Software Defect Prediction Based on Classification Rule Mining

Dulal Chandra Sahana

Department of Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela – 769 008, India

Software Defect Prediction Based on Classification Rule Mining

Dissertation submitted in

May 2013

to the department of

Computer Science and Engineering

of

National Institute of Technology Rourkela

in partial fulfillment of the requirements

for the degree of

Master of Technology

by

Dulal Chandra Sahana

(Roll 211CS3299)

under the supervision of

Prof. Korra Sathya Babu

Department of Computer Science and Engineering

National Institute of Technology Rourkela

Rourkela – 769 008, India

Computer Science and Engineering
National Institute of Technology Rourkela
Rourkela-769 008, India. www.nitrkl.ac.in

Prof. Korra Sathya Babu
Professor (Computer Science)

May 30, 2013

Certificate

This is to certify that the work in the thesis entitled *Software Defect Prediction Based on Classification Rule Mining* by *Dulal Chandra Sahana*, bearing roll number 211CS3299, is a record of an original research work carried out by him under my supervision and guidance in partial fulfillment of the requirements for the award of the degree of *Master of Technology* in *Computer Science and Engineering.* Neither this thesis nor any part of it has been submitted for any degree or academic award elsewhere.

Prof Korra Sathya Babu

Acknowledgment

I am gratefull to numerous local and global peers who have contributed towords shaping this thesis. At the outset, I would like to express my sincere thanks to Prof. K Sathya Babu for his advise during my thesis work. As my superviser , he has constantly encouraged me to remain focused on achieving my goal. His observations and comments help me to establish the overall direction of the research and to move forword with investigation in depth. He has help me greatly and been a source of knowledege.

I am very much indebted to Prof. Ashok Kumar Turuk, Head-CSE, for his continuous encouragement and support. He is always ready to help with a smile. I am also thankfull to all professors of the department for their support.

I am really thankfull to my all friends. My sincere thanks to everyone who has provided me with kind words, a welcome ear, new ideas, usefull criticism, or their invaluable time, I am truly indebted.

I must acknowledge the academic resources that I have got from NIT Rourkela. I would like to thank administrative and technical staff members of the department who have been kind enough to advise and help in their respective roles.

Last, but not the least, I would like to dedicate this thesis to my family, for their love patience, and understanding.

Dulal Chandra Sahana

dulal.sahana@gmail.com

Abstract

There has been rapid growth of software development. Due to various causes, the software comes with many defects. In Software development process, testing of software is the main phase which reduces the defects of the software. If a developer or a tester can predict the software defects properly then, it reduces the cost, time and effort. In this paper, we show a comparative analysis of software defect prediction based on classification rule mining. We propose a scheme for this process and we choose different classication algorithms. Showing the comparison of predictions in software defects analysis. This evaluation analyzes the prediction performance of competing learning schemes for given historical data sets(NASA MDP Data Set). The result of this scheme evaluation shows that we have to choose different classifier rule for different data set.

***Keywords*:** *Software defect prediction, classification Algorithm, Cofusion matrix.*

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Introduction to Software Defect Prediction

There has been a huge growth in the demand for software quality during recent ages. As a consequence, issues are related to testing, becoming increasingly critical. The ability to measure software defect can be extremely important for minimizing cost and improving the overall effectiveness of the testing process. The major amount of faults in a software system are found in a few of its components.

Although there is variety in the definition of software quality, it is truly accepted that a project with many defects lacks the quality of the software. Knowing the causes of possible defects as well as identifying general software process areas that may need attention from the initialization of a project could save money, time and working effort. The possibility of early estimating the probable faultiness of software could help on planning, controlling and executing software development activities. A low cost method for defect analysis is learning from past mistakes to prevent future ones. Today, there exist several data sets that could be mined in order to discover useful knowledge regarding defects.

Using this knowledge one should ideally be able to:–

a. Identify potential fault-prone software.

b. Estimate the distinct number of faults, and

c. Discover the possible causes of faults..

1.2 Motivation

Different data mining methods have been proposed for defect analysis in the past, but few of them manage to deal successfully with all of the above issues. Regression models estimates are difficult to interpret and also provide the exact number of faults which is too risky, especially in the beginning of a project when too little information is available. On the other hand classication models that predict possible faultiness can be specific, but not so much usefull to give clue about the actual number of faults. Many researcher used many techniques with different dataset that predict faultiness. But there are so many classification rule algorithms that can be effective to predict faultiness. All these issues motivates to our research in these field of software falult/defect prediction.

1.3 Objective

Keeping the research indications in view, it has been realized that there exists enough scope to improve the software defect prediction. In this research the objectives are confined to the followings: —

i. To utilize novel data set filtering mechanism for effective noise remove.

ii. To utilize novel classification algorithm for better prediction.

iii. To use better evalution measerment parameter to get better result.

iv. To decrease the software development cost, time and effort.

1.4 Structure of This Thesis

The remaining portion of this thesis is organized as follows:

- Chapter 2 describes the related background materials. This also includes a deffination of software defect prediction, classification, classifier etc. which are need to know for this research. This chapter also describes some of the broad categories of classification algorithms. It also describes the related works has been done in the past. It details the benefits and detriments of these different approaches.

- Chapter 3 describes an framework for software defect prediction. This also describes, how the framework has been evaluated in different steps. It also details the datasets that will be explored. And details the measerment parameters for defect prediction.

- Chapter 4 shows the results of the previously documented experiments. Here, we will show the difference in performance of global and locality based classifiers. Any discrepancies between the results shown here and prior results will be explained here.

- Chapter 6 lists the conclusions gathered from the experiments. We comment on the state of locality based learning as it pertains to software defect prediction. Finally, we detail what future research are needed to explore for software defect prediction.

Chapter 2

Background & Literature Survey

The purpose of this chapter is to establish a theoretical background for the project. The focus of this study will be on software defects and effort spent correcting software defects. However, it is necessary to explore research areas which influence or touches software defects. Poor software quality may be manifested through severe software defects, or software maintenance may be costly due to many defects requiring extensive effort to correct. Last, we explore relevant research methods for this study. The following digital sources was consulted:ACM Digital Library, IEEE Xplore, and Science Direct.

2.1 Data Mining for software Engineering

To improve the software productivity and quality, software engineers are applying data mining algorithms to various SE tasks. Many algorithms can help engineers figure out how to invoke API methods provided by a complex library or framework with insufficient documentation. In terms of maintenance, such type of data mining algorithms can assist in determining what code locations must be changed when another code location is changed. Software engineers can also use data mining algorithms to hunt for potential bugs that can cause future in-field failures as well as identify buggy lines of code (LOC) responsible for already-known failures. The second and third columns of Table 2.1 list several example data mining algorithms and the SE tasks to which engineers apply them [1].

Table 2.1: Example software engineering data, Mining algorithm, SE tasks

SE Data	Mining algo.	SE Tasks
Sequences: execution/ static traces, co-changes	Frequent itemset/ sequence/ partial-order mining, sequence matching/ clustering/ classification	Programming, maintenance, bug detection, debugging
Graphs: dynamic/ static call graphs, program dependence graphs	Frequent subgraph mining, graph matching/ clustering/ classification	Bug detection, debugging
Text: bug reports, e-mails, code comments, documentation	Text matching/ clustering/ classification	Maintenance, bug detection, debugging

2.2 Software defect predictor

A defect predictor is a tool or method that guides testing activities and software development lifecycle. According to Brooks, half the cost of software development is in unit and systems testing. Harold and Tahat also conform that testing phase requires approximately 50% or more of the whole project schedule. Therefore, the main challenge is the testing phase and practitioners seek predictors that indicate where the defects might exist before they start testing. This allows them to efficiently allocate their scarce resources. Defect predictors are used to make an ordering of modules to be inspected by veriffication and validation teams:

- In the case where there are insufficient resources to inspect all code (which is a very common situation in industrial developments), defect predictors can be used to increase the chances that the inspected code will have defects.
- In the case where all the code is to be inspected, but that inspection process will take weeks to months to complete, defect predictors can be used to increase the chances that defective modules will be inspected earlier. This is useful since it gives the development team earlier notification of what modules require rework, hence giving them more time to complete that rework prior to delivery.

