A Major Project Report On

CROSS PROJECT DEFECT PREDICTION FOR OPEN SOURCE SOFTWARE

Submitted in partial fulfilment of the requirements

for the award of the degree of

MASTER OF TECHNOLOGY

IN

SOFTWARE ENGINEERING

By

Anushree Agrawal

(Roll No. 2K12/SWE/09)

Under the guidance of

Dr. Ruchika Malhotra

Department of Software Engineering

Delhi Technological University, Delhi

Department of Computer Engineering

Delhi Technological University, Delhi

2012-2014

DELHI TECHNOLOGICAL UNIVERSITY

CERTIFICATE

This is to certify that the project report entitled **CROSS PROJECT DEFECT PREDICTION FOR OPEN SOURCE SOFTWARE** is a bona fide record of work carried out by Anushree Agrawal (2K12/SWE/09) under my guidance and supervision, during the academic session 2012-2014 in partial fulfilment of the requirement for the degree of Master of Technology in Software Engineering from Delhi Technological University, Delhi.

To the best of my knowledge, the matter embodied in the thesis has not been submitted to any other University/Institute for the award of any Degree or Diploma.

Dr.RuchikaMalhotra

Asst. Professor

Department of Software Engineering

Delhi Technological University

Delhi

DELHI TECHNOLOGICAL UNIVERSITY

ACKNOWLEDGEMENT

With due regards, I hereby take this opportunity to acknowledge a lot of people who have supported me with their words and deeds in completion of my research work as part of this course of Master of Technology in Software Engineering.

To start with I would like to thank the almighty for being with me in each and every step of my life. Next, I thank my parents and family for their encouragement and persistent support.

I would like to express my deepest sense of gratitude and indebtedness to my guide and motivator, **Dr. Ruchika Malhotra**, Assistant Professor, Department of Software Engineering, Delhi Technological University for her valuable guidance and support in all the phases from conceptualization to final completion of the project.

I wish to convey my sincere gratitude to all the faculties and PhD. Scholars of Computer Engineering Department, Delhi Technological University who have enlightened me during my project.

I humbly extend my grateful appreciation to my friends especially Aditya Jalan whose moral support made this project possible.

Last but not the least, I would like to thank all the people directly and indirectly involved in successfully completion of this project.

AnushreeAgrawal

Roll No. 2K12/SWE/09

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Software defect prediction is the process of identification of defects early in the life cycle so as to optimize the testing resources and reduce maintenance efforts. Defect prediction works well if sufficient amount of data is available to train the prediction model. However, not always this is the case. For example, when the software is the first release or the company has not maintained significant data. In such cases, cross project defect prediction may identify the defective classes. In this work, we have studied the feasibility of cross project defect prediction and empirically validated the same. We conducted our experiments on 12 open source datasets. The prediction model is built using 12 software metrics. After studying the various train test combinations, we found that cross project defect prediction was feasible in 35 out of 132 cases. The success of prediction is determined via precision, recall and AUC of the prediction model. We have also analysed 14 descriptive characteristics to construct the decision tree. The decision tree learnt from this data has 15 rules which describe the feasibility of successful cross project defect prediction.

INTRODUCTION

1.1 Introduction

A software defect is a disorder in the software system such that the software does not perform its intended task well or it does not meet its requirements or it does not meet customer expectations. It is the malfunctioning of the program to produce inappropriate or unexpected results due to error in the programming or logic. A defect is anything that has to be corrected in the software product in the developer's perspective. For example, the user expects that clicking of a button on a webpage should perform a specific task and clicking the button does nothing, then this is a software defect.

Defects may be introduced in the system due to specification errors, design errors or programming errors. Also, software is prone to change. No matter how brilliantly we design software, it is liable to change. New features may be added, existing functionalities may be modified and various other modifications are certain to happen. Unknowingly software defects are injected into the system by these modifications as well. These errors may lead to malfunctioning of the system which may lead to severe economic loss or even danger to life. There are numerous examples of software defects in the past which have made significant loss of currency or life.

Ariane 5, a giant rocket launched by a European space agency was intended to give Europe overwhelming supremacy in the commercial space business. The rocket was destroyed after 39 seconds of its launch due to overflow error. When the guidance system's own computer tried to convert one piece of data the sideways velocity of the rocket from a 64 bit format to a 16 bit format; the number was too big, and an overflow error resulted after 36.7 seconds. When the guidance system shutdown, it passed control to an identical, redundant unit, which was there to provide backup in case of just such a failure. Unfortunately, the second unit had also failed in the identical manner a few milliseconds before.

The Patriot missile which was first used in the Gulf war failed several times including one that killed 28 soldiers in Dahran, Saudi Arabia. The reason was a small timing error in the system's clock which when accumulated after 14 hours; the tracking system was no longer accurate. In the Dhahran attack, the system had been operating for more than 100 hours.

The Y2K problem in the commencement of twentieth century was simply the ignorance about the adequacy of using only last two digits of the year. This created problems in many systems when the year 2000 was taken as 00 which was unacceptable.

Many companies have experienced failures in their accounting system due to faults in the software itself. The failures range from producing the wrong information to the whole system crashing.

Thus early identification of defects in SDLC is very important to prevent catastrophic results. The aim of the software development process is to detect and correct errors before it is delivered to the customers. Defect prediction in software systems focuses on prediction of fault prone classes early in the software development life cycle. It is a 2- class classification system, where the aim is to assign each class into one of the two classes, 'defective' or 'non-defective'. The prediction model needs two inputs, software static code attributes or metrics and previous data. The prediction model is built using regression or machine learning methods. This helps in near to optimal allocation of testing and maintenance resources. Defect prediction works well if a large amount of data is available to train the prediction model. However, if the data is not preserved or if we are dealing with the first release of the software system, no training data is available. Thus defect prediction based on historical data of same project is not always feasible.

1.2 Motivation of the work:

Cross project defect prediction is the process of predicting defects in software systems using historical data of other projects [1]. This is important because sufficient data is not always available to train the prediction model from the same project. Also technologies change rapidly between different releases of the software systems. Hence the prediction may

not be very accurate in some cases with the training data of a previous release. Hence the need for cross project defect prediction arises. Very few Studies are available in literature for cross project defect prediction and they show that this is a serious challenging task. Cross project defect prediction can help to optimize the allocation of testing resources from the first release of the project. It may also minimize corrective maintenance efforts in software systems. Thus cross project defect prediction is very useful in optimizing testing resources and minimizing errors. In our work, we have attempted to study the feasibility of cross project defect prediction using open source software systems. The prediction model is build using logistic regression.

1.3 Aim of the work:

In our work we have tried to empirically find the answers to the following severe research problems

1. **Is the defect data of one project likely to derive defects of another project efficiently?**

   Various studies in the literature show that historical data from software repositories can be used in prediction of software defects for upcoming releases, but availability of this past defect data is not always possible. In this study, we have empirically validated that defect data from other projects can be used to identify the defective classes.

2. **Are the characteristics of datasets useful to identify potential defect predictors?**

   Cross project defect prediction is feasible in some cases, but this is not always possible. The major challenge in this field is how to identify the scenarios where cross project defect prediction is applicable. One solution to this problem given by Zimmerman [1] in his work is to study the relationship between the characteristics of the training and test set. In this work, we have studied 14 characteristics of software projects and illustrated this relationship with the help of decision tree where the characteristics determine the potential predictors.

3. **What are the acceptable criteria for successful cross project defect prediction?**
   There are numerous studies in literature in the area of software defect prediction. The prediction model is build using statistical or machine learning methods and the efficiency of the model is evaluated using various measures as sensitivity, specificity, precision, recall, AUC etc. analyzing the various studies, we have built the model using logistic regression and chosen appropriate cut off values of precision, recall and AUC to accept or reject the model.

1.4 Organization of the thesis:

This thesis report is divided into different chapters. To start with, after abstract we have the first chapter which is introduction. It defines the problem with the software systems and the impact of these problems on economics and human life. The importance of early defect prediction is explained in this section. We also broadly describe the defect prediction process which is traditionally used and the problems associated with the same. We have descried the cross project defect prediction process and discussed the need for the same.

Chapter 2 is the related work in the context of cross project defect prediction. Numerous studies are present in literature for defect prediction model trained from previous release of the same project, but very few cross project studies have been done in literature. This section describes the cross project prediction model proposed by Zimmermann et al [1]. We have also described the empirical validation and extension of Zimmermann et al's work by He et al [8]. Ma et al. have proposed a novel learning algorithm 'Transfer Naïve Bayes' for cross company defect prediction [9]. We have summarized their work in this section of the thesis.

Chapter 3 explains the "Research Methodologies" that are used in this project. We have explained the underlying concept behind each and every technology that we have used in our project in this chapter. It demonstrates the relevance of every technology why it has been used in the project and also discussed how these techniques have been applied in our project. The data collection process is done with the help of CMS tool [2]. The procedure is explained

in detail in this section of the thesis. Then we have explained the validation process of cross project defect prediction and the process of decision tree construction in this section of the thesis.

Following this we have chapter 4, "Result Analysis" that describes the implementation section of the project as well as the cross project prediction results. We have shown the acceptance criteria of each cross project model and also the decision tree which is learnt from data.

At last, in the last chapter, i.e. chapter 6 we provide the conclusion and related future work scopes on this project which could be taken as a subject to be worked upon. We have also discussed the threats to validity of our project in this section of our thesis.

RELATED WORK

Numerous studies are available in literature in the area of software defect prediction. The aim of most of them is to study the feasibility of defect prediction from the historical data of same project. Prediction models are built using the statistical and machine learning methods. Radjenovic et al. have done a systematic literature review for software fault prediction models in their work [3]. In this work, the authors have searched seven digital libraries to identify the most commonly used set of software metrics in software fault proneness prediction. This paper aims to identify software metrics and assess their applicability in software fault prediction. The authors investigated the influence of context on metrics selection and performance. This systematic literature review includes 106 papers published between 1991 and 2011. The selected papers are classified according to metrics and context properties. The results indicate that Object-oriented metrics (49%) were used nearly twice as often compared to traditional source code metrics (27%) or process metrics (24%). Chidamber and Kemerers (CK) object-oriented metrics were most frequently used. According to the selected studies there are significant differences between the metrics used in fault prediction performance. Object-oriented and process metrics have been reported to be more successful in finding faults compared to traditional size and complexity metrics. Process metrics seem to be better at predicting post-release faults compared to any static code metrics.

Gray and MacDonell have also compared the various techniques for software fault prediction models. The authors have discussed the inherent limitations of the techniques used in defect prediction models. The use of regression analysis to derive predictive equations for software metrics has been complemented by increasing numbers of studies using non-traditional methods, such as neural networks, fuzzy logic models, case-based reasoning systems, and regression trees. There has also been an increasing level of sophistication in the

regression-based techniques used, including robust regression methods, factor analysis, and more effective validation procedures. This paper examines the implications of using these methods and provides some recommendations as to when they may be appropriate. A comparison of the various techniques is also made in terms of their modeling capabilities with specific reference to software metrics [4].

