67	0.82	0.77	0.70	0.78	0.80

It seems that GGA and CFS may have a comparable performance for mid-sized projects. For Jedit4.1, the winner is GGA for the LR technique with an ROC-AUC value of 0.84. For Xerces1.3 and Jedit4.1, there is negligible improvement in most of the ML techniques except for IBk, KStar, and RT with feature selection techniques. In both these datasets, BASE performance gave better ROC-AUC performance. But for the rest of the three datasets, GGA or SGA have performed well.

Considering CFS, it gives the best performance with SL and LMT for Log4j1.0 gaining an ROC-AUC value of 0.85. Synapse1.1 also scored ROC-AUC value as 0.80 for KStar with CFS. Also, Ivy2.0 results reveal the ROC-AUC value as 0.80. Although ROC-AUC values obtained by CFS are not the highest ones in any of the remaining three datasets, they are competitive enough. Looking at individual ML technique performances, CFS has remarkably performed well for the number of ML techniques in Jedit4.0, Jedit4.2, and Ivy2.0. Jedit4.0, Ivy2.0, Jedit4.2 and Log4j1.1 have an approximately equivalent number of ML techniques that have ROC-AUC values more than 0.75. In Jedit4.2, 13 ML techniques have ROC-AUC more than 0.75 when features were selected using GGA as compared to 10 ML techniques in the original set of features represented by the BASE.

For small datasets, from Table 9.7, it can be noticed that different FS technique performed better for different datasets. There is an improvement in ROC-AUC performances of Log4j1.0, Log4j1.1, and Synapse1.1 when FS is done. Even though the highest value of ROC-AUC (0.85) is achieved in Log4j1.0 with CFS by SL, analysis of the results asserts that the evolutionary techniques (GGA and SGA) performed comparatively better than CFS. It is visible from Table 9.7 that GGA performed superior to other FS techniques and the BASE case for Log4j1.1. Talking of Synapse1.1 and Synapse1.2, again the highest ROC-AUC values are presented by evolutionary FS however ML technique-wise the maximum ROC-AUC values are achieved by CFS most of the time. CFS and GGA may likely have comparable potential to predict defects for these two small projects.

The comparison of ROC-AUC median values of all the datasets in four cases is pre-

sented as the bar graph in Figure 9.2. In terms of median values, only Xerces1.3 does not show improvement in the application of FS techniques. Color code for bars representing GGA, SGA, CFS, and BASE performances is shown in Figure 9.2. SDP models developed by employing GGA display the highest values for Ant1.7, Xalan2.4, Jedit4.0, Log4j1.0, Synapse1.1, Jedit4.1, Ivy2.0, and Jedit3.2. Figure 9.2 explicates the incompetency of FS techniques to amplify model performances for Xerces 1.1 and Xalan2.6. The highest median value for SGA is accomplished by small datasets Synapse1.2 and Jedit3.2. The median values of models developed using GGA range from 0.71 to 0.83. ROC-AUC value 0.83 is the highest amongst all median values in all the cases which are depicted by GGA for Log4j1.1.

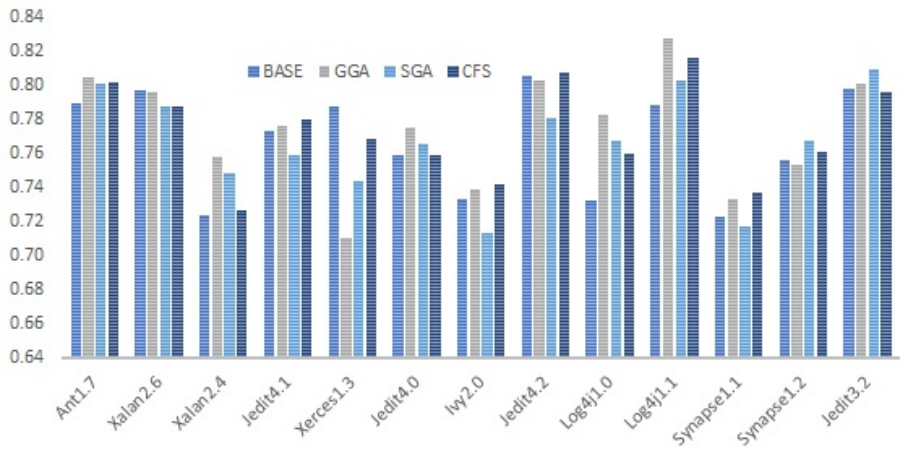

Figure 9.2: Dataset wise Median ROC-AUC values of the BASE, GGA, SGA, and CFS for 13 datasets

Balance Analysis:

Tables 9.8, 9.9 and 9.10 records the Balance values attained SDP models on large, mid-sized, and small projects respectively using original features and a reduced feature set with GGA, SGA, and CFS.

Table 9.8: Balance Performance of ML Models for Large Projects

Dataset	Case	NB	LR	SL	LB	MLP	IBk	KStar	ABM1	Bag	ICO	LMT	RT	RSS	PART	J48	Mean	Median
Ant1.7	BASE	67.05	56.81	55.58	67.77	61.76	59.34	58.25	**64.59**	62.92	68.61	55.58	61.45	62.24	58.43	64.75	61.68	61.76
	GGA	**69.68**	56.83	56.87	**68.35**	**62.15**	64.01	56.50	63.71	**63.47**	**69.61**	56.87	62.02	**62.57**	**69.80**	63.89	**63.09**	**63.47**
	SGA	60.46	**57.69**	**57.29**	65.52	61.71	**64.51**	57.35	62.10	60.9	69.15	**57.29**	**63.98**	59.64	62.57	66.74	61.79	61.71
	CFS	64.43	56.81	55.17	66.63	59.29	63.76	**58.59**	60.16	63.01	69.07	55.17	63.05	60.88	65.38	**68.55**	62.00	63.01
Xalan2.6	BASE	**62.04**	**70.10**	**70.10**	68.79	71.53	70.27	67.43	72.53	73.49	69.13	**73.83**	66.16	72.49	70.86	71.83	70.04	70.27
	GGA	60.01	69.06	69.06	**70.47**	69.53	**74.55**	**71.88**	**73.72**	**74.07**	69.78	69.72	**72.19**	72.52	**71.53**	71.90	70.67	71.53
	SGA	60.91	69.28	68.29	69.65	**71.84**	73.63	71.23	72.97	**74.07**	68.04	71.81	70.54	**72.84**	70.00	**72.35**	70.50	71.23
	CFS	59.91	70.07	69.97	68.91	70.29	69.55	68.61	63.78	70.03	**70.71**	69.97	66.90	69.99	60.51	62.58	67.45	69.55
Xalan2.4	BASE	53.82	**42.75**	38.28	38.26	46.40	52.53	48.25	**50.82**	42.68	33.75	38.91	52.24	**35.07**	36.29	43.83	43.59	42.75
	GGA	**54.45**	**42.75**	42.12	**42.10**	**51.01**	**55.22**	**54.01**	49.54	**47.88**	**40.84**	42.12	51.77	31.86	34.40	**43.95**	**45.60**	**43.95**
	SGA	53.82	40.20	**42.13**	39.53	43.97	55.12	52.14	48.29	41.42	33.76	**42.13**	**56.14**	30.57	32.48	40.81	43.50	42.13
	CFS	52.75	40.83	35.69	41.45	45.89	50.59	43.86	46.88	44.64	38.26	36.32	47.69	31.86	**38.88**	38.24	42.26	41.45

Table 9.9: Balance Performance of ML Models for Mid-sized Projects

Dataset	Case	NB	LR	SL	LB	MLP	IBk	KStar	ABM1	Bag	ICO	LMT	RT	RSS	PART	J48	Mean	Median
Jedit4.1	BASE	59.31	67.21	**65.76**	**64.14**	59.77	60.52	62.53	**64.28**	**63.74**	65.08	65.76	66.51	57.65	56.61	56.96	**62.39**	**63.74**
	GGA	**61.82**	64.71	56.73	63.27	62.01	64.00	**62.70**	63.06	59.89	62.53	57.62	58.41	**62.14**	54.02	54.5	60.49	62.01
	SGA	53.06	63.06	62.23	62.19	59.35	**67.11**	60.83	61.88	60.05	**66.02**	61.21	**66.64**	61.08	**60.23**	**61.77**	61.78	61.77
	CFS	59.47	**68.25**	60.42	60.10	**62.14**	59.51	62.53	61.10	63.40	65.95	61.31	63.68	58.66	59.50	59.43	61.70	61.10
Xerces1.3	BASE	53.51	**52.75**	**51.79**	56.87	**59.70**	61.63	**62.80**	**59.79**	50.65	56.89	52.79	57.40	**48.76**	48.64	**59.85**	55.59	56.87
	GGA	46.46	37.47	32.36	47.70	49.72	**68.00**	60.87	57.65	53.81	45.66	47.67	57.67	38.51	45.66	48.68	49.19	47.70
	SGA	51.50	47.71	47.72	**59.98**	55.88	67.90	60.85	59.70	57.90	**59.96**	**54.85**	61.76	48.75	49.75	53.80	**55.87**	55.88
	CFS	**64.83**	43.57	36.44	57.89	56.80	65.42	60.96	59.65	**58.92**	54.86	51.75	**63.53**	46.69	**54.80**	58.83	55.66	**57.89**
Jedit4.0	BASE	52.42	57.26	56.32	61.84	**62.50**	67.47	56.66	60.76	65.56	62.72	57.22	**64.69**	59.13	52.63	59.49	59.78	59.49
	GGA	48.82	57.29	53.63	60.1	61.07	**70.26**	56.02	**63.60**	**65.66**	**63.84**	59.13	61.34	56.32	48.68	57.92	58.91	59.13
	SGA	49.65	52.54	55.53	58.91	61.17	69.87	**57.22**	61.33	61.07	54.25	53.51	60.63	50.88	**55.73**	57.76	57.34	57.22
	CFS	**2.57**	**60.20**	**58.35**	**63.84**	59.3	63.6	54.55	63.43	64.58	63.79	**61.14**	56.66	**59.17**	54.37	**61.64**	**59.81**	**60.20**
Ivy2.0	BASE	60.40	**45.15**	38.12	41.58	**50.25**	53.69	48.45	45.11	**43.30**	39.79	38.12	51.65	29.29	**45.09**	36.36	44.42	**45.09**
	GGA	**60.67**	39.87	**39.89**	41.62	43.30	**62.76**	**52.18**	45.13	41.62	36.30	**39.89**	53.5	**31.06**	43.42	**46.94**	**45.21**	43.30
	SGA	39.79	31.06	29.29	41.63	39.79	44.97	43.27	43.28	38.1	38.12	31.05	48.34	29.29	43.37	36.31	38.51	39.79
	CFS	58.95	39.88	**39.89**	**45.09**	43.42	48.34	52.10	**48.59**	41.60	**43.28**	**39.89**	**53.74**	**31.06**	38.12	38.12	44.14	43.28
Jedit4.2	BASE	**58.44**	48.39	**52.81**	51.30	52.63	55.49	51.24	52.71	51.35	48.37	**52.81**	52.26	32.23	49.81	**59.93**	51.32	52.26
	GGA	56.86	**51.36**	48.41	**51.35**	51.36	**65.83**	55.61	**58.62**	**57.25**	**54.31**	48.41	56.89	**38.13**	**58.67**	58.67	**54.12**	**55.61**
	SGA	45.47	45.47	42.54	41.04	45.49	56.69	54.08	55.54	48.41	45.44	42.54	**59.43**	30.76	41.07	38.12	46.14	45.47
	CFS	57.00	45.44	46.92	49.83	**55.78**	58.22	**56.97**	57.07	51.34	45.44	46.92	55.37	35.18	41.05	46.85	49.96	49.83

Table 9.10: Balance Performance of ML Models for Small Projects

Dataset	Case	NB	LR	SL	LB	MLP	IBk	KStar	ABM1	Bag	ICO	LMT	RT	RSS	PART	J48	Mean	Median
Log4j1.0	BASE	64.47	63.82	**68.30**	61.12	61.29	62.91	53.48	**67.50**	**69.49**	53.6	**68.30**	52.35	54.16	57.42	**65.31**	61.57	62.91
	GGA	**70.21**	64.2	64.47	65.11	55.77	**69.69**	**57.93**	61.30	66.00	**68.30**	64.47	**67.30**	54.19	57.57	59.60	**63.07**	**64.47**
	SGA	60.33	**66.26**	64.47	64.47	**68.30**	61.78	55.53	60.93	63.96	62.33	64.47	59.45	**62.40**	**63.82**	60.26	62.58	62.40
	CFS	66.61	66.13	64.40	**65.68**	55.97	59.29	47.63	59.29	68.17	59.99	64.40	56.54	54.16	58.12	60.09	60.43	59.99
Log4j1.1	BASE	76.06	69.37	71.07	68.43	63.31	72.06	67.57	69.92	**71.82**	71.07	71.07	69.46	69.17	**72.50**	**71.50**	70.29	71.07
	GGA	76.06	**74.46**	72.11	71.50	**76.55**	**83.40**	**73.26**	**70.35**	68.86	70.00	72.11	**71.60**	66.79	64.92	64.71	71.78	71.60
	SGA	75.76	73.63	74.46	**74.22**	74.22	74.88	70.00	63.94	70.52	69	74.46	70.76	65.25	67.27	65.10	70.90	70.76
	CFS	**76.55**	**74.46**	**74.84**	73.63	75.07	74.33	69.03	69.92	66.33	**72.11**	**74.67**	66.06	**69.36**	70.73	70.52	**71.84**	**72.11**
Synapse1.1	BASE	**67.58**	**65.20**	**57.35**	**57.00**	60.14	64.96	62.87	60.86	64.19	53.44	**57.14**	59.93	50.35	**60.62**	**63.13**	**60.32**	**60.62**
	GGA	56.08	57.35	50.38	55.61	53.91	**69.39**	62.96	63.36	**65.03**	**54.74**	51.45	**64.81**	45.70	59.08	55.84	57.71	56.08
	SGA	64.50	56.18	55.11	50.98	60.41	68.87	63.48	**64.81**	61.65	50.27	55.11	63.08	46.82	58.35	59.49	58.61	59.49
	CFS	63.68	59.59	55.11	53.30	**63.09**	64.17	**64.50**	59.94	64.12	54.54	55.11	61.61	**51.59**	56.68	58.99	59.07	59.59
Synapse1.2	BASE	65.63	61.98	55.63	69.11	63.86	69.72	69.56	68.29	**70.32**	70.03	61.36	66.64	60.38	68.23	66.8	65.84	66.80
	GGA	64.48	57.72	55.27	**75.48**	65.75	**70.14**	71.39	**73.23**	70.04	70.59	59.49	**67.37**	**66.41**	**70.70**	**70.46**	**67.23**	**70.04**
	SGA	54.16	58.94	**60.05**	69.11	69.56	69.40	**72.31**	70.51	67.56	**71.05**	64.48	63.86	61.44	60.66	66.96	65.34	66.96
	CFS	**67.44**	**62.40**	59.83	67.12	**70.13**	64.95	68.76	66.83	69.14	68.49	**64.57**	64.78	62.36	67.56	69.04	66.23	67.12
Jedit3.2	BASE	61.5	**73.31**	**73.56**	67.43	67.35	69.26	**68.23**	68.07	69.81	68.23	**73.56**	**67.20**	67.33	64.85	64.74	**68.30**	**68.07**
	GGA	**69.42**	67.89	65.57	70.09	**69.47**	**70.83**	60.2	68.97	70.57	66.75	67.74	66.79	67.63	63.72	**68.36**	67.60	67.89
	SGA	61.5	67.63	64.57	68.48	66.46	70.54	66.67	**69.82**	**73.04**	**69.59**	64.57	66.79	61.57	**67.08**	65.71	66.93	66.79
	CFS	60.10	71.20	69.15	**71.30**	66.99	70.2	62.85	67.20	70.22	64.87	68.72	63.72	**68.39**	64.51	65	66.96	67.20

The maximum Balance value is portrayed by GGA for the Log4j1.1 dataset when classified with the IBk technique. Models built up by application of GGA experienced the maximum Balance values for all the datasets except for one of the mid-sized projects-Jedit4.1. Overall the best Balance value is 83.4 revealed by Log4j1.1 for IBk. It is interesting to note that models developed using the IBk technique gave the highest Balance value for seven datasets. Statistical ML techniques were also able to give the best predictions in terms of Balance for five datasets out of which three were mid-sized projects. For large projects, the results of Ant1.7 are improved for all ML techniques but ABM1. CFS gave the highest Balance value in Ant1.7 for only on ML technique, i.e., KStar. For remaining ML techniques, evolutionary FS projected outstanding Balance values. Three ML techniques achieved a Balance greater than 69% in Ant1.7 with GGA as compared to one in CFS and none in the BASE. Balance values of GGA and SGA based models are the highest for at least 10 ML techniques for Xalan2.6 and Xalan2.4. For Xalan2.4, two ML techniques gained 74% Balance value as compared to none in CFS and BASE.

Therefore, the analysis of Table 9.8 reveals the supremacy of evolutionary FS techniques in developing effective SDP models. Basis the insights from Table 9.8, it can be realized that except for Ant1.7, SGA and BASE have comparable Balance values for large datasets, if averaged performance of ML techniques is considered. The maximum Balance value achieved is 74.55 by the IBk technique for Xalan2.6 with GGA.

Though BASE was not able to get the maximum cumulative Balance value in any of the mid-sized datasets when seen ML technique-wise BASE has the greatest Balance values for Jedit4.1 and Xerces1.3 in many ML techniques. Jedit4.0 and Ivy2.0 showed the maximum values of Balance for seven ML techniques in the CFS case. Jedit4.2 and Ivy2.0 also depicted the maximum values of Balance for eight ML techniques in the GGA case. For Ivy2.0, GGA and CFS values are similar for eight ML techniques. Except for Jedit4.2, mid-sized projects (Jedit4.1, Xerces1.3, Jedit4.0, and Ivy2.0) have comparable Balance performances in the context of ML techniques. Seven techniques were able to attain a

Balance greater than 56% whereas only two ML techniques were capable of achieving this value for the BASE and SGA cases. This indicates that GGA and CFS may have not so significant difference between their defect prediction capabilities in terms of the Balance performance measure for Ivy2.0. The performance of Jedit4.2 seems to be improved by the GGA application according to Table 9.9.

From Table 9.10, GGA again emerged as the FS technique that provides the maximum Balance value for all five datasets in the small projects category. Effective ML techniques to achieve the maximum Balance value are NB, SL, LB, and IBk. The maximum value achieved by Log4j1.0, Synapse1.1, Synapse1.2, and Jedit3.2 is 70.21, 69.39, 75.48, and 73.56 respectively. The average Balance value achieved is the highest for Log4j1.0 and Synapse1.2 for the GGA case. Log4j1.1 has the maximum average Balance value for the CFS case. In Synapse1.2, eight ML techniques achieved a Balance value greater than 70% for the GGA case. The remaining two datasets have depicted the largest mean Balance values for the BASE case only. Therefore, it may be difficult to find any major increment in model performances with the application of FS techniques in small projects based on Balance measure.

Figure 9.3 reflects the median Balance values of models developed under the four cases. The color scheme of bars is the same as that of ROC-AUC and is represented in Figure 9.3.

Ant1.7, Xalan2.6, Xalan2.4, Jedit4.2, Ivy2.0, Jedit4.2 Log4j1.0, and Synapse1.2 have the tallest bar for GGA representing the highest median values achieved by these datasets through GGA. Jedit4.0 and Log4j1.1 have the highest median values of 60.2 and 72.11 respectively gained through CFS. For Synapse1.1, Jedit4.1, and Xerces1.3 we found no improvement in median values of Balance for ML models developed using FS techniques.

G-Mean Analysis:

G-Mean values are noted in Tables 9.11, 9.12 and 9.13 for small, mid-sized, and large projects respectively. The highest value achieved by a particular ML technique is boldfaced for each dataset. Tables 9.11, 9.12 and 9.13 are analyzed to understand the impact of FS

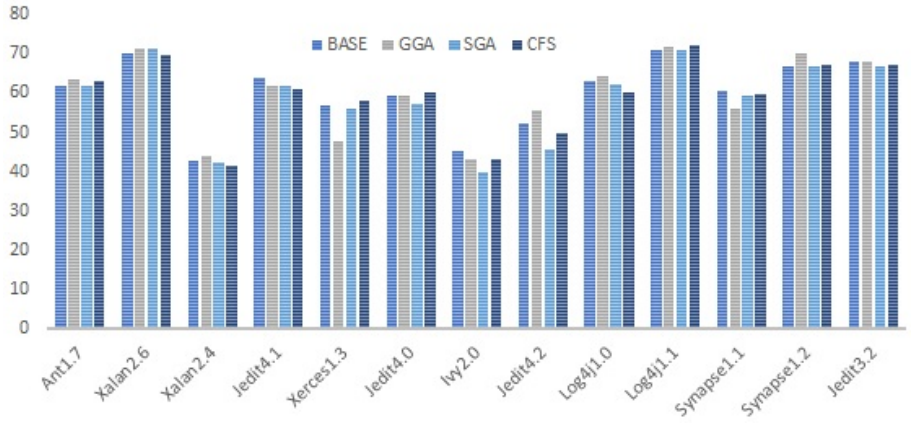

Figure 9.3: Median Balance values of the BASE, GGA, SGA, and CFS for 13 datasets

techniques in determining software defects.

GGA performed well for large and small projects. GGA and SGA depict the maximum ROC-AUC value for a particular ML technique for seven and two datasets respectively. Three datasets show the maximum G-Mean value for the BASE when models were constructed using statistical ML techniques and decision-based J48 classifier. The maximum G-Mean value of 0.84 is observed in the Log4j1.1 dataset for the GGA+IBk technique. Analysis of results according to projects is presented next.

Table 9.11: G-Mean Performance of ML Models for Large Projects

Dataset	Case	NB	LR	SL	LB	MLP	IBk	KStar	ABM1	Bag	ICO	LMT	RT	RSS	PART	J48	Mean	Median
Ant1.7	BASE	0.67	0.61	0.60	0.71	0.65	0.62	0.61	**0.67**	0.67	0.72	0.60	0.64	0.66	0.61	0.67	0.65	0.65
	GGA	**0.72**	0.61	0.61	**0.72**	**0.66**	0.67	0.60	0.66	**0.67**	**0.73**	0.61	0.64	**0.66**	**0.72**	0.67	**0.66**	**0.66**
	SGA	0.64	**0.62**	**0.62**	0.69	0.65	**0.67**	0.61	0.65	0.65	0.72	**0.62**	**0.67**	0.63	0.66	0.70	0.65	0.65
	CFS	0.68	0.61	0.59	0.70	0.63	0.66	**0.61**	0.63	0.67	0.72	0.59	0.65	0.65	0.69	**0.71**	0.65	0.65
Xalan2.6	BASE	**0.66**	0.71	0.71	0.70	0.72	0.70	0.68	0.73	0.74	0.70	**0.74**	0.66	0.73	0.71	0.72	0.71	0.71
	GGA	0.64	0.71	0.71	**0.72**	0.72	**0.75**	**0.73**	**0.74**	0.75	**0.72**	0.71	**0.72**	0.74	**0.72**	0.72	**0.72**	**0.72**
	SGA	0.65	0.71	0.70	0.70	**0.74**	0.74	0.72	0.73	**0.75**	0.69	0.73	0.71	**0.74**	0.71	**0.73**	0.72	0.72
	CFS	0.64	**0.72**	**0.72**	0.70	0.72	0.70	0.69	0.66	0.70	0.71	0.72	0.67	0.71	0.65	0.66	0.69	0.70
Xalan2.4	BASE	0.57	**0.43**	0.35	0.35	0.48	0.55	0.50	**0.53**	0.43	0.25	0.36	0.54	**0.28**	0.31	0.44	0.43	0.43
	GGA	**0.57**	0.43	0.42	**0.42**	**0.54**	**0.58**	**0.57**	0.52	**0.51**	**0.40**	0.42	0.54	0.19	0.27	**0.45**	**0.45**	**0.45**
	SGA	0.57	0.39	**0.42**	0.37	0.45	0.58	0.55	0.50	0.41	0.25	**0.42**	**0.59**	0.13	0.21	0.4	0.42	0.42
	CFS	0.56	0.40	0.30	0.41	0.47	0.53	0.44	0.48	0.46	0.35	0.31	0.49	0.19	**0.36**	0.35	0.41	0.41

Table 9.12: G-Mean Performance of ML Models for Mid-sized Projects

Dataset	Case	NB	LR	SL	LB	MLP	IBk	KStar	ABM1	Bag	ICO	LMT	RT	RSS	PART	J48	Mean	Median
Jedit4.1	BASE	0.63	0.71	**0.70**	**0.67**	0.63	0.63	0.66	**0.67**	**0.67**	0.68	**0.70**	0.67	0.61	0.6	0.59	**0.65**	**0.67**
	GGA	**0.65**	0.69	0.60	0.66	0.66	0.67	**0.66**	0.66	0.63	0.66	0.61	0.60	**0.66**	0.57	0.57	0.64	0.66
	SGA	0.56	0.67	0.67	0.65	0.63	**0.69**	0.64	0.64	0.63	**0.69**	0.65	**0.68**	0.65	0.63	**0.65**	0.65	0.65
	CFS	0.63	**0.72**	0.65	0.63	**0.66**	0.62	0.66	0.63	0.66	0.69	0.66	0.65	0.63	**0.64**	0.63	0.65	0.65
Xerces1.3	BASE	0.56	**0.56**	**0.56**	0.61	**0.63**	0.65	**0.67**	**0.64**	0.54	0.61	0.57	0.60	**0.52**	0.51	**0.64**	0.59	0.60
	GGA	0.48	0.34	0.21	0.50	0.53	**0.72**	0.65	0.61	0.58	0.48	0.5	0.61	0.36	0.48	0.51	0.5	0.5
	SGA	0.54	0.50	0.51	**0.65**	0.60	0.72	0.65	0.63	0.63	**0.65**	**0.59**	0.66	0.52	0.53	0.58	**0.60**	0.60
	CFS	**0.69**	0.44	0.31	0.62	0.61	0.68	0.66	0.63	**0.64**	0.59	0.55	**0.67**	0.49	**0.59**	0.63	0.59	**0.62**
Jedit4.0	BASE	0.55	0.61	0.60	0.65	**0.66**	0.70	0.59	0.64	0.69	0.66	0.61	**0.66**	0.63	0.56	0.62	0.63	0.63
	GGA	0.51	0.61	0.57	0.64	0.65	0.72	0.59	**0.67**	**0.69**	**0.68**	0.63	0.63	0.60	0.50	0.61	0.62	0.63
	SGA	0.52	0.55	0.59	0.62	0.66	**0.73**	**0.61**	0.64	0.65	0.57	0.57	0.63	0.54	**0.58**	0.61	0.60	0.61
	CFS	**0.56**	**0.64**	**0.63**	**0.68**	0.64	0.66	0.57	0.66	0.68	0.68	**0.65**	0.58	**0.63**	0.57	**0.65**	**0.63**	**0.64**
Ivy2.0	BASE	0.63	**0.47**	0.35	0.41	**0.53**	0.57	0.50	0.46	**0.43**	0.38	0.35	0.54	0.00	**0.46**	0.32	0.43	**0.46**
	GGA	**0.64**	0.38	**0.39**	0.41	0.43	**0.67**	**0.56**	0.47	0.41	0.31	**0.39**	0.56	**0.16**	0.44	**0.49**	**0.45**	0.43
	SGA	0.38	0.16	0.00	0.41	0.38	0.46	0.43	0.43	0.35	0.35	0.16	0.50	0.00	0.44	0.31	0.32	0.38
	CFS	0.63	0.38	0.38	**0.46**	0.44	0.50	0.55	**0.51**	0.41	**0.43**	0.38	**0.57**	0.16	0.35	0.35	0.43	0.43
Jedit4.2	BASE	**0.62**	0.51	**0.57**	0.55	0.56	0.59	0.54	0.56	0.55	0.51	**0.57**	0.55	0.20	0.53	0.64	0.54	0.55
	GGA	0.60	**0.55**	0.51	**0.55**	0.55	**0.70**	0.59	**0.63**	**0.62**	**0.59**	0.51	0.60	**0.35**	**0.63**	0.63	**0.58**	**0.59**
	SGA	0.47	0.47	0.43	0.40	0.48	0.60	0.57	0.59	0.51	0.47	0.43	**0.62**	0.14	0.41	0.35	0.46	0.47
	CFS	0.61	0.47	0.49	0.53	**0.61**	0.61	**0.61**	0.61	0.55	0.47	0.49	0.58	0.29	0.40	0.49	0.52	0.53