2.3 Defect Prediction as a Classification Problem

Software defect prediction can be viewed as a supervised binary classification problem [2] [3]. Software modules are represented with software metrics, and are labelled as either defective or non-defective. To learn defect predictors, data tables of historical examples are formed where one column has a boolean value for ”defects detected” (i.e. dependent variable) and the other columns describe software characteristics in terms of software metrics (i.e. independent variables).

2.4 Binary classification

In machine learning and statistics, classification is the problem of identifying to which of a set of categories (sub-populations) a new observation belongs, on the basis of a training set of data containing observations (or instances) whose category membership is known.

Binary or binomial classification is the task of classifying the members of a given set of objects into two groups on the basis of whether they have some property or not.

Data Classification is two step process. In the first step, a classifier is built describing a predetermined set of data classes or concepts. This is the learning step(or training phase), where a classification algorithm is builds the classifier by analyzing or "learning form" a training set made up of database tuples and there associated class labels.

In the second step the model is used for classification. Therefore, a test set is used, make up of test tupples and there associated class labels.

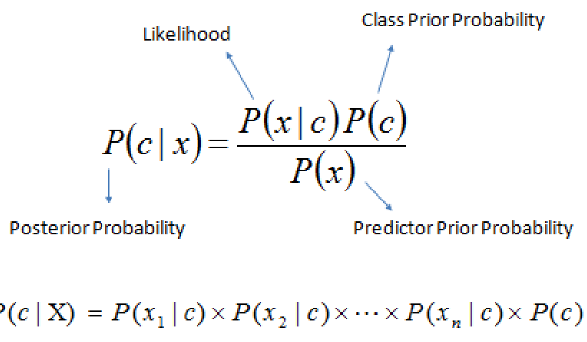
A classification rule [3] takes the form X=> C, where X is a set of data items, and C is the class (label) and a predetermined target. With such a rule, a transaction or data record t in a given database could be classified into class C if t contains X.

2.5 Binary Classification Algorithms

2.5.1 Bayesian Classification

The Naive Bayesian classifier is based on Bayes theorem with independence assumptions between predictors. A Naive Bayesian model is easy to build, with no complicated iterative parameter estimation which makes it particularly useful for very large datasets. Despite its simplicity, the Naive Bayesian classifier often does surprisingly well and is widely used because it often outperforms more sophisticated classification methods.

Algorithm:
Bayes theorem provides a way of calculating the posterior probability, P(c|x), from P(c), P(x), and P(x|c). Naive Bayes classifier assume that the effect of the value of a predictor (x) on a given class (c) is independent of the values of other predictors. This assumption is called class conditional independence.



$$P(c|x) = \frac{P(x|c)P(c)}{P(x)}$$

$$P(c|X) = P(x_1|c) \times P(x_2|c) \times \cdots \times P(x_n|c) \times P(c)$$

Figure 2.1: Bayes theorem

- P(c|x) is the posterior probability of class (target) given predictor (attribute).
- P(c) is the prior probability of class.
- P(x|c) is the likelihood which is the probability of predictor given class.
- P(x) is the prior probability of predictor.

Example:

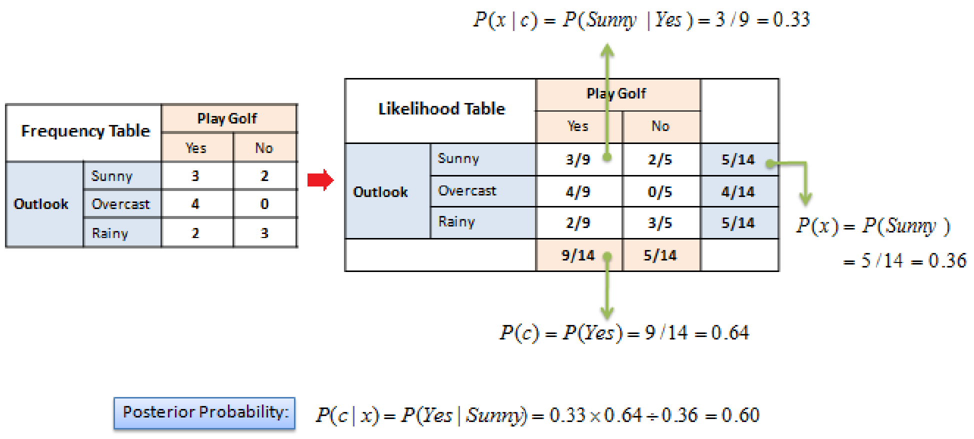
The posterior probability can be calculated by first, constructing a frequency table for each attribute against the target. Then, transforming the frequency tables to likelihood tables and finally use the Naive Bayesian equation to calculate the posterior probability for each class. The class with the highest posterior probability is the outcome of prediction.

2.5.2 Rule-Based Classification

Rules are a good way of representing information or bits of knowledge. A rule-based classifier uses a set of IF-THEN rules for classification. An IF-THEN rule is an expression of the form-

IF condition THEN conclusion

Example:



Frequency Table		Play Golf	
		Yes	No
Outlook	Sunny	3	2
	Overcast	4	0
	Rainy	2	3

Likelihood Table		Play Golf		
		Yes	No	
Outlook	Sunny	3/9	2/5	5/14
	Overcast	4/9	0/5	4/14
	Rainy	2/9	3/5	5/14
		9/14	5/14	

$P(x \mid c) = P(Sunny \mid Yes) = 3/9 = 0.33$

$P(x) = P(Sunny) = 5/14 = 0.36$

$P(c) = P(Yes) = 9/14 = 0.64$

Posterior Probability: $P(c \mid x) = P(Yes \mid Sunny) = 0.33 \times 0.64 \div 0.36 = 0.60$

Figure 2.2: Baye's theorem Example

IF age=youth AND student=yes THEN buys_computer=yes

There are many rule-based classifier algorithms are there. Some of them are: DecisionTable, OneR, PART, JRip, ZeroR.