Careful attribute selection is very important for the success of a fault prediction model. The authors have investigated the impact of attribute selection on naïve bayes based fault prediction model in their work. This research analyzes the impact of attribute selection on Naive Bayes (NB) based prediction model. the results are based on Eclipse and KC1 bug database. On the basis of experimental results, the authors show that careful combination of attribute selection and machine learning apparently useful and, on the Eclipse data set ,yield reasonable good performance with 88% probability of detection and 49% false alarm rate [5].

Very few studies are available in the area of cross project defect prediction. Turhan et al. have investigated the application of cross company defect data to build prediction model using static code features [10]. They have conducted their experiments on seven NASA and three SOFTLAB datasets.

Zimmermann et al. have studied the feasibility of cross project defect prediction and validated it using several versions of open source software. They have conducted their study on apache tomcat, apache Derby, Eclipse, Firefox, Direct-X, IIS, Printing, Windows Clustering, Windows File system, SQL Server 2005 and Windows Kernel [1]. The results indicate that the relationship of characteristics between the projects permits cross project defect prediction in some cases. This relationship is analyzed with the help of decision trees that can provide early estimates for precision, recall, and accuracy before a prediction is attempted.. Their results indicate that simply using models from projects in the same domain or with the same process does not lead to accurate predictions. The authors identified factors that do influence the success of cross-project predictions.

He et al. have also empirically validated cross project defect prediction using defect data from PROMISE repository [8]. They have conducted the experiment on 34 releases of 10 open source projects. This paper investigates defect predictions in the cross-project context

focusing on the selection of training data. Major conclusions from their experiments include: (1) in the best cases, training data from other projects can provide better prediction results than training data from the same project; (2) the prediction results obtained using training data from other projects meet our criteria for acceptance on the average level, defects in 18 out of 34 cases were predicted at a Recall greater than 70% and a Precision greater than 50%; (3) results of cross-project defect predictions are related with the distributional characteristics of data sets which are valuable for training data selection. they further propose an approach to automatically select suitable training data for projects without historical data. Prediction results provided by the training data selected by using our approach are comparable with those provided by training data from the same project.

Ma et al. have proposed a novel learning algorithm 'Transfer Naïve Bayes' for cross company defect prediction [9]. They have exploited all the cross company data in training the model. The results are validated on NASA datasets and Turkish local software datasets. It is difficult to employ defect prediction models which are built on the within-company data in practice, because of the lack of these local data repositories. Transfer learning is very useful in cases when distributions of training and test instances differ, but it is appropriate for cross-company software defect prediction. In this paper, the authors consider the cross-company defect prediction scenario where source and target data are drawn from different companies. In order to harness cross company data, they try to exploit the transfer learning method to build faster and highly effective prediction model.Unlike the prior works selecting training data which are similar from the test data, they proposed a novel algorithm called Transfer Naive Bayes (TNB), by using the information of all the proper features in training data. Their solution estimates the distribution of the test data, and transfers cross-company data information into the weights of the training data. On these weighted data, the defect prediction model is built. This article presents a theoretical analysis for the comparative methods, and shows the experiment results on the data sets from different organizations. It indicates that TNB is more accurate in terms of AUC (The area under the receiver operating characteristic curve), within less runtime than the state of the art methods. It is concluded that when there are too few local training data to train good classifiers, the useful knowledge from different-distribution training data on feature level may help. They are optimistic that transfer learning

method can guide optimal resource allocation strategies, which may reduce software testing cost and increase effectiveness of software testing process.

From this study we observed that cross project defect prediction is feasible with careful selection of code quality features. The relationship among the various characteristics of the datasets should be carefully analyzed to choose the potential defect predictors. We have attempted to extend this study by empirical validation of cross project defect data using defect data of twelve open source software and twelve Chidamber and Kemerer metrics [6].The prediction model is build using logistic regression.

RESEARCH METHODOLOGY

3.1 Data Collection

We have analyzed the logs of latest version for software to identify the faulty classes. We have developed a tool, Configuration Management System (CMS) in java language to fetch these logs [2]. CMS offers features to analyze the changes amongst two versions of software as well as fetch logs from software project repositories and process them to obtain bug count. In this study we have used CMS to obtain faulty classes only. Figure 3.1 explains the data collection method of CMS.

3.1.1 Source code checkout: the first step in data collection process is to obtain the source code from the remote repository. For this, we create a local copy of the software. We connect to the CVS repository of the software by logging in into the system and then download the source code on our local machine. This is done with the help of CVS "checkout" command.

3.1.2 Extraction of bugs: After we make a local repository for the code, we can request the logs using "log" command. The server replies with the software logs in response to this command, which is a huge file. We apply text mining on this file and search for text pattern "bug" and "fix" in the logs. If any of the two keywords is found, the class is assumed to be faulty. We repeat the process for each file in the source code to identify all the faulty classes in the software.

3.1.3 Metrics calculation: We obtain the metrics for software with the help of "Understand" tool. This tool calculates the object oriented metrics for each class. We have calculated seven object oriented metrics for software.

3.1.4 Preparation of dataset: we integrate the metric and bug report to obtain the dataset. Preprocessing is done to remove the unnecessary data points. Now we apply logistic regression on the collected data to build the prediction model.

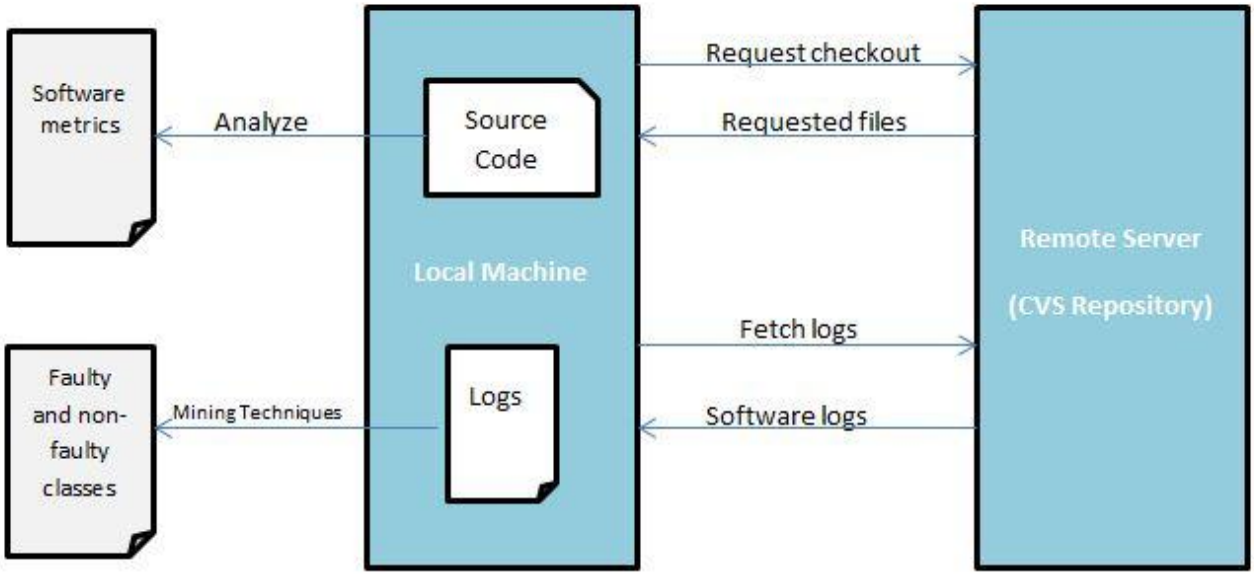

Fig.3.1. Data collection methodology

We have used 12 software for our study from sourceforge.net . These datasets vary in domain of application, size and percentage of faulty classes, while the programming language of all datasets is java. Tale 3.1 lists the programming language, version used for our experiments, and the count and percentage of faulty classes for all software under study.

- Amakihi: Amakihi supports the software testing activity of SDLC by helping the software developers in automation of test scripts. It is inspired from the software unit testing tools [11]. It ensures interaction of all interfaces with each other for thorough integration testing. It consists of 98 classes where 44 are faulty. The size of the software is 8 KLOC.

- Amber archer: Amber archer is a java class library to support corporate software development process. It helps in building UI, processing XML files, pooling, caching etc [12]. It consists of 693 java files with 9.7% of faulty classes. the size of the software is 29 KLOC.

- Abbot: Abbot is a java framework that is used to test UI for java applications. It can be used with scripts as well as compiled code. It facilitates to launch, explore and control an application [13]. It consists of 330 java classes out of which 46.1% classes are faulty and the size of the software is 27 KLOC.

- Apollo: Project Apollo is a data migration framework. It provides an editor and a compiler for data migration purpose for software systems. It consists of 292 java classes out of which 58 are faulty [14]. The size of the software is 20 KLOC.

- Avisync: Avisync is a utility developed in java language which is used to fix synchronization problems in audio/video while playing AVI files [15]. It is also small software with 67 classes with 37.3 % of faulty classes having one or more faults. The size of this software is only 3 KLOC and it is the smallest software under study.

- Jfreechart: Jfreechart is a chart library that can be used with java programs. It includes various representations of charts e.g. pie charts, bar chart, histograms, gantt charts etc [16]. It is developed using java and we have used the version 1.0.0. It consists of 689 classes out of which 59.2 % classes contain one or more faults. It is a relatively larger software with the size of 87 KLOC.

- Jgap: Jgap is a genetic programming component available as a java framework. It is used to design solutions based on evolutionary principles. It is developed in java language. It is user friendly and highly modular software. It consists of 173 classes out of which 35.3% (61) classes are having one or more than one faults [17]. The size of the software is 12 KLOC.

- Jtreeview: Jtreeview is a cross platform visualization tool which is used for visualization of gene expression data. It is developed in java language [18]. We have studied version 1.0.0 of this software for our study and 184 out of 405 classes (45.4%) are found faulty and the size of the software is 40 KLOC.

- Barcode4j: Barcode4j is available under the Apache license v2.0. It is a flexible generator of barcodes. We have used version 1.0 of this software for our experiments which consists of 170 classes out of which 31 classes had one or more than one faults [19]. The size of the software is 11 KLOC.

- Jtopen: it is a set of lightweight classes appropriate to be used on mobile devices. It supports the java classes to connect to IBM I, DDM access, basic JDBC access, command call, program call access, etc [20]. We have used v1.0 of this software for our study which consists of 1527 classes out of which 27.9% classes are faulty. This is the largest software under study with the size of 231 KLOC.

- Jung: JUNG provides a common and extendible language for the modeling, analysis, and visualization of data that can be represented as a graph or network. We have performed our experiments on JUNGv1.3 which consists of 51 faulty classes out of 149 [21]. The size of the software is 10 KLOC.