Table 9.13: G-Mean Performance of ML Models for Small Projects

Dataset	Case	NB	LR	SL	LB	MLP	IBk	KStar	ABM1	Bag	ICO	LMT	RT	RSS	PART	J48	Mean	Median
Log4j1.0	BASE	0.69	0.67	**0.72**	0.63	0.63	0.65	0.56	**0.70**	**0.72**	0.56	**0.72**	0.54	0.58	0.60	**0.68**	0.64	0.65
	GGA	**0.73**	0.68	0.69	0.67	0.59	**0.72**	**0.61**	0.64	0.69	**0.72**	0.69	**0.69**	0.59	0.60	0.62	**0.66**	**0.68**
	SGA	0.65	**0.70**	0.69	**0.69**	**0.72**	0.65	0.58	0.63	0.67	0.67	0.69	0.62	**0.67**	**0.67**	0.64	0.66	0.67
	CFS	0.71	0.69	0.69	0.68	0.59	0.62	0.49	0.62	0.71	0.63	0.69	0.58	0.58	0.62	0.64	0.64	0.63
Log4j1.1	BASE	0.78	0.71	0.75	0.71	0.65	0.73	0.69	0.71	**0.74**	**0.75**	0.75	0.7	0.73	0.74	0.73	0.72	0.73
	GGA	0.78	**0.77**	0.74	0.73	0.79	**0.84**	**0.74**	**0.71**	0.72	0.72	0.74	**0.72**	0.70	0.68	0.68	0.74	0.73
	SGA	0.78	0.75	0.77	0.75	0.77	0.75	0.72	0.66	0.73	0.70	0.77	0.72	0.69	0.71	0.69	0.73	0.73
	CFS	**0.79**	**0.77**	**0.78**	**0.75**	**0.79**	0.75	0.73	0.71	0.69	0.74	**0.78**	0.67	**0.74**	**0.74**	**0.73**	**0.74**	**0.74**
Synapse1.1	BASE	**0.69**	**0.68**	**0.61**	**0.60**	0.63	0.67	0.65	0.63	0.68	0.56	**0.60**	0.61	0.53	**0.63**	**0.65**	**0.63**	**0.63**
	GGA	0.60	0.61	0.53	0.58	0.58	**0.72**	0.66	0.66	**0.68**	**0.58**	0.54	**0.66**	0.47	0.62	0.59	0.61	0.60
	SGA	0.67	0.60	0.59	0.53	0.63	0.71	0.66	**0.66**	0.65	0.53	0.59	0.64	0.49	0.60	0.61	0.61	0.61
	CFS	0.66	0.63	0.59	0.56	**0.67**	0.65	**0.67**	0.62	0.68	0.57	0.59	0.63	**0.55**	0.59	0.62	0.62	0.62
Synapse1.2	BASE	0.68	0.65	0.58	0.71	0.65	0.7	0.71	0.69	0.72	0.72	0.64	0.67	0.63	0.7	0.69	0.68	0.69
	GGA	0.67	0.61	0.59	**0.76**	0.68	**0.71**	0.73	**0.75**	**0.72**	**0.73**	0.62	**0.68**	**0.69**	**0.72**	**0.72**	**0.69**	**0.71**
	SGA	0.57	0.62	**0.63**	0.71	0.71	0.71	**0.74**	0.71	0.7	0.72	0.67	0.65	0.64	0.63	0.68	0.67	0.68
	CFS	**0.70**	**0.66**	0.63	0.69	**0.71**	0.66	0.71	0.68	0.71	0.70	**0.67**	0.66	0.65	0.68	0.70	0.68	0.68
Jedit3.2	BASE	0.64	**0.75**	**0.76**	0.70	0.69	0.71	**0.69**	0.69	0.72	0.70	**0.76**	**0.68**	0.70	0.67	0.66	**0.70**	0.70
	GGA	**0.72**	0.71	0.69	0.72	**0.72**	**0.73**	0.62	0.70	0.72	0.69	0.70	0.68	0.70	0.67	**0.70**	0.70	**0.70**
	SGA	0.64	0.70	0.67	0.71	0.69	0.71	0.69	**0.71**	**0.75**	**0.72**	0.67	0.68	0.65	**0.69**	0.68	0.69	0.69
	CFS	0.63	0.73	0.72	**0.73**	0.69	0.71	0.64	0.68	0.72	0.67	0.71	0.65	**0.71**	0.66	0.67	0.69	0.69

When evolutionary FS were applied on three large datasets, as deduced from Table 9.11, the highest G-Mean value was achieved for at least 11 ML techniques in each dataset. In Ant1.7 and Xalan2.4, only one ML technique was able to score the maximum G-Mean value whereas, in Xalan2.6 there are two such instances. Also, the mean and median G-Mean value of CFS is the least for CFS. Hence, this points to the low predictive capability of CFS in large projects for the G-Mean measure. The overall performance of GGA seems promising followed by GGA and BASE. Xalan2.6 got the maximum G-Mean value of 0.75 for IBk with GGA. Ensemble learners ICO and RT classification with evolutionary FS gave the best G-Mean values for Ant1.7 and Xalan2.4. In mid-sized projects, Xerces1.3, Jedit4.0, and Ivy2.0 gave the highest G-Mean values of 0.72, 0.73, and 0.67 respectively with the IBk technique. Though Xerces1.1 exhibits the maximum value with GGA, it has the best predictions with seven ML techniques in the BASE case suggesting the non-effectiveness of FS techniques in this dataset. Jedit4.1 was classified the best by LR having a G-Mean value of 0.72 with filter method CFS, but again, the highest mean and median values are illustrated by the BASE. Table 9.12 shows comparatively better values of evolutionary FS for Jedit4.0 and Ivy2.0. CFS based models illustrate somewhat better G-Mean values than GGA based models for mid-sized projects.

For small projects, G-Mean performed well as a performance indicator. The G-Mean range for small projects is from 0.47 to 0.84 whereas for large projects range is from 0.13 to 0.75. Mid-sized projects achieved 0.73 as the maximum G-Mean value. Log4j1.0, Log4j1.1, and Synapse1.2 exemplified the highest G-Mean values for individual ML techniques for the evolutionary FS case. Synapse1.1 and Jedit3.2 have greater G-Mean values for BASE than other cases indicating less improvement in FS based models. Median G-Mean values of models are represented in Figure 9.4 for each dataset.

CFS has the highest median value for Xerces1.3, Jedit4.0, and Log4j1.1. The highest median value of G-Mean (0.74) is achieved by Log4j1.1 with CFS. GGA looks like a more capable FS technique as it produces the highest G-Mean median values for all large

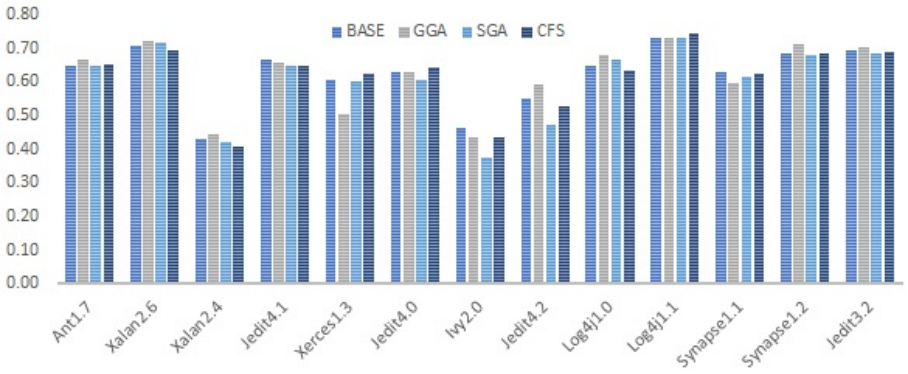

Figure 9.4: Median G-Mean values of the BASE, GGA, SGA, and CFS for 13 datasets

projects, one mid-sized project (Jedit4.2), and three small projects (Log4j1.0, Synapse1.2, and Jedit3.2). Jedit4.1, Ivy2.0, and Synapse1.1 did not show any improvement in median values of G-Mean on performing FS. Therefore, except for large projects, it is hard to conclude for one case that outperformed. In large datasets, GGA has a positive impact on the performance of models assessed using G-Mean.

Sensitivity Analysis:

Tables 9.14, 9.15 and 9.16 tells the Balance values attained by large, mid-sized, and small projects when models are developed for all ML techniques using original features and a reduced feature set with GGA, SGA, and CFS.

Table 9.14: Sensitivity Values for Large Projects

		NB	LR	SL	LB	MLP	IBk	KStar	ABM1	Bag	ICO	LMT	RT	RSS	PART	J48	Mean	Median
Ant1.7	BASE	0.81	0.82	0.80	0.82	0.77	0.78	**0.80**	0.82	0.83	0.83	**0.83**	0.79	0.79	0.82	0.77	0.81	0.81
	GGA	**0.83**	**0.83**	**0.82**	**0.83**	**0.80**	0.79	**0.80**	**0.83**	**0.84**	**0.84**	**0.83**	0.82	0.81	**0.83**	0.76	**0.82**	**0.83**
	SSGA	0.82	**0.83**	**0.82**	**0.83**	**0.80**	**0.80**	**0.80**	0.82	**0.84**	**0.84**	0.82	0.82	**0.84**	**0.83**	**0.79**	**0.82**	0.82
	CFS	0.82	0.82	**0.82**	**0.83**	0.78	0.78	**0.80**	**0.83**	0.83	0.83	0.82	**0.83**	0.83	**0.83**	0.77	0.81	0.82
Xalan2.6	BASE	**0.72**	**0.74**	0.74	**0.74**	0.73	0.7	**0.75**	**0.76**	0.72	0.73	**0.76**	**0.73**	**0.74**	**0.76**	0.69	0.73	**0.74**
	GGA	0.71	0.72	0.75	0.72	**0.76**	**0.74**	0.74	**0.76**	**0.74**	**0.75**	0.75	**0.73**	0.72	0.73	**0.73**	**0.74**	**0.74**
	SSGA	0.71	0.73	**0.76**	0.72	0.75	0.73	0.73	**0.76**	0.71	0.72	**0.76**	**0.73**	**0.74**	0.74	0.71	0.73	0.73
	CFS	0.71	**0.74**	0.73	0.73	0.7	0.71	0.70	0.71	0.71	0.71	0.72	0.72	0.71	0.73	0.68	0.72	0.71
Xalan2.4	BASE	0.82	0.85	0.82	0.85	0.81	0.82	**0.83**	0.83	0.83	0.84	**0.85**	0.83	0.82	0.85	0.79	0.83	0.83
	GGA	0.82	0.85	**0.85**	0.85	**0.83**	**0.83**	0.82	**0.86**	**0.86**	**0.85**	**0.85**	**0.84**	0.84	0.85	0.80	**0.84**	**0.85**
	SSGA	0.82	**0.86**	0.84	**0.86**	0.82	**0.83**	0.82	0.84	0.84	0.84	0.84	0.83	**0.85**	**0.86**	**0.81**	**0.84**	0.84
	CFS	**0.83**	0.85	0.84	0.84	0.81	0.82	0.81	0.85	0.84	0.84	**0.85**	**0.84**	0.84	0.84	0.78	0.83	0.84

Table 9.15: Sensitivity Values for Mid-sized Projects

		NB	LR	SL	LB	MLP	IBk	KStar	ABM1	Bag	ICO	LMT	RT	RSS	PART	J48	Mean	Median
Jedit4.1	BASE	0.80	0.82	0.77	**0.84**	0.77	0.79	**0.80**	**0.81**	**0.80**	**0.79**	0.80	0.78	0.75	**0.84**	0.73	0.79	**0.80**
	GGA	0.80	0.82	0.81	0.79	**0.78**	**0.80**	0.78	0.78	0.79	**0.79**	**0.82**	0.78	0.75	0.80	0.72	0.79	0.79
	SSGA	0.77	0.82	0.80	0.83	**0.78**	0.79	0.76	0.79	**0.80**	0.77	0.80	0.76	0.79	0.81	**0.75**	0.79	0.79
	CFS	**0.81**	**0.83**	**0.82**	0.82	0.76	0.79	0.76	0.79	**0.80**	**0.79**	0.81	**0.81**	**0.80**	0.82	0.74	**0.80**	**0.80**
Xerces1.3	BASE	0.83	0.86	0.85	**0.87**	0.85	0.86	**0.86**	0.85	0.88	0.87	**0.88**	0.85	**0.87**	0.87	0.83	0.86	0.86
	GGA	0.82	0.85	0.86	0.84	**0.89**	0.87	0.85	0.87	0.86	0.86	0.86	0.86	0.85	0.86	0.85	0.86	0.86
	SSGA	0.83	**0.87**	**0.88**	**0.87**	0.88	0.87	0.85	**0.88**	**0.89**	**0.89**	0.87	**0.87**	**0.87**	**0.88**	**0.86**	**0.87**	**0.87**
	CFS	**0.87**	0.85	0.86	0.84	0.84	**0.88**	0.85	**0.88**	0.88	0.88	0.87	**0.87**	**0.87**	0.86	0.84	0.86	**0.87**
Jedit4.0	BASE	0.77	0.80	0.79	0.79	0.79	0.76	0.79	0.82	0.81	0.81	0.80	**0.79**	0.77	0.79	**0.75**	0.79	0.79
	GGA	0.77	0.80	0.82	0.80	0.80	0.77	**0.81**	**0.83**	**0.83**	0.81	0.79	0.76	0.78	0.80	**0.75**	0.79	0.80
	SSGA	0.76	0.78	**0.83**	0.81	**0.82**	**0.79**	0.78	0.82	0.77	0.79	0.80	0.76	0.77	0.78	**0.75**	0.79	0.78
	CFS	**0.78**	**0.82**	0.82	**0.82**	0.77	0.74	0.80	0.81	0.82	**0.82**	**0.81**	0.78	**0.79**	**0.82**	0.72	**0.80**	**0.81**
Ivy2.0	BASE	0.84	0.88	0.86	0.89	0.86	0.85	0.87	0.86	0.86	0.87	**0.89**	0.87	0.89	0.89	0.83	0.87	0.87
	GGA	**0.86**	0.88	0.86	**0.90**	**0.90**	**0.89**	**0.88**	**0.88**	0.86	**0.88**	**0.89**	**0.89**	**0.90**	**0.90**	0.84	**0.88**	**0.88**
	SSGA	**0.86**	0.88	0.86	0.88	0.85	0.86	0.86	**0.88**	**0.89**	**0.88**	0.88	0.88	0.87	0.88	0.84	0.87	**0.88**
	CFS	**0.86**	**0.89**	**0.90**	0.89	0.84	0.87	0.87	0.87	0.86	0.87	**0.89**	**0.89**	0.89	0.89	**0.86**	**0.88**	0.87
Jedit4.2	BASE	0.86	0.88	0.86	**0.89**	0.85	0.86	0.87	0.89	0.87	0.88	0.87	0.87	0.87	**0.89**	0.82	0.87	0.87
	GGA	0.85	**0.89**	0.89	**0.89**	**0.88**	**0.87**	**0.88**	**0.90**	**0.90**	**0.89**	**0.88**	**0.89**	**0.89**	0.88	**0.85**	**0.88**	**0.89**
	SSGA	**0.88**	0.88	0.89	0.88	0.83	0.86	0.86	0.89	0.87	0.87	0.86	0.88	0.87	0.88	0.83	0.87	0.87
	CFS	0.86	0.87	**0.90**	0.88	0.84	0.86	0.87	0.88	0.87	0.87	**0.88**	0.87	0.86	0.88	0.84	0.87	0.87

Table 9.16: Sensitivity Values for Small Projects

		NB	LR	SL	LB	MLP	IBk	KStar	ABM1	Bag	ICO	LMT	RT	RSS	PART	J48	Mean	Median
Log4j1.0	BASE	0.84	0.79	0.74	0.83	0.76	0.75	**0.79**	0.80	0.76	0.76	0.81	0.76	0.78	0.83	0.70	0.78	0.78
	GGA	0.83	**0.82**	0.77	**0.84**	**0.81**	**0.79**	0.76	0.81	**0.83**	0.77	0.82	0.76	0.77	**0.84**	**0.79**	0.80	0.81
	SSGA	0.82	**0.82**	**0.83**	**0.84**	0.79	0.76	0.75	0.80	0.82	**0.84**	**0.83**	0.79	**0.82**	**0.84**	0.76	**0.81**	**0.82**
	CFS	**0.85**	**0.82**	0.79	0.83	0.76	0.75	0.76	**0.82**	0.79	0.79	0.81	**0.80**	0.80	0.83	0.72	0.79	0.80
Log4j1.1	BASE	0.83	0.76	0.72	0.83	0.76	0.75	0.74	0.78	**0.83**	0.78	0.82	0.77	0.78	0.83	0.73	0.78	0.78
	GGA	0.83	**0.83**	0.84	0.80	**0.84**	0.76	**0.75**	**0.80**	0.78	0.78	0.78	0.77	0.76	0.80	0.75	0.79	0.78
	SSGA	0.82	0.80	0.82	0.83	0.77	0.78	0.73	**0.80**	0.75	0.78	0.79	**0.81**	0.78	0.83	**0.76**	0.79	0.79
	CFS	**0.84**	**0.83**	**0.86**	**0.84**	0.76	**0.81**	0.74	0.76	0.80	**0.80**	**0.84**	**0.81**	**0.80**	**0.84**	0.72	**0.80**	**0.81**
Synapse1.1	BASE	0.75	**0.80**	0.76	0.79	0.75	0.75	0.75	**0.81**	0.75	**0.77**	0.77	0.74	**0.76**	0.78	0.72	0.76	0.76
	GGA	**0.78**	0.79	0.79	0.78	**0.80**	**0.80**	**0.77**	0.79	**0.76**	0.75	0.76	**0.76**	**0.76**	0.77	**0.75**	**0.77**	**0.77**
	SSGA	0.77	0.79	0.78	**0.80**	0.78	0.77	0.75	0.78	**0.76**	0.73	0.76	0.73	0.73	**0.80**	0.73	0.76	**0.77**
	CFS	**0.78**	0.79	**0.81**	**0.80**	0.73	0.77	0.75	0.80	0.75	0.74	**0.78**	0.75	**0.76**	**0.80**	0.74	**0.77**	**0.77**
Synapse1.2	BASE	0.75	0.74	0.70	0.72	0.73	0.75	0.74	0.77	0.77	0.76	0.73	**0.77**	0.75	0.75	0.72	0.74	**0.75**
	GGA	0.76	0.75	**0.75**	0.74	0.75	0.77	**0.79**	**0.79**	**0.78**	**0.80**	**0.75**	0.75	**0.78**	0.73	**0.73**	**0.76**	**0.75**
	SSGA	0.72	0.73	**0.75**	**0.75**	**0.77**	**0.78**	0.75	0.77	0.76	0.76	**0.75**	0.72	0.73	**0.76**	0.70	0.75	**0.75**
	CFS	**0.77**	**0.76**	0.74	0.74	0.72	0.77	0.72	0.78	0.76	0.74	0.73	0.72	0.74	**0.76**	0.71	0.74	0.74
Jedit3.2	BASE	0.75	**0.80**	0.74	**0.81**	0.75	0.74	0.74	0.79	0.76	0.77	0.77	0.75	0.73	**0.81**	0.73	**0.76**	0.75
	GGA	**0.80**	0.79	**0.78**	0.78	**0.78**	0.72	0.74	0.77	0.75	0.78	**0.78**	**0.76**	**0.76**	0.76	**0.74**	**0.76**	**0.77**
	SSGA	0.75	0.78	0.76	0.77	0.75	**0.77**	**0.75**	**0.81**	**0.78**	**0.79**	0.75	0.74	**0.76**	0.77	**0.74**	**0.76**	0.76
	CFS	0.75	0.79	0.76	0.79	0.75	0.71	0.73	0.78	0.73	0.78	**0.78**	0.74	0.74	0.77	0.72	0.75	0.75

Analyzing the performance of large projects in terms of Sensitivity, four statistical classifiers, two instance-based ML techniques, LMT, RT, and PART exhibited no visual impact of feature selection in SDP. Considering the mean and median values from Table 9.14, 9.15, and 9.16 no remarkable difference is observed in the performance of BASE and evolutionary feature selection cases. An increase in GGA values is too small to make any remarkable impact. Compared with CFS, GGA or SGA has performed better for a majority of datasets.

For Ant1.7, almost all ML techniques except RSS have similar performance for the BASE, GGA, SGA, and CFS. The mean value is either 0.81 or 0.82 for all four cases. For Xalan2.6, models based on statistical ML techniques have similar performance for the four cases. The Sensitivity observed in the MLP model is 0.76 as compared to 0.70 with CFS. Instance-based classifiers also exhibited better performance in terms of Sensitivity when features were selected using GGA. Models based on ABM1, LMT, RT, and RSS showed similar performance with evolutionary feature selection and BASE whereas, for the same ML techniques, the performance of models built on CFS-selected features decreased in terms of Sensitivity. For ICO and Bag, GGA depicted a maximum Sensitivity value of 0.74 and 0.75 respectively. Xalan2.4 depicted a maximum Sensitivity value of 0.86 in GGA case with ensembles (ABM1 and Bag) and SGA case with LR and LB. But as the Sensitivity value with BASE case, when no feature is ignored, is 0.85 with both LR and LB, therefore we did not consider feature selection as the better option.

Similar observations can be deduced by the investigation of model performances of mid-sized and small projects.

The median Sensitivity values of the developed defect prediction models on the investigated datasets in the chapter for the BASE, GGA, SGA, and CFS scenario are presented in Figure 9.5.

In Figure 9.5, on x-axis we have datasets and y-axis represents Sensitivity value starting at 0.65. The minimum median value for Sensitivity is observed in Xalan2.6 (0.71) when CFS was used to select relevant features. Except for Jedit4.1, Jedit4.0, Synapse1.1, and

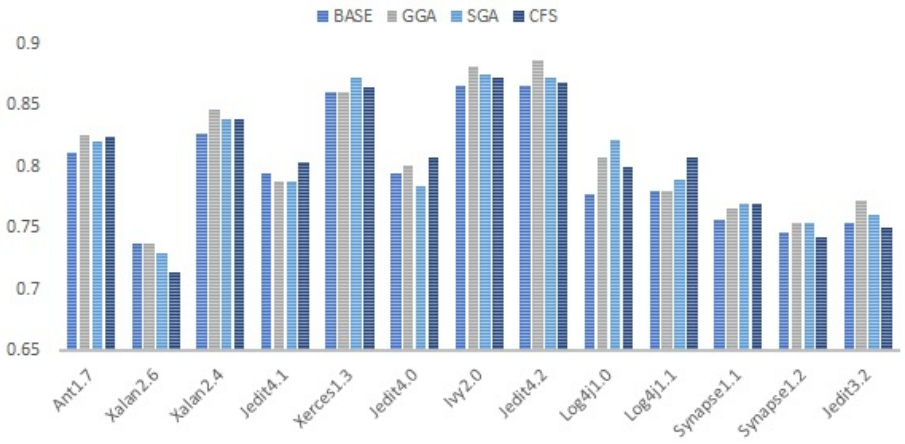

Figure 9.5: Median Sensitivity values of the BASE, GGA, SGA, and CFS for 13 datasets

Log4j1.1, evolutionary feature selection has resulted in better SDP models than CFS.

9.4.3 RQ3: Which FS technique can be categorized as the best amongst all for SDP?

Models developed for SDP are judged against the Friedman test and Wilcoxon signed-rank test statistically to get the response to RQ3. We opted for stable and robust performance measures for answering this RQ as they are more reliable than Sensitivity.

Tables 9.17, 9.18 and 9.19 presents the Friedman rankings of FS cases for all the datasets (Overall), size-wise categories of projects, and individual projects. AUC_O, Bal_O, and GM_O represents the ROC-AUC, Balance and G-Mean performance measure for all the projects. AUC_L, Bal_L, and GM_L represents the ROC-AUC, Balance and G-Mean performance measure for large projects. AUC_M, Bal_M, and GM_M represents the ROC-AUC, Balance and G-Mean performance measure for mid-sized projects. AUC_S, Bal_S, and GM_S represents the ROC-AUC, Balance and G-Mean performance measure for small projects. Mean ranks of GGA, SGA, CFS, and BASE are written in brackets. P-values and Kendall C are also recorded in tables for their easy interpretations. GGA is boldfaced

in Tables 9.17, 9.18 and 9.19. SGA is boldfaced and italicized only in the cases where SGA performed better than GGA. The null hypothesis for the Friedman test states that the performances of ML models in which FS techniques are involved are not statistically different from the performances of ML models that use all features to perform SDP in terms of ROC-AUC, Balance, and G-Mean.