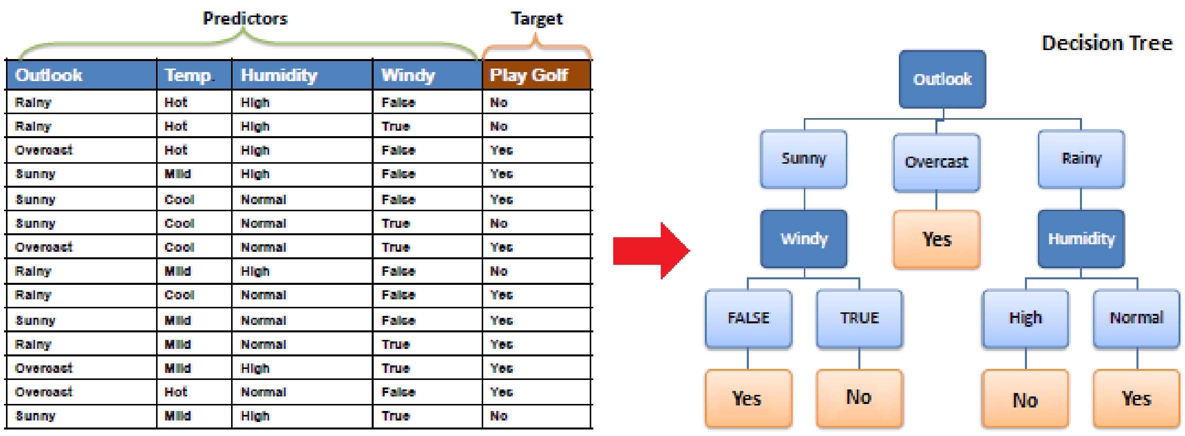
2.5.3 Logistic Regression

In statistics, logistic regression or logit regression is a type of regression analysis used for predicting the outcome of a categorical dependent variable (a dependent variable that can take on a limited number of values, whose magnitudes are not meaningful but whose ordering of magnitudes may or may not be meaningful) based on one or more predictor variables.
An explanation of logistic regression begins with an explanation of the logistic function, which always takes on values between zero and one:

$$f(t)=\frac{1}{1+e^{-t}}$$

2.5.4 Decision Tree classification

Decision tree induction is the learning of decision trees from class-labled training tuples. A desision tree is a flow chart like tree structure, where each internal nodes(non-leaf node) denotes a test on an attribute , each brunch represents an outcome of the test, each leaf node(or internal node) holds a class label. The topmost node in a tree is the root node.



Outlook	Temp.	Humidity	Windy	Play Golf
Rainy	Hot	High	False	No
Rainy	Hot	High	True	No
Overcast	Hot	High	False	Yes
Sunny	Mild	High	False	Yes
Sunny	Cool	Normal	False	Yes
Sunny	Cool	Normal	True	No
Overcast	Cool	Normal	True	Yes
Rainy	Mild	High	False	No
Rainy	Cool	Normal	False	Yes
Sunny	Mild	Normal	False	Yes
Rainy	Mild	Normal	True	Yes
Overcast	Mild	High	True	Yes
Overcast	Hot	Normal	False	Yes
Sunny	Mild	High	True	No

Figure 2.3: Example of Decision Tree

There are many algorithms developed using decision tree for classification with some differences. Some of them like BFTree, C4.8/J48, J48Graft,and SimpleCart are very popular.

2.6 Related Works

2.6.1 Regression via classification

In 2006, Bibi, Tsoumakas, Stamelos, Vlahavas, apply a machine learning approach to the problem of estimating the number of defects called Regression via Classification (RvC) [4].The whole process of Regression via Classification (RvC) comprises two important stages:

a) The discretization of the numeric target variable in order to learn a classification model,
b) the reverse process of transforming the class output of the model into a numeric prediction.

2.6.2 Static Code Attribute

Menzies, Greenwald, and Frank (MGF) [5] published a study in this journal in 2007 in which they compared the performance of two machine learning techniques (Rule Induction and Naive Bayes) to predict software components containing defects. To do this, they used the NASA MDP repository, which, at the time of their research, contained 10 separate data sets.

2.6.3 ANN

In 2007, Iker Gondra [6]used a machine learning methods for defect prediction. He used Artificial neural network as a machine learner.

2.6.4 Embedded software defect prdiction

In 2007, Oral and Bener [7] used Multilayer Perception (MLP), NB, VFI(Voting Feature Intervals) for Embedded softwaredefect prediction. there they used only 7 data sets for evaluation.

2.6.5 Association rule classification

In 2011 Baojun, Karel [3] used classification based association rule named CBA2 for software defect prediction.In these research they used assocition rule for clssafication. and they compare with other classification rules such as C4.5 and Ripper.

2.6.6 Defect-proneness Prediction framework

In 2011, Song, Jia, Ying, and Liu propased a general framework for software defect-pronness prediction. in this research they use M*N cross validation with the dataset(NASA, Softlab Dataset) for learining process. and they used 3 classification algorithms(Naive baysed, OneR, J48). and they copared with MGF [5] framework.

In 2010 a research has benn done by Chen, Sen, Du Ge, [8] on software defect prediction using datamining. In this reseach they used probabilistic Relational model and Baysean Network.

Chapter 3

Proposed Scheme

3.1 Overview Of the Framework

In General, before building defect prediction model and using them for prediction purposes, we first need to decide which learning scheme or learning algorithm should be used to construct the model. Thus, the predictive performance of the learning scheme should be determined, especially for future data. However, this step is often neglected and so the resultant prediction model may not be Reliable. As a consequence, we use a software defect prediction framework that provides guidance to address these potential shortcomings.

The framework consists of two components:

1) scheme evaluation and

2) defect prediction.

Figure 3.1 contains the details. At the scheme evaluation stage, the performances of the different learning schemes are evaluated with historical data to determine whether a certain learning scheme performs sufficiently well for prediction purposes or to select the best from a set of competing schemes.

From Figure 3.1, we can see that the historical data are divided into two parts: a training set for building learners with the given learning schemes, and a test set

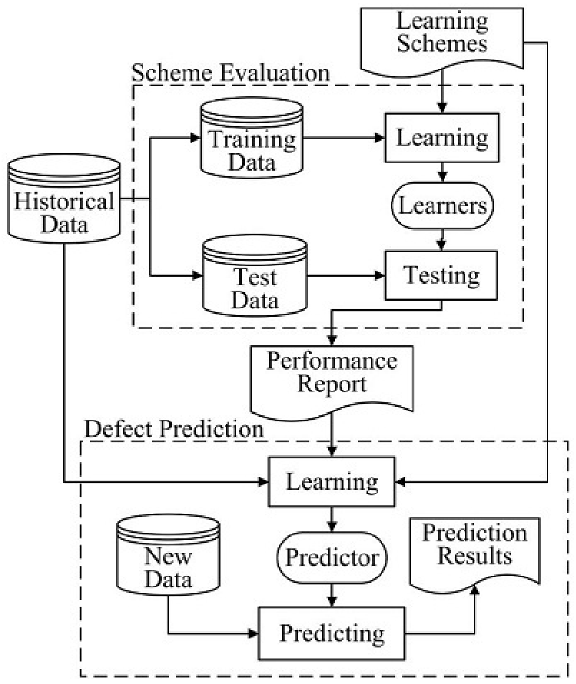


Figure 3.1: Proposed framework

for evaluating the performances of the learners. It is very important that the test data are not used in any way to build the learners. This is a necessary condition to assess the generalization ability of a learner that is built according to a learning scheme and to further determine whether or not to apply the learning scheme or select one best scheme from the given schemes.

At the defect prediction stage, according to the performance report of the first stage, a learning scheme is selected and used to build a prediction model and predict software defect. From Fig. 3.1, we observe that all of the historical data are used to build the predictor here. This is very different from the first stage; it is very useful for improving the generalization ability of the predictor. After the predictor is built, it can be used to predict the defect-proneness of new software components.

MGF proposed [5] a baseline experiment and reported the performance of the Naive Bayes data miner with log-filtering as well as attribute selection, which performed the scheme evaluation but with in appropriate data. This is because they used both the training (which can be viewed as historical data) and test (which can be viewed as new data) data to rank attributes, while the labels of the new data are unavailable when choosing attributes in practice.