- Geotag: it is a portable; GUI based intelligent matching software system. It allows us to match date time information from photos with the location information from a map or GPS unit. We have used v 0.07 of this software for our study which consists of 628 classes, 89 of which are faulty [22]. The size of this software is 45 KLOC.

Dataset	Programming language	Version	Total LOC	No. of faulty classes	No. of non-faulty classes	Total no of classes	Percentage of faulty classes
Amakihi	Java	1.0alpha1	8059	44	54	98	44.9
Amber archer	Java	1.1	29316	67	626	693	9.7
Abbot	Java	1.0.0rc1	27235	152	178	330	46.1
Apollo	Java	0.1	20281	58	234	292	19.9
Avisync	Java	1.0	3096	25	42	67	37.3
Jfreechart	Java	1.0.0	87189	408	281	689	59.2
Jgap	Java	3.4.4	12847	61	112	173	35.3
Jtreeview	Java	1.0.0	40278	184	221	405	45.4
Barcode4j	Java	1.0	11161	31	139	170	18.2
Jtopen	Java	1.0	231246	426	1101	1527	27.9
Jung	Java	1.3	10164	51	98	149	34.2
Geotag	Java	0.07	45246	89	539	628	14.2

Table 3.1: datasets used for experiments

3.2 Prediction model

The prediction model is build using the logistic regression technique. Logistic regression is a type of probabilistic statistical classification model, which is used to predict a binary response from a binary predictor based on one or more predictor variables. It measures

the relationship between the independent variable and the categorical independent variable. "Logistic regression" is used to refer specifically to the problem in which the dependent variable is binary—that is, the number of available categories is two—while problems with more than two categories are referred to as multinomial logistic regression or, if the multiple categories are ordered, as ordered logistic regression. Logistic regression measures the relationship between a categorical dependent variable and one or more independent variables, which are usually (but not necessarily) continuous, by using probability scores as the predicted values of the dependent variable. Logistic regression can be binomial or multinomial. Binomial or binary logistic regression deals with situations in which the observed outcome for a dependent variable can have only two possible types (for example, "dead" vs. "alive"). Multinomial logistic regression deals with situations where the outcome can have three or more possible types (e.g., "disease A" vs. "disease B" vs. "disease C"). In binary logistic regression, the outcome is usually coded as "0" or "1", as this leads to the most straightforward interpretation. If a particular observed outcome for the dependent variable is the noteworthy possible outcome (referred to as a "success" or a "case") it is usually coded as "1" and the contrary outcome (referred to as a "failure" or a "non-case") as "0". Logistic regression is used to predict the odds of being a case based on the values of the independent variables (predictors). The odds are defined as the probability that a particular outcome is a case divided by the probability that it is a non-case [7].

We have studied various object oriented software metrics and selected 12 of them to build our prediction model. Table 3.2 lists these software metrics. These metrics are the independent variables to construct the prediction variable and the binary dependent variable is fault proneness.

Metrics Studied	Description
AVG_CC(Average Cyclomatic Complexity)	There are a large number of functions or program modules in a project. The average of all such Cyclomatic Complexities is known as average Cyclomatic Complexity.
CBO(Coupling between objects)	CBO is measured only for object oriented systems and it is defined as the number of other classes that a class is coupled to.
NOC(Number of children)	It is the count of number of immediate subclasses that inherit the class. This gives an idea about the influence of the class on software design.
NIM(Number of instance methods)	This is the count of total number of methods defined in a class that are only accessible through an object of that class.
NIV(Number of instance variables)	This is the count of total number of variables defined in a class that are only accessible through an object of that class.
RFC(Response for a class)	The response set (RS) of a class is a set of methods that can potentially be executed in response to a message received by an object of that class.
NPM(Number of public methods)	It is the count of total public methods in a class.
LOC(Lines of code)	The total number of executable lines of code excluding blank lines and comments.
MAX_CC(Maximum Cyclomatic Complexity)	It is the maximum cyclomatic complexity possessed by any function or program in the entire software. This gives the information about most complex part of the project.
DIT(Depth of inheritance Tree)	DIT is the path length from root node to the farthest leaf node of the inheritance tree. The higher value of DIT denotes a greater number of classes that it inherits, making it complex to predict the class behavior.
LCOM(Lack of cohesion amongst method)	It is the difference between method not having common attribute usage and methods having common attribute usage.
WMC(Weighted methods per class)	WMC is defined as the weighted sum of the complexities of all the methods defined in a class.

Table 3.2: metrics description

3.3 Descriptive statistics

We have calculated 14 indicators to describe the distribution of each metric in a training/test set. These indicators and their description are listed in table 3.3. We combine these indicators with the metrics to make a set of (14 indicators * 12 metrics) 168 metric indicators. These 168 indicators describe the distributional characteristics of the training and test sets under study. We have listed these characteristics in table 3.4-3.15 for all the datasets.

Indicator	Description
Mean	The average value of the data points. It is given as $\mu = \sum_{i=1}^{n} \frac{x_i}{n}$
Median	The middle value in the sorted dataset.
Mode	The value with maximum occurrence in the dataset.
Std. Deviation	It measures the distance of data points from the mean. It is given as $\sqrt{\frac{\sum(x-\mu)^2}{n}}$
Variance	It is a measure of variability and computed by squaring the std. deviation.
Skewness	It is the measure of asymmetry in the dataset.
Kurtosis	It is the measure of peakedness in the dataset.
Minimum	The minimum value among all the data points.
Maximum	The maximum value among all the data points.
Range	The numeric difference between the minimum and maximum.
First Quartile	The first quartile is obtained by computing the median of the dataset and then re-computing the median of the lower half.
Third Quartile	The third quartile is obtained by computing the median of the dataset and then re-computing the median of the upper half.
Interquartile range	The difference between the third quartile and the first quartile.
Coff. Of variation	It is given as the ratio of std. deviation to the arithmetic mean.

Table 3.3: indicators of software attributes

Following below are the lists of all the distributive characteristics by different datasets. Let us start with *Amakihi*.

Amakihi	Mean	Median	Mode	Std. Deviation	Variance_	Skewness	Kurtosis	Range	Minimum	Maximum	First Quartile	Third Quartile	Interquartile range	Coff. Of variation
AVG_CC	1.79	1.00	1.00	1.71	2.93	3.57	15.76	11.00	1.00	12.00	1.00	2.00	1.00	0.96
CBO	2.05	1.00	0.00	2.97	8.83	3.40	15.60	18.00	0.00	18.00	0.00	3.00	3.00	1.45
NOC	0.30	0.00	0.00	0.85	0.73	3.27	11.51	5.00	0.00	5.00	0.00	0.00	0.00	2.88
NIM	6.13	4.00	1.00	7.33	53.77	3.88	23.39	57.00	0.00	57.00	1.00	9.00	8.00	1.20
NIV	1.46	1.00	0.00	2.30	5.30	2.30	5.31	11.00	0.00	11.00	0.00	2.00	2.00	1.58
RFC	9.20	8.00	4.00	7.92	62.76	2.99	16.33	59.00	1.00	60.00	4.00	13.00	9.00	0.86
NPM	0.21	0.00	0.00	0.56	0.31	2.89	8.54	3.00	0.00	3.00	0.00	0.00	0.00	2.62
LOC	82.23	44.50	5.00	147.10	21638.02	5.86	44.07	1263.00	5.00	1268.00	17.00	83.00	66.00	1.79
MAX_CC	3.80	2.00	1.00	4.25	18.10	2.47	7.72	24.00	1.00	25.00	1.00	5.00	4.00	1.12
DIT	1.72	2.00	2.00	0.69	0.47	0.81	1.03	3.00	1.00	4.00	1.00	2.00	1.00	0.40
LCOM	51.06	64.00	0.00	35.53	1262.12	-0.50	-1.37	100.00	0.00	100.00	0.00	80.00	80.00	0.70
WMC	13.81	9.00	4.00	15.28	233.44	2.17	5.11	73.00	1.00	74.00	4.00	18.00	14.00	1.11

Table 3.4: Distributive characteristics of amakihi

Amberarcher	Mean	Median	Mode	Std. Deviation	Variance_	Skewness	Kurtosis	Range	Minimum	Maximum	First Quartile	Third Quartile	Interquartile range	Coff. Of variation
AVG_CC	1.23	1.00	1.00	0.75	0.56	3.02	13.77	7.00	0.00	7.00	1.00	1.00	0.00	0.61
CBO	2.62	2.00	0.00	3.20	10.23	2.06	5.37	20.00	0.00	20.00	0.00	4.00	4.00	1.22
NOC	0.45	0.00	0.00	2.03	4.13	9.26	109.88	30.00	0.00	30.00	0.00	0.00	0.00	4.53
NIM	0.31	0.00	0.00	1.26	1.60	14.39	288.85	27.00	0.00	27.00	0.00	0.00	0.00	4.08
NIV	0.39	0.00	0.00	1.37	1.88	5.91	41.61	14.00	0.00	14.00	0.00	0.00	0.00	3.54
RFC	10.25	8.00	2.00	10.88	118.38	2.80	10.53	76.00	0.00	76.00	3.00	13.00	10.00	1.06
NPM	0.57	0.00	0.00	1.54	2.38	5.01	34.66	16.00	0.00	16.00	0.00	0.00	0.00	2.72
LOC	42.30	23.00	11.00	50.84	2584.21	2.79	11.03	401.00	1.00	402.00	11.00	53.00	42.00	1.20
MAX_CC	2.39	1.00	1.00	2.43	5.93	2.91	11.38	20.00	0.00	20.00	1.00	3.00	2.00	1.02
DIT	1.91	2.00	1.00	0.95	0.91	0.70	-0.45	4.00	1.00	5.00	1.00	3.00	2.00	0.50
LCOM	30.80	16.00	0.00	33.77	1140.40	0.44	-1.43	100.00	0.00	100.00	0.00	64.00	64.00	1.10
WMC	8.35	5.00	2.00	9.94	98.73	2.77	10.89	83.00	0.00	83.00	2.00	10.00	8.00	1.19