Table 9.17: Friedman Rankings of Feature Selection Techniques with respect to ROC-AUC

ROC-AUC		Rank 1	Rank 2	Rank 3	Rank 4	p-value	Kendall C
AUC_O		**GGA(2.84)**	CFS (2.5)	SGA(2.37)	BASE(2.27)	**0.000**	0.038
AUC_L		**GGA(3.12)**	SGA (2.53)	CFS (2.17)	BASE(2.16)	**0.001**	0.122
AUC_M		CFS (2.74)	**GGA (2.66)**	BASE(2.36)	SGA(2.22)	**0.044**	0.036
AUC_S		**GGA(2.86)**	CFS (2.46)	SGA(2.42)	BASE(2.25)	**0.027**	0.041
Large Projects	Ant1.7	**GGA (3.26)**	SGA (2.96)	CFS (2.16)	BASE (1.60)	**0.001**	0.352
	Xalan2.6	BASE (3.03)	**GGA (2.70)**	SGA (2.20)	CFS (2.06)	0.141	0.141
	Xalan2.4	**GGA (3.40)**	SGA (2.43)	CFS (2.30)	BASE (1.86)	**0.010**	0.255
Mid-sized Projects	Jedit4.1	BASE (2.9)	**GGA (2.86)**	CFS (2.7)	SGA (1.53)	**0.009**	0.255
	Xerces1.3	BASE (3.06)	CFS (2.66)	**SGA (2.33)**	**GGA (1.93)**	**0.099**	0.14
	Jedit4.0	GGA (3.13)	CFS (2.66)	BASE (2.26)	SGA (1.93)	0.062	0.163
	Ivy2.0	**SGA (3.13)**	CFS (2.86)	**GGA (2.4)**	BASE (1.6)	**0.007**	0.271
	Jedit4.2	**GGA (2.96)**	CFS (2.83)	SGA (2.2)	BASE (2)	0.107	0.136
Small Projects	Log4j1.0	SGA (3.46)	**GGA (2.83)**	CFS (2.06)	BASE (1.63)	**0.001**	0.225
	Log4j1.1	**GGA (3.26)**	SGA (2.40)	CFS (2.4)	BASE (1.93)	**0.039**	0.186
	Synapse1.1	CFS (3.06)	**SGA (2.66)**	**GGA (2.13)**	BASE (2.13)	**0.001**	0.21

ROC-AUC		Rank 1	Rank 2	Rank 3	Rank 4	p-value	Kendall C
	Synapse1.2	**GGA (3.13)**	BASE (2.73)	CFS (2.43)	SGA (1.7)	**0.019**	0.221
	Jedit3.2	**GGA (2.96)**	BASE (2.83)	CFS (2.33)	SGA (1.86)	**0.076**	0.153

Table 9.18: Friedman Rankings of Feature Selection Techniques with respect to Balance

Balance		Rank 1	Rank 2	Rank 3	Rank 4	p-value	Kendall C
Bal_O		**GGA (2.68)**	BASE (2.53)	SGA (2.38)	CFS (2.38)	0.051	0.013
Bal_L		**GGA (3.30)**	BASE (2.58)	SGA (2.05)	CFS (2.05)	**0.000**	0.152
Bal_M		**SGA (2.56)**	CFS (2.56)	BASE (2.52)	GGA (2.34)	0.644	0.007
Bal_S		**GGA (2.64)**	BASE (2.52)	SGA (2.41)	CFS (2.41)	0.596	0.008
Large Projects	Ant1.7	**GGA (3.40)**	SGA (2.20)	CFS (2.20)	BASE (2.20)	**0.011**	0.245
	Xalan2.6	**GGA (3.00)**	BASE (2.80)	SGA (2.10)	CFS (2.10)	**0.086**	0.147
	Xalan2.4	**GGA (3.50)**	BASE (2.76)	SGA (1.86)	CFS (1.86)	**0.000**	0.429
Mid-sized Projects	Jedit4.1	BASE (2.66)	**SGA (2.6)**	CFS (2.6)	**GGA (2.13)**	0.601	0.041
	Xerces1.3	BASE (3.00)	**SGA (2.83)**	CFS (2.83)	**GGA (1.33)**	**0.000**	0.407
	Jedit4.0	**SGA (2.70)**	CFS (2.70)	GGA (2.33)	BASE (2.26)	0.654	0.036
	Ivy2.0	**GGA (2.66)**	SGA (2.60)	CFS (2.60)	BASE (2.13)	0.582	0.043
	Jedit4.2	**GGA (3.26)**	BASE (2.53)	SGA (2.10)	CFS (2.10)	**0.028**	0.202
Small Projects	Log4j1.0	**GGA (2.80)**	BASE (2.53)	SGA (2.33)	CFS (2.33)	0.682	0.033
	Log4j1.1	**SGA (2.76)**	CFS (2.76)	GGA (2.50)	BASE (1.96)	0.212	0.1
	Synapse1.1	BASE (2.93)	**SGA (2.36)**	CFS (2.36)	**GGA (2.33)**	0.473	0.056

Balance		Rank 1	Rank 2	Rank 3	Rank 4	p-value	Kendall C
	Synapse1.2	**GGA (3.00)**	BASE (2.40)	SGA (2.30)	CFS (2.30)	0.334	0.076
	Jedit3.2	BASE (2.80)	**GGA (2.60)**	SGA (2.30)	CFS (2.30)	0.615	0.04

Table 9.19: Friedman Rankings of Feature Selection Techniques with respect to G-Mean

G-Mean		Rank 1	Rank 2	Rank 3	Rank 4	p-value	Kendall C
GM_O		**GGA (2.68)**	BASE (2.52)	SGA (2.39)	CFS (2.39)	0.053	0.013
GM_L		**GGA (3.33)**	BASE (2.53)	SGA (2.06)	CFS (2.06)	**0.000**	0.24
GM_M		**SGA (2.54)**	CFS (2.54)	BASE (2.52)	**GGA (2.38)**	0.829	0.004
GM_S		**GGA (2.60)**	BASE (2.51)	SGA (2.44)	CFS (2.44)	0.813	0.004
Large Projects	Ant1.7	**GGA (3.40)**	BASE (2.20)	SGA (2.20)	CFS (2.20)	**0.011**	0.245
	Xalan2.6	**GGA (3.20)**	BASE (2.60)	SGA (2.10)	CFS (2.10)	**0.042**	0.182
	Xalan2.4	**GGA (3.40)**	BASE (2.80)	SGA (1.90)	CFS (1.90)	**0.000**	0.542
Mid-sized Projects	Jedit4.1	**SGA (2.60)**	BASE (2.60)	CFS (2.60)	**GGA (2.13)**	0.601	0.041
	Xerces1.3	BASE (3.00)	**SGA (2.83)**	CFS (2.83)	**GGA (1.33)**	**0.000**	0.407
	Jedit4.0	**SGA (2.70)**	CFS (2.70)	GGA (2.33)	BASE (2.26)	0.654	0.036
	Ivy2.0	**GGA (2.86)**	SGA (2.50)	CFS (2.50)	BASE (2.13)	0.442	0.06
	Jedit4.2	**GGA (3.26)**	BASE (2.53)	SGA (2.10)	CFS (2.10)	**0.028**	0.202
Small Projects	Log4j1.0	**GGA (2.80)**	BASE (2.53)	SGA (2.33)	CFS (2.33)	0.682	0.033
	Log4j1.1	**SGA (2.90)**	CFS (2.90)	GGA (2.23)	BASE (1.96)	0.068	0.158
	Synapse1.1	BASE (2.93)	**SGA (2.36)**	CFS (2.36)	GGA (2.33)	0.473	0.056

G-Mean		Rank 1	Rank 2	Rank 3	Rank 4	p-value	Kendall C
	Synapse1.2	**GGA** **(3.13)**	SGA (2.30)	CFS (2.30)	BASE (2.26)	0.148	0.119
	Jedit3.2	BASE (15.00)	**GGA** **(5.35)**	SGA (3.00)	CFS (0.14)	0.148	0.119

ROC-AUC Analysis for Friedman Rankings:

For ROC-AUC, when large and small projects are considered, as evident from Table 9.17, GGA emerged as the best case for model development with the BASE as the lowest ranker but in mid-sized projects, CFS outperformed the other techniques. However, when all projects are considered together, then GGA performance was statistically better than other techniques. FS has improved the defect prediction capability of the models as the BASE case has the lowest Friedman rank. Even when size categories are looked upon, GGA and BASE are statistically better in all three categories. Though CFS results were better than GGA in mid-sized projects, still overall GGA scored the highest rank. The reason behind this is the minor difference between their mean ranks.

For mid-sized projects, ROC-AUC mean ranks for CFS and GGA are 2.74 and 2.66 respectively. P–values are also less than 0.05, thus we refute the null hypothesis and ascertain that results are statistically significant at α = 0.05. Models built using the evolutionary FS technique GGA are statistically proven to be the best predictors. There exists a statistical difference but low Kendall C indicates that although there is a significant difference, it is only a small difference. The range of Kendall C is from 0 to 1.

Now, considering individual datasets, approximately 70% of datasets were classified as the best using evolutionary techniques, either GGA or SGA. Except for Xalan2.6 and Jedit4.2, all the rankings are statistically significant. The significant p-values are less than 0.05 and are in bold typeface. GGA attained the first rank in seven datasets, out of which one dataset illustrates comparable performance amongst all the cases as its p-value is greater than 0.05. GGA also achieved the second rank for three more datasets. SGA was

also able to secure the first position for two datasets. Though in Xalan2.6, the BASE comes first, but as the p-value is 0.141, there is no statistical difference between its performance and the performance of GGA/SGA/CFS. Analysis of Friedman rankings of FS techniques and BASE ensure the positive impact of evolutionary techniques in constructing effectual ML models for defect prediction when ROC–AUC measure is used to assess the models' performances.

Balance and G-Mean Analysis for Friedman Rankings:

The mapping of Table 9.18 and Table 9.17 results in the discovery of a similar pattern in the Friedman rankings of FS techniques and their significance. Rankings ensure that GGA has remarkable performance when all projects are taken together and also for large and small projects. For mid-sized projects, another evolutionary technique SGA is ranked the best. So, overall we can conclude their fitness to be the best if the p-value is less than 0.05. The p-value is remarkably lower than 0.05 for large projects, so statistically; Balance and G-Mean values depicting ML performances are greatest with the application of GGA. There is an increase in predictive capabilities of ML models evaluated with G-Mean and Balance when evolutionary FS techniques are applied for small, mid-sized, large projects, and even for overall projects. But as the p-value is less than 0.05, their performances are not statistically better. All the FS techniques are comparable with the BASE case, except for large projects. Their overall p-value is 0.051 and 0.053 for Balance and G-Mean respectively. Therefore, the results are 94.9% and 94.7% significant for Balance and G-Mean respectively. Based on this, performance variations can be considered approximately valid.

Discussing the individual datasets' performances, defects of all datasets from large projects are best classified using GGA. Performances of models constructed with GGA for large projects are statistically better than the BASE and other FS techniques. Moreover, for mid-sized and small projects, models that incorporated evolutionary FS have performed non significantly better than those BASE and CFS-based models.

Wilcoxon Signed-rank Test Results:

To verify the results acquired by the Friedman test, this study executed post-hoc analysis. Wilcoxon signed-rank test tends to uncover whether a particular FS technique X that is ranked the best in Friedman testing is statistically better than others by performing pairwise comparisons. The null hypothesis is set as that there is no statistical difference between the performance of X and other FS techniques including the BASE case and is tested for 95% significance. Only the performance of X and another FS technique or BASE is compared at a time. The test was evaluated on the results of all the 15 ML techniques. For overall projects, large projects, and small projects, the best prediction models were based on GGA, therefore Wilcoxon signed-rank test is conducted concerning GGA for performance measures- ROC-AUC, Balance, and G-Mean. For mid-sized projects, the Wilcoxon test is conducted for CFS regarding ROC-AUC and SGA on Balance and G-Mean both. Table 9.20 states the Wilcoxon signed-rank results for ROC-AUC, G-Mean, and Balance.

- ROC-AUC Analysis: For mid-sized projects, CFS was ranked no. 1 but there was a minor difference in their mean ranks. So Wilcoxon signed-rank test will help us to determine the effectiveness of resultant models and will assist in establishing whether there is an actual difference between the two techniques or not. According to results shown in Table 9.20, ROC-AUC results are very encouraging as GGA is better than the BASE for overall considered datasets. GGA is significantly better than CFS and SGA for all projects and large projects. For small projects, ROC-AUC values have significantly increased than BASE performance values but have a similar impact as CFS. About mid-sized projects, though CFS performed significantly well when Friedman ranks were calculated but it failed to distinguish amongst CFS, GGA, and BASE when Wilcoxon signed-rank test is performed to confirm its usefulness. Even though ROC-AUC values for mid-sized projects illustrated comparable performances in all four cases, we reject the null hypothesis based on the overall performance of

projects for ROC-AUC and advocate researchers to employ GGA, an evolutionary FS technique, for selecting the important features before building the ML model.

- Balance and G-Mean Analysis: Balance and G-Mean values project the related behavior in all pairs for overall, large and small projects. As can be seen from Table 9.20, p-values are significant only for large projects. In large projects, therefore, GGA has boosted the model performances with a significant increase in Balance and G-Mean values. However, when comprehending the results of mid-sized projects, their behaviors differ and another evolutionary technique SGA gave better results. Considering Balance, when models of mid-sized projects were compared on basis of SGA (first ranker in Friedman test) values with others, SGA appeared to be highly significant than CFS and BASE with a p-value of 0.000 for both cases. Balance and G-Mean performances of models for small projects are comparable with each other and have illustrated a non-significant increase in model prepared using GGA.

Table 9.20: Wilcoxon signed-rank results for projects in terms of ROC-AUC, Balance, and G-Mean

Case	GGA_BASE	GGA_SGA	GGA_CFS
AUC_O	**0.000**	**0.001**	**0.024**
AUC_L	**0.000**	**0.037**	**0.001**
AUC_S	**0.000**	0.058	0.068
Bal_O	0.822	**0.003**	0.170
Bal_L	**0.008**	**0.007**	**0.000**
Bal_S	0.85	0.255	0.457
GM_O	0.361	0.160	0.160
GM_L	**0.001**	**0.000**	**0.000**
GM_S	0.172	0.056	0.056
-	CFS - BASE	CFS - GGA	CFS - SGA
AUC_M	0.545	0.562	**0.045**
-	SGA-BASE	SGA-GGA	SGA-CFS
Bal_M	**0.000**	0.066	**0.000**
GM_M	0.325	0.172	0.172

9.4.4 RQ4: Which ML technique performs the best with the best FS technique for the classification of software defects?

This study evaluates SDP models developed using 15 ML techniques of five different categories. It is important to investigate the predictive capabilities of various ML techniques to ascertain the model efficacy. Tables 9.5, 9.6, and 9.7 states the ROC-AUC values for small, mid-sized, and large project. Balance values obtained for all ML techniques to BASE, GGA, SGA, and CFS are depicted in Tables 9.8, 9.9, and 9.10 . Tables 9.11, 9.12, and 9.13 portrays the G-Mean values for all ML techniques. These tables were discussed in detail in subsection 9.4.2 considering the feature selection techniques. Results were analyzed based on four scenarios: BASE, GGA, SGA, and CFS. The cumulative analysis was performed for size-based datasets. As ROC-AUC proved to be a more promising measure when compared to FS techniques, so to find the best performer in ML techniques, we will restrict our analysis to ROC-AUC only. Balance and G-Mean values for ML techniques can be easily evaluated by given tabular statistics in Tables 9.8 - 9.13.

ROC-AUC Analysis:

The maximum ROC-AUC value is gained by small projects in all four cases. In the BASE, the maximum ROC-AUC value of 0.85 is attained by LR for the Jedit3.2 project. , On comparing GGA performance also, LR gave the best ROC-AUC value of 0.872 for Log4j1.1. In SGA, Synapse1.2 has the maximum ROC-AUC value of 0.848 when classified with LB. Naïve Bayes shows its capability to predict defects in Log4j1.1 with an ROC-AUC value of 0.86. When all the datasets are considered cumulatively, we use median values to assess the machine learner's performance. The median value is almost the same in all the four cases of the BASE, GGA, SGA, and CFS and is approximately 0.80. In the BASE, the median ROC-AUC value is the best for LMT (0.808). RSS, MLP, and Logistic illustrate the maximum median values for GGA, SGA, and CFS respectively.

This section compares the performance of different ML techniques that are used for

developing SDP models amongst themselves with aid of statistical techniques. We exploit only ROC-AUC to find the Friedman rankings of 15 ML techniques as it is a threshold independent stable performance measure used in this study. The comparative performance was evaluated dataset-wise. Furthermore, the Wilcoxon signed–rank test was performed with α = 0.05 to statistically evaluate the compared results. Friedman rankings with the p-value for ML techniques and their mean ranks (in brackets) are tabulated in Table 9.21. Ranks are calculated for the BASE, GGA, SGA, and CFS to explore which techniques perform better with the FS technique selected or when no FS is done, Base.

Table 9.21: Friedman Rankings for ML Techniques with p-values in terms of ROC-AUC

	BASE	GGA	SGA	CFS	Overall	Large	Mid-sized	Small
Rank 1	RSS (12.38)	LR (11.00)	KStar (12.65)	LR (12.15)	RSS (11.18)	RSS (12.29)	RSS (11.55)	SL (10.47)
Rank 2	SL (11.57)	RSS (10.80)	LB (10.88)	RSS (11.80)	LB (10.51)	Bag (12.00)	LB (10.82)	KStar (10.22)
Rank 3	LMT (11.34)	LB (10.34)	ICO (9.92)	NB (10.92)	LR (10.26)	LB (10.83)	LR (10.45)	RSS (10.15)
Rank 4	Bag (11.15)	SL (9.96)	RSS (9.73)	LMT (10.88)	Bag (10.23)	LR (10.29)	LMT (10.4)	LR (10.07)
Rank 5	LB (10.5)	Bag (9.84)	Bag (9.34)	SL (10.84)	LMT (10.06)	LMT (9.62)	NB (9.87)	Bag (10.05)
Rank 6	LR (10.42)	KStar (9.69)	ABM1 (9.23)	Bag (10.57)	SL (9.80)	SL (9.41)	SL (9.37)	LB (10.02)
Rank 7	ABM1 (9.69)	ICO (9.07)	LMT (8.96)	LB (10.34)	KStar (9.45)	ICO (9.20)	Bag (9.35)	LMT (10.00)
Rank 8	NB (9.19)	LMT (9.07)	MLP (8.26)	MLP (9.15)	NB (9.01)	ABM1 (8.83)	KStar (9.25)	NB (9.02)
Rank 9	ICO (8.00)	NB (8.76)	LR (7.5)	ICO (8.00)	ICO (8.75)	KStar (8.50)	ICO (9.10)	ABM1 (8.47)
Rank 10	KStar (8.00)	MLP (8.69)	NB (7.19)	KStar (7.46)	ABM1 (8.36)	MLP (7.83)	MLP (9.05)	ICO (8.12)
Rank 11	MLP (6.46)	ABM1 (8.30)	IBk (7.07)	ABM1 (6.23)	MLP (8.14)	NB (7.58)	ABM1 (7.97)	MLP (7.42)

	BASE	GGA	SGA	CFS	Overall	Large	Mid-sized	Small
Rank 12	PART (4.46)	IBk (5.65)	SL (6.84)	PART (4.76)	PART (4.73)	PART (5.66)	IBk (4.05)	IBk (5.92)
Rank 13	IBk (2.69)	PART (4.76)	PART (4.92)	J48 (2.80)	IBk (4.50)	J48 (3.58)	PART (4.02)	PART (4.87)
Rank 14	J48 (2.42)	J48 (2.23)	RT (3.76)	IBk (2.61)	J48 (2.78)	IBk (2.91)	J48 (2.57)	RT (2.62)
Rank 15	RT (1.69)	RT (1.76)	J48 (3.69)	RT (1.42)	RT (2.16)	RT (1.41)	RT (2.15)	J48 (2.52)
p-value	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**

LR is statistically found better ML technique to construct models when GGA or CFS is to be used by researchers. Results are true with a confidence level of 95% as for every column in Table 9.21, the p-value is 0.000. Focusing on the size of projects, RSS is considered the first ranker ML for the large, mid-sized, and overall situation. SL scored the highest rank for small-sized projects. However, the worst ranks are achieved by RT, J48, IBk, and PART. Now, as Friedman test results are found significant, we need to perform the Wilcoxon signed-rank test to do a pairwise comparison. The null hypothesis is set as that there is no difference in RSS and other ML techniques for overall, large, and mid-sized projects. For small projects, the null hypothesis states that there is no difference between the performance of SL and other ML techniques.

Wilcoxon signed-rank test results are exemplified in Table 9.22 with p-values and significant pairs are boldfaced. The null hypothesis is rejected for the pairs that are in bold as their p-value is less than 0.05.

Table 9.22: Wilcoxon signed-rank results for RSS (Overall, Large, Mid-sized) and for SL (Small projects)

RSS with	NB	LR	SL	LB	MLP	IBk	KStar	ABM1	Bag	ICO	LMT	RT
Overall	**0.003**	0.467	0.27	**0.004**	**0.000**	**0.000**	**0.001**	**0.000**	0.316	**0.000**	0.15	**0.000**
Large	**0.002**	0.109	**0.050**	**0.028**	**0.002**	**0.002**	**0.002**	**0.003**	0.724	**0.002**	**0.050**	**0.002**

RSS with	NB	LR	SL	LB	MLP	IBk	KStar	ABM1	Bag	ICO	LMT	RT
Mid-sized	**0.004**	0.271	0.076	**0.035**	**0.015**	**0.000**	**0.042**	**0.001**	0.086	**0.001**	0.073	**0.000**
SL with	NB	LR	LB	MLP	IBk	KStar	ABM1	Bag	ICO	LMT	RT	RSS
Small	**0.038**	0.247	0.094	**0.005**	**0.000**	0.108	**0.025**	0.351	**0.004**	0.213	**0.000**	0.145

When all the projects are considered together, then RSS is statistically better than eight ML techniques. For large and midsized projects, RSS is statistically better than ten and eight ML techniques respectively. When SL is checked for small projects, SL is observed to have comparable performance with four more ML techniques. RSS technique generates random subtrees with a different subset of features and projects the subset that gives the best ROC-AUC value as an optimal solution. This is one of the possible reasons why RSS outperformed the other ML techniques. LR and Bag are as good as RSS for all scenarios. RSS and Bag are ensemble techniques whereas LR is a statistical technique.

For small datasets, SL, LR, LB, KStar, Bag, LMT, and RSS are the best performers. Out of these, the first three are statistical ML techniques, KStar is the nearest neighbor based algorithm, and the remaining three are ensemble-based learners.

9.5 Discussion

To the best of our knowledge, no one has explored GGA and SGA as the candidate for feature selection for the software engineering datasets in the literature to date. Therefore, in this chapter GA based FS techniques are used to assess their impact on defect prediction. The most acceptable CFS is included in this study to provide a fair comparison. The objective of the study was to ascertain the impact of evolutionary FS techniques on the predictive capability of ML techniques. Datasets were categorized as small, mid-sized, and large datasets based on the number of classes. We provide an in-depth analysis of results

emphasizing the usage of GGA for FS to remove unnecessary features. It is empirically validated that GGA performs better than CFS and does improve the performance by eliminating unnecessary and uncorrelated features from the original datasets.

A large number of ML techniques are evaluated and results based on ROC-AUC, Balance, and G-Mean indicate that ensemble techniques-RSS, Bag, and statistical technique-LR statistically outperformed other ML techniques in all the size categories of datasets. Results validate the superiority of GGA over CFS when classifying large projects in terms of ROC-AUC, Balance, and G-Mean. Furthermore, the overall performance of GGA also emerged as statistically better than CFS with ROC-AUC assessment. This study encourages researchers and software practitioners to explore evolutionary FS for SDP and provides them an insight to select the ML technique wisely. Despite the comparable performance of G-Mean and Balance, we advocate the use of ROC-AUC for performance assessment as it is versatile and threshold independent. It gave better prediction results for software defects than G-Mean and Balance.

Besides, this study provides developers and software practitioners the guideline for selecting the appropriate FS technique for their application. They can decide the FS technique based on the importance of IQAs in their software. Preferred ML techniques for effective SDP are RSS, NB, LB, MLP, IBk, ICO, and RT. For small projects, SL also outperformed LR, LB, KStar, Bag, LMT, and RT.

Future direction includes replication of the study with inter-project or cross-project validation. Resampling methods can be incorporated to further reduce the imbalanced classification problem before performing evolutionary feature selection. Research studies lack benchmarking for evolutionary feature selection. Therefore, more studies are required to be contributed by researchers that include datasets belonging to projects of different programming languages.

Chapter 10

Conclusion

10.1 Summary of the Work

Software Quality Assurance is an important activity that need to be carried out with limited resources. Changes in software are inevitable and they led to increase in the complexity of the software. The modifications give rise to the possibilities of defects which may lead to failure of software systems. Software Defect Prediction aims at identifying the software parts that are vulnerable to be defective. Exposing of such defect-prone areas in early stages of software development tends to save time and resources. Software metrics are exploited in OO software for predictive modelling. The impact of internal quality attributes like cohesion, coupling etc on the defect proneness is established and this assists in unveiling the complex part of software in design phase which may have some defects. Cost involved in rectifying these defects increases exponentially if they remained untreated and gets propagated in coding or implementation phase. Additionally, imbalanced ratio of defective and non-defective classes in software makes it difficult for model to train and learn properly. The primary aim of the work conducted in this thesis is to construct the SDP models which aids in timely delivery of software with reliable quality by efficiently allocating limited

software resources.

A wide array of ML techniques and SBTs have been investigated in this work for effective identification of defective classes. Software data is mostly imbalanced in the real world and the accurate results are desirable for the predictive modelling in software engineering. Imbalanced data, if not treated, can lead to wrong predictions and incorrect model generation and it will adversely affect the company profile. This thesis work analyzes various imbalance learning methods like resampling methods, cost-sensitive learning, and hybrid ensemble methods in this direction. Moreover, predictive modelling with large number of software metrics may hamper the model evaluation process. Therefore, in this work feature selection techniques are used to recognize the most relevant and important features that are not redundant and are highly correlated with defect. CFS as well as evolutionary feature selection techniques are assessed for constructing the SDP models. Important features are identified and impact of OO metrics on defect prediction is established. This enlightens the researchers and developers to concentrate on those features while designing the software. The work is manifested as organized empirical evaluations with easy interpretations and take-aways for researchers and software practitioners.

Software defect prediction is one of the dominant research field. Number of systematic literature reviews exist in literature that covers the classification techniques involved and various issues in SDP. Catal [95] found only two studies related to imbalanced data problem in the time period of 1990-2009. Since imbalanced data problem has recently gained the attention of researchers, this area requires benchmark studies and the guidelines. With no supporting survey in literature pertaining to the imbalanced data problem, we performed a literature review in which we systematically reviewed 48 primary studies in the period from January 2000 to September 2020 that address the imbalanced classification issue. The selected studies are limited to binary defect classification only. The data is extracted from these studies and analyzed with respect to various RQs framed to cater to the need for the systematic review. These studies were analyzed to answer various RQs with respect to the

type of variables used for developing the models, categories of various data analysis techniques used for developing models, datasets used, the predictive performance of various techniques (most popular category of data analysis techniques found), validation methods used for validation of the models, the statistical tests used for results verification and the threats encountered and addressed model development.

We next presented the research methodology followed in the consequent chapters in the thesis. We summarized the research questions addressed in this work along with the brief description of datasets.Descriptive statistics are included for all the datasets. A description of the various variables involved in the studies along with a specification of all the classification techniques used in the thesis is presented. Data need to be preprocessed for better model development. Details of feature selection techniques that are used in this work are included. Imbalanced nature of datasets may lead to wrong predictions, therefore, number of imbalance learning methods are involved in the formulation of this work. We provided details of all these methods with their parameter settings. Models are build using ten-fold cross-validation and the results are evaluated based on stable performance measures: ROC-AUC, BAlance and G-Mean. We provided the description of the validation methods and performance measures that are used to assess the efficacy of the models. Conduction of statistical tests is an important part of any research validation and we also employed non-parametric tests in each chapter to validate the results statistically. Details of these statistical tests are also included for the understanding.

While developing effective ML models we first established the relationship of OO metrics and dependent variable of the study. Coupling and cohesion metrics were found to be the highly related to the dependent variable 'Defect'. LCOM, Ca and RFC metrics were evaluated as the most significant predictors of defect. We built ML models for imbalanced data with 15 ML techniques. Results were improved with engagement of resampling methods. Oversampling methods as well as undersampling methods were explored and results were analyzed using ROC-AUC, G-Mean, Balance, and Sensitivity. The results evalu-

ated using these stable measures indicate the effectiveness of resampling methods to deal with imbalanced classification problem. Experimental results confirmed that oversampling methods performed better than undersampling methods for detecting defects. ROS method exhibited the best prediction capability and the results were further statistically validated. From the varied set of ML techniques, ensemble methods- ABM1, Bag, RT and RSS outperforms the other ML techniques. Their performances were comparable with the nearest neighbor classifiers- KStar and IBk.

As the results of models developed using ML techniques and ensemble methods were found suitable for imbalanced classification, we explored more of ensemble methods based on AdaBoost and Bagging with 15 datasets. Boosting-based and bagging-based ensemble methods were used to develop SDP models for imbalanced data and the results advocate the effectiveness of hybrid ensemble methods. Bagging-based hybrid ensembles performed better than boosting-based hybrid ensembles. Amongst bagging-based ensembles, Underbagging2 performed best with median G-Mean of 0.72, median Balance of 71.15, and median ROC-AUC value of 0.72. Underbagging secured maximum median Sensitivity value of 0.72. UBAG2, UBAG, SMTBAG, and MSBAG showcased better defect prediction capability than all others and have comparable performances with each other. In the category of Boosting based ensembles, RUSBoost outperformed with median Sensitivity value of 61.76, median G-Mean of 0.70, median Balance of 68.83, and median ROC-AUC value of 0.70.

Imbalanced classification is effectively handled by the resampling methods in previous studies. We further tried to solve the imbalanced classification issue at algorithm level by addressing cost-sensitivity. All ML techniques assigns equal costs to False Negatives and False Positives. Performance of ML techniques is enhanced in terms of robust performance measures by penalizing the misclassification cost of defective classes. Five meta cost learners- MC, 10, MC15, MC20, MC25, and MC30 were employed for tackling imbalanced data. 56.48% of cost-sensitive models have a Balance value greater than 0.65

as compared to 1.85% of cost-insensitive models. 60.19% of cost-sensitive models have a Balance value greater than 0.65 as compared to 32.41% of cost-insensitive models. 52.78% of cost-sensitive models have a Balance value greater than 65 as compared to 19.44% of cost-insensitive models. 51.85% of cost-sensitive models have a Balance value greater than 0.75 as compared to 43.52% of cost-insensitive models. The performance of cost-insensitive models and cost-sensitive models was compared with help of Wilcoxon signed-rank test and results show significant difference in terms of Sensitivity, G-Mean, Balance and ROC-AUC. Friedman test was incorporated to find the best ML performer with cost-sensitive learning. Bag, an ensemble method, again emerged as the best ML technique for defect prediction in imbalanced data.