3.2 Scheme Evaluation

The scheme evaluation is a fundamental part of the software defect prediction framework. At this stage, different learning schemes are evaluated by building and evaluating learners with them. The first problem of scheme evaluation is how to divide historical data into training and test data. As mentioned above, the test data should be independent of the learner construction. This is a necessary precondition to evaluate the performance of a learner for new data. Cross-validationis usually used to estimate how accurately a predictive model will perform in practice. One round of cross-validation involves partitioning a data set into complementary subsets, performing the analysis on one subset, and validating the analysis on the other subset. To reduce variability, multiple rounds of cross-validation are performed using different partitions, and the validation results are averaged over the rounds.

In our framework, an percentage split used for estimating the performance of each predictive model, that is, each data set is first divided into 2 parts, and after that a predictor is learned on 60% intances, and then tested on the remaining 40%. To overcome any ordering effect and to achieve reliable statistics, each holdout experiment is also repeated M times and in each repetition the data sets are randomized. So overall, M*N(N=Data sets) models are built in all during the period of evaluation; thus M*N results are obtained on each data set about the performance of the each learning scheme.

After the training-test splitting is done each round, both the training data and learning scheme(s) are used to build a learner. A learning scheme consists of a data preprocessing method, an attribute selection method, and a learning algorithm.

Evaluation of the proposed framework is comprised of:

1. A data preprocessor
   - The training data are preprocessed, such as removing outliers, handling missing values, and discretizing or transforming numeric attributes.
   - Here Preprocessor used-
     NASA Preprocessing Tool
2. An attribute selector
   - Here we have considered all the attributes pvovided by the NASA MDP Data Set.
3. Learning Algorithms
   - NaiveBayseSimple from bayse classification
   - Logistic classification
   - From Rule based classification
     - DecisionTable
     - OneR
     - JRip
     - PART
   - From Tree based classification–
     - J48
     - J48Graft

3.3 Scheme Evaluation Algoritm

```
   Data: Historical Data Set
   Result: The mean performance values
1  M=12 :No of Data Set
2  i=1;
3  while $i<=M$ do
4      Read Historical Data Set D[i];
5      Split Data set Intances using % split;
6      Train[i]=60% of D; % Training Data;
7      Learning(Train[i],scheme);
8      Test Data=D[i]-Train[i];% Test Data;
9      Result=TestClassifier(Test[i],Learner);
10 end
```

Algorithm 1: Scheme Evaluation

3.4 Defect prediction

The defect prediction part of our framework is straightforward; it consists of predictor construction and defect prediction. During the period of the predictor construction:

1. A learning scheme is chosen according to the Performance Report.

2. A predictor is built with the selected learning scheme and the whole historical data. While evaluating a learning scheme, a learner is built with the training data and tested on the test data. Its final performance is the mean over all rounds. This reveals that the evaluation indeed covers all the data. Therefore, as we use all of the historical data to build the predictor, it is expected that the constructed predictor has stronger generalization ability.

3. After the predictor is built, new data are preprocessed in same way as historical data, then the constructed predictor can be used to predict software defect with preprocessed new data.

3.5 Difference between Our Framework and Others

So, to summarize, the main difference between our framework and that of others in the following:
1) We choose the entire learning scheme, not just one out of the learning algorithm, attribute selector, or data preprocessor;
2) we use the appropriate data to evaluate the performance of a scheme.
—-NASA MDP Data Set [9].
3)We choose percentage split for training data set(60%) and test dataset(40%).

3.6 Data Set

We used the data taken from the public NASA MDP repository, which was also used by MGF and many others, e.g., [10], [11], [12], [13].Thus, there are 12 data sets in total from NASA MDP repository.

Table 4.1, and 4.2 provides some basic summary information. Each data set is comprised of a number of software modules (cases), each containing the corresponding number of defects and various software static code attributes. After preprocessing, modules that contain one or more defects were labeled as defective. A more detailed description of code attributes or the origin of the MDP data sets can be obtained from [5].

Table 3.1: NASA MDP Data Sets

Data Set	System	Language	Total Loc
CM1-5	Spacecraft Instrument	C	17K
KC3-4	Storage management for ground data	JAVA	8K and 25K
KC1-2	Storage management for ground data	C++	*
MW1	Database	C	8K
PC1,2,5	Flight Software for Earth orbiting Software	C	26K
PC3,4	Flight Software for Earth orbiting Software	C	30-36K

Table 3.2: Data Sets

Data Set	Attribute	Module	Defect	Defect(%)
CM1	38	344	42	1.22
JM1	22	9593	1759	18.34
KC1	22	2096	325	15.5
KC3	40	200	36	18
MC1	39	9277	68	0.73
MC2	40	127	44	34.65
MW1	38	264	27	10.23
PC1	38	759	61	8.04
PC2	37	1585	16	1.0
PC3	38	1125	140	12.4
PC4	38	1399	178	12.72
PC5	39	17001	503	2.96

3.7 Performance Measurement

The Performance measured according to the Confusion matix given in table:3.3, whis is used by many researchers e.g [14], [5]. Table 3.3 illustartes a confusion matrix for a two class problem having positive and negative class values.

Table 3.3: Confusion Matix

		Predicted Class	
		Positive	Negative
Actual class	Positive	Trure Positive	False Negative
	Negative	False Positive	True negative

Software defect predictor performance of the proposed scheme based on Accuracy, Sensitivity, Specificity, Balance, ROC Area defined as —

- Accuracy = $\frac{TP+TN}{TP+FP+TN+FN}$
  $=\frac{TruePositive+TrueNegative}{TruePositive+FalsePositive+TrueNegative+FalseNegative}$
  =The percentage of prediction that are correct.
- pd=True Positive Rate(tpr)=Sensitivity = $\frac{TP}{TP+FN}$
  =The percentage of positive labled instances that predicted as positive
- Specificity = $\frac{TN}{FP+TN}$
  =The percentage of positive labled instances that predicted as negative.
- pf=False Positive Rate(fpr)=1-specificity
  =The percentage of Negative labled instances that predicted as negative

Formal definitions for pd and pf are given in the formula. Obviously, higher pds and lower pfs are desired. The point (pd=1, pf=0) is the ideal position where we recognize all defective modules and never make mistakes.
MGF introduced a performance measure called balance, which is used to choose the optimal (pd, pf) pairs. The definition is shown bellow from which we can see that it is equivalent to the normalized euclidean distance from the desired point (0, 1) to (pf,pd) in a ROC curve.

- Balance = $1 - \frac{\sqrt{(1-pd)^2+(0-pf)^2}}{\sqrt{2}}$

The receiver operating characteristic(ROC) [15] [28], curve is often used to evaluate the performance of binary predictors. A typical ROC curve is shown in Fig. 3.2. The y-axis shows probability of detection (pd) and the x-axis shows probability of false alarms (pf).

Formal definitions for pd and pf are given above. Obviously, higher pds and lower pfs are desired. The point (pf=0, pd=1) is the ideal position where we recognize all defective modules and never make mistakes.