Table 3.5: Distributive characteristics of amberarcher

Abbot	Mean	Median	Mode	Std. Deviation	Variance_	Skewness	Kurtosis	Range	Minimum	Maximum	First Quartile	Third Quartile	Interquartile range	Coff. Of variation
AVG_CC	1.75	1.00	1.00	1.41	1.98	2.81	10.97	10.00	0.00	10.00	1.00	2.00	1.00	0.81
CBO	3.22	1.00	1.00	7.16	51.24	11.40	169.16	113.00	0.00	113.00	1.00	4.00	3.00	2.22
NOC	0.56	0.00	0.00	2.78	7.74	12.48	185.08	44.00	0.00	44.00	0.00	0.00	0.00	4.96
NIM	6.26	2.00	2.00	11.13	123.82	5.36	38.42	112.00	0.00	112.00	2.00	7.00	5.00	1.78
NIV	1.76	0.00	0.00	4.34	18.80	6.89	64.39	52.00	0.00	52.00	0.00	2.00	2.00	2.47
RFC	34.73	6.00	5.00	59.53	3544.07	2.18	3.45	221.00	0.00	221.00	3.00	35.25	32.25	1.71
NPM	1.05	0.00	0.00	5.23	27.34	13.18	204.55	85.00	0.00	85.00	0.00	1.00	1.00	4.99
LOC	82.53	25.50	5.00	203.82	41543.89	7.94	84.96	2655.00	1.00	2656.00	9.00	83.00	74.00	2.47
MAX_CC	3.96	2.00	1.00	4.60	21.19	3.30	15.58	37.00	0.00	37.00	1.00	5.00	4.00	1.16
DIT	2.47	2.00	2.00	1.30	1.69	0.81	-0.04	5.00	1.00	6.00	1.00	3.00	2.00	0.53
LCOM	30.44	0.00	0.00	36.14	1305.94	0.55	-1.42	98.00	0.00	98.00	0.00	66.00	66.00	1.19
WMC	16.63	6.00	2.00	38.13	1453.52	6.25	47.94	369.00	0.00	369.00	2.00	15.25	13.25	2.29

Table 3.6:Distributive characteristics of abbot

apollo	Mean	Median	Mode	Std. Deviation	Variance_	Skewness	Kurtosis	Range	Minimum	Maximum	First Quartile	Third Quartile	Interquartile range	Coff. Of variation
AVG_CC	1.73	1.00	1.00	1.65	2.72	3.89	18.81	13.00	0.00	13.00	1.00	2.00	1.00	0.95
CBO	4.85	4.00	0.00	5.02	25.17	2.05	7.12	35.00	0.00	35.00	1.00	8.00	7.00	1.03
NOC	0.60	0.00	0.00	3.37	11.35	9.58	107.82	44.00	0.00	44.00	0.00	0.00	0.00	5.62
NIM	6.43	5.00	5.00	7.44	55.28	5.00	40.18	82.00	0.00	82.00	3.00	8.00	5.00	1.16
NIV	2.79	2.00	0.00	3.48	12.14	2.75	13.62	30.00	0.00	30.00	0.00	4.00	4.00	1.25
RFC	13.15	9.00	1[a]	11.88	141.03	1.59	4.09	83.00	0.00	83.00	4.00	20.50	16.50	0.90
NPM	0.21	0.00	0.00	0.78	0.61	5.06	30.31	7.00	0.00	7.00	0.00	0.00	0.00	3.80
LOC	69.46	39.50	17.00	97.06	9421.09	4.84	36.28	1022.00	2.00	1024.00	17.25	82.75	65.50	1.40
MAX_CC	4.31	3.00	1.00	5.35	28.60	3.99	22.07	48.00	0.00	48.00	1.00	5.00	4.00	1.24
DIT	1.90	2.00	2.00	0.81	0.65	0.78	0.97	4.00	1.00	5.00	1.00	2.00	1.00	0.42
LCOM	39.73	44.00	0.00	32.20	1036.71	-0.03	-1.43	100.00	0.00	100.00	0.00	68.00	68.00	0.81
WMC	13.82	9.00	5.00	19.47	379.10	5.85	53.45	229.00	0.00	229.00	5.00	16.00	11.00	1.41

Table 3.7: Distributive characteristics of Apollo

avisync	Mean	Median	Mode	Std. Deviation	Variance_	Skewness	Kurtosis	Range	Minimum	Maximum	First Quartile	Third Quartile	Interquartile range	Coff. Of variation
AVG_CC	1.13	1.00	1.00	0.42	0.18	2.09	6.36	3.00	0.00	3.00	1.00	1.00	0.00	0.37
CBO	3.07	1.00	0.00	4.75	22.52	2.04	3.97	21.00	0.00	21.00	0.00	4.00	4.00	1.54
NOC	0.61	0.00	0.00	1.53	2.33	3.37	13.97	9.00	0.00	9.00	0.00	0.00	0.00	2.50
NIM	7.69	6.00	1.00	7.74	59.85	1.32	1.17	32.00	0.00	32.00	1.00	11.00	10.00	1.01
NIV	2.06	1.00	0.00	2.81	7.88	2.15	5.61	14.00	0.00	14.00	0.00	3.00	3.00	1.36
RFC	14.93	8.00	5.00	12.76	162.89	0.86	-0.59	44.00	0.00	44.00	5.00	24.00	19.00	0.86
NPM	1.57	0.00	0.00	4.12	17.01	3.80	15.24	23.00	0.00	23.00	0.00	2.00	2.00	2.63
LOC	46.21	32.00	5.00	55.89	3123.99	2.17	4.81	247.00	4.00	251.00	5.00	61.00	56.00	1.21
MAX_CC	2.28	1.00	1.00	2.52	6.33	3.45	17.01	17.00	0.00	17.00	1.00	3.00	2.00	1.10
DIT	2.36	2.00	1.00	1.35	1.81	0.54	-1.02	4.00	1.00	5.00	1.00	4.00	3.00	0.57
LCOM	68.04	81.00	100.00	34.97	1222.74	-1.04	-0.24	100.00	0.00	100.00	57.00	100.00	43.00	0.51
WMC	11.07	7.00	1.00	12.38	153.28	1.78	3.29	58.00	0.00	58.00	1.00	15.00	14.00	1.12

Table 3.8: Distributive characteristics of avisync

jfreechart	Mean	Median	Mode	Std. Deviation	Variance_	Skewness	Kurtosis	Range	Minimum	Maximum	First Quartile	Third Quartile	Interquartile range	Coff. Of variation
AVG_CC	1.45	1.00	1.00	0.93	0.86	3.18	14.34	8.00	0.00	8.00	1.00	2.00	1.00	0.64
CBO	4.51	2.00	1.00	6.36	40.47	3.87	31.87	81.00	0.00	81.00	1.00	6.00	5.00	1.41
NOC	0.31	0.00	0.00	1.33	1.76	6.25	45.75	14.00	0.00	14.00	0.00	0.00	0.00	4.33
NIM	9.93	5.00	4.00	15.17	230.14	5.52	43.64	172.00	0.00	172.00	4.00	11.00	7.00	1.53
NIV	2.23	1.00	0.00	4.87	23.74	5.12	35.08	46.00	0.00	46.00	0.00	2.50	2.50	2.19
RFC	33.82	7.00	5.00	62.26	3876.49	2.51	5.05	264.00	1.00	265.00	5.00	27.00	22.00	1.84
NPM	0.49	0.00	0.00	1.04	1.08	2.55	7.54	7.00	0.00	7.00	0.00	0.00	0.00	2.11
LOC	126.54	74.00	5.00	186.01	34600.39	4.77	33.46	2148.00	4.00	2152.00	42.00	133.00	91.00	1.47
MAX_CC	5.29	2.00	2.00	6.54	42.75	2.98	11.56	51.00	0.00	51.00	2.00	6.00	4.00	1.23
DIT	2.03	2.00	2.00	0.88	0.78	1.35	2.60	5.00	1.00	6.00	2.00	2.00	0.00	0.43
LCOM	38.90	41.00	0.00	37.86	1433.32	0.14	-1.68	100.00	0.00	100.00	0.00	75.00	75.00	0.97
WMC	21.86	9.00	8.00	38.07	1449.36	6.00	53.51	489.00	0.00	489.00	6.00	22.00	16.00	1.74

Table 3.9: Distributive characteristics of jfreechart

jgap	Mean	Median	Mode	Std. Deviation	Variance_	Skewness	Kurtosis	Range	Minimum	Maximum	First Quartile	Third Quartile	Interquartile range	Coff. Of variation
AVG_CC	1.31	1.00	1.00	0.73	0.53	2.03	7.06	5.00	0.00	5.00	1.00	2.00	1.00	0.56
CBO	4.18	4.00	5.00	4.36	18.98	3.16	16.04	34.00	0.00	34.00	1.00	5.00	4.00	1.04
NOC	0.61	0.00	0.00	3.70	13.70	8.77	82.26	39.00	0.00	39.00	0.00	0.00	0.00	6.04
NIM	8.80	7.00	7.00	9.41	88.51	3.14	13.46	65.00	0.00	65.00	3.00	10.00	7.00	1.07
NIV	2.31	1.00	0.00	3.60	12.95	3.08	12.67	25.00	0.00	25.00	0.00	3.00	3.00	1.56
RFC	31.92	12.00	2[a]	31.87	1015.67	0.44	-1.70	83.00	0.00	83.00	4.00	71.00	67.00	1.00
NPM	0.17	0.00	0.00	0.77	0.59	7.34	65.90	8.00	0.00	8.00	0.00	0.00	0.00	4.60
LOC	74.26	46.00	36.00	109.82	12059.72	5.28	36.83	1011.00	3.00	1014.00	25.00	77.00	52.00	1.48
MAX_CC	3.40	2.00	2.00	3.57	12.74	2.98	12.20	26.00	0.00	26.00	1.00	4.00	3.00	1.05
DIT	1.69	1.00	1.00	0.80	0.63	0.62	-1.15	2.00	1.00	3.00	1.00	2.00	1.00	0.47
LCOM	76.64	84.00	100.00	27.02	730.05	-1.43	1.74	100.00	0.00	100.00	64.00	100.00	36.00	0.35
WMC	15.42	10.00	8.00	21.35	455.83	4.47	26.14	179.00	0.00	179.00	5.00	17.00	12.00	1.38

Table 3.10:Distributive characteristics of jgap

jtree view	Mean	Median	Mode	Std. Deviation	Variance_	Skewness	Kurtosis	Range	Minimum	Maximum	First Quartile	Third Quartile	Interquartile range	Coff. Of variation
AVG_CC	1.61	1.00	1.00	1.20	1.44	3.05	12.51	10.00	0.00	10.00	1.00	2.00	1.00	0.75
CBO	3.51	2.00	0.00	4.78	22.84	3.83	24.01	48.00	0.00	48.00	1.00	4.00	3.00	1.36
NOC	0.24	0.00	0.00	0.95	0.90	6.16	51.21	11.00	0.00	11.00	0.00	0.00	0.00	3.92
NIM	8.13	5.00	1.00	9.99	99.76	3.33	17.23	92.00	0.00	92.00	2.00	10.00	8.00	1.23
NIV	3.16	2.00	0.00	4.56	20.83	2.74	9.95	33.00	0.00	33.00	0.00	4.00	4.00	1.44
RFC	13.84	8.00	1.00	18.26	333.44	2.71	8.42	104.00	0.00	104.00	3.00	17.00	14.00	1.32
NPM	0.85	0.00	0.00	1.73	2.99	2.85	9.52	12.00	0.00	12.00	0.00	1.00	1.00	2.03
LOC	99.45	54.00	10[a]	131.52	17296.98	3.67	19.68	1153.00	3.00	1156.00	27.00	124.50	97.50	1.32
MAX_CC	3.95	3.00	1.00	3.64	13.26	1.76	3.09	20.00	0.00	20.00	1.00	5.00	4.00	0.92
DIT	1.87	2.00	2.00	0.70	0.49	0.70	1.17	4.00	1.00	5.00	1.00	2.00	1.00	0.37
LCOM	41.39	50.00	0.00	35.01	1225.38	-0.02	-1.58	100.00	0.00	100.00	0.00	75.00	75.00	0.85
WMC	16.02	8.00	1.00	21.06	443.50	3.09	12.90	153.00	0.00	153.00	4.00	20.00	16.00	1.31