Through systematic literature review we found very few defect prediction studies that used search-based techniques. We explored sixteen SBTs to find their competency to generate effective defect prediction models. BIOHEL achieved the mean G-Mean value of 0.63, mean Balance value of 61.83, and mean ROC-AUC value of 0.63. The performance of SBTs are comparable with ML techniques. Friedman test and Wilcoxon signed-rank test were applied to find out the effective subset of SBTs that can be used for predictive modelling in SDP. UCS, BIOHEL, CHC, GA_ADI, GA_INT, MPLCS, PBIL, and SGA emerged as better SBTs statistically and these techniques projected comparable performance in terms of G-Mean, Balance, and ROC-AUC. GA variants performed statistically better than PSO variants. Next, imbalanced classification can be addressed using this identified subset of SBTs.

As performance of SBTs was found comparable with ML techniques, we provided the resampling solution for the imbalanced classification with SBTs. To bridge the research gaps found in literature review, we tried to incorporate resampling methods with SBTs that performed well in previous work and increased the predictive capabilities of SBT models. We performed the empirical study with effective experimental setup that involves multiple runs of SBTs to handle their stochastic behavior, involvement of stable performance

measures, parameter settings of techniques and rigorous statistical tests. The pairwise comparisons of the models developed using the GA_INT technique and the other investigated techniques was evaluated using the Wilcoxon test. It was found that the models developed using the GA_INT technique were statistically better than XCS, MPLCS, and GA_ADI. Performances of UCS, BIOHEL, LDWPSO, and CPSO were found to be comparable with GA_INT for G-Mean, Balance and ROC-AUC. With resampling methods, performance of PSO variants also improved for classifying defects in imbalanced data. With GA_INT, SMT achieved mean G-Mean value of 0.72, mean Balance value of 71.91, and mean ROC-AUC value of 0.73. When all SBTS are considered, SLSMT method was concluded the best resampling methods based on the Friedman ranks when overall performance is considered. On pair-wize comparison of SLSMT with other resampling methods, we concluded that the performances of SLSMT, ADSYN, RUS, SMT and NCL are statistically comparable.

As SBTs were found to be effective for model development, we also explored SBTs for feature selection. Genetic algorithm variants constructed reliable SDP models, hence we employed variant of this evolutionary technique for feature selection and compared the results with the well established and acceptable CFS technique. In this work, we grouped the datasets into three categories (small, mid-sized, and large) denoting the size based on the number of classes in it. Models were developed using features generated by GGA, SGA, and CFS and their performances are compared for 15 ML techniques using Sensitivity, G-Mean, Balance, and ROC-AUC. Friedman Test followed by Wilcoxon signed-rank test was conducted to rank the models according to their performances and to perform pairwize comparison with the best ranker. GGA-based models performed statistically better than others for large projects in terms of ROC-AUC, Balance, and G-Mean. Considering all the datasets together, GGA-based models were significantly better than others in terms of ROC-AUC. GGA was better than CFS for small projects but not significantly in terms of stable performance measures. For mid-sized projects,CFS performed well in terms of ROC-AUC and SGA performed better in terms of Balance and G-Mean. SGA was

significantly better than CFS with Balance values for mid-sized projects. This empirical investigation ascertains the impact of evolutionary FS on ML-based SDP models. SBTs are comparatively less explored classification techniques and we utilized them for selecting useful features that help in better predictions of software defects.

10.2 Application of the Work

The empirical investigation conducted in the thesis would aid the software practitioners and researchers in the following ways:

- The work provides a well-defined and systematic approach for the development of effective software defect prediction models. Software practitioners and researchers can follow it for further experiments or replication of existing experiments.
- Identification of subset of object-oriented metrics useful for defect prediction will assist design engineers to focus on important aspects of the software. They can design the software better once they understand the intricate relationship between the internal quality attributes and defect proneness.
- Considering that the multiple classification techniques are evaluated in the research work, it provides guidelines to researchers for selecting an appropriate modelling technique for defect prediction task.
- This work corroborates the employment of resampling methods, metacost learners and hybrid ensemble methods for tackling the imbalanced classification problem. The extensive model development and validation in this direction provides the empirical framework to software developers to wisely select the solution they desire for delivering mature and reliable software systems.

- This work enables project managers to strategically plan effective resource allocation for assessment and improvement of existing products and practices.
- Recognition of defect-prone areas in early stages of software development lifecycle helps software testers to utilize their assets efficiently. The early planning and execution of their course of action will expedite the development process. Testing efforts in the weak spots lead to in-time defect removal.
- Identification of design and coding defects in early stages can lead software practitioners to efficacious allocation of limited resources, timely release of better software and higher customer satisfaction.

10.3 Future Directions

The work carried out in this thesis makes a conclusive contribution in providing strong background for selection of SDP models. However, it opens several research possibilities for the future.

- We have used search-based techniques for the model development and feature selection. More studies need to be conducted in both areas, specially feature selection, with the techniques and datasets other than those used in this study.
- In this thesis, we explored resampling methods with search-based methods for effective defect prediction modelling. The future research should leverage other imbalance learning methods combined with search-based techniques.
- In addition to machine learning techniques and search-based techniques, hybridized techniques can be investigated in order to improve the performance of defect prediction models.

- This work focused on binary defect classification problem. Future research should be conducted to extend defect classification as multiclass problem by exploring the impact of defect severity.

- We built SDP models based on different resampling methods. In literature the effect of these parameters on the overall performance measure are little known and explored. We intend to perform a detailed study of such dependencies analyzing the defect prediction capabilities of resampling-based models with different parameter values of resampling methods. Parameter optimization shall result in development of better SDP models.

- Replication is important in order to examine the set of the same hypothesis in different contexts or with the aim to improve the experiment and validate the findings of previous experiments. Data collected through replications can be used to refute or accept well-formed theories and increase the evidence on which software practitioners and researchers can base their decisions. Therefore, future studies may replicate our experiments to yield generalized conclusions.

Appendices

Appendix A

Datasetwize Values of Performance Measures achieved with NS and Resampling Methods

Table A.1: Tomcat6.0 ROC-AUC Values

Tomcat6.0	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
NB	0.78	0.78	0.79	0.77	0.77	0.8	0.79	0.78	0.73	**0.86**	0.68
LR	0.8	0.8	0.81	0.8	0.8	0.82	0.82	0.8	0.75	**0.86**	0.69
SL	0.81	0.8	0.81	0.8	0.8	0.82	0.82	0.8	0.77	**0.85**	0.68
LB	0.79	0.83	0.84	0.81	0.86	**0.88**	**0.88**	0.75	0.68	**0.88**	0.62
MLP	0.78	0.83	0.81	0.79	0.82	**0.88**	0.87	0.75	0.71	0.86	0.59
IBk	0.64	0.83	0.79	0.78	0.93	**0.95**	**0.95**	0.68	0.67	0.8	0.63
Kstar	0.72	0.88	0.86	0.78	0.96	**0.99**	0.98	0.72	0.64	0.85	0.64
ABM1	0.73	0.91	0.9	0.83	0.97	**0.99**	0.96	0.71	0.7	0.86	0.61
BAG	0.77	0.93	0.91	0.84	0.97	**0.99**	0.97	0.76	0.74	0.88	0.68
ICO	0.81	0.83	0.84	0.82	0.85	**0.88**	**0.88**	0.74	0.71	0.86	0.61
LMT	0.81	0.85	0.84	0.79	0.93	**0.96**	0.91	0.78	0.75	0.83	0.69
RT	0.58	0.84	0.8	0.77	0.92	**0.96**	0.9	0.64	0.63	0.76	0.57
RSS	0.78	0.91	0.9	0.85	0.96	**1**	0.97	0.79	0.74	0.87	0.71
PART	0.75	0.86	0.85	0.8	0.91	**0.96**	0.92	0.71	0.67	0.8	0.6

Tomcat6.0	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
J48	0.67	0.86	0.81	0.81	0.92	**0.95**	0.9	0.66	0.68	0.71	0.6

Table A.2: Tomcat6.0 Balance Values

Tomcat6.0	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
NB	55.6	58.27	57.18	56.29	51.87	56.58	56.78	60.47	53.95	**64.96**	50.34
LR	42.13	73.79	73.61	72.67	51.7	**75.74**	73.78	66.26	60.81	58.64	48.7
SL	38.47	73.26	73.04	72.32	51.03	**76.58**	73.74	65.36	63.9	54.97	45.33
LB	43.06	73.12	75.31	74.46	60.5	**80.24**	79.38	67.67	60.44	59.56	52.32
MLP	36.63	77.83	75.79	74.6	61.02	**81.99**	81.23	65.7	60.33	62.32	48.45
IBk	45.53	81.63	77.34	75.54	93.06	**94**	91.89	64.16	64.78	73.11	58.92
Kstar	41.17	81.25	79.39	77.13	**91.38**	90.48	90.43	61.35	60.83	57.71	55.2
ABM1	44.78	83.85	82.88	77.39	93.61	**94.99**	91.41	65.99	61.02	62.28	53.74
Bag	43.04	86.08	84.67	78.73	86.48	**94.35**	90.88	68.2	62.35	64.14	52.14
ICO	40.3	73.19	75.31	74.63	60.49	**80.31**	79.38	65.14	62.39	62.32	51.08
LMT	38.47	81.52	80.05	75.6	90.76	**92.73**	88.98	65.27	62.72	65.95	47.15
RT	45.57	84.01	79.62	75.77	91.62	**94.09**	90.11	61.66	60.64	70.3	54.03
RSS	32.05	83.7	83.39	76.7	79.29	**94.56**	90.07	64.16	61.88	54.08	50.1
PART	35.72	78.32	77.58	73.67	84.95	**91.66**	86.79	66.18	58.29	59.49	50.97
J48	33.88	82.66	79.64	76.69	87.7	**92.82**	89.03	64.77	59.57	62.31	49.87

Table A.3: Tomcat6.0 G-Mean Values

Tomcat6.0	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
NB	0.59	0.61	0.6	0.59	0.55	0.6	0.6	0.63	0.56	**0.7**	0.52
LR	0.42	0.75	0.75	0.73	0.55	**0.77**	0.75	0.69	0.63	0.64	0.5
SL	0.36	0.74	0.74	0.73	0.54	**0.77**	0.75	0.69	0.66	0.6	0.46
LB	0.44	0.73	0.76	0.75	0.64	**0.8**	0.79	0.69	0.62	0.65	0.54
MLP	0.32	0.78	0.76	0.75	0.65	**0.82**	0.81	0.68	0.63	0.68	0.49
IBk	0.46	0.82	0.78	0.76	0.93	**0.95**	0.93	0.65	0.65	0.77	0.6
Kstar	0.4	0.82	0.8	0.78	0.92	**0.93**	0.92	0.63	0.62	0.63	0.57
ABM1	0.46	0.84	0.83	0.78	0.94	**0.96**	0.92	0.67	0.63	0.67	0.56
Bag	0.44	0.87	0.85	0.79	0.88	**0.96**	0.92	0.7	0.64	0.69	0.54
ICO	0.39	0.73	0.76	0.75	0.64	**0.8**	0.79	0.66	0.63	0.68	0.52
LMT	0.36	0.82	0.8	0.76	0.91	**0.94**	0.89	0.68	0.65	0.71	0.48
RT	0.47	0.84	0.8	0.76	0.92	**0.95**	0.9	0.63	0.62	0.74	0.55
RSS	0.2	0.84	0.83	0.77	0.83	**0.96**	0.91	0.67	0.63	0.59	0.52
PART	0.3	0.78	0.78	0.74	0.85	**0.94**	0.87	0.68	0.6	0.64	0.52

Tomcat6.0	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
J48	0.25	0.83	0.8	0.77	0.88	**0.94**	0.9	0.66	0.62	0.68	0.51

Table A.4: Tomcat6.0 Sensitivity Values

Tomcat6.0	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
NB	0.38	0.42	0.4	0.39	0.32	0.39	0.4	0.45	0.36	**0.51**	0.31
LR	0.18	0.67	0.67	0.66	0.32	**0.69**	0.67	0.53	0.47	0.42	0.29
SL	0.13	0.66	0.66	0.65	0.31	**0.7**	0.67	0.52	0.51	0.36	0.23
LB	0.19	0.71	0.72	0.69	0.45	**0.81**	0.78	0.58	0.48	0.43	0.35
MLP	0.1	0.78	0.74	0.75	0.45	**0.85**	0.8	0.53	0.45	0.47	0.29
IBk	0.23	0.87	0.83	0.82	0.94	**0.99**	0.98	0.55	0.57	0.62	0.45
Kstar	0.17	0.89	0.85	0.87	0.9	**0.99**	0.98	0.48	0.48	0.4	0.39
ABM1	0.22	0.87	0.85	0.85	0.92	**0.99**	0.94	0.56	0.48	0.47	0.36
Bag	0.19	0.91	0.89	0.85	0.82	**0.99**	0.95	0.57	0.51	0.49	0.34
ICO	0.16	0.72	0.72	0.69	0.45	**0.81**	0.78	0.58	0.53	0.47	0.34
LMT	0.13	0.87	0.85	0.8	0.91	**0.99**	0.93	0.52	0.49	0.52	0.26
RT	0.23	0.87	0.82	0.83	0.92	**0.99**	0.88	0.51	0.49	0.58	0.42
RSS	0.04	0.87	0.85	0.77	0.71	**0.99**	0.94	0.51	0.49	0.35	0.3
PART	0.09	0.81	0.85	0.69	0.86	**0.99**	0.91	0.56	0.43	0.43	0.32
J48	0.06	0.88	0.84	0.84	0.85	**0.99**	0.93	0.55	0.44	0.47	0.31

Table A.5: Synapse1.0 ROC-AUC Values

Synapse1.0	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
NB	0.74	0.83	0.85	0.83	0.86	0.88	**0.91**	0.76	0.73	0.77	0.72
LR	0.72	0.86	0.85	0.83	0.84	0.89	**0.91**	0.59	0.59	0.75	0.59
SL	0.67	0.82	0.85	0.83	0.83	0.88	**0.9**	0.75	0.78	0.79	0.69
LB	0.73	0.88	0.89	0.84	0.93	**0.94**	0.93	0.75	0.76	0.79	0.58
MLP	0.74	0.88	0.88	0.8	0.87	**0.94**	0.92	0.6	0.74	0.88	0.68
IBk	0.66	0.81	0.81	0.79	0.89	**0.94**	0.93	0.72	0.73	0.76	0.63
KStar	0.67	0.85	0.88	0.8	0.96	**0.98**	0.97	0.73	0.76	0.82	0.7
ABM1	0.64	0.91	0.91	0.82	0.95	**0.99**	0.94	0.79	0.73	0.8	0.56
Bag	0.59	0.9	0.92	0.85	0.96	**0.99**	0.95	0.79	0.78	0.77	0.63
ICO	0.73	0.88	0.88	0.83	0.93	**0.94**	0.93	0.74	0.71	0.77	0.59
LMT	0.68	0.88	0.87	0.84	0.92	**0.94**	0.92	0.75	0.71	0.79	0.61
RT	0.58	0.83	0.85	0.76	0.89	**0.95**	0.87	0.72	0.68	0.77	0.57
RSS	0.59	0.92	0.9	0.84	0.94	**0.99**	0.95	0.73	0.79	0.83	0.63
PART	0.52	0.85	0.87	0.82	0.94	**0.94**	0.9	0.65	0.64	0.61	0.48

Synapse1.0	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
J48	0.59	0.84	0.83	0.82	0.9	**0.95**	0.89	0.65	0.68	0.72	0.52

Table A.6: Synapse1.0 Balance Values

Synapse1.0	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
NB	68.04	80.75	82.01	80.38	67.37	83.87	**84.44**	74.55	68.01	68.67	67.01
LR	60.17	76.89	77.54	77.39	60.44	81.12	**84.05**	62.46	63.68	68.91	57.39
SL	42.51	77.79	78.08	76.39	63.73	81.5	**84.8**	67.46	63.01	60.15	59.09
LB	42.49	81.48	82.15	79.87	84.95	**89.92**	88.42	76.64	77.7	46.88	53.24
MLP	51.26	81.98	82.15	75.39	82.92	**89.25**	86.72	55.39	74.05	72.77	62.83
IBk	50.95	80.02	79.48	73.17	91.17	**92.96**	89.92	68.13	68.86	73.13	58.04
KStar	50.95	79.65	80.38	75.81	**89.7**	87.92	85.86	71.7	75.37	68.76	66.3
ABM1	42.33	85.42	82.92	75.17	90.6	**94.38**	91.09	74.55	70.2	68.76	54.79
Bag	42.49	86.24	82.97	80.12	93.38	**93.76**	89.96	70.63	74.02	55.7	50.05
ICO	42.53	81.48	82.15	81.26	84.76	**89.72**	88.3	70.63	81.01	55.7	46.25
LMT	42.51	85.4	83.44	79.65	91.64	**92.4**	92.14	67.46	62.12	60.15	54.34
RT	46.5	82.73	84.16	73.6	88.72	**93.76**	86.44	70.63	67.36	72.9	55.01
RSS	29.29	85.4	82.3	79.46	83.29	**90.45**	89.96	72.42	79.24	29.29	29.06
PART	29.2	80.97	82.84	78.36	92.21	**93.76**	87.43	65.55	70.83	46.84	51.06
J48	33.62	84.7	84.46	79.34	86.92	**92.79**	90.69	65.55	74.02	51.19	44.81

Table A.7: Synapse1.0 G-Mean Values

Synapse1.0	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
NB	0.71	0.81	0.82	0.8	0.7	0.84	**0.85**	0.75	0.71	0.72	0.69
LR	0.65	0.77	0.78	0.78	0.64	0.81	**0.84**	0.64	0.64	0.73	0.59
SL	0.43	0.78	0.79	0.77	0.68	0.82	**0.85**	0.69	0.65	0.65	0.62
LB	0.43	0.81	0.83	0.8	0.86	**0.91**	0.89	0.78	0.78	0.49	0.54
MLP	0.55	0.83	0.83	0.76	0.84	**0.91**	0.88	0.56	0.75	0.76	0.65
IBk	0.53	0.8	0.8	0.74	0.91	**0.95**	0.91	0.68	0.69	0.77	0.6
Kstar	0.53	0.81	0.81	0.77	**0.91**	0.9	0.88	0.72	0.76	0.73	0.68
ABM1	0.42	0.86	0.83	0.76	0.91	**0.95**	0.91	0.75	0.71	0.73	0.57
Bag	0.43	0.86	0.83	0.81	0.93	**0.95**	0.9	0.72	0.74	0.6	0.51
ICO	0.43	0.81	0.83	0.83	0.86	**0.9**	0.89	0.72	0.81	0.6	0.46
LMT	0.43	0.86	0.84	0.8	0.92	**0.94**	0.92	0.69	0.64	0.65	0.56
RT	0.47	0.83	0.85	0.74	0.89	**0.95**	0.87	0.72	0.68	0.76	0.55
RSS	0	0.86	0.83	0.81	0.85	**0.93**	0.9	0.75	0.8	0	0
PART	0	0.83	0.83	0.81	0.92	**0.95**	0.88	0.66	0.71	0.49	0.51

Synapse1.0	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
J48	0.24	0.85	0.84	0.82	0.88	**0.94**	0.91	0.66	0.74	0.54	0.44

Table A.8: Synapse1.0 Sensitivity Values

Synapse1.0	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
NB	0.56	0.82	0.86	0.81	0.56	0.88	**0.89**	0.69	0.56	0.56	0.56
LR	0.44	0.75	0.74	0.72	0.44	0.81	**0.86**	0.5	0.56	0.56	0.44
SL	0.19	0.78	0.74	0.7	0.49	0.82	**0.85**	0.56	0.5	0.44	0.44
LB	0.19	0.81	0.88	0.86	0.8	**0.95**	0.91	0.69	0.75	0.25	0.38
MLP	0.31	0.9	0.88	0.8	0.78	**0.97**	0.94	0.44	0.69	0.63	0.5
IBk	0.31	0.85	0.88	0.81	0.91	**0.99**	0.95	0.63	0.63	0.63	0.44
KStar	0.31	0.9	0.89	0.85	0.96	**0.98**	**0.98**	0.69	0.69	0.56	0.56
ABM1	0.19	0.88	0.85	0.81	0.89	**0.98**	0.91	0.69	0.63	0.56	0.38
Bag	0.19	0.88	0.85	0.9	0.93	**0.98**	0.91	0.63	0.75	0.38	0.31
ICO	0.19	0.81	0.88	0.91	0.8	**0.95**	0.91	0.63	0.81	0.38	0.31
LMT	0.19	0.91	0.88	0.81	0.96	**0.98**	0.94	0.56	0.5	0.44	0.38
RT	0.25	0.87	0.88	0.81	0.87	**0.98**	0.9	0.63	0.63	0.63	0.44
RSS	0	0.91	0.87	0.89	0.78	**0.99**	0.94	0.63	0.75	0	0
PART	0	0.91	0.86	0.93	0.93	**0.98**	0.91	0.56	0.69	0.25	0.38
J48	0.06	0.88	0.85	0.93	0.82	**0.98**	0.92	0.56	0.75	0.31	0.25

Table A.9: Ivy2.0 ROC-AUC Values

Ivy2.0	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
NB	0.79	0.8	0.83	0.83	0.8	0.84	**0.86**	0.8	0.76	0.85	0.74
LR	0.79	0.8	0.84	0.84	0.79	0.84	**0.85**	0.76	0.73	0.82	0.68
SL	0.76	0.79	0.84	**0.84**	0.8	0.84	**0.84**	0.78	0.74	0.84	0.69
LB	0.77	0.88	0.87	0.83	0.88	**0.9**	0.89	0.76	0.72	0.84	0.6
MLP	0.73	0.87	0.86	0.84	0.85	**0.92**	0.91	0.78	0.67	0.79	0.6
IBk	0.68	0.82	0.82	0.78	0.93	**0.95**	**0.95**	0.72	0.67	0.83	0.64
KStar	0.73	0.86	0.87	0.78	0.97	**0.99**	0.98	0.74	0.77	0.81	0.71
ABM1	0.73	0.92	0.92	0.84	0.96	**0.99**	0.98	0.71	0.66	0.77	0.61
Bag	0.81	0.93	0.92	0.85	0.95	**0.99**	0.97	0.81	0.69	0.86	0.63
ICO	0.72	0.87	0.87	0.82	0.88	**0.9**	0.89	0.73	0.71	0.8	0.59
LMT	0.73	0.85	0.87	0.82	0.89	**0.93**	0.89	0.78	0.74	0.81	0.67
RT	0.61	0.82	0.82	0.77	0.9	**0.95**	0.9	0.69	0.64	0.73	0.55
RSS	0.83	0.9	0.93	0.86	0.93	**0.98**	0.96	0.79	0.73	0.85	0.7
PART	0.72	0.83	0.86	0.83	0.9	**0.93**	0.91	0.74	0.74	0.78	0.58

Ivy2.0	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
J48	0.72	0.85	0.84	0.83	0.9	**0.95**	0.88	0.61	0.58	0.61	0.6

Table A.10: Ivy2.0 Balance Values

Ivy2.0	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
NB	58.88	67.68	67.04	**68.46**	56.19	62.58	67.1	60.2	61.76	65.98	59.89
LR	45.18	73.64	75.19	75.07	58.22	**76.49**	76.27	67.39	59.69	61.05	56.45
SL	38.12	73.25	75.4	75.78	57.47	**77.9**	75.84	63.39	57.55	57.54	58.17
LB	43.4	79.22	79.92	79.32	70.31	**83.16**	81.04	65.53	57.37	66.25	49.75
MLP	41.65	82.59	80.06	77.68	77.88	**86.9**	84.6	71.17	60.41	73.15	51.23
IBk	57.25	81.67	80.7	76.88	90.1	**93.76**	92.76	70.04	67.08	78.53	67.14
KStar	48.55	81.03	81.59	77.98	90.89	**91.78**	90.77	65.35	73.27	71.52	67
ABM1	50.35	83.93	84.07	79.06	92.62	**95.36**	92	66.06	60.14	66.35	54.76
Bag	46.84	84.53	82.83	79.25	86.97	**95.58**	90.38	74.25	59.16	64.56	54.76
ICO	43.33	79.39	79.75	79.84	70.97	**83.15**	81.04	58.76	62.35	71.45	48.79
LMT	38.1	83.3	81.74	77.87	88.41	**91.13**	87.93	63.39	57.55	57.52	58.17
RT	51.69	81.56	81.34	77.43	90.38	**93.39**	89.27	67.59	64.01	67.44	51.64
RSS	32.82	83.27	84.38	79.61	80.59	**94.03**	88.76	65.35	60.63	62.83	49.88
PART	38.12	80.09	79.13	79.56	86.17	**91.33**	88.97	66.03	59.01	71.31	51.47
J48	38.12	84.03	80.9	78.93	87.98	**90.88**	87.11	63.4	54.29	67.98	52.3

Table A.11: Ivy2.0 G-Mean Values

Ivy2.0	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
NB	0.62	0.69	0.69	**0.71**	0.59	0.65	0.7	0.63	0.64	0.7	0.62
LR	0.47	0.74	0.76	0.76	0.63	**0.77**	**0.77**	0.69	0.61	0.66	0.59
SL	0.35	0.74	0.76	0.76	0.62	**0.78**	0.77	0.66	0.58	0.62	0.61
LB	0.44	0.79	0.8	0.79	0.73	**0.84**	0.81	0.66	0.58	0.71	0.5
MLP	0.41	0.83	0.8	0.78	0.8	**0.87**	0.85	0.72	0.61	0.77	0.52
IBk	0.61	0.82	0.81	0.78	0.9	**0.95**	0.94	0.7	0.67	0.82	0.67
KStar	0.51	0.81	0.82	0.79	0.92	**0.93**	0.92	0.67	0.73	0.76	0.67
ABM1	0.53	0.84	0.84	0.79	0.93	**0.96**	0.92	0.66	0.6	0.71	0.56
Bag	0.49	0.85	0.83	0.79	0.88	**0.96**	0.91	0.75	0.6	0.7	0.56
ICO	0.44	0.79	0.8	0.8	0.74	**0.84**	0.81	0.6	0.63	0.75	0.49
LMT	0.35	0.85	0.82	0.78	0.88	**0.93**	0.89	0.66	0.58	0.62	0.61
RT	0.54	0.82	0.82	0.79	0.9	**0.95**	0.9	0.68	0.64	0.7	0.52
RSS	0.22	0.84	0.85	0.8	0.83	**0.95**	0.89	0.67	0.62	0.68	0.52
PART	0.35	0.8	0.8	0.8	0.86	**0.93**	0.9	0.67	0.63	0.75	0.52

Ivy2.0	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
J48	0.35	0.85	0.81	0.8	0.89	**0.93**	0.88	0.63	0.55	0.72	0.54

Table A.12: Ivy2.0 Sensitivity Values

Ivy2.0	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
NB	0.43	**0.57**	0.55	**0.57**	0.39	0.48	0.55	0.45	0.48	0.53	0.45
LR	0.23	0.68	0.69	0.69	0.41	**0.71**	0.7	0.58	0.48	0.45	0.4
SL	0.13	0.68	0.69	0.71	0.4	**0.73**	0.7	0.5	0.45	0.4	0.43
LB	0.2	0.81	0.8	0.78	0.59	**0.87**	0.84	0.58	0.48	0.53	0.33
MLP	0.18	0.84	0.78	0.73	0.7	**0.9**	0.85	0.65	0.53	0.63	0.38
IBk	0.4	0.86	0.86	0.88	0.88	**0.98**	0.98	0.65	0.68	0.7	0.65
KStar	0.28	0.86	0.87	0.85	0.88	**0.99**	0.98	0.55	0.73	0.6	0.63
ABM1	0.3	0.88	0.84	0.83	0.91	**0.99**	0.94	0.65	0.58	0.53	0.4
Bag	0.25	0.92	0.86	0.83	0.83	**0.99**	0.95	0.68	0.48	0.5	0.4
ICO	0.2	0.81	0.8	0.79	0.6	**0.88**	0.84	0.45	0.53	0.6	0.35
LMT	0.13	0.94	0.87	0.84	0.88	**0.99**	0.93	0.5	0.45	0.4	0.43
RT	0.33	0.81	0.86	0.89	0.9	**0.99**	0.93	0.6	0.6	0.55	0.38
RSS	0.05	0.89	0.88	0.83	0.73	**0.99**	0.9	0.55	0.48	0.48	0.3
PART	0.13	0.81	0.86	0.81	0.83	**0.99**	0.93	0.58	0.43	0.6	0.4
J48	0.13	0.91	0.86	0.86	0.84	**0.99**	0.92	0.6	0.4	0.55	0.35