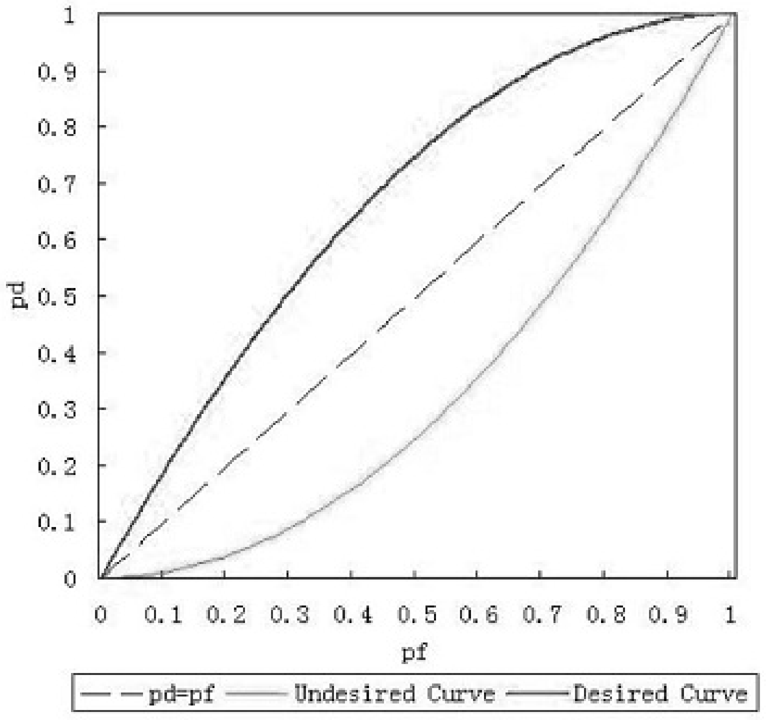


Figure 3.2: Scheme evaluation of the proposed framework

The Area Under ROC Curve (AUC) is often calculated to compare different ROC curves. Higher AUC values indicate the classifier is, on average, more to the upper left region of the graph. AUC represents the most informative and commonly used, thus it is used as another performance measure in this paper.

Chapter 4

Result Discussion

This section provides simulation results of some of the Classification algorithm techniques collected by simulation on Software tool named weka(virsion 3.6.9). In the thesis, however, proposed schemes are more comprehensively compared with competent schemes.

According to best accuracy value we choose 8 classification algorithm among many classification algorithms. All the evaluted values are collected and compare with different performance measurement parameter.

4.1 Accuracy

From the accuracy table 4.1 we can see different algorithm giving diffrent accuracy on different data set. But the average performane nearly same.
For Storage management software(KC1-3) LOG, J48G giving better Accuracy value.
For database software written in c programming language (MW1) only PART giving better accuracy value.
The performance graph is given in the figure 4.3.

Table 4.1: Accuracy

Methods	NB	LOG	DT	JRip	OneR	PART	J48	J48G
CM1	83.94	87.68	**89.13**	86.23	**89.13**	73.91	86.23	86.96
JM1	81.28	**82.02**	81.57	81.42	79.67	81.13	79.8	79.83
KC1	83.05	**86.87**	84.84	84.84	83.29	83.89	85.56	85.56
KC3	77.5	71.25	75	76.25	71.25	81.25	80	**82.5**
MC1	**94.34**	99.27	99.25	99.22	99.3	99.19	99.3	99.3
MC2	66	66.67	56.86	56.86	56.86	**70.59**	52.94	54.9
MW1	79.25	77.36	85.85	86.79	85.85	**88.68**	85.85	85.85
PC1	88.82	92.11	**92.43**	89.14	91.45	89.8	87.83	88.49
PC2	94.29	99.05	**99.37**	99.21	**99.37**	**99.37**	98.9	98.9
PC3	34.38	**84.67**	80.22	82.89	82.89	82.67	82.22	83.56
PC4	87.14	**91.79**	90.18	90.36	90.18	88.21	88.21	88.93
PC5	96.56	96.93	97.01	**97.28**	96.9	96.93	97.13	97.16

4.2 Sensitivity

From the accuracy table 4.2 we see that NB algorithm gives better performance in maximum data set.
In case of DecisionTable gives the sensitivity zero(sometimes), that means it considering all the class as a true negetive. It can not be cosider for defect prediction. LOG, OneR, PART, J48, J48G algorithms giving average performance.

Table 4.2: Sensitivity

Methods	NB	LOG	DT	JRip	OneR	PART	J48	J48G
CM1	**0.4**	0.267	0	0.2	0.133	0.333	0.2	0.2
JM1	**0.198**	0.102	0.07	0.157	0.109	0.03	0.131	0.123
KC1	**0.434**	0.238	0.197	0.328	0.254	0.32	0.32	0.32
KC3	**0.412**	**0.412**	0.118	0.118	0.176	0.353	0.353	0.353
MC1	**0.548**	0.161	0.194	0.161	0.161	0.194	0.161	0.161
MC2	**0.571**	0.545	0	0	0.091	0.5	0.045	0.045
MW1	**0.429**	0.286	**0.429**	0.143	.071	0.286	0.214	0.214
PC1	0.28	0.24	0.16	0.16	0.08	**0.36**	0.24	0.24
PC2	**0.333**	0	0	0	0	0	0	0
PC3	**0.986**	0.178	0	0.233	0.014	0.137	0.288	0.288
PC4	0.431	0.538	0.231	0.508	0.323	0.677	**0.692**	0.677
PC5	0.427	0.308	0.332	**0.521**	0.303	0.474	0.498	0.479

4.3 Specificity

From the specificity table we can see some of the algoritm are giving 100 percent specificity, that can not be cosider as there respective sensitivity zero. These algorithms can give wrong predictin.
So According to the sensitivity and specificty DecisionTable algorithm should not cosider for software defect prediction as they giving high 100% specificity bt 0% sensitivity.

Table 4.3: Specificity

Methods	NB	LOG	DT	JRip	OneR	PART	J48	J48G
CM1	0.893	0.951	1	0.943	**0.984**	0.789	0.943	0.951
JM1	0.956	0.988	0.99	0.968	0.957	**0.994**	0.954	0.956
KC1	0.898	**0.976**	0.959	0.937	0.932	0.927	0.947	0.947
KC3	0.873	0.794	0.921	0.937	0.857	0.937	0.921	**0.952**
MC1	0.947	**1**	0.999	0.999	**1**	0.999	**1**	**1**
MC2	0.724	0.759	**1**	**1**	0.931	0.862	0.897	0.931
MW1	0.848	0.848	0.924	**0.978**	**0.978**	**0.978**	0.957	0.957
PC1	0.943	0.982	**0.993**	0.957	0.989	0.946	0.935	0.943
PC2	0.946	0.997	**1**	0.998	**1**	**1**	0.995	0.995
PC3	0.219	0.976	0.958	0.944	**0.987**	0.96	0.926	0.942
PC4	0.929	0.968	**0.99**	0.956	0.978	0.909	0.907	0.917
PC5	0.983	0.99	**0.991**	0.987	0.99	0.985	0.986	0.987

4.4 Balance

looking to the Accuracy, Sensitivity and Specificty performance table we cosider the NB, LOG, JRip, OneR, PART, J48, J48G, as there performance are average. From the graph figure 4.1 we see that, in maximum of cases the OneR algorithm giving lowest balance value than others. So, no need to use for defect prediction.