Table 3.11: Distributive characteristics of jtreeview

barcode4j	Mean	Median	Mode	Std. Deviation	Variance_	Skewness	Kurtosis	Range	Minimum	Maximum	First Quartile	Third Quartile	Interquartile range	Coff. Of variation
AVG_CC	1.79	1.00	1.00	1.27	1.62	2.03	4.50	7.00	1.00	8.00	1.00	2.00	1.00	0.71
CBO	3.05	3.00	0.00	2.72	7.41	0.84	0.02	11.00	0.00	11.00	1.00	5.00	4.00	0.89
NOC	0.34	0.00	0.00	1.18	1.40	4.96	28.52	9.00	0.00	9.00	0.00	0.00	0.00	3.52
NIM	4.77	3.00	1.00	4.56	20.82	1.74	3.40	24.00	0.00	24.00	2.00	6.00	4.00	0.96
NIV	1.10	0.00	0.00	2.16	4.68	2.69	7.98	12.00	0.00	12.00	0.00	1.00	1.00	1.97
RFC	9.79	7.00	3.00	9.79	95.77	1.61	1.91	41.00	1.00	42.00	3.00	12.00	9.00	1.00
NPM	0.84	0.00	0.00	2.05	4.22	3.78	17.74	15.00	0.00	15.00	0.00	1.00	1.00	2.46
LOC	65.65	43.00	4.00	82.98	6885.97	4.72	34.94	788.00	3.00	791.00	18.00	87.25	69.25	1.26
MAX_CC	4.42	3.00	1.00	4.64	21.55	2.64	10.08	32.00	1.00	33.00	1.00	6.00	5.00	1.05
DIT	1.76	2.00	2.00	0.71	0.50	0.57	-0.09	3.00	1.00	4.00	1.00	2.00	1.00	0.40
LCOM	31.55	0.00	0.00	37.14	1379.04	0.49	-1.55	96.00	0.00	96.00	0.00	75.00	75.00	1.18
WMC	12.17	7.00	1.00	13.57	184.11	2.37	6.99	77.00	1.00	78.00	3.00	17.00	14.00	1.11

Table 3.12: Distributive characteristics of barcode4j

jtopen	Mean	Median	Mode	Std. Deviation	Variance_	Skewness	Kurtosis	Range	Minimum	Maximum	First Quartile	Third Quartile	Interquartile range	Coff. Of variation
AVG_CC	1.82	1.00	1.00	1.96	3.83	6.69	75.59	34.00	0.00	34.00	1.00	2.00	1.00	1.08
CBO	5.09	3.00	0.00	6.19	38.32	2.51	12.00	72.00	0.00	72.00	1.00	7.00	6.00	1.22
NOC	0.43	0.00	0.00	2.35	5.54	10.68	143.99	39.00	0.00	39.00	0.00	0.00	0.00	5.47
NIM	10.09	5.00	1.00	16.02	256.73	5.43	46.58	217.00	0.00	217.00	2.00	11.00	9.00	1.59
NIV	3.59	1.00	0.00	5.56	30.96	3.19	15.51	53.00	0.00	53.00	0.00	5.00	5.00	1.55
RFC	20.80	15.00	7.00	22.42	502.82	2.85	13.35	217.00	0.00	217.00	6.00	27.00	21.00	1.08
NPM	0.88	0.00	0.00	1.74	3.02	4.22	26.22	20.00	0.00	20.00	0.00	1.00	1.00	1.96
LOC	151.44	74.00	35.00	248.11	61559.40	5.18	40.21	3135.00	1.00	3136.00	35.00	163.00	128.00	1.64
MAX_CC	6.10	4.00	1.00	9.73	94.77	6.00	52.07	135.00	0.00	135.00	1.00	7.00	6.00	1.60
DIT	2.03	2.00	2.00	0.96	0.92	0.79	0.13	4.00	1.00	5.00	1.00	3.00	2.00	0.47
LCOM	73.63	84.00	100.00	28.80	829.28	-1.33	0.82	100.00	0.00	100.00	62.00	95.00	33.00	0.39
WMC	23.73	11.00	1.00	38.90	1513.07	4.53	29.70	443.00	0.00	443.00	4.00	28.00	24.00	1.64

Table 3.13: Distributive characteristics of jtopen

jung	Mean	Median	Mode	Std. Deviation	Variance_	Skewness	Kurtosis	Range	Minimum	Maximum	First Quartile	Third Quartile	Interquartile range	Coff. Of variation
AVG_CC	1.60	1.00	1.00	1.08	1.17	2.04	4.27	6.00	0.00	6.00	1.00	2.00	1.00	0.68
CBO	5.07	4.00	0.00	4.63	21.43	0.91	0.64	23.00	0.00	23.00	1.00	8.00	7.00	0.91
NOC	0.39	0.00	0.00	0.99	0.98	3.20	11.42	6.00	0.00	6.00	0.00	0.00	0.00	2.55
NIM	7.21	4.00	1.00	7.56	57.15	1.69	2.96	40.00	0.00	40.00	2.00	10.00	8.00	1.05
NIV	2.22	1.00	0.00	2.85	8.15	2.35	6.98	16.00	0.00	16.00	0.00	3.00	3.00	1.28
RFC	17.55	9.00	4.00	18.42	339.32	1.15	0.13	66.00	0.00	66.00	4.00	30.50	26.50	1.05
NPM	0.47	0.00	0.00	1.11	1.24	3.11	11.03	7.00	0.00	7.00	0.00	0.00	0.00	2.37
LOC	68.21	40.00	4.00	67.13	4506.33	1.28	0.87	276.00	2.00	278.00	16.50	105.50	89.00	0.98
MAX_CC	3.92	3.00	1.00	3.40	11.55	1.91	5.49	22.00	0.00	22.00	1.00	5.00	4.00	0.87
DIT	1.78	1.00	1.00	1.03	1.05	1.33	1.08	4.00	1.00	5.00	1.00	2.00	1.00	0.58
LCOM	35.64	35.00	0.00	34.04	1158.53	0.24	-1.53	100.00	0.00	100.00	0.00	68.00	68.00	0.95
WMC	14.68	9.00	1.00	14.61	213.56	1.69	3.96	87.00	0.00	87.00	4.00	21.50	17.50	1.00

Table 3.14: Distributive characteristics of jung

geotag	Mean	Median	Mode	Std. Deviation	Variance_	Skewness	Kurtosis	Range	Minimum	Maximum	First Quartile	Third Quartile	Interquartile range	Coff. Of variation
AVG_CC	1.82	1.00	1.00	1.72	2.95	2.88	11.86	15.00	0.00	15.00	1.00	2.00	1.00	0.94
CBO	2.74	1.00	1.00	4.20	17.65	4.75	38.19	52.00	0.00	52.00	1.00	3.00	2.00	1.53
NOC	0.34	0.00	0.00	1.64	2.69	8.68	87.32	19.00	0.00	19.00	0.00	0.00	0.00	4.84
NIM	5.38	3.00	1.00	7.65	58.59	4.89	35.79	88.00	0.00	88.00	1.00	6.00	5.00	1.42
NIV	1.67	0.00	0.00	3.25	10.58	4.06	22.49	29.00	0.00	29.00	0.00	2.00	2.00	1.95
RFC	11.47	5.00	2.00	14.39	207.06	2.04	4.08	88.00	0.00	88.00	2.00	14.00	12.00	1.25
NPM	0.68	0.00	0.00	1.89	3.57	5.64	44.13	22.00	0.00	22.00	0.00	1.00	1.00	2.80
LOC	72.05	35.00	10.00	127.28	16200.42	7.76	98.49	2039.00	2.00	2041.00	15.00	79.50	64.50	1.77
MAX_CC	4.29	2.00	1.00	6.90	47.55	8.23	109.87	113.00	0.00	113.00	1.00	5.00	4.00	1.61
DIT	1.93	2.00	2.00	1.05	1.10	1.64	2.97	5.00	1.00	6.00	1.00	2.00	1.00	0.54
LCOM	37.39	44.00	0.00	36.23	1312.27	0.21	-1.54	100.00	0.00	100.00	0.00	70.00	70.00	0.97
WMC	12.87	6.00	2.00	22.32	498.23	7.83	103.00	365.00	0.00	365.00	3.00	14.00	11.00	1.73

Table 3.15: Distributive characteristics of geotag

3.4 Performance evaluation measures

3.4.1 Precision and Recall

- **Precision:** precision is the ratio of number of classes that are correctly classified as faulty and the no. of classes that are classified as faulty.

- **Recall:** recall is the ratio of the number of classes that are correctly classified to the total no. of faulty classes.

Precision and recall are predictive parameters of information retrieval system. Precision is given by the fraction of retrieved instances that are correctly identified out of the total retrieved instances whereas recall is given by the fraction of relevant instances that are retrieved out of the total number of instances. Let us elaborate this using an example. Suppose in a game of quiz contest. A contestant X answers 6 questions out of a total 10 questions. He answers 4 of them correct and 2 incorrect. So, in this scenario Precision (X) = 4/7 and Recall (X) = 4/10.

From the above concept, we can easily conclude that if in any problem the Value of Precision is high it implies that most of the information retrieved is relevant and very less is irrelevant. If the recall is high, then that implies most of the information was correctly retrieved.

3.4.2 Area under ROC curve

ROC curve is a graphical representation of binary classifiers when we vary the discriminating threshold points. It is a plot in between the true positives out of the total actual positives vs. the false positives out of the total actual negatives. There are two more terminologies related to the above concept.

- Sensitivity: Sensitivity or true positive rate is the fraction of true positives and total actual positives.

- Specificity: It is the false positive rate or the fraction of false positives and total actual negatives subtracted from 1.

From here, we deduce that ROC curve is a graphical plot between sensitivity and 1-specificity at varied discriminating thresholds.

We often use Area Under the Curve (AUC) to denote the results obtained by ROC curve. AUC is actually the probability that the classifier will rank a randomly chosen positive instance higher than a randomly chosen negative instance.

3.4.3 Construction of decision tree

Although cross project defect prediction works in several cases, but successful defect prediction is not feasible in all cases. After studying the various combinations of training and testing datasets, we have constructed a decision tree to validate the relationship between feasibility of cross project defect prediction and distributional characteristics of training and testing datasets.