Table A.13: Jedit4.2 ROC-AUC Values

Jedit4.2	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
NB	0.83	0.78	0.82	0.79	0.81	0.85	0.85	0.8	0.74	**0.89**	0.72
LR	0.8	0.81	0.85	0.81	0.83	0.87	0.87	0.78	0.71	**0.89**	0.7
SL	0.84	0.78	0.83	0.81	0.83	0.85	0.87	0.83	0.78	**0.89**	0.68
LB	0.82	0.83	0.86	0.82	0.87	0.91	**0.92**	0.8	0.75	0.85	0.71
MLP	0.82	0.84	0.83	0.79	0.84	0.9	**0.92**	0.74	0.69	0.87	0.62
IBk	0.66	0.78	0.81	0.77	0.9	**0.95**	0.93	0.61	0.68	0.82	0.67
KStar	0.74	0.84	0.87	0.76	0.97	**0.99**	**0.99**	0.7	0.59	0.83	0.64
ABM1	0.75	0.89	0.9	0.8	0.98	**1**	0.97	0.72	0.71	0.85	0.63
Bag	0.82	0.9	0.9	0.83	0.96	**0.99**	0.96	0.77	0.74	0.87	0.68
ICO	0.83	0.83	0.86	0.82	0.87	**0.91**	**0.91**	0.77	0.78	0.83	0.6
LMT	0.84	0.82	0.81	0.81	0.92	**0.93**	0.92	0.83	0.76	0.89	0.68
RT	0.59	0.8	0.8	0.75	0.88	**0.96**	0.88	0.67	0.61	0.73	0.61
RSS	0.84	0.89	0.9	0.84	0.94	**0.99**	0.95	0.82	0.77	0.88	0.71
PART	0.84	0.84	0.83	0.8	0.88	**0.94**	0.89	0.69	0.73	0.77	0.62

Jedit4.2	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
J48	0.69	0.81	0.79	0.78	0.91	**0.94**	0.89	0.7	0.62	0.71	0.61

Table A.14: Jedit4.2 Balance Values

Jedit4.2	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
NB	57.05	59.88	60.33	55.49	53.31	58.73	58.94	59.68	57.61	**61.37**	54.99
LR	48.4	71.61	74.7	73.91	58.25	77.47	**78.78**	70.79	66.42	68.89	66.19
SL	45.47	71.62	73.5	73.3	58.3	76.32	**76.67**	68.71	72.41	66.01	56.56
LB	48.37	74.95	77.73	72.33	71.96	84.85	**85.55**	67.71	66.32	67.33	61.79
MLP	51.35	74.3	74.98	72.05	65.43	81.14	**83.05**	63.41	60.89	67.4	55.04
IBk	55.58	77.88	79.91	71.63	91.45	**94.16**	90.66	63.27	68.21	78.7	64.13
KStar	49.72	76.8	78.33	73.85	91.62	**92.18**	89.92	69.06	58.29	67.37	57.82
ABM1	55.58	81.63	84.33	75.56	93.9	**94.19**	90.63	67.46	65.02	70.2	61.28
Bag	51.31	81.25	83.16	75.83	88.15	**94.41**	87.91	66.46	66.83	65.96	62.81
ICO	46.92	74.33	77.73	72.01	71.9	85.04	**85.72**	67.27	66.71	70.11	57.37
LMT	45.47	76.27	76.34	74.53	**89.89**	88.88	87.68	68.71	70.69	66.01	56.56
RT	49.28	79.46	79.48	72.74	87.6	**95.1**	87.45	66.67	60.31	67	58.08
RSS	38.12	80.57	81.48	76.1	82.16	**92**	87.13	66.36	67.55	67.46	55.86
PART	44	76.11	77.73	74.6	88.16	**92.65**	88.16	69.06	69.28	55.61	56
J48	45.39	77.03	77.14	73.89	89.27	**91.34**	88.44	69.58	64.64	70.11	56.29

Table A.15: Jedit4.2 G-Mean Values

Jedit4.2	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
NB	0.61	0.63	0.63	0.59	0.57	0.62	0.63	0.63	0.6	**0.65**	0.57
LR	0.51	0.72	0.75	0.74	0.62	0.78	**0.79**	0.72	0.68	0.73	0.67
SL	0.47	0.72	0.74	0.74	0.62	**0.77**	**0.77**	0.72	0.75	0.71	0.58
LB	0.51	0.75	0.78	0.73	0.74	0.85	**0.86**	0.69	0.67	0.71	0.63
MLP	0.55	0.75	0.75	0.72	0.68	0.81	**0.84**	0.66	0.61	0.72	0.56
IBk	0.59	0.78	0.81	0.72	0.91	**0.95**	0.92	0.64	0.68	0.81	0.64
KStar	0.52	0.78	0.79	0.75	0.92	**0.94**	0.92	0.7	0.59	0.72	0.58
ABM1	0.59	0.82	0.85	0.76	0.94	**0.95**	0.91	0.68	0.66	0.74	0.62
Bag	0.55	0.82	0.84	0.76	0.88	**0.96**	0.89	0.67	0.68	0.7	0.64
ICO	0.49	0.75	0.78	0.72	0.74	**0.86**	**0.86**	0.68	0.7	0.74	0.59
LMT	0.47	0.78	0.77	0.75	0.9	**0.91**	0.88	0.72	0.73	0.71	0.58
RT	0.51	0.8	0.8	0.73	0.88	**0.96**	0.88	0.67	0.61	0.7	0.59
RSS	0.35	0.81	0.82	0.76	0.84	**0.93**	0.87	0.68	0.7	0.72	0.58
PART	0.45	0.78	0.79	0.76	0.89	**0.94**	0.88	0.7	0.72	0.59	0.57

Jedit4.2	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
J48	0.47	0.79	0.77	0.75	0.89	**0.93**	0.89	0.72	0.66	0.74	0.58

Table A.16: Jedit4.2 Sensitivity Values

Jedit4.2	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
NB	0.4	0.45	0.45	0.38	0.34	0.42	0.42	0.44	0.42	**0.46**	0.38
LR	0.27	0.65	0.69	0.69	0.41	0.72	**0.75**	0.63	0.56	0.56	0.56
SL	0.23	**0.75**	0.69	0.68	0.41	0.73	0.72	0.56	0.63	0.52	0.42
LB	0.27	0.81	0.77	0.68	0.62	**0.90**	0.88	0.58	0.58	0.54	0.5
MLP	0.31	0.85	0.76	0.69	0.53	0.84	**0.91**	0.5	0.56	0.54	0.42
IBk	0.38	0.82	0.87	0.76	0.92	**0.99**	0.97	0.56	0.65	0.71	0.58
KStar	0.29	0.85	0.87	0.83	0.92	**0.99**	0.98	0.6	0.48	0.54	0.5
ABM1	0.38	0.86	0.87	0.8	0.95	**0.99**	0.94	0.63	0.56	0.58	0.52
Bag	0.31	0.9	0.88	0.83	0.86	**0.99**	0.93	0.58	0.56	0.52	0.52
ICO	0.25	0.81	0.77	0.68	0.62	**0.9**	0.88	0.6	0.54	0.58	0.44
LMT	0.23	0.88	0.83	0.72	0.89	**0.99**	0.92	0.56	0.6	0.52	0.42
RT	0.29	0.83	0.86	0.77	0.9	**0.99**	0.91	0.67	0.52	0.54	0.46
RSS	0.13	0.86	0.84	0.8	0.76	**0.96**	0.91	0.56	0.56	0.54	0.4
PART	0.21	0.9	0.87	0.86	0.92	**0.99**	0.9	0.6	0.58	0.38	0.42
J48	0.23	0.9	0.81	0.85	0.89	**0.99**	0.91	0.58	0.54	0.58	0.42

Table A.17: Xerces1.3 ROC-AUC Values

Xerces1.3	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
NB	0.78	0.73	0.78	0.75	0.75	0.79	0.81	0.78	0.8	**0.83**	0.78
LR	0.76	0.73	0.78	0.75	0.73	0.79	**0.8**	0.74	0.74	0.79	0.73
SL	0.73	0.73	0.78	0.75	0.74	0.8	**0.81**	0.74	0.74	**0.81**	0.73
LB	0.81	0.81	0.89	0.8	0.84	0.87	**0.9**	0.79	0.79	0.86	0.77
MLP	0.72	0.79	0.85	0.75	0.8	0.87	**0.88**	0.77	0.79	0.82	0.72
IBk	0.77	0.86	0.87	0.78	0.92	**0.95**	**0.95**	0.74	0.81	0.83	0.81
KStar	0.82	0.88	0.9	0.78	0.95	**0.97**	**0.97**	0.79	0.84	0.87	0.84
ABM1	0.76	0.9	0.92	0.78	0.94	**0.97**	0.96	0.76	0.8	0.87	0.75
Bag	0.8	0.92	0.91	0.81	0.92	**0.96**	**0.96**	0.78	0.81	0.86	0.82
ICO	0.79	0.81	0.89	0.79	0.79	0.87	**0.9**	0.76	0.77	0.85	0.78
LMT	0.76	0.89	0.87	0.74	0.86	**0.94**	0.92	0.7	0.68	0.78	0.72
RT	0.68	0.86	0.83	0.74	0.92	**0.95**	0.88	0.66	0.72	0.78	0.66
RSS	0.83	0.89	0.92	0.81	0.89	**0.96**	0.94	0.82	0.82	0.87	0.81
PART	0.77	0.86	0.87	0.77	0.87	**0.93**	0.91	0.77	0.76	0.82	0.73

Xerces1.3	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
J48	0.6	0.85	0.87	0.73	0.82	**0.94**	0.89	0.7	0.72	0.74	0.65

Table A.18: Xerces1.3 Balance Values

Xerces1.3	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
NB	56.67	55.21	63.66	57.55	63.11	68.53	67.53	55.57	63.67	**69.98**	62.52
LR	44.62	71.01	72.02	66.2	62.08	76.13	**77.39**	68.01	65.43	61.88	65.11
SL	39.51	70.47	72.53	65.77	60.12	77.63	**77.73**	68.6	66.4	62.98	65.54
LB	56.87	74.49	**82.84**	72.6	64.53	78.49	79.48	69.15	71.63	63.92	69.52
MLP	53.74	71.77	**79.84**	66.12	68.09	76.97	79.03	65.74	68.9	58.82	63.92
IBk	63.66	83.24	83.4	69.17	86.4	92.25	**92.52**	70.16	77.79	74.91	71.57
KStar	60.95	82.52	82.44	71.66	84.8	**91.05**	90.45	71.7	72.79	72.21	72.53
ABM1	58.71	83.5	86	68.04	88.3	**91.67**	89.83	65.86	74.25	72.84	72.31
Bag	56.84	84.44	83.55	72.35	80.11	**91.3**	89.95	70.91	71.63	67.03	72.03
ICO	53.84	74.22	**82.47**	72.2	59.32	78.49	79.48	71.99	73.52	59.87	62.82
LMT	51.79	80.92	82.82	67.78	80.86	**89.94**	88.55	68.6	65.97	67.92	63.44
RT	59.77	85.5	82.53	69.9	87.77	**92.79**	88.48	65.42	72.28	74.67	64.99
RSS	47.73	80.56	83.19	70.97	70.25	**90.36**	86.99	72.31	73.84	61	64.57
PART	50.72	79.55	81.02	68.91	74.49	**89.9**	88.87	72.6	69.45	69.16	59.82
J48	54.8	79.16	83.78	69.36	79.67	**90.21**	88.84	72.6	72.28	66.65	59.69

Table A.19: Xerces1.3 G-Mean Values

Xerces1.3	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
NB	0.6	0.58	0.67	0.61	0.67	0.72	0.71	0.59	0.67	**0.74**	0.66
LR	0.46	0.71	0.73	0.67	0.66	0.77	**0.78**	0.69	0.66	0.66	0.67
SL	0.38	0.71	0.73	0.66	0.64	**0.78**	**0.78**	0.69	0.67	0.68	0.68
LB	0.61	0.75	**0.83**	0.73	0.68	0.79	0.8	0.7	0.72	0.68	0.7
MLP	0.57	0.72	**0.81**	0.67	0.72	0.79	**0.81**	0.68	0.71	0.63	0.66
IBk	0.67	0.83	0.83	0.69	0.87	0.92	**0.93**	0.71	0.78	0.78	0.73
KStar	0.66	0.83	0.82	0.72	0.87	**0.91**	**0.91**	0.73	0.74	0.77	0.74
ABM1	0.63	0.84	0.86	0.68	0.89	**0.92**	0.9	0.67	0.75	0.76	0.73
Bag	0.61	0.85	0.84	0.72	0.82	**0.91**	0.9	0.71	0.72	0.71	0.73
ICO	0.58	0.74	**0.83**	0.72	0.64	0.79	0.8	0.72	0.74	0.64	0.67
LMT	0.56	0.82	0.83	0.68	0.82	**0.9**	0.89	0.69	0.66	0.72	0.65
RT	0.64	0.86	0.83	0.7	0.88	**0.93**	0.89	0.66	0.72	0.77	0.65
RSS	0.51	0.81	0.83	0.71	0.75	**0.9**	0.87	0.73	0.74	0.66	0.67
PART	0.54	0.8	0.81	0.7	0.77	**0.9**	0.89	0.73	0.7	0.74	0.62

Xerces1.3	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
J48	0.59	0.81	0.84	0.69	0.81	**0.9**	0.89	0.73	0.72	0.7	0.63

Table A.20: Xerces1.3 Sensitivity Values

Xerces1.3	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
NB	0.39	0.37	0.49	0.41	0.48	0.56	0.55	0.38	0.49	**0.58**	0.48
LR	0.22	0.66	0.65	0.56	0.47	0.71	**0.73**	0.59	0.57	0.46	0.54
SL	0.14	0.65	0.67	0.58	0.44	0.73	**0.75**	0.61	0.59	0.48	0.54
LB	0.39	0.77	0.79	0.75	0.5	**0.82**	0.75	0.62	0.65	0.49	0.64
MLP	0.35	0.68	**0.75**	0.56	0.55	0.69	0.72	0.54	0.58	0.42	0.52
IBk	0.49	0.86	0.83	0.69	0.83	0.94	**0.95**	0.62	0.77	0.65	0.62
KStar	0.45	0.86	0.82	0.73	0.79	**0.92**	**0.92**	0.64	0.65	0.61	0.64
ABM1	0.42	0.86	0.86	0.67	0.86	**0.94**	0.9	0.57	0.7	0.62	0.67
Bag	0.39	0.91	0.83	0.71	0.73	**0.92**	0.91	0.68	0.65	0.54	0.64
ICO	0.35	0.76	0.79	0.71	0.43	**0.82**	0.75	0.71	0.77	0.43	0.48
LMT	0.32	0.91	0.79	0.63	0.75	**0.93**	0.9	0.61	0.59	0.55	0.52
RT	0.43	0.86	0.81	0.71	0.85	**0.94**	0.91	0.58	0.7	0.65	0.58
RSS	0.26	0.84	0.81	0.66	0.58	**0.9**	0.88	0.65	0.72	0.45	0.52
PART	0.3	0.86	0.82	0.79	0.65	**0.93**	**0.93**	0.7	0.64	0.57	0.45
J48	0.36	0.91	0.81	0.71	0.73	**0.93**	0.9	0.7	0.7	0.54	0.43

Table A.21: Camel1.6 ROC-AUC Values

Camel1.6	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
NB	0.68	0.67	0.68	0.63	0.69	0.68	0.68	0.65	0.63	**0.74**	0.62
LR	0.69	0.69	0.69	0.63	0.72	0.7	0.7	0.68	0.65	**0.76**	0.63
SL	0.68	0.67	0.68	0.63	0.71	0.69	0.7	0.68	0.64	**0.75**	0.6
LB	0.73	0.71	0.74	0.69	0.74	0.73	0.76	0.66	0.72	**0.78**	0.69
MLP	0.69	0.73	0.69	0.66	0.75	0.76	0.76	0.67	0.64	**0.78**	0.65
IBk	0.64	0.77	0.74	0.62	0.89	**0.91**	0.88	0.61	0.68	0.83	0.68
KStar	0.67	0.78	0.79	0.57	0.95	**0.98**	0.94	0.59	0.67	0.82	0.69
ABM1	0.71	0.83	0.83	0.65	0.94	**0.97**	0.91	0.64	0.65	0.84	0.69
Bag	0.72	0.82	0.82	0.68	0.92	**0.96**	0.92	0.67	0.69	0.83	0.67
ICO	0.73	0.71	0.74	0.68	0.74	0.73	0.76	0.62	0.71	**0.78**	0.67
LMT	0.68	0.75	0.76	0.66	0.86	**0.89**	0.81	0.64	0.63	0.75	0.57
RT	0.59	0.72	0.73	0.58	0.85	**0.9**	0.78	0.57	0.57	0.69	0.59
RSS	0.69	0.81	0.83	0.68	0.91	**0.96**	0.88	0.66	0.69	0.81	0.72
PART	0.68	0.74	0.77	0.66	0.85	**0.89**	0.82	0.62	0.61	0.77	0.65

Camel1.6	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
J48	0.59	0.73	0.73	0.63	0.85	**0.88**	0.81	0.6	0.57	0.69	0.61

Table A.22: Camel1.6 Balance Values

Camel1.6	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
NB	43.82	44.7	45.65	41.66	**47.94**	46.05	43.61	46.32	46.86	46.46	45.35
LR	36.42	61.6	**61.91**	57.26	53.15	61.68	61.72	55.5	58.28	51.11	51.31
SL	36.04	60.95	**61.34**	57.41	51.87	60.43	61	56.21	55.06	45.29	47.11
LB	37.9	64.68	68.91	64.47	56.68	65.59	**70.24**	56.47	64.85	59.34	60.84
MLP	38.21	65.96	63.65	59.28	66.28	65.59	**68.81**	60.19	61.1	64.5	52.83
IBk	52.16	74.42	71.69	58.31	88.87	**89.11**	85.41	59.59	67.37	80.29	66.68
KStar	47.66	69.46	70.27	58.75	**84.47**	84.4	81.33	55.97	60.42	68.32	60.71
ABM1	51.13	75.81	75.28	61.41	87.78	**88.56**	83.53	61.3	62.11	71.26	61.91
Bag	44.95	73.74	74.51	62.28	85.01	**88.08**	81.94	61.7	62.41	66.86	59.8
ICO	38.65	65.57	68.59	62.69	56.68	65.5	**69.59**	53.87	63.74	56.86	53.37
LMT	37.54	69.72	70.1	62.81	83.84	**84.82**	78.4	55.55	58.33	58.02	47.17
RT	52.48	71.18	72.91	56.24	83.93	**87.15**	78.4	56.45	56.61	66.28	58.55
RSS	31.54	72.98	74.85	62.23	81.59	**87.56**	81.02	56.71	63.94	50.94	55.95
PART	38.64	65.19	68.84	61.54	78.38	**83.32**	76.73	60.54	56.48	63.87	57.58
J48	42.22	69.19	70.9	61.81	81.02	**82.63**	78.44	57.61	53.66	64.46	56.39

Table A.23: Camel1.6 G-Mean Values

Camel1.6	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
NB	0.44	0.45	0.47	0.41	**0.5**	0.47	0.44	0.48	0.48	0.48	0.47
LR	0.31	0.62	**0.62**	0.58	0.55	0.62	0.62	0.57	0.59	0.54	0.53
SL	0.31	0.61	**0.62**	0.58	0.54	0.61	0.61	0.57	0.56	0.46	0.48
LB	0.34	0.65	0.69	0.64	0.59	0.66	**0.7**	0.57	0.65	0.62	0.62
MLP	0.35	0.66	0.64	0.6	0.66	0.67	**0.69**	0.61	0.61	0.67	0.54
IBk	0.54	0.75	0.72	0.58	0.89	**0.91**	0.86	0.6	0.67	0.81	0.67
KStar	0.49	0.7	0.71	0.59	0.86	**0.88**	0.84	0.56	0.6	0.7	0.61
ABM1	0.53	0.76	0.75	0.61	0.88	**0.9**	0.84	0.61	0.62	0.73	0.62
Bag	0.46	0.74	0.75	0.62	0.85	**0.89**	0.82	0.62	0.62	0.7	0.6
ICO	0.36	0.66	0.69	0.63	0.59	0.66	**0.7**	0.54	0.64	0.6	0.55
LMT	0.34	0.7	0.7	0.63	0.85	**0.87**	0.79	0.57	0.59	0.61	0.48
RT	0.54	0.71	0.73	0.56	0.85	**0.89**	0.79	0.57	0.57	0.67	0.59
RSS	0.18	0.73	0.75	0.62	0.82	**0.88**	0.81	0.57	0.64	0.54	0.58
PART	0.36	0.66	0.69	0.62	0.79	**0.85**	0.77	0.61	0.57	0.66	0.58

Camel1.6	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
J48	0.42	0.7	0.71	0.62	0.81	**0.85**	0.79	0.58	0.54	0.66	0.57

Table A.24: Camel1.6 Sensitivity Values

Camel1.6	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
NB	0.21	0.22	0.23	0.18	**0.27**	0.24	0.2	0.24	0.26	0.24	0.23
LR	0.1	**0.58**	0.56	0.49	0.35	0.54	0.56	0.41	0.5	0.31	0.33
SL	0.1	**0.57**	0.56	0.49	0.33	0.52	0.55	0.42	0.43	0.23	0.26
LB	0.12	0.73	0.74	0.63	0.4	0.64	**0.75**	0.45	0.62	0.44	0.49
MLP	0.13	0.62	0.56	0.49	0.69	0.55	**0.73**	0.51	0.56	0.52	0.37
IBk	0.34	0.81	0.77	0.57	0.93	**0.98**	0.92	0.56	0.68	0.75	0.62
KStar	0.27	0.81	0.8	0.64	0.94	**0.98**	0.95	0.5	0.57	0.57	0.52
ABM1	0.32	0.79	0.79	0.63	0.91	**0.97**	0.86	0.59	0.61	0.62	0.55
Bag	0.22	0.79	0.76	0.64	0.86	**0.95**	0.86	0.55	0.62	0.54	0.5
ICO	0.13	0.75	0.75	0.57	0.4	0.63	**0.75**	0.4	0.65	0.4	0.36
LMT	0.12	0.78	0.74	0.62	0.91	**0.97**	0.83	0.41	0.51	0.42	0.27
RT	0.35	0.75	0.74	0.55	0.91	**0.98**	0.81	0.51	0.52	0.57	0.52
RSS	0.03	0.75	0.76	0.6	0.77	**0.93**	0.82	0.45	0.63	0.31	0.39
PART	0.13	0.76	0.65	0.62	0.85	**0.94**	0.81	0.59	0.59	0.51	0.45
J48	0.19	0.8	0.73	0.66	0.85	**0.95**	0.81	0.56	0.48	0.52	0.46

Table A.25: Ant1.7 ROC-AUC Values

Ant1.7	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
NB	0.8	0.74	0.81	0.81	0.77	0.81	0.8	0.8	0.76	**0.87**	0.77
LR	0.82	0.77	0.83	0.83	0.79	0.83	0.83	0.82	0.78	**0.89**	0.78
SL	0.83	0.77	0.83	0.82	0.8	0.83	0.83	0.83	0.79	**0.89**	0.79
LB	0.8	0.78	0.83	0.81	0.83	0.86	0.84	0.8	0.77	**0.86**	0.78
MLP	0.81	0.77	0.83	0.82	0.79	0.85	0.83	0.8	0.74	**0.89**	0.77
IBk	0.69	0.78	0.74	0.7	0.87	**0.91**	0.89	0.68	0.65	0.83	0.68
KStar	0.78	0.81	0.86	0.7	0.94	**0.98**	0.96	0.71	0.75	0.89	0.73
ABM1	0.78	0.84	0.87	0.79	0.96	**0.97**	0.92	0.77	0.75	0.89	0.75
Bag	0.8	0.85	0.88	0.81	0.95	**0.96**	0.92	0.81	0.77	0.9	0.77
ICO	0.79	0.78	0.83	0.81	0.83	**0.86**	0.84	0.79	0.74	**0.86**	0.79
LMT	0.83	0.8	0.83	0.81	0.88	**0.91**	0.82	0.82	0.76	0.89	0.77
RT	0.67	0.74	0.77	0.69	0.88	**0.90**	0.79	0.66	0.62	0.78	0.66
RSS	0.8	0.82	0.87	0.83	0.93	**0.93**	0.88	0.81	0.78	0.88	0.78
PART	0.76	0.79	0.83	0.8	0.87	**0.88**	0.85	0.79	0.64	0.87	0.77

Ant1.7	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
J48	0.74	0.79	0.79	0.77	0.88	**0.88**	0.83	0.7	0.68	0.79	0.72

Table A.26: Ant1.7 Balance Values

Ant1.7	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
NB	64.16	59.67	65.92	64.48	59.94	64.46	65.1	67.84	62.86	**71.04**	63.16
LR	56.81	67.47	73.4	73.28	63.01	73.84	73.59	70.5	72.71	**76.6**	70.92
SL	55.17	67.01	73.74	72.7	62.5	72.67	73.12	69.39	74.99	**76.06**	71.15
LB	67.82	70.12	75.94	73.07	65.94	76.33	75.31	75.3	69.11	**81.51**	70.69
MLP	61.32	69.53	75.95	75.23	68.22	76.6	74.72	71.95	70.92	**76.89**	70.69
IBk	63.66	75.77	74.63	66.69	87.47	**88.99**	86.66	67.35	67.28	81.34	67.38
KStar	58.32	73.44	77.53	69.5	87.17	**87.47**	85.03	66.17	66.28	78.39	64.43
ABM1	62.07	77.18	79.21	72.77	90.68	**90.89**	83.96	69.57	64.9	79.02	69.73
Bag	63.32	76.95	79.89	74.19	**90.64**	89.45	83.79	75.02	66.87	80.4	69.87
ICO	69.46	70.12	75.94	72.91	65.12	76.29	75.2	74.67	72.7	**81.35**	71.41
LMT	55.17	74.77	76.02	72.99	**86.7**	86.56	80.61	68.92	72.94	76.06	69.64
RT	63.27	73.66	76.23	68.39	87.63	**88.31**	79.42	64.76	61.68	76.79	65.27
RSS	62.13	74.67	79.98	74.8	81.29	**87.11**	80.34	74.42	74.28	78.59	70.69
PART	65.82	72.27	73.51	72.09	73.67	**82.5**	77.36	72.02	65.11	78.55	71.03
J48	68.55	74.79	77.7	73.4	84.92	**85.75**	81.75	72.03	69.56	75.15	68.8