Table 4.4: Balance

Methods	NB	LOG	DT	JRip	OneR	PART	J48	J48G
CM1	**0.569**	0.481	0.293	0.433	0.387	0.505	0.433	0.433
JM1	**0.432**	0.365	0.342	0.403	0.369	0.314	0.385	0.379
KC1	**0.593**	0.461	0.431	0.523	0.47	0.516	0.518	0.518
KC3	**0.575**	0.559	0.374	0.375	0.409	0.54	0.539	0.541
MC1	**0.678**	0.407	0.43	0.407	0.407	0.43	0.407	0.407
MC2	**0.639**	0.636	0.293	0.293	0.355	0.633	0.321	0.323
MW1	0.582	0.484	**0.593**	0.394	0.343	0.495	0.443	0.443
PC1	0.489	0.462	0.406	0.405	0.349	**0.546**	0.461	0.461
PC2	**0.527**	0.293	0.293	0.293	0.293	0.293	0.293	0.293
PC3	0.448	0.419	0.292	0.456	0.303	0.389	0.494	**0.495**
PC4	0.595	0.673	0.456	0.651	0.521	0.763	**0.772**	0.764
PC5	0.595	0.511	0.528	**0.661**	0.507	0.628	0.645	0.631

Depending on Accuracy, Sensitivity, Specificity, Balance performance we choosen 6 Algoritms from 8 algoritms are–

- NaiveBayesSimple
- Logistic
- JRip
- PART
- J48 and J48Graft

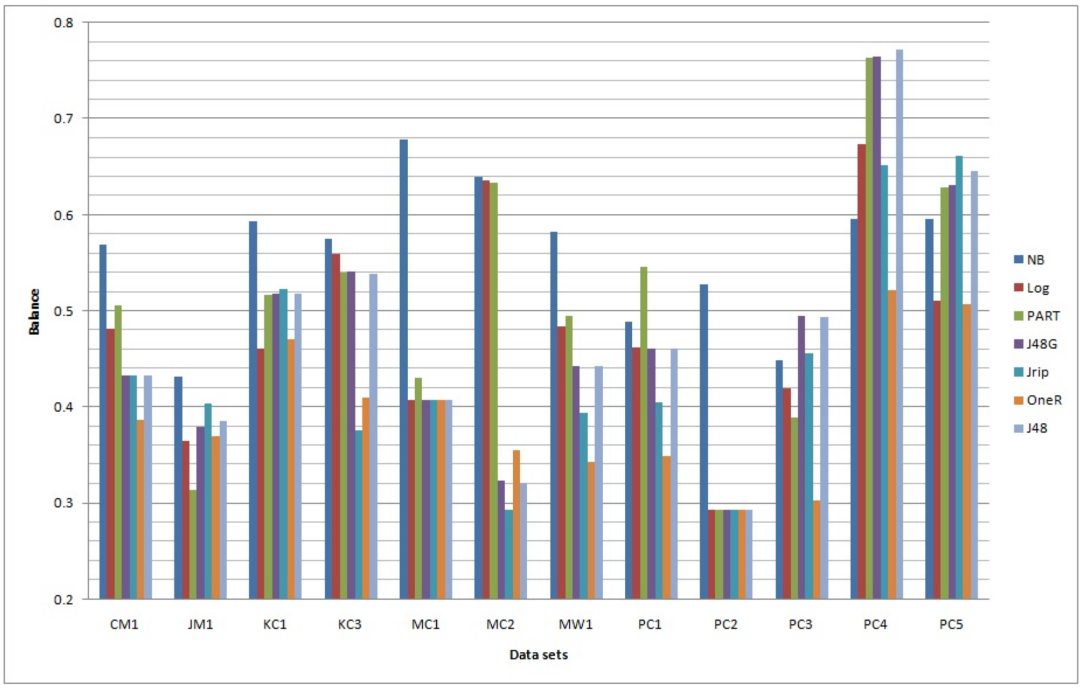


Figure 4.1: Balance

4.5 ROC Area

And the Software defect prediction performance based on ROC Area simulated by our scheme given in the table:4.5..

According to ROC Area Logistic and Nayevbased algorithm gives the better performance for software defect prediction.

Table 4.5: Comparative Performance(ROC Area) of Software defect prediction.

Methods	CM1	JM1	KC1	KC3	MC1	MC2	MW1	PC1	PC2	PC3	PC4	PC5
NB	**0.685**	0.681	0.801	**0.745**	0.861	**0.745**	**0.666**	0.736	**0.846**	0.793	0.84	0.804
Log	0.668	0.709	**0.808**	0.604	**0.893**	0.686	0.592	**0.821**	0.7	**0.802**	**0.911**	**0.958**
JRip	0.572	0.562	0.633	0.527	0.58	0.5	0.561	0.561	0.499	0.589	0.735	0.755
PART	0.492	**0.713**	0.709	0.612	0.773	0.639	0.611	0.566	0.481	0.728	0.821	0.942
J48	0.537	0.67	0.698	0.572	0.819	0.259	0.5	0.646	0.39	0.727	0.784	0.775
j48G	0.543	0.666	0.698	0.587	0.819	0.274	0.5	0.651	0.39	0.738	0.778	0.775

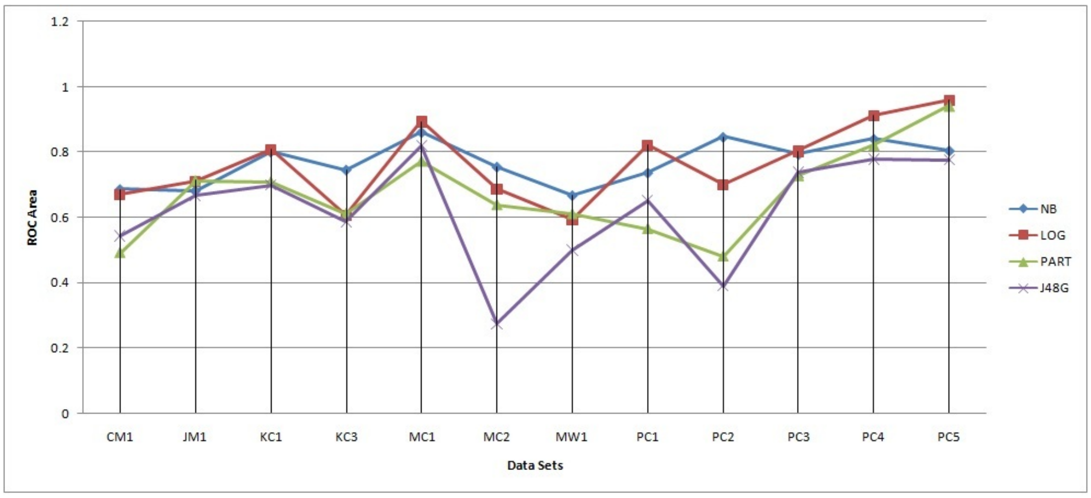


Figure 4.2: ROC Area

4.6 Comparision with other's results

- In 2011 Song, Jia, Ying, and Liu propased a general framework. In that framework they used OneR algortms for defect prediction, But that shuld no be consider for defect prediction as it gives 0 sensitivity sometimes, and balance values are very low than others.
- In 2007 MGF used considers only 10 data set, whereas in our research we used 12 data set with more modules in every data set. And in our result the balance values are also greater than there results.
- In others works different machine learning algorithms are used. In our research