3.4.3.1 Decision tree

A decision tree is a decision support tool that uses a tree-like graph or model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. It is one way to display an algorithm. A decision tree is a flowchart-like structure in which internal node represents test on an attribute, each branch represents outcome of test and each leaf node represents class label (decision taken after computing all attributes). A path from root to leaf represents classification rules. In decision analysis a decision tree and the closely related influence diagram is used as a visual and analytical decision support tool, where the expected values (or expected utility) of competing alternatives are calculated. A decision tree consists of 3 types of nodes:

1. Decision nodes - commonly represented by squares
2. Chance nodes - represented by circles
3. End nodes - represented by triangles

Decision trees are commonly used in operations research, specifically in decision analysis, to help identify a strategy most likely to reach a goal. If in practice decisions have to be taken online with no recall under incomplete knowledge, a decision tree should be paralleled by a probability model as a best choice model or online selection model algorithm. Another use of decision trees is as a descriptive means for calculating conditional probabilities.

We have conducted our experiments on all possible permutations of the datasets. One set is chosen as training set which is used to build the prediction model and the remaining 11 sets are test sets. They are chosen one by one to evaluate the model. This process is repeated y choosing all the datasets as training sets one at a time. Thus we get 132 (12X11) combinations from 12 datasets. Precision, recall and AUC are analyzed to predict whether prediction is possible or not. **If precision>0.6 and recall> 0.7 and AUC >0.6, then we assume that prediction is possible, otherwise not.** These cut off values are chosen by analyzing the acceptance criterion of various prediction models that are available in literature. Choosing these cut off values of precision, recall and AUC, prediction was found possible in 35 out of 132 permutations. Then we used the distributive characteristics of these datasets to build the decision node of the decision tree and the leaf node tells whether prediction is possible or not.

To construct the decision tree, we have used weka 3.6.10. Random tree algorithm is used to construct the decision tree with 10X validation on the dataset.

3.4.3.2 Random Tree:

These are machine learning methods that operate by constructing a multitude of decision trees at training time and giving output as individual trees. These are basically used in supervised learning techniques. The training algorithm for Random forest uses bagging technique. Bagging method is used to generate multiple version of a predictor and to aggregate this predictor. Bootstrap replicates the learning set that is used to form multiple version of predictor then after used it as new learning set. Bagging is also known as bootstrap aggregating that repeatedly samples from a data set according to uniform probability distribution. Despite its popular usage in many real-world applications, existing research is mainly concerned with studying unstable learners as the key to ensure the performance gain of a bagging predictor, with many key factors remaining unclear. For example, it is not clear when a bagging predictor can outperform a single learner and what is the expected performance gain when different learning algorithms were used to form a bagging predictor.

However, in random forest the algorithm used differs from bagging slightly. It uses a modified tree learning algorithm that selects a random subset of the features, at each candidate split. It provides for the correlation of the trees in an ordinary bootstrap sample.

3.4.3.3 Validation method

Validation is the process of checking whether the statistical machine learning techniques that we have applied is actually acceptable or not. The validation process can involve analyzing the goodness of fit of the regression, analyzing whether the results are random, and checking whether the model's predictive performance deteriorates substantially when applied to data that were not used in model estimation.

We have applied k-fold cross validation, with k=10.

In k-fold cross-validation, the original sample is randomly partitioned into k equal size subsamples. Of the k subsamples, a single subsample is retained as the validation data for testing the model, and the remaining k − 1 sub samples are used as training data. The cross-validation process is then repeated k times (the folds), with each of the k subsamples used exactly once as the validation data. The k results from the folds can then be averaged (or otherwise combined) to produce a single estimation.

The dataset is constructed in the following manner: first we List all the distributive characteristics for all the metrics for the training data set followed by the distributive characteristics of test dataset. The last column is a binary variable which tells prediction is possible for this permutation or not. Assuming we have m distribution characteristics for n metrics, the total number of columns in the dataset will be 2(m*n) +1. In our case, m=14 and n=12, hence the total number of columns in the dataset = 337. The number of rows is equal to the number of permutations of the training and test sets. Thus the size of our dataset is 337 X 132. The procedure for construction of dataset for decision tree is shown in Figure 3.2. The prediction model is built by training from a software system and tested on all remaining datasets. The result is marked “yes” if the criterion for successful prediction is satisfied else

"no". Now we calculate distributive characteristics for all metrics of both train and test sets and combine them with the prediction result as shown in figure 3.2. This gives one row of the combined dataset. Now we repeat the process for all combinations to complete the dataset for learning of the decision tree.

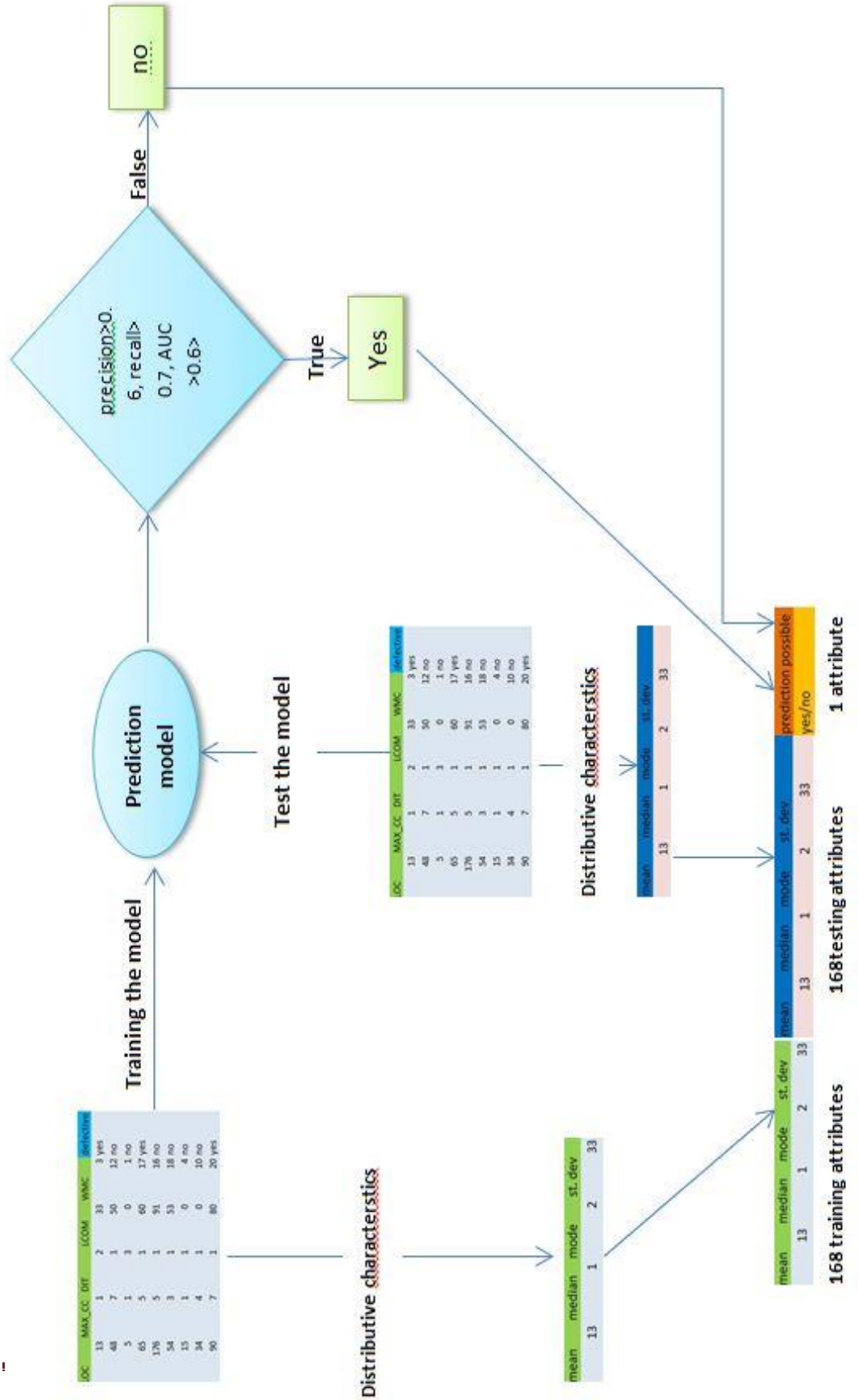

Figure 3.2: Generation of training-test instance from the dataset combination

RESULT ANALYSIS

4.1 Experimental Results

We generated 132 train-test instances from the various combinations of the datasets. Out of these 132 instances, 35 were successful with the values of precision, recall and AUC greater than the cut off values. Thus we get only 26.5% successful cross project defect prediction scenarios. The best prediction results are observed with Amberarcher as test set and various training sets. The highest values of precision recall and AUC are obtained with barcode4j and Geotag as training sets and Amberarcher as test set. Precision and recall for both these models is greater than 80% and AUC is greater than 70%. Table 4.1 lists the successful train-test combinations and corresponding values of precision, recall and AUC.

The size of the decision tree learnt from these train test instances is 75. It consists of 38 leaf nodes out of which 15 are labeled "yes" and 23 are labeled "no". We performed 10 – fold cross validation and observed precision 74.7%, recall 74.2% and AUC 67.9%. The decision tree is built using random tree algorithm. Table 4.2 lists the rules derived from the decision tree for successful cross project defect prediction. The support indicates the no. of instances which satisfy the rule. The decision tree is shown in figure 4.1. Only 37 out of 336 project characteristics are found significant in the construction of decision tree. 24 of these 37 deciding characteristics are of training set and the rest 13 of test set. These characteristics are compared with a cut off value at each deciding node and the value decides the class whether "yes" or "no".