Table A.27: Ant1.7 G-Mean Values

Ant1.7	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
NB	0.68	0.62	0.69	0.68	0.64	0.68	0.68	0.7	0.66	**0.74**	0.66
LR	0.61	0.68	0.74	0.74	0.67	0.75	0.75	0.72	0.73	**0.8**	0.71
SL	0.59	0.68	0.75	0.73	0.66	0.74	0.74	0.72	0.75	**0.79**	0.72
LB	0.71	0.7	0.76	0.73	0.7	0.78	0.76	0.75	0.69	**0.83**	0.71
MLP	0.65	0.7	0.76	0.75	0.7	0.77	0.75	0.72	0.71	**0.8**	0.71
IBk	0.66	0.76	0.75	0.67	0.87	**0.91**	0.88	0.67	0.68	0.83	0.67
KStar	0.61	0.74	0.78	0.7	0.87	**0.89**	0.86	0.67	0.67	0.8	0.64
ABM1	0.65	0.77	0.79	0.73	0.91	**0.92**	0.84	0.7	0.65	0.8	0.7
Bag	0.67	0.77	0.8	0.74	**0.91**	0.9	0.84	0.75	0.67	0.82	0.7
ICO	0.72	0.7	0.76	0.73	0.69	0.78	0.76	0.75	0.73	**0.82**	0.73
LMT	0.59	0.75	0.76	0.73	0.87	**0.89**	0.81	0.7	0.73	0.79	0.7
RT	0.65	0.74	0.76	0.68	0.88	**0.9**	0.79	0.65	0.62	0.78	0.66
RSS	0.66	0.75	0.8	0.75	0.83	**0.87**	0.8	0.75	0.74	0.8	0.71
PART	0.69	0.72	0.75	0.72	0.76	**0.83**	0.78	0.73	0.65	0.81	0.71

Ant1.7	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
J48	0.71	0.75	0.78	0.74	0.85	**0.87**	0.82	0.72	0.7	0.77	0.69

Table A.28: Ant1.7 Sensitivity Values

Ant1.7	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
NB	0.5	0.44	0.53	0.51	0.44	0.51	0.52	0.56	0.49	**0.6**	0.49
LR	0.39	0.6	0.67	0.67	0.48	0.67	0.67	0.61	**0.77**	0.67	0.65
SL	0.37	0.59	0.67	0.66	0.48	0.64	0.66	0.58	**0.78**	0.67	0.65
LB	0.55	0.71	0.73	0.71	0.52	0.68	0.71	0.72	0.75	**0.76**	0.65
MLP	0.46	0.68	0.74	0.74	0.58	0.72	0.71	0.67	**0.76**	0.68	0.66
IBk	0.51	0.82	0.78	0.68	0.88	**0.97**	0.94	0.64	0.75	0.75	0.7
KStar	0.42	0.81	0.83	0.73	0.88	**0.97**	0.94	0.58	0.73	0.71	0.63
ABM1	0.48	0.8	0.81	0.74	0.91	**0.97**	0.87	0.65	0.74	0.73	0.67
Bag	0.49	0.83	0.82	0.74	0.89	**0.93**	0.85	0.71	0.77	0.74	0.69
ICO	0.58	0.71	0.73	0.71	0.51	0.68	0.71	0.72	0.67	**0.76**	0.63
LMT	0.37	0.79	0.75	0.71	0.89	**0.97**	0.84	0.59	0.78	0.67	0.64
RT	0.51	0.76	0.8	0.69	0.89	**0.97**	0.8	0.58	0.7	0.7	0.73
RSS	0.47	0.75	0.78	0.74	0.75	**0.89**	0.8	0.71	0.75	0.71	0.66
PART	0.52	0.76	0.66	0.68	0.65	**0.85**	0.83	0.64	0.64	0.7	0.67
J48	0.57	0.8	0.79	0.78	0.85	**0.95**	0.84	0.69	0.72	0.66	0.66

Table A.29: Jedit4.0 ROC-AUC Values

Jedit4.0	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
NB	0.76	0.69	0.77	0.74	0.74	0.78	0.77	0.75	0.65	**0.79**	0.71
LR	0.78	0.74	0.79	0.73	0.79	0.82	0.81	0.75	0.62	**0.83**	0.7
SL	0.76	0.72	0.78	0.73	0.78	0.81	0.8	0.75	0.68	**0.82**	0.74
LB	0.78	0.75	0.83	0.79	0.84	0.84	0.84	0.76	0.64	**0.87**	0.72
MLP	0.79	0.75	0.78	0.72	**0.84**	0.84	0.84	0.69	0.67	0.83	0.69
IBk	0.72	0.76	0.78	0.66	0.91	**0.92**	0.89	0.68	0.71	0.79	0.65
KStar	0.75	0.83	0.81	0.65	0.96	**0.98**	0.95	0.72	0.73	0.85	0.68
ABM1	0.78	0.85	0.85	0.75	0.95	**0.97**	0.94	0.76	0.7	0.86	0.68
Bag	0.74	0.86	0.87	0.76	0.95	**0.97**	0.93	0.78	0.71	0.86	0.7
ICO	0.76	0.74	0.82	0.78	0.84	0.84	0.84	0.72	0.59	**0.87**	0.65
LMT	0.77	0.79	0.78	0.74	0.9	**0.9**	0.87	0.75	0.68	0.84	0.73
RT	0.66	0.71	0.73	0.66	0.87	**0.91**	0.82	0.69	0.61	0.74	0.64
RSS	0.77	0.82	0.86	0.79	0.94	**0.96**	0.88	0.76	0.7	0.85	0.73
PART	0.71	0.77	0.77	0.77	0.88	**0.88**	0.84	0.66	0.61	0.72	0.64

Jedit4.0	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
J48	0.67	0.76	0.75	0.75	**0.9**	0.87	0.87	0.7	0.61	0.76	0.6

Table A.30: Jedit4.0 Balance Values

Jedit4.0	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
NB	50.67	47.89	54.72	52.16	52.14	54.49	52.2	57.64	54.49	55.22	**57.94**
LR	57.26	66.14	69.43	65.39	65.84	**74.06**	72.02	64.5	55.61	70.43	63.8
SL	56.48	64.93	68.79	66.64	65.96	**73.16**	71.82	66.46	56.83	71.95	66.88
LB	60.06	68.47	74.13	75.4	73.07	**75.98**	75.28	70.88	59.6	74.33	66
MLP	58.23	68.43	69.84	66.85	73.48	77.14	**77.81**	63.31	62.19	77.7	63.08
IBk	67.65	75.4	78.04	62.64	90.11	**91.27**	87.72	68.12	69.06	80.21	68.22
KStar	57.81	74.6	75.93	65.47	**90.32**	89.76	85.65	62.42	66.84	73.14	62.74
ABM1	60.81	76.32	76.44	72.76	**90.77**	89.56	87.17	70.11	61.45	77.98	61.9
Bag	62.86	77.16	80.05	72.66	**89.82**	88	83.39	71.17	62.35	76.41	60.89
ICO	65.56	69.3	72.27	74.15	72.91	**75.5**	75.02	71.78	53.26	72.78	72.45
LMT	57.33	72.85	72.83	71.53	**88.29**	86.84	82.62	66.95	56.83	72.78	66.88
RT	62.14	71.09	72.34	63.78	86.56	**90.13**	81.84	69.14	58.04	72.8	64.4
RSS	57.26	73.67	76.25	72.52	89.54	**90.27**	78.93	72.98	52.49	70.45	65.87
PART	53.61	69.74	69.85	73.41	84.89	**86.63**	80.92	63.1	57.29	71.77	59.49
J48	62.24	75	72.82	75.95	**88.36**	82.51	82.3	68.12	58.72	72.17	57.08

Table A.31: Jedit4.0 G-Mean Values

Jedit4.0	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
NB	0.53	0.49	0.58	0.55	0.55	0.57	0.55	**0.6**	0.56	0.58	0.6
LR	0.61	0.66	0.7	0.66	0.67	**0.74**	0.72	0.65	0.57	0.72	0.64
SL	0.61	0.66	0.69	0.67	0.68	**0.73**	0.72	0.67	0.59	0.73	0.67
LB	0.64	0.69	0.74	0.76	0.74	**0.76**	0.75	0.71	0.6	0.76	0.66
MLP	0.62	0.69	0.7	0.67	0.74	0.77	**0.78**	0.63	0.63	0.78	0.63
IBk	0.7	0.76	0.78	0.63	0.9	**0.93**	0.89	0.68	0.69	0.81	0.68
KStar	0.61	0.76	0.76	0.65	0.9	**0.91**	0.87	0.64	0.67	0.75	0.63
ABM1	0.64	0.77	0.77	0.73	**0.91**	**0.91**	0.88	0.7	0.63	0.79	0.62
Bag	0.67	0.77	0.8	0.73	**0.9**	0.89	0.83	0.71	0.63	0.77	0.61
ICO	0.69	0.69	0.72	0.74	0.74	**0.76**	0.75	0.72	0.55	0.74	0.73
LMT	0.61	0.73	0.73	0.72	**0.89**	0.88	0.83	0.67	0.59	0.74	0.67
RT	0.64	0.71	0.73	0.64	0.87	**0.91**	0.82	0.69	0.59	0.74	0.64
RSS	0.61	0.74	0.76	0.73	0.9	**0.9**	0.79	0.74	0.54	0.73	0.66
PART	0.57	0.71	0.7	0.73	0.85	**0.88**	0.81	0.63	0.58	0.72	0.59

Jedit4.0	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
J48	0.65	0.75	0.73	0.76	**0.88**	0.83	0.83	0.68	0.59	0.73	0.57

Table A.32: Jedit4.0 Sensitivity Values

Jedit4.0	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
NB	0.31	0.27	0.37	0.33	0.33	0.36	0.33	0.41	0.37	0.37	**0.43**
LR	0.4	0.66	0.65	0.59	0.55	0.72	0.7	0.56	**0.77**	0.61	0.72
SL	0.39	0.76	0.68	0.63	0.55	0.71	0.78	0.61	**0.83**	0.64	0.71
LB	0.44	0.67	0.71	0.72	0.66	**0.75**	0.73	0.67	**0.75**	0.67	0.71
MLP	0.41	0.73	0.67	0.62	0.67	0.75	**0.79**	0.64	0.73	0.76	0.63
IBk	0.56	0.84	0.79	0.6	0.91	**0.98**	0.94	0.63	0.76	0.76	0.67
KStar	0.41	0.85	0.79	0.66	0.91	**0.98**	0.93	0.51	0.71	0.64	0.65
ABM1	0.45	0.83	0.79	0.71	0.95	**0.96**	0.91	0.65	0.81	0.72	0.72
Bag	0.48	0.81	0.81	0.73	0.91	**0.94**	0.85	0.68	0.79	0.71	0.69
ICO	0.52	0.7	0.69	0.7	0.65	0.74	0.72	0.65	**0.81**	0.64	0.68
LMT	0.4	0.75	0.72	0.66	0.92	**0.94**	0.86	0.6	0.83	0.64	0.71
RT	0.49	0.72	0.8	0.62	0.91	**0.95**	0.83	0.68	0.75	0.65	0.65
RSS	0.4	0.77	0.74	0.67	0.91	**0.92**	0.8	0.67	0.87	0.6	0.67
PART	0.35	0.83	0.63	0.72	0.88	**0.93**	0.8	0.59	0.68	0.67	0.59
J48	0.48	0.79	0.69	0.75	**0.90**	0.89	0.86	0.63	0.72	0.64	0.69

Table A.33: Log4j1.0 ROC-AUC Values

Log4j1.0	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
NB	0.83	0.8	0.83	0.79	0.83	0.86	0.85	0.82	0.89	**0.9**	0.85
LR	0.82	0.8	0.81	0.75	0.82	0.86	0.86	0.77	0.85	**0.88**	0.83
SL	0.85	0.81	0.82	0.76	0.83	0.88	0.87	0.81	0.89	**0.9**	0.87
LB	0.78	0.79	0.86	0.72	0.86	**0.88**	**0.88**	0.73	0.86	0.86	0.85
MLP	0.73	0.82	0.82	0.72	0.82	0.85	0.87	0.76	0.79	**0.89**	0.75
IBk	0.64	0.78	0.81	0.66	0.87	**0.92**	0.9	0.72	0.75	0.84	0.7
KStar	0.73	0.82	0.86	0.66	0.96	**0.97**	0.96	0.7	0.75	0.89	0.74
ABM1	0.71	0.81	0.84	0.69	0.93	**0.95**	0.87	0.71	0.77	0.86	0.76
Bag	0.77	0.82	0.86	0.69	0.91	**0.94**	0.87	0.76	0.8	0.92	0.85
ICO	0.76	0.79	0.85	0.71	0.85	**0.88**	0.87	0.7	0.81	0.85	0.83
LMT	0.85	0.81	0.79	0.76	0.87	**0.91**	0.82	0.81	0.89	0.91	0.87
RT	0.62	0.77	0.8	0.62	0.86	**0.9**	0.8	0.66	0.69	0.86	0.71
RSS	0.78	0.83	0.85	0.74	**0.91**	**0.91**	0.88	0.75	0.78	0.9	0.85
PART	0.73	0.78	0.77	0.63	0.87	**0.91**	0.84	0.65	0.77	0.81	0.75

Log4j1.0	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
J48	0.65	0.7	0.75	0.61	**0.93**	0.86	0.83	0.72	0.7	0.82	0.74

Table A.34: Log4j1.0 Balance Values

Log4j1.0	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
NB	66.61	63.46	70.26	61.4	67.2	74.08	72.48	70.42	73.25	**74.48**	72.11
LR	66.13	71.04	75.95	71.15	72.72	**80.22**	76.1	72.98	79.65	78.46	72.33
SL	64.4	70.51	75.72	70.5	71.36	82.04	76.14	76.77	**82.54**	80.86	74.44
LB	65.68	72.31	77.3	67.2	78.15	79.98	78.99	67.58	75.84	**81.54**	78.08
MLP	55.97	73.75	76.96	63.18	73.99	**84.7**	79.9	69.15	74.27	79.08	68.6
IBk	59.29	77.31	79.98	63.6	86.63	**90.62**	87.52	65.95	70.45	83.22	72.1
KStar	47.63	76.42	81.55	63.02	89.36	**91.98**	88.66	62.64	63.2	70.13	61.21
ABM1	59.29	73.75	77.44	64.4	88.91	**89.26**	80.95	70.28	72.02	72.82	73.86
Bag	68.17	74.89	77.94	66.17	**85.6**	84.63	80.51	70.28	70.45	78.46	80.02
ICO	59.99	70.68	77.94	68.62	75.25	**83.45**	80.37	62.58	69.38	82.08	76.82
LMT	64.4	71.87	74.71	70.5	84.38	**86.99**	73.69	76.77	82.54	80.54	74.44
RT	56.54	76.94	79.99	61.04	84.47	**86.61**	80.22	65.46	68.2	83.15	70.78
RSS	54.16	75.41	78.38	67.37	80.82	**84.91**	79.2	67	73.2	82.53	78.39
PART	58.12	71.7	76.42	61.89	**84.84**	84.63	80.52	61.39	74.27	77.2	67.69
J48	60.09	70.33	76.13	60.86	**89.36**	86.34	84.63	69.67	64.72	81.95	69.27

Table A.35: Log4j1.0 G-Mean Values

Log4j1.0	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
NB	0.71	0.67	0.74	0.66	0.71	**0.77**	0.76	0.74	0.75	**0.77**	0.75
LR	0.69	0.71	0.76	0.72	0.74	**0.8**	0.77	0.73	**0.8**	0.79	0.72
SL	0.69	0.71	0.76	0.71	0.73	0.82	0.77	0.78	**0.83**	**0.83**	0.75
LB	0.68	0.72	0.77	0.67	0.78	0.8	0.79	0.68	0.76	**0.82**	0.79
MLP	0.59	0.74	0.77	0.64	0.74	**0.85**	0.8	0.69	0.75	0.8	0.69
IBk	0.62	0.77	0.8	0.64	0.87	**0.92**	0.89	0.66	0.71	0.83	0.72
KStar	0.49	0.76	0.82	0.63	0.9	**0.93**	0.89	0.64	0.63	0.73	0.62
ABM1	0.62	0.74	0.77	0.64	0.9	**0.91**	0.81	0.7	0.72	0.73	0.74
Bag	0.71	0.75	0.78	0.66	**0.86**	0.85	0.81	0.7	0.71	0.79	0.8
ICO	0.63	0.71	0.78	0.69	0.75	**0.84**	0.8	0.63	0.7	0.83	0.77
LMT	0.69	0.72	0.75	0.71	0.84	**0.88**	0.74	0.78	0.83	0.82	0.75
RT	0.58	0.77	0.8	0.61	0.85	**0.89**	0.8	0.66	0.68	0.86	0.71
RSS	0.58	0.76	0.79	0.68	0.81	**0.85**	0.79	0.68	0.74	0.84	0.78
PART	0.62	0.72	0.76	0.62	**0.86**	0.85	0.81	0.62	0.75	0.78	0.68

Log4j1.0	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
J48	0.64	0.73	0.76	0.61	**0.9**	0.87	0.85	0.7	0.66	0.82	0.69

Table A.36: Log4j1.0 Sensitivity Values

Log4j1.0	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
NB	0.53	0.49	0.59	0.46	0.54	0.64	0.62	0.59	**0.65**	**0.65**	0.62
LR	0.53	0.66	0.72	0.65	0.64	0.78	0.71	0.71	**0.82**	0.74	0.71
SL	0.5	0.63	0.71	0.64	0.61	**0.79**	0.7	0.71	**0.79**	0.74	0.68
LB	0.53	0.74	0.76	0.68	0.76	**0.82**	0.8	0.65	0.79	0.76	0.74
MLP	0.38	0.72	0.73	0.55	0.69	0.82	**0.83**	0.68	0.82	0.74	0.62
IBk	0.44	0.81	0.82	0.64	0.91	**0.97**	0.95	0.62	0.79	0.81	0.68
KStar	0.26	0.78	0.82	0.67	0.91	**0.97**	0.9	0.5	0.59	0.59	0.5
ABM1	0.44	0.72	0.78	0.62	0.94	**0.97**	0.8	0.68	0.76	0.68	0.74
Bag	0.56	0.76	0.78	0.68	**0.89**	0.85	0.82	0.68	0.79	0.74	0.79
ICO	0.44	0.74	0.78	0.67	0.71	**0.82**	0.79	0.62	0.76	0.76	0.74
LMT	0.5	0.68	0.7	0.64	0.86	**0.93**	0.69	0.71	0.79	0.74	0.68
RT	0.41	0.78	0.8	0.62	0.91	**0.98**	0.78	0.68	0.74	0.76	0.71
RSS	0.35	0.82	0.74	0.62	0.77	**0.83**	0.77	0.59	0.79	0.76	0.79
PART	0.41	0.7	0.78	0.6	**0.93**	0.85	0.81	0.5	0.82	0.71	0.65
J48	0.44	0.89	0.73	0.59	0.91	**0.93**	0.85	0.65	0.79	0.79	0.68

Table A.37: Synapse1.1 ROC-AUC Values

Synapse1.1	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
NB	0.75	0.69	0.73	0.71	0.7	0.77	0.75	0.7	0.7	**0.78**	0.67
LR	0.74	0.69	0.73	0.7	0.71	**0.78**	0.76	0.73	0.67	**0.78**	0.67
SL	0.77	0.71	0.73	0.72	0.7	0.79	0.78	0.74	0.72	**0.82**	0.7
LB	0.69	0.75	0.76	0.67	0.75	0.79	**0.8**	0.71	0.67	**0.8**	0.68
MLP	0.77	0.79	0.77	0.68	0.82	**0.84**	**0.84**	0.79	0.65	0.82	0.65
IBk	0.7	0.77	0.75	0.63	**0.87**	**0.87**	0.86	0.67	0.64	0.86	0.68
KStar	0.76	0.79	0.84	0.62	**0.95**	**0.95**	0.92	0.73	0.67	0.84	0.79
ABM1	0.78	0.82	0.83	0.69	**0.95**	0.93	0.9	0.76	0.64	0.81	0.65
Bag	0.77	0.83	0.84	0.69	**0.94**	0.93	0.9	0.75	0.72	0.82	0.76
ICO	0.7	0.75	0.75	0.66	0.75	0.78	**0.8**	0.57	0.67	0.78	0.63
LMT	0.77	0.78	0.75	0.72	**0.87**	0.85	0.84	0.74	0.72	0.81	0.71
RT	0.66	0.73	0.75	0.6	**0.86**	0.84	0.81	0.63	0.6	0.72	0.59
RSS	0.75	0.82	0.8	0.7	**0.93**	0.9	0.84	0.72	0.68	0.79	0.68
PART	0.67	0.8	0.75	0.64	**0.87**	**0.87**	0.83	0.68	0.66	0.8	0.64

Synapse1.1	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
J48	0.66	0.73	0.75	0.63	**0.87**	**0.87**	0.82	0.6	0.62	0.69	0.66

Table A.38: Synapse1.1 Balance Values

Synapse1.1	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
NB	63.68	64.18	67.35	63.87	56.69	66.87	66.45	64.31	62.74	**71.88**	63.89
LR	61.93	66.5	69.53	66.25	65.21	73.1	72.06	69.29	61.32	**77.09**	65.07
SL	56.22	67.47	68.78	66.85	64.01	75.73	73.43	67.74	61.03	**77.64**	63.21
LB	53.22	67.37	71.47	64.87	65.53	**73.88**	73.74	61.55	61.85	72.59	62.77
MLP	61.99	71.35	71.46	63.46	76.99	**78.68**	78.65	77.36	62.43	74.32	60.33
IBk	65.25	75.81	73.4	59.87	**85.36**	85.28	83.01	66.19	61.03	85.28	69.91
KStar	61.39	73.48	76.42	63.93	**86.01**	85.77	84.33	67.74	63.45	72.51	67.16
ABM1	62.43	76.12	78.65	64.47	**88.66**	87.9	84.08	68.05	63.45	73.92	60.27
Bag	61.65	74.48	76	64.85	**87.11**	84.09	83.44	65.82	65.8	76.1	66.87
ICO	52.13	67.37	70.65	65.59	65.53	**74.22**	73.8	51.35	67.5	71.88	57.66
LMT	56.22	74.82	68.99	66.85	**82.8**	81.82	77.1	67.74	61.03	78.48	63.89
RT	64.51	72.62	74.51	58.83	83.2	**83.6**	81.19	62.48	58.53	71.5	58.46
RSS	51.59	71.65	73.82	69.01	**86.02**	81.16	79.1	68.44	55.05	68.82	67.43
PART	58.69	75.45	70.18	62.85	83.09	**84.19**	77.08	61.29	63.45	72.59	60.27
J48	58.89	70.69	70.6	62.34	**82.8**	82.02	76.75	58.98	59.62	66.85	63.57

Table A.39: Synapse1.1 G-Mean Values

Synapse1.1	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
NB	0.66	0.65	0.68	0.65	0.59	0.68	0.68	0.65	0.63	**0.73**	0.64
LR	0.66	0.67	0.7	0.66	0.66	0.73	0.72	0.69	0.62	**0.78**	0.65
SL	0.6	0.68	0.69	0.67	0.65	0.76	0.74	0.68	0.62	**0.79**	0.63
LB	0.55	0.68	0.72	0.65	0.66	**0.74**	**0.74**	0.62	0.63	**0.74**	0.63
MLP	0.66	0.72	0.72	0.64	0.77	**0.79**	**0.79**	0.77	0.63	0.75	0.6
IBk	0.66	0.76	0.74	0.6	0.86	**0.87**	0.84	0.66	0.62	0.85	0.7
KStar	0.64	0.74	0.77	0.64	**0.87**	**0.87**	0.85	0.68	0.64	0.73	0.67
ABM1	0.65	0.76	0.79	0.65	**0.9**	0.89	0.84	0.68	0.64	0.74	0.6
Bag	0.65	0.75	0.76	0.65	**0.88**	0.84	0.83	0.66	0.67	0.77	0.67
ICO	0.54	0.68	0.71	0.66	0.66	**0.74**	**0.74**	0.51	0.68	0.73	0.58
LMT	0.6	0.75	0.69	0.67	**0.84**	0.83	0.77	0.68	0.62	0.8	0.64
RT	0.65	0.73	0.75	0.59	**0.86**	0.84	0.81	0.63	0.59	0.72	0.59
RSS	0.55	0.72	0.74	0.69	**0.87**	0.81	0.79	0.69	0.57	0.71	0.68
PART	0.61	0.76	0.72	0.64	0.84	**0.85**	0.77	0.62	0.64	0.74	0.6

Synapse1.1	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
J48	0.61	0.71	0.71	0.63	**0.84**	0.83	0.77	0.59	0.6	0.67	0.64

Table A.40: Synapse1.1 Sensitivity Values

Synapse1.1	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
NB	0.5	0.57	0.6	0.53	0.41	0.57	0.57	0.55	0.63	**0.65**	0.58
LR	0.47	0.61	0.65	0.61	0.57	0.71	0.68	0.65	**0.75**	0.7	0.62
SL	0.38	0.61	0.64	0.61	0.55	0.72	0.68	0.63	**0.8**	0.7	0.58
LB	0.35	0.63	0.69	0.63	0.6	0.76	0.72	0.57	**0.77**	0.65	0.65
MLP	0.47	0.77	0.64	0.54	**0.8**	**0.8**	**0.8**	0.75	0.73	0.68	0.6
IBk	0.55	0.83	0.78	0.6	0.92	**0.93**	0.89	0.67	0.8	0.85	0.75
KStar	0.47	0.8	0.84	0.6	**0.93**	0.92	0.89	0.63	0.72	0.67	0.62
ABM1	0.48	0.76	0.8	0.59	**0.95**	0.94	0.84	0.65	0.72	0.68	0.58
Bag	0.47	0.77	0.74	0.59	**0.94**	0.88	0.84	0.6	0.78	0.7	0.65
ICO	0.33	0.63	0.67	0.65	0.6	0.75	0.72	0.62	**0.78**	0.65	0.68
LMT	0.38	0.8	0.67	0.61	**0.93**	0.89	0.74	0.63	0.8	0.72	0.58
RT	0.55	0.74	0.74	0.57	**0.95**	0.9	0.83	0.58	0.72	0.65	0.63
RSS	0.32	0.72	0.69	0.65	**0.91**	0.8	0.77	0.62	0.87	0.58	0.63
PART	0.43	0.81	0.6	0.51	**0.92**	0.91	0.78	0.52	0.72	0.65	0.58
J48	0.43	0.73	0.64	0.51	**0.93**	0.89	0.77	0.57	0.75	0.6	0.57

Table A.41: Synapse1.2 ROC-AUC Values

Synapse1.2	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
NB	0.78	0.73	0.75	0.75	0.83	0.76	0.77	0.76	0.78	**0.86**	0.8
LR	0.8	0.75	0.78	0.77	0.85	0.79	0.79	0.75	0.79	**0.87**	0.8
SL	0.79	0.75	0.76	0.77	0.85	0.78	0.78	0.76	0.77	**0.88**	0.8
LB	0.78	0.75	0.77	0.79	0.83	0.83	0.83	0.77	0.8	**0.9**	0.8
MLP	0.76	0.74	0.74	0.74	**0.84**	0.79	0.8	0.75	0.76	**0.84**	0.82
IBk	0.67	0.76	0.73	0.65	**0.85**	**0.85**	0.84	0.65	0.72	0.79	0.74
KStar	0.77	0.79	0.81	0.67	0.93	**0.94**	0.93	0.76	0.79	0.9	0.79
ABM1	0.75	0.76	0.81	0.72	**0.93**	**0.93**	0.88	0.78	0.79	0.91	0.83
Bag	0.8	0.8	0.83	0.75	**0.91**	0.9	0.88	0.79	0.78	**0.91**	0.82
ICO	0.76	0.74	0.77	0.77	0.84	0.82	0.82	0.75	0.76	**0.89**	0.78
LMT	0.82	0.71	0.74	0.76	0.82	**0.88**	0.82	0.77	0.7	**0.88**	0.8
RT	0.67	0.7	0.74	0.63	**0.83**	0.82	0.76	0.64	0.64	0.77	0.68
RSS	0.76	0.78	0.82	0.78	**0.9**	0.87	0.84	0.76	0.77	0.86	0.8
PART	0.73	0.7	0.73	0.71	0.83	0.84	0.8	0.76	0.69	**0.85**	0.68

Synapse1.2	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
J48	0.74	0.68	0.79	0.67	**0.85**	0.84	0.76	0.67	0.74	0.82	0.71