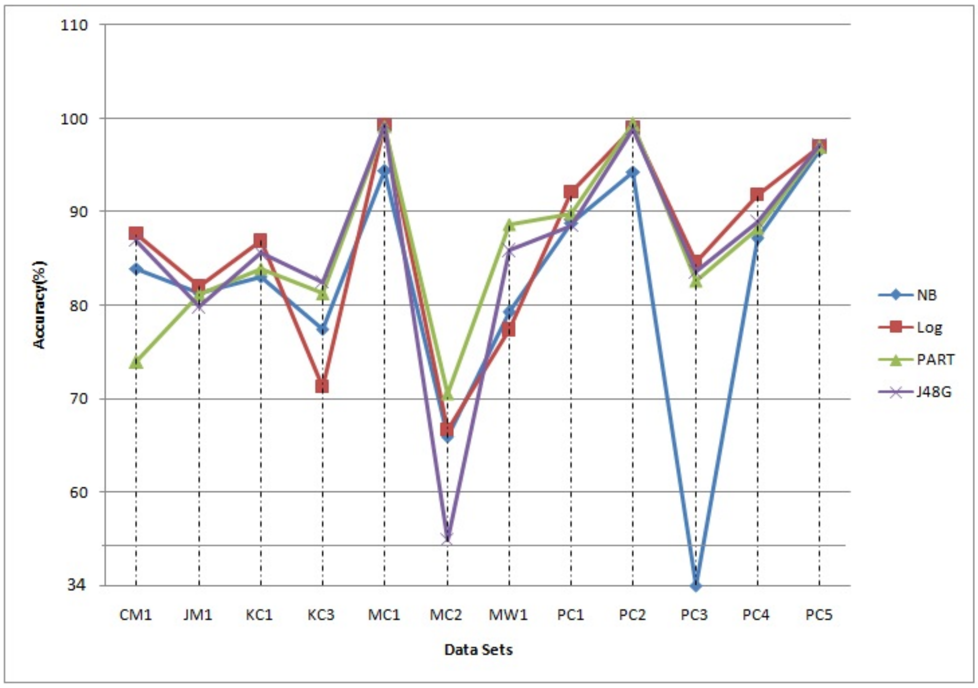


Figure 4.3: Accuracy

the reults of comparative measurement values are increases.Mainly in accuracy inceases as we used percentage split.