Training	Testing	Precision	Recall	AUC
Amakihi	Amberarcher	0.848	0.791	0.63
Amakihi	barcode4j	0.74	0.706	0.721
amberarcher	Abbot	0.763	0.7	0.793
amberarcher	Apollo	0.743	0.733	0.648

amberarcher	Avisync	0.697	0.701	0.763
amberarcher	barcode4j	0.769	0.782	0.775
Abbot	Jtreeview	0.738	0.738	0.797
Apollo	Amberarcher	0.842	0.886	0.754
Apollo	Abbot	0.763	0.7	0.895
Apollo	Jgap	0.717	0.705	0.78
Apollo	barcode4j	0.769	0.818	0.809
Avisync	Amakihi	0.724	0.724	0.754
jfreechart	Amberarcher	0.857	0.848	0.656
jfreechart	Abbot	0.777	0.776	0.87
jfreechart	Jtreeview	0.72	0.721	0.786
Jgap	Amberarcher	0.832	0.89	0.335
Jgap	Abbot	0.792	0.721	0.713
Jgap	Apollo	0.764	0.781	0.713
Jtreeview	Abbot	0.811	0.809	0.901
Jtreeview	barcode4j	0.797	0.724	0.788
barcode4j	Amberarcher	0.86	0.84	0.711
barcode4j	Abbot	0.739	0.7	0.826
barcode4j	Apollo	0.707	0.709	0.678
barcode4j	Avisync	0.708	0.701	0.729
Jung	Amberarcher	0.854	0.851	0.62
Jung	Apollo	0.755	0.767	0.604
Jung	barcode4j	0.82	0.841	0.67
Jung	Geotag	0.802	0.841	0.504
Geotag	Amberarcher	0.859	0.887	0.709
Geotag	Apollo	0.761	0.801	0.673
Geotag	barcode4j	0.788	0.824	0.812
amberarcher	Geotag	0.781	0.788	0.596
Apollo	Geotag	0.797	0.83	0.687
Jgap	Geotag	0.776	0.728	0.535
barcode4j	Geotag	0.787	0.815	0.708

Table 4.1: successful prediction results

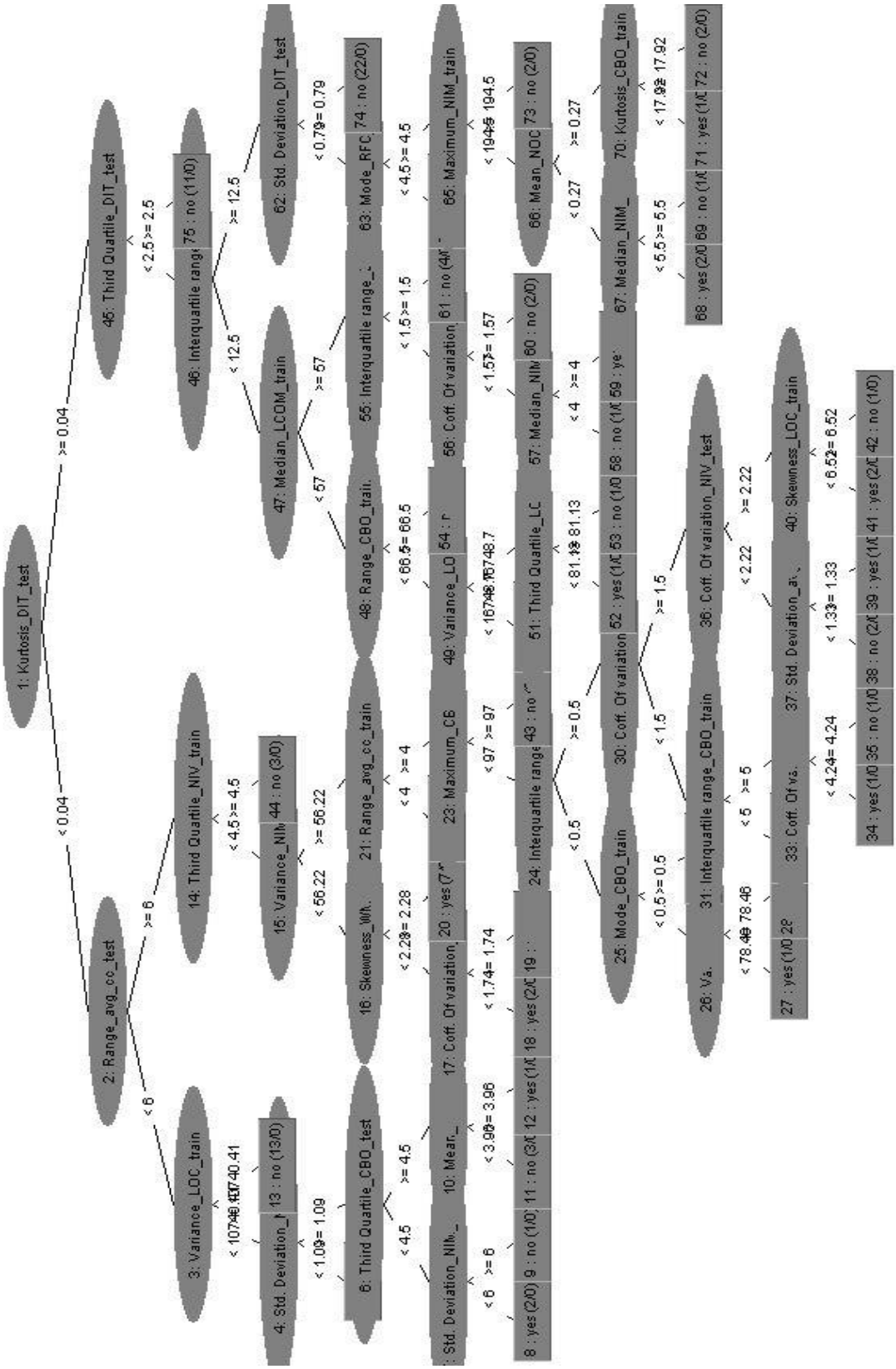

Figure 4.2: Decision Tree

RULE	SUPPORT
Kurtosis_DIT_test >= 0.04 AND Third Quartile_DIT_test < 2.5 AND Interquartile range_WMC_test < 12.5 AND Median_LCOM_train < 57 AND Range_CBO_train < 66.5 AND Variance_LOC_train < 16748.7	8
Kurtosis_DIT_test < 0.04 AND Range_avg_cc_test >= 6 AND Third Quartile_NIV_train < 4.5 AND Variance_NIM_train < 56.22 AND Skewness_WMC_train >= 2.28	7
Kurtosis_DIT_test < 0.04 AND Range_avg_cc_test >= 6 AND Third Quartile_NIV_train < 4.5 AND Variance_NIM_train >= 56.22 AND Range_avg_cc_train >= 4 AND Maximum_CBO_train < 97 AND Interquartile range_NPM_test < 0.5 AND Mode_CBO_train >= 0.5	3
Kurtosis_DIT_test < 0.04 AND Range_avg_cc_test < 6 AND Variance_LOC_train < 10740.41 AND Std. Deviation_NOC_train >= 1.09 AND Third Quartile_CBO_test < 4.5 AND Std. Deviation_NIM_train < 6	2
Kurtosis_DIT_test < 0.04 AND Range_avg_cc_test >= 6 AND Third Quartile_NIV_train < 4.5 AND Variance_NIM_train < 56.22 AND Skewness_WMC_train < 2.28 AND Coff. Of variation_WMC_test < 1.74	2
Kurtosis_DIT_test < 0.04 AND Range_avg_cc_test >= 6 AND Third Quartile_NIV_train < 4.5 AND Variance_NIM_train >= 56.22 AND Range_avg_cc_train >= 4 AND Maximum_CBO_train < 97 AND Interquartile range_NPM_test >= 0.5 AND Coff. Of variation_NIV_train < 1.5 AND Interquartile range_CBO_train < 5	2
Kurtosis_DIT_test < 0.04 AND Range_avg_cc_test >= 6 AND Third Quartile_NIV_train < 4.5 AND Variance_NIM_train >= 56.22 AND Range_avg_cc_train >= 4 AND Maximum_CBO_train < 97 AND Interquartile range_NPM_test >= 0.5 AND Coff. Of variation_NIV_train >= 1.5 AND Coff. Of variation_NIV_test >= 2.22 AND Skewness_LOC_train < 6.52	2
Kurtosis_DIT_test >= 0.04 AND Third Quartile_DIT_test < 2.5 AND Interquartile range_WMC_test >= 12.5 AND Std. Deviation_DIT_test < 0.79 AND Mode_RFC_train >= 4.5 AND Maximum_NIM_train < 194.5 AND Mean_NOC_test < 0.27 AND Median_NIM_train < 5.5	2

Kurtosis_DIT_test < 0.04 AND Range_avg_cc_test < 6 AND Variance_LOC_train < 10740.41 AND Std. Deviation_NOC_train >= 1.09 AND Third Quartile_CBO_test >= 4.5 AND Mean_CBO_train >= 3.96	1
Kurtosis_DIT_test < 0.04 AND Range_avg_cc_test >= 6 AND Third Quartile_NIV_train < 4.5 AND Variance_NIM_train >= 56.22 AND Range_avg_cc_train >= 4 Maximum_CBO_train < 97 AND Interquartile range_NPM_test < 0.5 AND Mode_CBO_train < 0.5 AND Variance_NIM_train < 78.46	1
Kurtosis_DIT_test < 0.04 AND Range_avg_cc_test >= 6 AND Third Quartile_NIV_train < 4.5 AND Variance_NIM_train >= 56.22 AND Range_avg_cc_train >= 4 AND Maximum_CBO_train < 97 AND Interquartile range_NPM_test >= 0.5 AND Coff. Of variation_NIV_train < 1.5 AND Interquartile range_CBO_train >= 5 AND Coff. Of variation_NOC_test < 4.24	1
Kurtosis_DIT_test < 0.04 AND Range_avg_cc_test >= 6 AND Third Quartile_NIV_train < 4.5 AND Variance_NIM_train >= 56.22 AND Range_avg_cc_train >= 4 AND Maximum_CBO_train < 97 AND Interquartile range_NPM_test >= 0.5 AND Coff. Of variation_NIV_train >= 1.5 AND Coff. Of variation_NIV_test < 2.22 AND Std. Deviation_avg_cc_train >= 1.33	1
Kurtosis_DIT_test >= 0.04 AND Third Quartile_DIT_test < 2.5 AND Interquartile range_WMC_test < 12.5 AND Median_LCOM_train < 57 AND Range_CBO_train < 66.5 AND Variance_LOC_train >= 16748.7 AND Third Quartile_LOC_test < 81.13	1
Kurtosis_DIT_test >= 0.04 AND Third Quartile_DIT_test < 2.5 AND Interquartile range_WMC_test < 12.5 AND Median_LCOM_train >= 57 AND Interquartile range_DIT_train < 1.5 AND Coff. Of variation_NIV_train < 1.57 AND Median_NIM_test >= 4	1
Kurtosis_DIT_test >= 0.04 AND Third Quartile_DIT_test < 2.5 AND Interquartile range_WMC_test >= 12.5 AND Std. Deviation_DIT_test < 0.79 AND Mode_RFC_train >= 4.5 AND Maximum_NIM_train < 194.5 AND Mean_NOC_test >= 0.27 AND Kurtosis_CBO_train < 17.92	1

Table 4.2: Rules for successful prediction learnt from DT

4.2 Discussion of results

From the results obtained from our experiments, some of the common observations we concluded are:

1. Software with lower percentage of defective classes has a very large set of potential defect predictors.
2. Defects for large software systems cannot be predicted by relatively smaller software systems.
3. Datasets with huge difference in the number of classes cannot be used in cross project defect prediction.

Table 4.3 lists the datasets which can be used for identification of defective classes for each of the training dataset. Here we can see that jtopen is not useful in defect proneness prediction of any of the software under study. Amber archer, Barcode4j and Jung are potential predictors for 5,5 and 4 datasets respectively. Abbot and Avisync are predictors for only 1 dataset while jtopen for none.