Table A.42: Synapse1.2 Balance Values

Synapse1.2	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
NB	67.44	60.63	64.09	65.33	69.23	66.41	63.68	65.92	71.3	**75.5**	73.74
LR	62.4	66.77	68.54	68.72	75.24	69.7	71.07	66.29	67.28	**81.33**	74.37
SL	59.83	66.95	67.35	69.51	75.64	69.89	69.11	64.76	62.09	**83.21**	71.99
LB	67.12	68.22	70.08	71.75	76.97	73.72	72.8	71.49	60.56	**80.82**	62.07
MLP	70.13	69.72	71.48	71.73	81.34	75.53	74.53	75.05	70.25	**81.54**	73.01
IBk	64.95	73.29	70.68	61.93	**84.02**	83.27	82.7	65.49	66.27	79.43	73.02
KStar	68.76	73.38	73.42	66.41	**84.32**	84.21	83.46	71.07	64.03	83.36	72.03
ABM1	66.83	69.89	71.91	67.3	**85.18**	82.77	80.9	71.16	56.44	84.39	70.82
Bag	69.14	73.71	74.53	69.72	83.6	82.28	80.94	72.27	56.04	**83.85**	62.39
ICO	68.49	66.58	70.1	70.99	76.85	74.01	72.34	71.96	45.23	**78.34**	61.81
LMT	64.57	67.58	67.71	70.66	77.87	**84.19**	75.53	68.33	57.76	83.21	71.99
RT	64.78	69.8	74.32	61.85	**81.71**	81.19	75.54	63.9	59.68	76.45	67.28
RSS	62.36	71.81	73.95	71.99	**83.6**	81.26	78.56	71.75	43.22	79.66	62.54
PART	67.56	62.97	69.28	70.09	79.78	76.48	76.11	73.68	72.35	**83.96**	65.92
J48	69.04	68.37	76.05	66.72	**81.96**	80.84	74.01	66.83	72.88	78.37	67.74

Table A.43: Synapse1.2 G-Mean Values

Synapse1.2	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
NB	0.7	0.62	0.66	0.67	0.71	0.68	0.65	0.68	0.71	**0.77**	0.75
LR	0.66	0.68	0.69	0.7	0.76	0.7	0.72	0.68	0.69	**0.82**	0.75
SL	0.63	0.68	0.68	0.7	0.77	0.71	0.7	0.66	0.64	**0.83**	0.73
LB	0.69	0.68	0.7	0.72	0.77	0.74	0.73	0.72	0.63	**0.81**	0.64
MLP	0.71	0.7	0.72	0.72	0.82	0.76	0.75	0.75	0.71	**0.82**	0.74
IBk	0.66	0.74	0.71	0.62	**0.84**	**0.84**	**0.84**	0.66	0.67	0.79	0.73
KStar	0.71	0.74	0.73	0.66	**0.85**	**0.85**	0.84	0.71	0.66	0.83	0.72
ABM1	0.68	0.7	0.72	0.67	**0.86**	0.83	0.81	0.71	0.59	0.84	0.71
Bag	0.71	0.74	0.75	0.7	**0.84**	0.83	0.81	0.72	0.58	**0.84**	0.64
ICO	0.7	0.67	0.7	0.71	0.77	0.74	0.73	0.72	0.46	**0.78**	0.64
LMT	0.67	0.68	0.68	0.71	0.78	**0.85**	0.76	0.69	0.59	0.83	0.73
RT	0.66	0.7	0.74	0.62	**0.83**	0.82	0.76	0.64	0.61	0.77	0.68
RSS	0.65	0.72	0.74	0.72	**0.84**	0.81	0.79	0.72	0.43	0.8	0.65
PART	0.68	0.64	0.7	0.7	0.8	0.77	0.77	0.74	0.73	**0.84**	0.67

Synapse1.2	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
J48	0.7	0.69	0.76	0.67	**0.82**	0.81	0.74	0.67	0.73	0.79	0.68

Table A.44: Synapse1.2 Sensitivity Values

Synapse1.2	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
NB	0.56	0.47	0.52	0.54	0.59	0.56	0.52	0.53	**0.69**	0.67	0.66
LR	0.48	0.58	0.6	0.6	0.69	0.62	0.64	0.56	**0.83**	0.78	0.81
SL	0.44	0.58	0.58	0.61	0.68	0.63	0.61	0.53	**0.85**	0.8	0.83
LB	0.57	0.67	0.7	0.7	0.74	0.75	0.73	0.66	**0.87**	0.81	0.84
MLP	0.63	0.68	0.7	0.66	0.79	0.76	0.78	0.71	0.8	0.77	**0.81**
IBk	0.55	0.83	0.73	0.57	0.87	0.89	**0.9**	0.63	0.79	0.81	0.78
KStar	0.58	0.81	0.75	0.65	0.87	**0.9**	**0.9**	0.66	0.85	0.83	0.76
ABM1	0.58	0.75	0.73	0.66	**0.91**	0.87	0.83	0.7	0.86	0.86	0.79
Bag	0.58	0.79	0.78	0.68	**0.87**	0.86	0.83	0.71	0.84	0.85	0.79
ICO	0.58	0.64	0.71	0.68	0.73	0.74	0.68	0.69	**0.93**	0.79	0.83
LMT	0.51	0.66	0.61	0.64	0.76	**0.91**	0.77	0.6	0.83	0.8	0.83
RT	0.55	0.7	0.76	0.6	**0.89**	0.88	0.76	0.62	0.83	0.81	0.77
RSS	0.49	0.75	0.73	0.68	0.87	0.83	0.79	0.67	**0.93**	0.78	0.86
PART	0.59	0.77	0.76	0.7	0.8	**0.84**	**0.84**	0.69	0.77	**0.84**	0.77
J48	0.6	0.73	0.73	0.61	0.83	**0.84**	0.76	0.6	0.74	0.76	0.74

Table A.45: Log4j1.1 ROC-AUC Values

Log4j1.1	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
NB	0.86	0.78	0.85	0.84	0.81	0.86	0.85	0.84	0.85	**0.89**	0.86
LR	0.84	0.76	0.83	0.82	0.79	0.84	0.84	0.81	0.77	**0.88**	0.8
SL	0.83	0.73	0.81	0.79	0.76	0.8	0.84	0.81	0.75	**0.87**	0.79
LB	0.83	0.84	0.89	0.83	**0.92**	0.89	0.84	0.8	0.86	0.86	0.8
MLP	0.82	0.77	0.86	0.8	0.8	0.87	**0.88**	0.8	0.78	0.87	0.78
IBk	0.75	0.73	0.78	0.7	**0.87**	0.86	0.85	0.71	0.7	**0.87**	0.71
KStar	0.82	0.8	0.89	0.77	**0.96**	0.95	0.93	0.76	0.77	0.95	0.85
ABM1	0.76	0.78	0.84	0.79	**0.97**	0.93	0.86	0.71	0.71	0.88	0.75
Bag	0.82	0.82	0.87	0.84	**0.93**	0.91	0.89	0.79	0.77	0.9	0.77
ICO	0.81	0.83	0.87	0.82	**0.91**	0.87	0.82	0.81	0.83	0.87	0.76
LMT	0.81	0.78	0.79	0.77	**0.92**	0.84	0.83	0.81	0.76	0.85	0.79
RT	0.68	0.73	0.75	0.71	**0.92**	0.88	0.79	0.69	0.8	0.79	0.69
RSS	0.85	0.8	0.87	0.83	**0.92**	0.9	0.87	0.83	0.85	0.92	0.8
PART	0.8	0.78	0.84	0.74	**0.92**	0.85	0.82	0.73	0.82	0.82	0.72

Log4j1.1	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
J48	0.72	0.76	0.81	0.75	**0.91**	0.84	0.82	0.7	0.84	0.82	0.73

Table A.46: Log4j1.1 Balance Values

Log4j1.1	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
NB	76.55	61.77	76.7	72.91	67.92	72.91	76.7	75.42	78.57	**82.11**	76.91
LR	74.46	68.81	72.77	72.72	64.68	73.36	76.39	71.41	61.79	**81.81**	64.5
SL	74.84	69.56	73.65	74.09	68.37	71.83	78.11	73.94	56.86	**83.38**	76.29
LB	73.63	73.76	**80.46**	77.24	79.59	79.38	73.92	71.05	65.86	78.35	67.73
MLP	75.07	69.94	77.41	71.38	72.4	76.93	**80.89**	72.24	56.86	78.35	67.73
IBk	74.33	69.09	77.07	68.82	83.14	82.21	80.92	72.69	68.7	**84.87**	72.27
KStar	69.03	67.7	80.86	72.15	85.79	**87.84**	85.68	68.87	59.56	82.49	78.01
ABM1	69.92	67.72	78.96	73.36	**88.9**	82.73	75.16	66.15	58.8	80.86	65.49
Bag	66.33	75.82	76.54	73.07	**84.93**	83.69	80.86	72.09	64.13	76.03	67.73
ICO	72.11	72.93	77.23	78.4	**79.7**	78.71	69.94	72.53	62.13	79.31	67.73
LMT	74.67	70.74	73.19	72.21	**85.34**	76.99	77.56	73.94	60.85	81.81	76.29
RT	66.06	70.54	73.76	68.09	**88.76**	84.04	76.83	68.45	79.79	75.04	67.82
RSS	69.36	71.12	73.48	72.72	81.74	81.21	75.58	70.47	54.2	**82.64**	71.85
PART	70.73	72.24	78.38	71.67	**86.39**	83.63	73.36	70	61.79	75.83	68.66
J48	70.52	69.5	77.73	70.02	**84.41**	82.22	72.77	68.78	61.79	76.03	71.67

Table A.47: Log4j1.1 G-Mean Values

Log4j1.1	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
NB	0.79	0.66	0.79	0.76	0.72	0.76	0.79	0.77	0.81	**0.84**	0.81
LR	0.77	0.69	0.73	0.73	0.65	0.74	0.76	0.72	0.65	**0.82**	0.65
SL	0.78	0.7	0.75	0.75	0.68	0.72	0.78	0.75	0.59	**0.84**	0.76
LB	0.75	0.74	**0.81**	0.78	0.8	0.79	0.74	0.71	0.69	0.78	0.68
MLP	0.79	0.7	0.79	0.72	0.73	0.8	**0.81**	0.73	0.59	0.78	0.68
IBk	0.75	0.69	0.77	0.69	**0.86**	0.84	0.82	0.73	0.7	0.85	0.72
KStar	0.73	0.68	0.81	0.72	0.87	**0.88**	0.86	0.69	0.61	0.85	0.79
ABM1	0.71	0.68	0.79	0.74	**0.91**	0.84	0.75	0.66	0.6	0.81	0.66
Bag	0.69	0.76	0.77	0.73	**0.85**	0.84	0.81	0.72	0.66	0.77	0.68
ICO	0.74	0.73	0.77	0.79	**0.8**	0.79	0.7	0.73	0.66	0.8	0.68
LMT	0.78	0.71	0.74	0.73	**0.86**	0.77	0.78	0.75	0.63	0.82	0.76
RT	0.67	0.71	0.74	0.68	**0.91**	0.85	0.77	0.69	0.8	0.75	0.68
RSS	0.74	0.71	0.76	0.73	0.82	0.81	0.76	0.73	0.59	**0.83**	0.72
PART	0.74	0.72	0.8	0.72	**0.88**	0.84	0.74	0.72	0.65	0.76	0.69

Log4j1.1	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
J48	0.73	0.7	0.78	0.71	**0.85**	0.82	0.73	0.7	0.65	0.77	0.72

Table A.48: Log4j1.1 Sensitivity Values

Log4j1.1	NS	ADSYN	SMT	SLSMT	SPD	ROS	AHC	RUS	CNNTL	NCL	OSS
NB	0.68	0.46	0.68	0.62	0.55	0.62	0.68	0.68	0.7	**0.76**	0.68
LR	0.65	0.7	0.7	0.67	0.64	0.7	0.74	0.65	**0.89**	0.78	0.73
SL	0.65	0.62	0.67	0.68	0.65	0.65	0.74	0.68	**0.84**	0.81	0.76
LB	0.65	0.74	0.75	0.72	0.77	0.78	0.7	0.68	**0.89**	0.76	0.7
MLP	0.65	0.67	0.7	0.64	0.67	0.68	0.77	0.65	**0.84**	0.76	0.7
IBk	0.7	0.75	0.81	0.77	**0.96**	0.91	0.9	0.76	0.84	0.84	0.76
KStar	0.57	0.74	0.81	0.7	**0.93**	0.9	0.84	0.62	0.78	0.76	0.73
ABM1	0.62	0.64	0.75	0.7	**0.99**	0.9	0.77	0.65	0.76	0.78	0.76
Bag	0.54	0.78	0.75	0.68	**0.86**	0.84	0.81	0.68	0.81	0.7	0.7
ICO	0.62	0.71	0.8	0.75	0.78	0.78	0.67	0.7	**0.92**	0.76	0.7
LMT	0.65	0.68	0.67	0.67	**0.88**	0.74	0.74	0.68	0.84	0.78	0.76
RT	0.57	0.67	0.74	0.7	**0.97**	0.91	0.83	0.65	0.84	0.78	0.76
RSS	0.57	0.67	0.64	0.67	0.84	0.78	0.72	0.59	**0.97**	0.78	0.68
PART	0.59	0.75	0.71	0.67	**0.96**	0.86	0.7	0.59	0.89	0.73	0.62
J48	0.59	0.7	0.72	0.62	0.88	0.84	0.7	0.59	**0.89**	0.7	0.65

Bibliography

[1] R. Malhotra, *Empirical research in software engineering: concepts, analysis, and applications*. CRC Press, 2016.

[2] V. R. Basili, L. C. Briand, and W. L. Melo, "A validation of object-oriented design metrics as quality indicators," *IEEE Transactions on software engineering*, vol. 22, no. 10, pp. 751–761, 1996.

[3] 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, p. 3, 2010.

[4] R. Malhotra, "An extensive analysis of search-based techniques for predicting defective classes," *Computers & Electrical Engineering*, vol. 71, pp. 611–626, 2018.

[5] 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.

[6] B. Kitchenham and S. L. Pfleeger, "Software quality: the elusive target [special issues section]," *IEEE software*, vol. 13, no. 1, pp. 12–21, 1996.

[7] Y. Singh and R. Malhotra, *Object-oriented software engineering*. PHI Learning Pvt. Ltd., 2012.

[8] T. Dyba, B. A. Kitchenham, and M. Jorgensen, "Evidence-based software engineering for practitioners," *IEEE software*, vol. 22, no. 1, pp. 58–65, 2005.

[9] S. L. Pfleeger, "Understanding and improving technology transfer in software engineering," *Journal of Systems and Software*, vol. 47, no. 2-3, pp. 111–124, 1999.

[10] A. E. Hassan and T. Xie, "Software intelligence: the future of mining software engineering data," in *Proceedings of the FSE/SDP workshop on Future of software engineering research*, 2010, pp. 161–166.

[11] S. Planning, "The economic impacts of inadequate infrastructure for software testing," *National Institute of Standards and Technology*, 2002.

[12] C. Lewis, Z. Lin, C. Sadowski, X. Zhu, R. Ou, and E. J. Whitehead, "Does bug prediction support human developers? findings from a google case study," in *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 2013, pp. 372–381.

[13] B. Caglayan, B. Turhan, A. Bener, M. Habayeb, A. Miransky, and E. Cialini, "Merits of organizational metrics in defect prediction: an industrial replication," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 2. IEEE, 2015, pp. 89–98.

[14] T. Zimmermann and N. Nagappan, "Predicting defects with program dependencies," in *2009 3rd international symposium on empirical software engineering and measurement*. IEEE, 2009, pp. 435–438.

[15] R. B. Grady, "Software failure analysis for high-return process improvement decisions," *Hewlett Packard Journal*, vol. 47, pp. 15–24, 1996.

[16] L. Chen, B. Fang, Z. Shang, and Y. Tang, "Tackling class overlap and imbalance problems in software defect prediction," *Software Quality Journal*, vol. 26, no. 1, pp. 97–125, 2018.

[17] H. D. Rombach, "Design measurement: Some lessons learned," *IEEE Software*, vol. 7, no. 2, pp. 17–25, 1990.

[18] B. Henderson-Sellers, *Object-oriented metrics: measures of complexity*. Prentice-Hall, Inc., 1995.

[19] M. A. Mazurowski, P. A. Habas, J. M. Zurada, J. Y. Lo, J. A. Baker, and G. D. Tourassi, "Training neural network classifiers for medical decision making: The effects of imbalanced datasets on classification performance," *Neural networks*, vol. 21, no. 2-3, pp. 427–436, 2008.

[20] M. Khalilia, S. Chakraborty, and M. Popescu, "Predicting disease risks from highly imbalanced data using random forest," *BMC medical informatics and decision making*, vol. 11, no. 1, p. 51, 2011.

[21] C. Phua, D. Alahakoon, and V. Lee, "Minority report in fraud detection: classification of skewed data," *Acm sigkdd explorations newsletter*, vol. 6, no. 1, pp. 50–59, 2004.

[22] M. Vasu and V. Ravi, "A hybrid under-sampling approach for mining unbalanced datasets: applications to banking and insurance," *International Journal of Data Mining, Modelling and Management*, vol. 3, no. 1, pp. 75–105, 2011.

[23] A. C. Bahnsen, A. Stojanovic, D. Aouada, and B. Ottersten, "Cost sensitive credit card fraud detection using bayes minimum risk," in *2013 12th international conference on machine learning and applications*, vol. 1. IEEE, 2013, pp. 333–338.

[24] A. Moreo, A. Esuli, and F. Sebastiani, "Distributional random oversampling for imbalanced text classification," in *Proceedings of the 39th International ACM SIGIR conference on Research and Development in Information Retrieval*, 2016, pp. 805–808.

[25] Z. Zheng, X. Wu, and R. Srihari, "Feature selection for text categorization on imbalanced data," *ACM Sigkdd Explorations Newsletter*, vol. 6, no. 1, pp. 80–89, 2004.

[26] P. C. Lane, D. Clarke, and P. Hender, "On developing robust models for favourability analysis: Model choice, feature sets and imbalanced data," *Decision Support Systems*, vol. 53, no. 4, pp. 712–718, 2012.

[27] J. Burez and D. Van den Poel, "Handling class imbalance in customer churn prediction," *Expert Systems with Applications*, vol. 36, no. 3, pp. 4626–4636, 2009.

[28] S. Kotsiantis, D. Kanellopoulos, P. Pintelas *et al.*, "Handling imbalanced datasets: A review," *GESTS International Transactions on Computer Science and Engineering*, vol. 30, no. 1, pp. 25–36, 2006.

[29] V. S. Spelmen and R. Porkodi, "A review on handling imbalanced data," in *2018 International Conference on Current Trends towards Converging Technologies (ICCTCT)*. IEEE, 2018, pp. 1–11.

[30] M. Galar, A. Fernandez, E. Barrenechea, H. Bustince, and F. Herrera, "A review on ensembles for the class imbalance problem: bagging-, boosting-, and hybrid-based approaches," *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 42, no. 4, pp. 463–484, 2011.

[31] C. Seiffert, T. M. Khoshgoftaar, and J. Van Hulse, "Improving software-quality predictions with data sampling and boosting," *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, vol. 39, no. 6, pp. 1283–1294, 2009.

[32] N. V. Chawla, N. Japkowicz, and A. Kotcz, "Special issue on learning from imbalanced data sets," *ACM SIGKDD explorations newsletter*, vol. 6, no. 1, pp. 1–6, 2004.

[33] 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.

[34] F. Akiyama, "An example of software system debugging." in *IFIP Congress (1)*, vol. 71, 1971, pp. 353–359.

[35] 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.

[36] 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.

[37] 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.

[38] M. Lorenz and J. Kidd, *Object-oriented software metrics: a practical guide*. Prentice-Hall, Inc., 1994.

[39] L. C. Briand, C. Bunse, and J. W. Daly, "A controlled experiment for evaluating quality guidelines on the maintainability of object-oriented designs," *IEEE Transactions on Software Engineering*, vol. 27, no. 6, pp. 513–530, 2001.

[40] R. C. Martin, *Agile software development: principles, patterns, and practices*. Prentice Hall, 2002.

[41] M.-H. Tang, M.-H. Kao, and M.-H. Chen, "An empirical study on object-oriented metrics," in *Proceedings sixth international software metrics symposium (Cat. No. PR00403)*. IEEE, 1999, pp. 242–249.

[42] B. Ghotra, S. McIntosh, and A. E. Hassan, "A large-scale study of the impact of feature selection techniques on defect classification models," in *2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*. IEEE, 2017, pp. 146–157.

[43] M. Kondo, C.-P. Bezemer, Y. Kamei, A. E. Hassan, and O. Mizuno, "The impact of feature reduction techniques on defect prediction models," *Empirical Software Engineering*, vol. 24, no. 4, pp. 1925–1963, 2019.

[44] S. Liu, X. Chen, W. Liu, J. Chen, Q. Gu, and D. Chen, "Fecar: A feature selection framework for software defect prediction," in *2014 IEEE 38th Annual Computer Software and Applications Conference*. IEEE, 2014, pp. 426–435.

[45] C. Ni, X. Chen, F. Wu, Y. Shen, and Q. Gu, "An empirical study on pareto based multi-objective feature selection for software defect prediction," *Journal of Systems and Software*, vol. 152, pp. 215–238, 2019.

[46] R. Malhotra, "An empirical framework for defect prediction using machine learning techniques with android software," *Applied Soft Computing*, vol. 49, pp. 1034–1050, 2016.

[47] T. Zhou, X. Sun, X. Xia, B. Li, and X. Chen, "Improving defect prediction with deep forest," *Information and Software Technology*, vol. 114, pp. 204–216, 2019.

[48] Z. Xu, J. Liu, Z. Yang, G. An, and X. Jia, "The impact of feature selection on defect prediction performance: An empirical comparison," in *2016 IEEE 27th Interna-*

tional Symposium on Software Reliability Engineering (ISSRE). IEEE, 2016, pp. 309–320.

[49] K. Muthukumaran, A. Rallapalli, and N. B. Murthy, "Impact of feature selection techniques on bug prediction models," in *Proceedings of the 8th India Software Engineering Conference*, 2015, pp. 120–129.

[50] 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.

[51] R. Malhotra, "A systematic review of machine learning techniques for software fault prediction," *Applied Soft Computing*, vol. 27, pp. 504–518, 2015.

[52] A. O. Balogun, S. Basri, S. J. Abdulkadir, and A. S. Hashim, "Performance analysis of feature selection methods in software defect prediction: a search method approach," *Applied Sciences*, vol. 9, no. 13, p. 2764, 2019.

[53] A. O. Balogun, S. Basri, S. Mahamad, S. J. Abdulkadir, M. A. Almomani, V. E. Adeyemo, Q. Al-Tashi, H. A. Mojeed, A. A. Imam, and A. O. Bajeh, "Impact of feature selection methods on the predictive performance of software defect prediction models: An extensive empirical study," *Symmetry*, vol. 12, no. 7, p. 1147, 2020.

[54] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, "A systematic literature review on fault prediction performance in software engineering," *IEEE Transactions on Software Engineering*, vol. 38, no. 6, pp. 1276–1304, 2011.

[55] R. Malhotra and Y. Singh, "On the applicability of machine learning techniques for object oriented software fault prediction," *Software Engineering: An International Journal*, vol. 1, no. 1, pp. 24–37, 2011.

[56] A. Iqbal, S. Aftab, U. Ali, Z. Nawaz, L. Sana, M. Ahmad, and A. Husen, "Performance analysis of machine learning techniques on software defect prediction using nasa datasets," *Int. J. Adv. Comput. Sci. Appl*, vol. 10, no. 5, 2019.

[57] D. Bowes, T. Hall, and J. Petrić, "Software defect prediction: do different classifiers find the same defects?" *Software Quality Journal*, vol. 26, no. 2, pp. 525–552, 2018.

[58] 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.

[59] M. Harman, "Why the virtual nature of software makes it ideal for search based optimization," in *International Conference on Fundamental Approaches to Software Engineering*. Springer, 2010, pp. 1–12.

[60] Z. Li, X.-Y. Jing, and X. Zhu, "Progress on approaches to software defect prediction," *IET Software*, vol. 12, no. 3, pp. 161–175, 2018.

[61] R. Reena and R. T. Selvi, "A survey of genetic feature selection for software defect prediction," *i-Manager's Journal on Software Engineering*, vol. 10, no. 3, p. 20, 2016.

[62] S. Hosseini, B. Turhan, and M. Mäntylä, "Search based training data selection for cross project defect prediction," in *Proceedings of the The 12th International Conference on Predictive Models and Data Analytics in Software Engineering*, 2016, pp. 1–10.

[63] C. Xiang, L. Lingjiao, J. Ren, and W. Shixin, "Sbfs: search based feature selection framework for software defect prediction," *Application Research of Computers*, no. 4, p. 35, 2017.

[64] Ö. F. Arar and K. Ayan, "Software defect prediction using cost-sensitive neural network," *Applied Soft Computing*, vol. 33, pp. 263–277, 2015.

[65] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, "Automated parameter optimization of classification techniques for defect prediction models," in *Proceedings of the 38th International Conference on Software Engineering*, 2016, pp. 321–332.

[66] D. Ryu and J. Baik, "Effective harmony search-based optimization of cost-sensitive boosting for improving the performance of cross-project defect prediction," *KIPS Transactions on Software and Data Engineering*, vol. 7, no. 3, pp. 77–90, 2018.

[67] X. Cai, Y. Niu, S. Geng, J. Zhang, Z. Cui, J. Li, and J. Chen, "An under-sampled software defect prediction method based on hybrid multi-objective cuckoo search," *Concurrency and Computation: Practice and Experience*, vol. 32, no. 5, p. e5478, 2020.

[68] D. Wu, J. Zhang, S. Geng, X. Cai, and G. Zhang, "A multi-objective bat algorithm for software defect prediction," in *International Conference on Bio-Inspired Computing: Theories and Applications*. Springer, 2019, pp. 269–283.

[69] 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.

[70] Y. Singh, A. Kaur, and R. Malhotra, "Prediction of software quality model using gene expression programming," in *International Conference on Product-Focused Software Process Improvement*. Springer, 2009, pp. 43–58.

[71] 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.

[72] S. Chatterjee, S. Nigam, and A. Roy, "Software fault prediction using neuro-fuzzy network and evolutionary learning approach," *Neural Computing and Applications*, vol. 28, no. 1, pp. 1221–1231, 2017.

[73] C. Manjula and L. Florence, "Deep neural network based hybrid approach for software defect prediction using software metrics," *Cluster Computing*, vol. 22, no. 4, pp. 9847–9863, 2019.

[74] 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.

[75] M. Tan, L. Tan, S. Dara, and C. Mayeux, "Online defect prediction for imbalanced data," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 2. IEEE, 2015, pp. 99–108.

[76] L. Pelayo and S. Dick, "Applying novel resampling strategies to software defect prediction," in *NAFIPS 2007-2007 Annual meeting of the North American fuzzy information processing society*. IEEE, 2007, pp. 69–72.

[77] Y. Liu, A. An, and X. Huang, "Boosting prediction accuracy on imbalanced datasets with svm ensembles," in *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 2006, pp. 107–118.

[78] Y. Kamei, A. Monden, S. Matsumoto, T. Kakimoto, and K.-i. Matsumoto, "The effects of over and under sampling on fault-prone module detection," in *First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)*. IEEE, 2007, pp. 196–204.

[79] T. M. Khoshgoftaar and K. Gao, "Feature selection with imbalanced data for software defect prediction," in *2009 International Conference on Machine Learning and Applications*. IEEE, 2009, pp. 235–240.

[80] J. Riquelme, R. Ruiz, D. Rodríguez, and J. Moreno, "Finding defective modules from highly unbalanced datasets," *Actas de los Talleres de las Jornadas de Ingeniería del Software y Bases de Datos*, vol. 2, no. 1, pp. 67–74, 2008.

[81] L. Pelayo and S. Dick, "Evaluating stratification alternatives to improve software defect prediction," *IEEE transactions on reliability*, vol. 61, no. 2, pp. 516–525, 2012.

[82] C. Seiffert, T. M. Khoshgoftaar, J. Van Hulse, and A. Folleco, "An empirical study of the classification performance of learners on imbalanced and noisy software quality data," *Information Sciences*, vol. 259, pp. 571–595, 2014.