Training Dataset	Prediction possible for
Amakihi	Amber archer, Barcode4j
Amber archer	Abbot, Apollo , Avisync, Barcode4j, Geotag
Abbot	Jtreeview
Apollo	Amber archer, Abbot, Jgap, Barcode4j, Geotag
Avisync	Amakihi
Jfreechart	Amber archer, Abbot, Jtreeview
Jgap	Abbot, Apollo, Geotag
Jtreeview	Abbot, Barcode4j
Barcode4j	Amber archer , Abbot, Apollo, Avisync, Geotag
Jung	Amber archer, Apollo, Barcode4j, Geotag
Geotag	Amber archer, Apollo, Barcode4j

Table 4.3: Performance of training sets

Table 4.4 is another representation of table 18, where the relationship between dataset and its potential predictors is shown. From this table, we can see that Amber archer can be predicted by most (seven out of eleven in this case) of the datasets. Thus it is good training as well as test set. Similarly Abbot and Barcode4j can be predicted by six datasets. Thus they are good test datasets for which defects can be easily identified. Apollo and Geotag can also be predicted by five datasets and hence defective classes are identifiable in these cases.

Dataset	Potential Predictors
Amakihi	Avisync
Amber archer	Amakihi, Apollo, Jfreechart, Jgap, Barcode4j, Jung, Geotag
Abbot	Amber archer, Apollo, Jfreechart, Jgap,Jtreeviw, Barcode4j
Apollo	Amber archer, Jgap, Barcode4j, Jung, Geotag
Avisync	Amber archer, Barcode4j
Jgap	Apollo
Jtreeview	Abbot, Jfreechart
Barcode4j	Amakihi, Amber archer, Apollo,Jtreeview, Jung, Geotag
Geotag	Amber archer, Apollo, Jgap, Barcode4j, Jung

Table 4.4: Potential predictors for test sets

Figure 4.2 shows the diagrammatic representation of the potential predictors for software. The X-axis shows the test dataset and Y-axis shows the count of the potential predictors. This helps in the relative study of the potential predictors for each test software. Amber archer has the highest number of predictors while jfreechart, jtopen and jung can't be predicted by any of the training sets. However, if we relax some of the acceptance criterion, we obtain better results with these software systems. Apollo, avisync, barcode4j and geotag also have ample training sets. Thus we can see that 9 out of 12 software systems under study can be successfully predicted by one or more training sets.

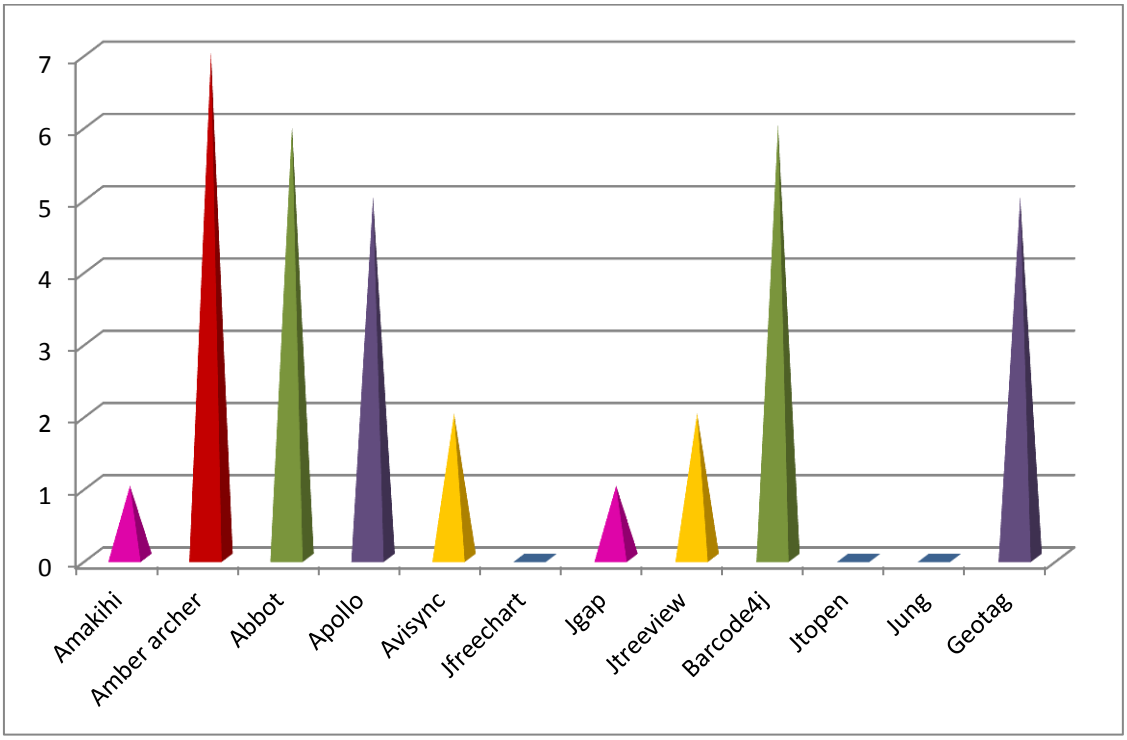

Figure 4.2: potential defect predictors for dataset under study

4.3 Threats to validity

One of the major threats to validity of our work is the acceptance criteria of the successful model. We have selected three parameters for successful model i.e. precision, recall and AUC. The selection of these parameters is based on previous studies in literature about defect prediction and our own analysis. However, the acceptance criteria may vary depending on various factors. In such a case, some of our observations and conclusions may change.

Another threat is the selection of static code metrics to build the defect prediction model. Studies in literature show that these metrics can be used for defect prediction models, but it is not always the case. The appropriate selection of these metrics may vary depending on the dataset. A subset of these metrics is found significant in a large number of studies. Thus we

can conclude that our experiments and observations may vary depending on the selection of these static code metrics.

One other threat may be related to our data collection methodology. We have assumed the presence of a bug by the keywords “bug” and “fix” in the change logs. Previous researches have observed that these keywords indicate the presence of bug in a class; however this may depend on the commenting style of the developer as well. If the information is logged via different keywords other than these, then some of the bugs may remain undetected. This may lead to mistakes in defect assignment and identification

CONCLUSION AND FUTURE WORK

5.1 Conclusions

Cross project defect prediction is very useful for projects where sufficient training data is unavailable to train the model. Cross project defect prediction enables identification of faulty classes in first release of software systems. We have discussed software defect and its impact on software systems in chapter 1. Defects may lead to failure of the software and cause financial loss of and may even lead to danger to life. The need for early defect prediction is discussed in this section. The prediction model may be built using training data from previous release of same software or using data of different software. The first approach is not always possible. We have empirically validated the second approach it this thesis. We have discussed the previous work in literature in cross project defect prediction in chapter 2. The research methodology is explained in Chapter 3. The data collection is done with the help of 'CMS tool' developed y us which calculates the defect data for each class. Software metrics are collected using 'Understand for Java'. The prediction model is built with this using logistic regression. We conducted our experiments on 12 open source projects. 132 combinations of train-test instances are generated from these 12 projects and the feasibility of cross project defect prediction is analyzed. The results show that cross project defect prediction is not always feasible. Only 35 out of 132 instances exhibit successful cross project prediction behavior in our experiments. Thus careful selection of training set needs to be done in order to identify defective classes correctly. The decision tree, constructed in our experiment learns from the distributive characteristics of software projects to generate rules for successful defect prediction. This may help in selection of appropriate training sets. The following are the important conclusions drawn from our study:

1. Chances of successful cross project defect prediction are more likely for comparable number of classes in the training and test sets. It is observed that jtopen; with very high number of classes and high LOC than other software systems is neither a good training nor a test set.

2. Cross project defect prediction may provide acceptable results, if careful selection of training set is done.

5.2 Application of the work

Cross project defect prediction is the process of learning from one project to improve another project. It is applicable to the Software Development Life Cycle to improve the software quality and make the software system more reliable and gain more confidence and customer's satisfaction. Also choosing more than one prediction model trained by different sets will increase the confidence of the prediction results. This is not possible in the traditional defect prediction method because model is trained from the previous release of same system. Training the model with different data will increase the reliance on prediction results. This will increase the reliability, traceability, usability and maintainability of the software systems and will help in mitigating software crisis.

5.3 Future Work

Previous studies show that application of some machine learning algorithms builds better prediction models than statistical method. We may apply machine learning methods as bagging, naïve bayes etc to build the prediction model instead of logistic regression in future. This may improve the performance of the model as well as decision tree.

Our experiments for cross project do not take into account the programming language of the software under study. All the projects under study are developed using same programming language i.e. java. We may extend our work where combinations of different programming languages are taken and verify if the prediction works as well in such scenarios or not.

We may also extend the work on real life corporate software. The process followed and the complexity of industrial software is different from that of open source software and hence we may take these features also into account. This will make the application of cross project defect prediction more realistic and applicable to software industry.

REFERENCES

[1]. D. Radjenović, M. Heričko, R. Torkar, and A. Živkovič, "Software fault prediction metrics: A systematic literature review," Inf. Softw. Technol., vol. 55, no. 8, pp. 1397–1418, Aug. 2013.

[2]. A. R. Gray and S. G. Macdonell, "A comparison of techniques for developing predictive models of software metrics," vol. 5849, no. 96, pp. 6–7, 1997.

[3]. B. Mishra and K. K. Shukla, "Impact of attribute selection on defect proneness prediction in OO software," 2011 2nd Int. Conf. Comput. Commun. Technol., pp. 367–372, Sep.

[4]. Shyam R. Chidamber and Chris F. Kemerer, "A Metrics Suite for Object Oriented Design", IEEE Trans. Softw. Eng., vol. 20, no. 6, pp. 476-493, June 1994.

[5]. David G. Kleinbaum and Mitchel Klein, "Logistic Regression-A Self-Learning Text", Third edition.

[6]. Z. He, F. Shu, Y. Yang, M. Li, and Q. Wang, An investigation on the feasibility of cross-project defect prediction, vol. 19, no. 2. 2011, pp. 167–199.

[7]. Y. Ma, G. Luo, X. Zeng, and A. Chen, "Transfer learning for cross-company software defect prediction," Inf. Softw. Technol., vol. 54, no. 3, pp. 248–256, Mar. 2012.

[10]. B. Turhan, T. Menzies, A. B. Bener, and J. Di Stefano, "On the relative value of cross-company and within-company data for defect prediction," Empir. Softw. Eng., vol. 14, no. 5, pp. 540–578, Jan. 2009

[11]. http://amakihi.sourceforge.net/

[12]. http://sourceforge.net/projects/amberarcher/

[13]. http://abbot.sourceforge.net/doc/overview.shtml

[14]. http://sourceforge.net/projects/startec-apollo

[15]. http://sourceforge.net/projects/avisync/

[16]. http://sourceforge.net/projects/jfreechart/

[17]. http://sourceforge.net/projects/jgap/

[18]. http://sourceforge.net/projects/jtreeview/

[19]. http://sourceforge.net/projects/barcode4j/

[20]. http://sourceforge.net/projects/jt400/

[21]. http://sourceforge.net/projects/jung/

[22]. http://sourceforge.net/projects/geotag/