[83] D. Rodriguez, I. Herraiz, R. Harrison, J. Dolado, and J. C. Riquelme, "Preliminary comparison of techniques for dealing with imbalance in software defect prediction," in *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, 2014, pp. 1–10.

[84] R. Shatnawi, "Improving software fault-prediction for imbalanced data," in *2012 International Conference on Innovations in Information Technology (IIT)*. IEEE, 2012, pp. 54–59.

[85] S. Wang and X. Yao, "Using class imbalance learning for software defect prediction," *IEEE Transactions on Reliability*, vol. 62, no. 2, pp. 434–443, 2013.

[86] W. Jindaluang, V. Chouvatut, and S. Kantabutra, "Under-sampling by algorithm with performance guaranteed for class-imbalance problem," in *2014 International Computer Science and Engineering Conference (ICSEC)*. IEEE, 2014, pp. 215–221.

[87] P. Lingden, A. Alsadoon, P. Prasad, O. H. Alsadoon, R. S. Ali, and V. T. Q. Nguyen, "A novel modified undersampling (mus) technique for software defect prediction," *Computational Intelligence*, vol. 35, no. 4, pp. 1003–1020, 2019.

[88] K. K. Bejjanki, J. Gyani, and N. Gugulothu, "Class imbalance reduction (cir): A novel approach to software defect prediction in the presence of class imbalance," *Symmetry*, vol. 12, no. 3, p. 407, 2020.

[89] R. Malhotra and S. Kamal, "An empirical study to investigate oversampling methods for improving software defect prediction using imbalanced data," *Neurocomputing*, vol. 343, pp. 120–140, 2019.

[90] M. J. Siers and M. Z. Islam, "Software defect prediction using a cost sensitive decision forest and voting, and a potential solution to the class imbalance problem," *Information Systems*, vol. 51, pp. 62–71, 2015.

[91] ——, "Addressing class imbalance and cost sensitivity in software defect prediction by combining domain costs and balancing costs," in *International Conference on Advanced Data Mining and Applications*. Springer, 2016, pp. 156–171.

[92] Y. Peng, G. Kou, G. Wang, H. Wang, and F. I. Ko, "Empirical evaluation of classifiers for software risk management," *International Journal of Information Technology & Decision Making*, vol. 8, no. 04, pp. 749–767, 2009.

[93] Q. Song, Y. Guo, and M. Shepperd, "A comprehensive investigation of the role of imbalanced learning for software defect prediction," *IEEE Transactions on Software Engineering*, vol. 45, no. 12, pp. 1253–1269, 2018.

[94] T. M. Khoshgoftaar and E. B. Allen, "Logistic regression modeling of software quality," *International Journal of Reliability, Quality and Safety Engineering*, vol. 6, no. 04, pp. 303–317, 1999.

[95] C. Catal, "Software fault prediction: A literature review and current trends," *Expert systems with applications*, vol. 38, no. 4, pp. 4626–4636, 2011.

[96] G. Haixiang, L. Yijing, J. Shang, G. Mingyun, H. Yuanyue, and G. Bing, "Learning from class-imbalanced data: Review of methods and applications," *Expert Systems with Applications*, vol. 73, pp. 220–239, 2017.

[97] H. Kaur, H. S. Pannu, and A. K. Malhi, "A systematic review on imbalanced data challenges in machine learning: Applications and solutions," *ACM Computing Surveys (CSUR)*, vol. 52, no. 4, pp. 1–36, 2019.

[98] R. Malhotra and J. Jain, "Systematic literature review on imbalanced software defect prediction," *Frontiers of Computer Science*, 2021.

[99] B. A. Kitchenham, D. Budgen, and P. Brereton, *Evidence-based software engineering and systematic reviews*. CRC press, 2015, vol. 4.

[100] A. Arshad, S. Riaz, L. Jiao, and A. Murthy, "Semi-supervised deep fuzzy c-mean clustering for software fault prediction," *IEEE Access*, vol. 6, pp. 25 675–25 685, 2018.

[101] V. López, A. Fernández, J. G. Moreno-Torres, and F. Herrera, "Analysis of preprocessing vs. cost-sensitive learning for imbalanced classification. open problems on intrinsic data characteristics," *Expert Systems with Applications*, vol. 39, no. 7, pp. 6585–6608, 2012.

[102] K. Kaminsky and G. Boetticher, "How to predict more with less, defect prediction using machine learners in an implicitly data starved domain," in *The 8th world multiconference on systemics, cybernetics and informatics, Orlando, FL*. Citeseer, 2004.

[103] R. Shatnawi, "The application of roc analysis in threshold identification, data imbalance and metrics selection for software fault prediction," *Innovations in Systems and Software Engineering*, vol. 13, no. 2-3, pp. 201–217, 2017.

[104] M. Akour, I. Alsmadi, and I. Alazzam, "Software fault proneness prediction: a comparative study between bagging, boosting, and stacking ensemble and base learner methods," *International Journal of Data Analysis Techniques and Strategies*, vol. 9, no. 1, pp. 1–16, 2017.

[105] X. Yang, D. Lo, X. Xia, and J. Sun, "Tlel: A two-layer ensemble learning approach for just-in-time defect prediction," *Information and Software Technology*, vol. 87, pp. 206–220, 2017.

[106] C. W. Yohannese, T. Li, M. Simfukwe, and F. Khurshid, "Ensembles based combined learning for improved software fault prediction: A comparative study," in *2017 12th International Conference on Intelligent Systems and Knowledge Engineering (ISKE)*. IEEE, 2017, pp. 1–6.

[107] D. R. Ibrahim, R. Ghnemat, and A. Hudaib, "Software defect prediction using feature selection and random forest algorithm," in *2017 International Conference on New Trends in Computing Sciences (ICTCS)*. IEEE, 2017, pp. 252–257.

[108] K. Gao, T. M. Khoshgoftaar, and A. Napolitano, "A hybrid approach to coping with high dimensionality and class imbalance for software defect prediction," in *2012 11th international conference on machine learning and applications*, vol. 2. IEEE, 2012, pp. 281–288.

[109] K. E. Bennin, J. Keung, and A. Monden, "Impact of the distribution parameter of data sampling approaches on software defect prediction models," in *2017 24th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 2017, pp. 630–635.

[110] R. S. Wahono and N. Suryana, "Combining particle swarm optimization based feature selection and bagging technique for software defect prediction," *International Journal of Software Engineering and Its Applications*, vol. 7, no. 5, pp. 153–166, 2013.

[111] 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.

[112] A. Kalsoom, M. Maqsood, M. A. Ghazanfar, F. Aadil, and S. Rho, "A dimensionality reduction-based efficient software fault prediction using fisher linear discriminant analysis (flda)," *The Journal of Supercomputing*, vol. 74, no. 9, pp. 4568–4602, 2018.

[113] S. Huda, K. Liu, M. Abdelrazek, A. Ibrahim, S. Alyahya, H. Al-Dossari, and S. Ahmad, "An ensemble oversampling model for class imbalance problem in software defect prediction," *IEEE access*, vol. 6, pp. 24 184–24 195, 2018.

[114] R. Mousavi, M. Eftekhari, and F. Rahdari, "Omni-ensemble learning (oel): utilizing over-bagging, static and dynamic ensemble selection approaches for software defect prediction," *International Journal on Artificial Intelligence Tools*, vol. 27, no. 06, p. 1850024, 2018.

[115] R. Li and S. Wang, "An empirical study for software fault-proneness prediction with ensemble learning models on imbalanced data sets." *JSW*, vol. 9, no. 3, pp. 697–704, 2014.

[116] D.-L. Miholca, G. Czibula, and I. G. Czibula, "A novel approach for software defect prediction through hybridizing gradual relational association rules with artificial neural networks," *Information Sciences*, vol. 441, pp. 152–170, 2018.

[117] R. S. Wahono and N. S. Herman, "Genetic feature selection for software defect prediction," *Advanced Science Letters*, vol. 20, no. 1, pp. 239–244, 2014.

[118] J. Suntoro, F. W. Christanto, and H. Indriyawati, "Software defect prediction using aweig+ adacost bayesian algorithm for handling high dimensional data and class imbalance problem," *International Journal of Information Technology and Business*, vol. 1, no. 1, pp. 36–41, 2018.

[119] R. S. Wahono, N. S. Herman, and S. Ahmad, "Neural network parameter optimization based on genetic algorithm for software defect prediction," *Advanced Science Letters*, vol. 20, no. 10-11, pp. 1951–1955, 2014.

[120] T. T. Khuat and M. H. Le, "Ensemble learning for software fault prediction problem with imbalanced data," *International Journal of Electrical and Computer Engineering*, vol. 9, no. 4, p. 3241, 2019.

[121] S. Hussain, J. Keung, A. A. Khan, and K. E. Bennin, "Performance evaluation of ensemble methods for software fault prediction: An experiment," in *Proceedings of the ASWEC 2015 24th Australasian Software Engineering Conference*, 2015, pp. 91–95.

[122] B. A. O, S. Basri, S. J. Abdulkadir, V. E. Adeyemo, A. A. Imam, and A. O. Bajeh, "Software defect prediction: Analysis of class imbalance and performance stability," *Journal of Engineering Science and Technology*, vol. 14, no. 6, pp. 3294–3308, 2019.

[123] A. Alsaeedi and M. Z. Khan, "Software defect prediction using supervised machine learning and ensemble techniques: A comparative study," *Journal of Software Engineering and Applications*, vol. 12, no. 5, pp. 85–100, 2019.

[124] X.-Y. Jing, F. Wu, X. Dong, and B. Xu, "An improved sda based defect prediction framework for both within-project and cross-project class-imbalance problems," *IEEE Transactions on Software Engineering*, vol. 43, no. 4, pp. 321–339, 2016.

[125] C. Pan, M. Lu, B. Xu, and H. Gao, "An improved cnn model for within-project software defect prediction," *Applied Sciences*, vol. 9, no. 10, p. 2138, 2019.

[126] F. Wu, X.-Y. Jing, X. Dong, J. Cao, B. Xu, and S. Ying, "Cost-sensitive local collaborative representation for software defect prediction," in *2016 International Conference on Software Analysis, Testing and Evolution (SATE)*. IEEE, 2016, pp. 102–107.

[127] O. M. Maruf, "The impact of parameter optimization of ensemble learning on defect prediction," *Computer Science Journal of Moldova*, vol. 79, no. 1, pp. 85–128, 2019.

[128] K. E. Bennin, J. Keung, A. Monden, Y. Kamei, and N. Ubayashi, "Investigating the effects of balanced training and testing datasets on effort-aware fault prediction models," in *2016 IEEE 40th annual Computer software and applications conference (COMPSAC)*, vol. 1. IEEE, 2016, pp. 154–163.

[129] W. Li, Z. Huang, and Q. Li, "Three-way decisions based software defect prediction," *Knowledge-Based Systems*, vol. 91, pp. 263–274, 2016.

[130] P. R. Bal and S. Kumar, "Wr-elm: Weighted regularization extreme learning machine for imbalance learning in software fault prediction," *IEEE Transactions on Reliability*, 2020.

[131] S. K. Pandey, R. B. Mishra, and A. K. Tripathi, "Bpdet: An effective software bug prediction model using deep representation and ensemble learning techniques," *Expert Systems with Applications*, vol. 144, p. 113085, 2020.

[132] Y. Ma, L. Guo, and B. Cukic, "A statistical framework for the prediction of fault-proneness," in *Advances in Machine Learning Applications in Software Engineering*. IGI Global, 2007, pp. 237–263.

[133] P. Yu, T. Systa, and H. Muller, "Predicting fault-proneness using oo metrics. an industrial case study," in *Proceedings of the Sixth European Conference on Software Maintenance and Reengineering*. IEEE, 2002, pp. 99–107.

[134] 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.

[135] 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.

[136] M. Jureczko and D. Spinellis, "Using object-oriented design metrics to predict software defects," *Models and Methods of System Dependability. Oficyna Wydawnicza Politechniki Wrocławskiej*, pp. 69–81, 2010.

[137] 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.

[138] A. Singh, R. Bhatia, and A. Singhrova, "Taxonomy of machine learning algorithms in software fault prediction using object oriented metrics," *Procedia computer science*, vol. 132, pp. 993–1001, 2018.

[139] R. S. Wahono, "A systematic literature review of software defect prediction," *Journal of Software Engineering*, vol. 1, no. 1, pp. 1–16, 2015.

[140] Z. Li, X.-Y. Jing, and X. Zhu, "Progress on approaches to software defect prediction," *IET Software*, vol. 12, no. 3, pp. 161–175, 2018.

[141] L. H. Son, N. Pritam, M. Khari, R. Kumar, P. T. M. Phuong, P. H. Thong *et al.*, "Empirical study of software defect prediction: A systematic mapping," *Symmetry*, vol. 11, no. 2, p. 212, 2019.

[142] R. Martin, "Oo design quality metrics," *An analysis of dependencies*, vol. 12, no. 1, pp. 151–170, 1994.

[143] W. Li and S. Henry, "Object-oriented metrics that predict maintainability," *Journal of systems and software*, vol. 23, no. 2, pp. 111–122, 1993.

[144] A. H. Watson, D. R. Wallace, and T. J. McCabe, *Structured testing: A testing methodology using the cyclomatic complexity metric*. US Department of Commerce, Technology Administration, National Institute of Standards and Technology, 1996, vol. 500, no. 235.

[145] M. Jureczko and L. Madeyski, "Towards identifying software project clusters with regard to defect prediction," in *Proceedings of the 6th international conference on predictive models in software engineering*, 2010, pp. 1–10.

[146] M. A. Hall, "Correlation-based feature selection of discrete and numeric class machine learning," 2000.

[147] J. Yang and V. Honavar, "Feature subset selection using a genetic algorithm," in *Feature extraction, construction and selection*. Springer, 1998, pp. 117–136.

[148] J. Casillas, O. Cordón, M. J. Del Jesus, and F. Herrera, "Genetic feature selection in a fuzzy rule-based classification system learning process for high-dimensional problems," *Information Sciences*, vol. 136, no. 1-4, pp. 135–157, 2001.

[149] H. He, Y. Bai, E. A. Garcia, and S. Li, "Adasyn: Adaptive synthetic sampling approach for imbalanced learning," in *2008 IEEE international joint conference on neural networks (IEEE world congress on computational intelligence)*. IEEE, 2008, pp. 1322–1328.

[150] 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.

[151] C. Bunkhumpornpat, K. Sinapiromsaran, and C. Lursinsap, "Safe-level-smote: Safe-level-synthetic minority over-sampling technique for handling the class imbalanced problem," in *Pacific-Asia conference on knowledge discovery and data mining*. Springer, 2009, pp. 475–482.

[152] J. Stefanowski and S. Wilk, "Selective pre-processing of imbalanced data for improving classification performance," in *International Conference on Data Warehousing and Knowledge Discovery*. Springer, 2008, pp. 283–292.

[153] G. E. Batista, R. C. Prati, and M. C. Monard, "A study of the behavior of several methods for balancing machine learning training data," *ACM SIGKDD explorations newsletter*, vol. 6, no. 1, pp. 20–29, 2004.

[154] G. Cohen, M. Hilario, H. Sax, S. Hugonnet, and A. Geissbuhler, "Learning from imbalanced data in surveillance of nosocomial infection," *Artificial intelligence in medicine*, vol. 37, no. 1, pp. 7–18, 2006.

[155] J. Laurikkala, "Improving identification of difficult small classes by balancing class distribution," in *Conference on Artificial Intelligence in Medicine in Europe*. Springer, 2001, pp. 63–66.

[156] M. Kubat, S. Matwin *et al.*, "Addressing the curse of imbalanced training sets: one-sided selection," in *Icml*, vol. 97. Citeseer, 1997, pp. 179–186.

[157] P. Domingos, "Metacost: A general method for making classifiers cost-sensitive," in *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining*, 1999, pp. 155–164.

[158] Y. Freund and R. E. Schapire, "A decision-theoretic generalization of on-line learning and an application to boosting," *Journal of computer and system sciences*, vol. 55, no. 1, pp. 119–139, 1997.

[159] S. Wang and X. Yao, "Multiclass imbalance problems: Analysis and potential solutions," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 42, no. 4, pp. 1119–1130, 2012.

[160] Y. Freund, R. E. Schapire *et al.*, "Experiments with a new boosting algorithm," in *icml*, vol. 96. Citeseer, 1996, pp. 148–156.

[161] R. E. Schapire and Y. Singer, "Improved boosting algorithms using confidence-rated predictions," *Machine learning*, vol. 37, no. 3, pp. 297–336, 1999.

[162] R. Quinlan, "u¿ c4. 5: Programs for machine learning.¡/u¿ morgan kaufmann publishers," *San Mateo, CA*, 1993.

[163] L. Breiman, "Bagging predictors," *Machine learning*, vol. 24, no. 2, pp. 123–140, 1996.

[164] H. Guo and H. L. Viktor, "Learning from imbalanced data sets with boosting and data generation: the databoost-im approach," *ACM Sigkdd Explorations Newsletter*, vol. 6, no. 1, pp. 30–39, 2004.

[165] S. Hu, Y. Liang, L. Ma, and Y. He, "Msmote: Improving classification performance when training data is imbalanced," in *2009 second international workshop on computer science and engineering*, vol. 2. IEEE, 2009, pp. 13–17.

[166] C. Seiffert, T. M. Khoshgoftaar, J. Van Hulse, and A. Napolitano, "Rusboost: A hybrid approach to alleviating class imbalance," *IEEE Transactions on Systems, Man, and Cybernetics-Part A: Systems and Humans*, vol. 40, no. 1, pp. 185–197, 2009.

[167] N. V. Chawla, A. Lazarevic, L. O. Hall, and K. W. Bowyer, "Smoteboost: Improving prediction of the minority class in boosting," in *European conference on principles of data mining and knowledge discovery*. Springer, 2003, pp. 107–119.

[168] S. Wang and X. Yao, "Diversity analysis on imbalanced data sets by using ensemble models," in *2009 IEEE Symposium on Computational Intelligence and Data Mining*. IEEE, 2009, pp. 324–331.

[169] R. Barandela, R. M. Valdovinos, and J. S. Sánchez, "New applications of ensembles of classifiers," *Pattern Analysis & Applications*, vol. 6, no. 3, pp. 245–256, 2003.

[170] J. Błaszczyński, M. Deckert, J. Stefanowski, and S. Wilk, "Integrating selective preprocessing of imbalanced data with ivotes ensemble," in *International conference on rough sets and current trends in computing*. Springer, 2010, pp. 148–157.

[171] G. John and P. Langley, "Estimating continuous distributions in bayesian classifiers. in proceedings of the eleventh conference on uncertainty in artificial intelligence," 1995.

[172] S. Le Cessie and J. C. Van Houwelingen, "Ridge estimators in logistic regression," *Journal of the Royal Statistical Society: Series C (Applied Statistics)*, vol. 41, no. 1, pp. 191–201, 1992.

[173] M. Sumner, E. Frank, and M. Hall, "Speeding up logistic model tree induction," in *European conference on principles of data mining and knowledge discovery*. Springer, 2005, pp. 675–683.

[174] 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.

[175] R. Rojas, *Neural networks: a systematic introduction*. Springer Science & Business Media, 2013.

[176] D. W. Aha, D. Kibler, and M. K. Albert, "Instance-based learning algorithms," *Machine learning*, vol. 6, no. 1, pp. 37–66, 1991.

[177] J. G. Cleary and L. E. Trigg, "K*: An instance-based learner using an entropic distance measure," in *Machine Learning Proceedings 1995*. Elsevier, 1995, pp. 108–114.

[178] N. Landwehr, M. Hall, and E. Frank, "Logistic model trees," *Machine learning*, vol. 59, no. 1-2, pp. 161–205, 2005.

[179] B. Leo, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[180] T. K. Ho, "The random subspace method for constructing decision forests," *IEEE transactions on pattern analysis and machine intelligence*, vol. 20, no. 8, pp. 832–844, 1998.

[181] E. Frank and I. H. Witten, "Generating accurate rule sets without global optimization," 1998.

[182] J. Bacardit and N. Krasnogor, "Performance and efficiency of memetic pittsburgh

learning classifier systems," *Evolutionary computation*, vol. 17, no. 3, pp. 307–342, 2009.

[183] J. R. Cano, F. Herrera, and M. Lozano, "Using evolutionary algorithms as instance selection for data reduction in kdd: an experimental study," *IEEE transactions on evolutionary computation*, vol. 7, no. 6, pp. 561–575, 2003.

[184] S.-U. Guan and F. Zhu, "An incremental approach to genetic-algorithms-based classification," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 35, no. 2, pp. 227–239, 2005.

[185] 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.

[186] 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.

[187] S. W. Wilson, "Classifier fitness based on accuracy," *Evolutionary computation*, vol. 3, no. 2, pp. 149–175, 1995.

[188] J. Bacardit, E. K. Burke, and N. Krasnogor, "Improving the scalability of rule-based evolutionary learning," *Memetic computing*, vol. 1, no. 1, pp. 55–67, 2009.

[189] Y. Liu, Z. Qin, Z. Shi, and J. Chen, "Rule discovery with particle swarm optimization," in *Advanced Workshop on Content Computing*. Springer, 2004, pp. 291–296.

[190] K. C. Tan, Q. Yu, and J. H. Ang, "A coevolutionary algorithm for rules discovery in data mining," *International Journal of Systems Science*, vol. 37, no. 12, pp. 835–864, 2006.

[191] M. Stone, "Cross-validatory choice and assessment of statistical predictions," *Journal of the Royal Statistical Society: Series B (Methodological)*, vol. 36, no. 2, pp. 111–133, 1974.

[192] 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.

[193] K. Gao, T. M. Khoshgoftaar, and A. Napolitano, "Combining feature subset selection and data sampling for coping with highly imbalanced software data." in *SEKE*, 2015, pp. 439–444.

[194] C. Tantithamthavorn and A. E. Hassan, "An experience report on defect modelling in practice: Pitfalls and challenges," in *Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice*, 2018, pp. 286–295.

[195] T. Fawcett, "An introduction to roc analysis," *Pattern recognition letters*, vol. 27, no. 8, pp. 861–874, 2006.

[196] 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.

[197] 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.

[198] C. R. Kothari, *Research methodology: Methods and techniques*. New Age International, 2004.

[199] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *Journal of Machine learning research*, vol. 7, no. Jan, pp. 1–30, 2006.

[200] 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.

[201] 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*, vol. 95, pp. 296–312, 2018.

[202] F. Wilcoxon, "Individual comparisons by ranking methods," in *Breakthroughs in statistics*. Springer, 1992, pp. 196–202.

[203] T. M. Khoshgoftaar, K. Gao, and N. Seliya, "Attribute selection and imbalanced data: Problems in software defect prediction," in *2010 22nd IEEE International conference on tools with artificial intelligence*, vol. 1. IEEE, 2010, pp. 137–144.

[204] R. Malhotra and J. Jain, "Predicting defects in imbalanced data using resampling methods: An empirical investigation," *PeerJ Computer Science*, 2021.

[205] J. Alcalá-Fdez, A. Fernández, J. Luengo, J. Derrac, S. García, L. Sánchez, and F. Herrera, "Keel data-mining software tool: data set repository, integration of algorithms and experimental analysis framework." *Journal of Multiple-Valued Logic & Soft Computing*, vol. 17, 2011.

[206] R. Malhotra and J. Jain, "Handling imbalanced data using ensemble learning in software defect prediction," in *2020 10th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*. IEEE, 2020, pp. 300–304.

[207] R. Malhotra and J. Jain, "On alleviating class imbalance problem using ensembles

in software defect prediction: An empirical investigation," *Romanian Journal of Information Science and Technology*, 2021.

[208] T. Menzies, R. Krishna, and D. Pryor, "The promise repository of empirical software engineering data. north carolina state university, department of computer science," 2016.

[209] F. Wilcoxon, S. Katti, and R. A. Wilcox, "Critical values and probability levels for the wilcoxon rank sum test and the wilcoxon signed rank test," *Selected tables in mathematical statistics*, vol. 1, pp. 171–259, 1970.

[210] X. Wu, V. Kumar, J. R. Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, S. Y. Philip *et al.*, "Top 10 algorithms in data mining," *Knowledge and information systems*, vol. 14, no. 1, pp. 1–37, 2008.

[211] R. Malhotra and J. Jain, "Predicting defects in object-oriented software using cost-sensitive classification," in *2020 1st International Conference on Computational Research and Data Analytics*. IOP Conference Series, 2020.

[212] Y. Ma, G. Luo, X. Zeng, and A. Chen, "Transfer learning for cross-company software defect prediction," *Information and Software Technology*, vol. 54, no. 3, pp. 248–256, 2012.

[213] M. Friedman, "The use of ranks to avoid the assumption of normality implicit in the analysis of variance," *Journal of the american statistical association*, vol. 32, no. 200, pp. 675–701, 1937.

[214] R. Malhotra, "Search based techniques for software fault prediction: current trends and future directions," in *Proceedings of the 7th International Workshop on Search-Based Software Testing*, 2014, pp. 35–36.

[215] R. Malhotra and J. Jain, "Predicting software defects for object-oriented software using search-based techniques," *International Journal of Software Engineering and Knowledge Engineering*, vol. 31, no. 02, pp. 193–215, 2021.

[216] 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.

[217] R. Özakıncı and A. Tarhan, "Early software defect prediction: A systematic map and review," *Journal of Systems and Software*, vol. 144, pp. 216–239, 2018.

[218] M. Harman, "The relationship between search based software engineering and predictive modeling," in *Proceedings of the 6th International Conference on Predictive Models in Software Engineering*, 2010, pp. 1–13.

[219] M. Harman and B. F. Jones, "Search-based software engineering," *Information and software Technology*, vol. 43, no. 14, pp. 833–839, 2001.

[220] T. M. Khoshgoftaar, X. Yuan, and E. B. Allen, "Balancing misclassification rates in classification-tree models of software quality," *Empirical Software Engineering*, vol. 5, no. 4, pp. 313–330, 2000.

[221] R. Malhotra and J. Jain, "Using class imbalance learning and search - based techniques for object oriented software defect prediction," in *International Conference on Recent Trends in Engineering, Technology and Business Management)*. Amity University, Noida, India, 2019.

[222] R. Barandela, R. M. Valdovinos, J. S. Sánchez, and F. J. Ferri, "The imbalanced training sample problem: Under or over sampling?" in *Joint IAPR international workshops on statistical techniques in pattern recognition (SPR) and structural and syntactic pattern recognition (SSPR)*. Springer, 2004, pp. 806–814.

[223] 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, 2009.

[224] R. Malhotra and J. Jain, "Empirical validation of evolutionary feature selection techniques for software defect prediction," *Advances in Electrical and Computer Engineering*, 2021.

Supervisor's Biography

Prof. Ruchika Malhotra
Professor and Head of Department
Department of Software Engineering
Delhi Technological University
Email: ruchikamalhotra@dtu.ac.in

Educational Qualifications:
Postdoc (Indiana University-Purdue University Indianapolis, USA), Ph.D (Computer Applications)

Ruchika Malhotra is Professor and Head of Department of Software Engineering, Delhi Technological University, Delhi, India. She is Associate Dean in Industrial Research and Development, Delhi Technological University. She 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, USA. She is listed in the 2021 list of world's top 2% scientists by Stanford University. She was an Associate Professor in Discipline of Software Engineering, Department of Computer Science & Engineering, Delhi Technological University,

Delhi and 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 four consecutive years (2017-2020). 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. She has published more than 220 research papers in international journals and conferences. Her h-index is 33 as reported by Google Scholar.

Author's Biography

Juhi Jain
Research Scholar
Department of Computer Science & Engineering
Delhi Technological University
Email: erjuhijain@gmail.com

Educational Qualifications:
M.Tech (Information Technology), B.Tech (Information Technology)

Juhi Jain is currently pursuing her doctoral degree from Delhi Technological University. She completed her master's degree in Information Technology in 2010 from Guru Gobind Singh Indraprastha University, Delhi, India. She received her bachelor degree in Information Technology in 2004 from Hindu College of Engineering, Maharishi Dayanand University, Rohtak, India. Her research interests are in software quality improvement, validation of software metrics, predictive modeling in software engineering domain and applications of machine learning techniques in defect prediction.