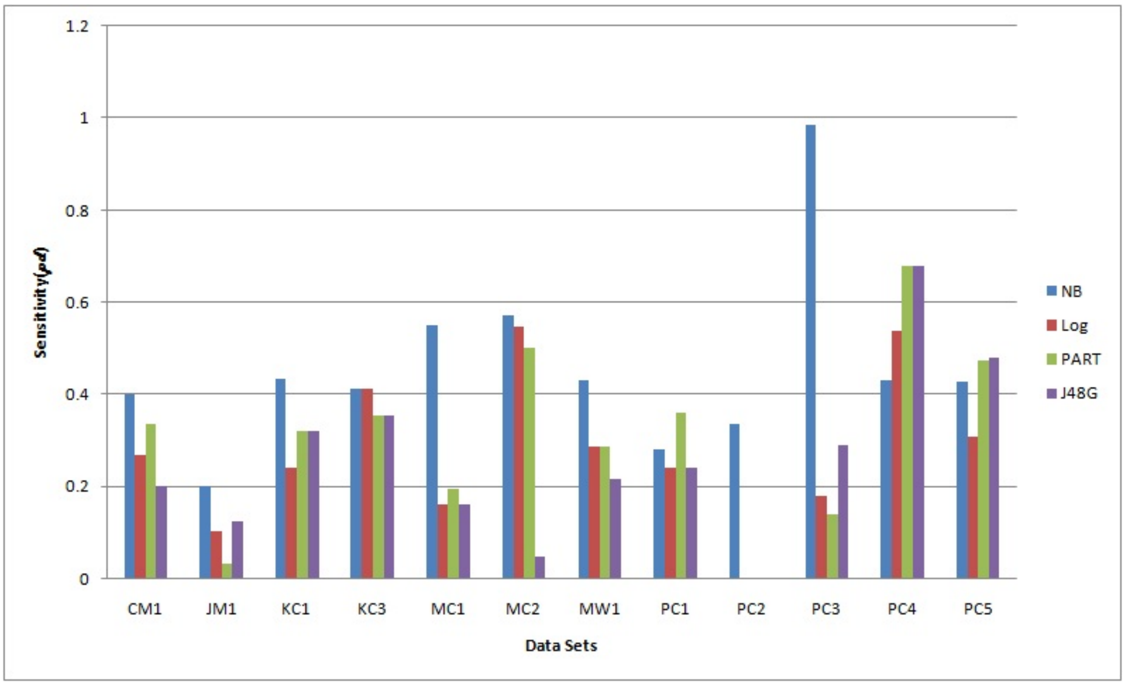


Figure 4.4: Sensitivity

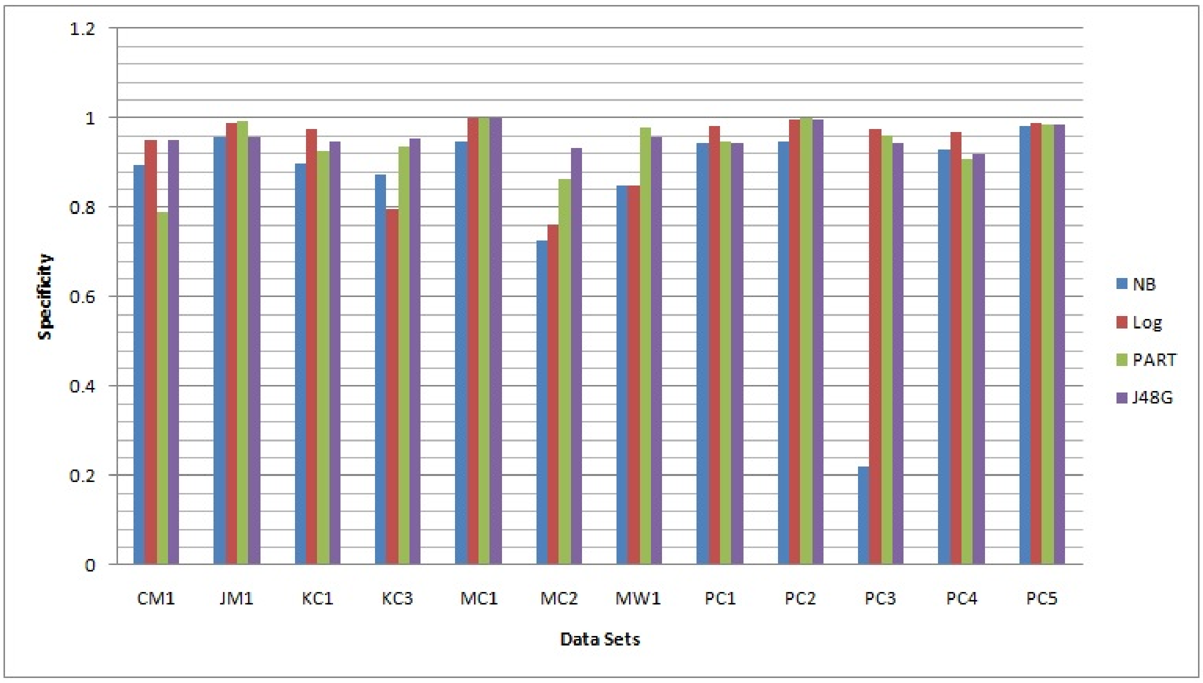


Figure 4.5: Specificity

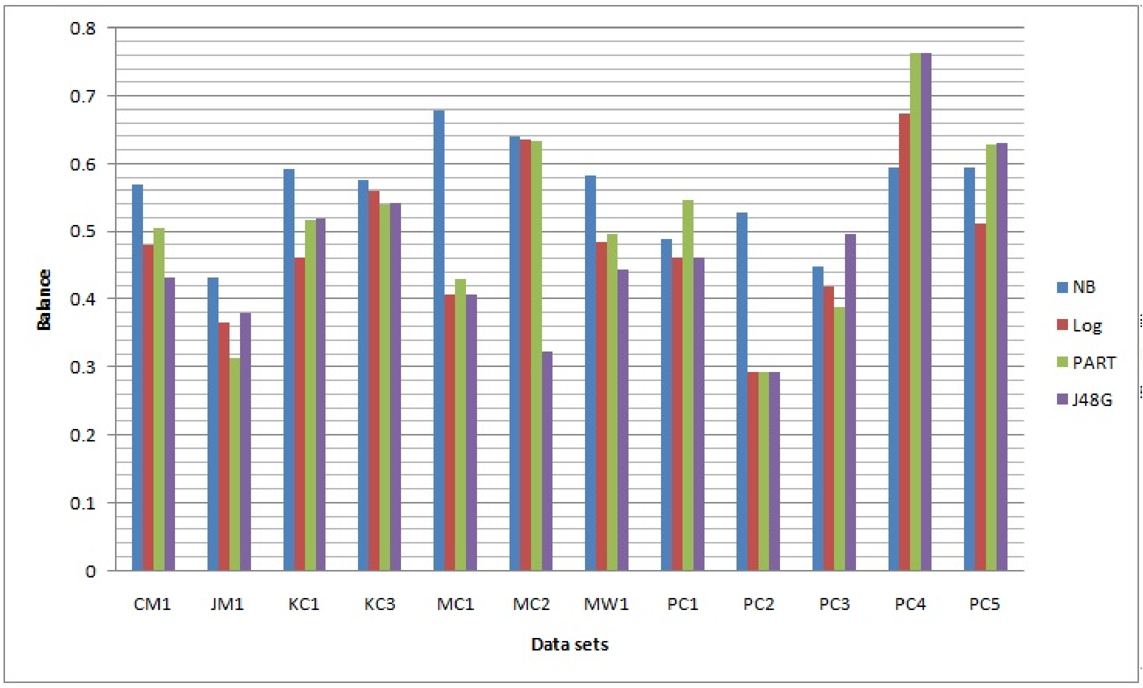


Figure 4.6: Balance

Chapter 5

Conclusion

5.1 Concluding Remarks

In our research work we have attempted to solve the Software defect prediction problem through different Data mining (Classification) algorithms.
In our research NB and Logistic algorithm gives the overall better performance for defect prediction. PART and J48 gives better performance than OneR and JRip .

From these results, we see that a data preprocessor/attribute selector can play different roles with different learning algorithms for different data sets and that no learning scheme dominates, i.e., always outperforms the others for all data sets. This means we should choose different learning schemes for different data sets, and consequently, the evaluation and decision process is important.

In order to improve the efficiency and quality of software development, we can make use of the advantage of data mining to analysis and predict large number of defect data collected in the software development. This paper reviewed the current state of software defect management, software defect prediction models and data mining technology briefly. Then proposed an ideal software defect management and prediction system, researched and analyzed several software defect prediction methods based on data mining techniques and specific models(NB, Logistic, PART, J48G)

5.2 Scope for Further Research

- Clustering based classification can be used.
- Future studies could focus on comparing more classification methods and improving association rule based classification methods
- Furthermore, the pruning of rules for association rule based classification methods can be considered.

Bibliography

[1] Tao Xie, Suresh Thummalapenta, David Lo, and Chao Liu. Data mining for software engineering. *Computer*, 42(8):55–62, 2009.

[2] Qinbao Song, Zihan Jia, Martin Shepperd, Shi Ying, and Jin Liu. A general software defect-proneness prediction framework. *Software Engineering, IEEE Transactions on*, 37(3):356–370, 2011.

[3] Ma Baojun, Karel Dejaeger, Jan Vanthienen, and Bart Baesens. Software defect prediction based on association rule classification. *Available at SSRN 1785381*, 2011.

[4] S Bibi, G Tsoumakas, I Stamelos, and I Vlahavas. Software defect prediction using regression via classification. In *IEEE International Conference on*, pages 330–336, 2006.

[5] Tim Menzies, Jeremy Greenwald, and Art Frank. Data mining static code attributes to learn defect predictors. *Software Engineering, IEEE Transactions on*, 33(1):2–13, 2007.

[6] Iker Gondra. Applying machine learning to software fault-proneness prediction. *Journal of Systems and Software*, 81(2):186–195, 2008.

[7] Ataç Deniz Oral and Ayşe Başar Bener. Defect prediction for embedded software. In *Computer and information sciences, 2007. iscis 2007. 22nd international symposium on*, pages 1–6. IEEE, 2007.

[8] Yuan Chen, Xiang-heng Shen, Peng Du, and Bing Ge. Research on software defect prediction based on data mining. In *Computer and Automation Engineering (ICCAE), 2010 The 2nd International Conference on*, volume 1, pages 563–567. IEEE, 2010.

[9] Martin Shepperd, Qinbao Song, Zhongbin Sun, and Carolyn Mair. Data quality: Some comments on the nasa software defect data sets. 2013.

[10] Stefan Lessmann, Bart Baesens, Christophe Mues, and Swantje Pietsch. Benchmarking classification models for software defect prediction: A proposed framework and novel findings. *Software Engineering, IEEE Transactions on*, 34(4):485–496, 2008.

[11] Yue Jiang, Bojan Cukic, and Tim Menzies. Fault prediction using early lifecycle data. In *Software Reliability, 2007. ISSRE'07. The 18th IEEE International Symposium on*, pages 237–246. IEEE, 2007.

[12] Yue Jiang, Bojan Cuki, Tim Menzies, and Nick Bartlow. Comparing design and code metrics for software quality prediction. In *Proceedings of the 4th international workshop on Predictor models in software engineering*, pages 11–18. ACM, 2008.

[13] Hongyu Zhang, Xiuzhen Zhang, and Ming Gu. Predicting defective software components from code complexity measures. In *Dependable Computing, 2007. PRDC 2007. 13th Pacific Rim International Symposium on*, pages 93–96. IEEE, 2007.

[14] Gustavo EAPA Batista, Ronaldo C Prati, and Maria Carolina Monard. A study of the behavior of several methods for balancing machine learning training data. *ACM SIGKDD Explorations Newsletter*, 6(1):20–29, 2004.

[15] Charles E Metz, Benjamin A Herman, and Jong-Her Shen. Maximum likelihood estimation of receiver operating characteristic (roc) curves from continuously-distributed data. *Statistics in medicine*, 17(9):1033–1053, 1998.

[16] Qinbao Song, Martin Shepperd, Michelle Cartwright, and Carolyn Mair. Software defect association mining and defect correction effort prediction. *Software Engineering, IEEE Transactions on*, 32(2):69–82, 2006.

[17] Norman E. Fenton and Martin Neil. A critique of software defect prediction models. *Software Engineering, IEEE Transactions on*, 25(5):675–689, 1999.

[18] Naeem Seliya and Taghi M Khoshgoftaar. Software quality estimation with limited fault data: a semi-supervised learning perspective. *Software Quality Journal*, 15(3):327–344, 2007.

[19] Frank Padberg, Thomas Ragg, and Ralf Schoknecht. Using machine learning for estimating the defect content after an inspection. *Software Engineering, IEEE Transactions on*, 30(1):17–28, 2004.

[20] Venkata UB Challagulla, Farokh B Bastani, I-Ling Yen, and Raymond A Paul. Empirical assessment of machine learning based software defect prediction techniques. In *Object-Oriented Real-Time Dependable Systems, 2005. WORDS 2005. 10th IEEE International Workshop on*, pages 263–270. IEEE, 2005.

[21] Norman Fenton, Paul Krause, and Martin Neil. A probabilistic model for software defect prediction. *IEEE Trans Software Eng*, 2001.

[22] Raimund Moser, Witold Pedrycz, and Giancarlo Succi. A comparative analysis of the efficiency of change metrics and static code attributes for defect prediction. In *Software Engineering, 2008. ICSE'08. ACM/IEEE 30th International Conference on*, pages 181–190. IEEE, 2008.

[23] Ganesh J Pai and Joanne Bechta Dugan. Empirical analysis of software fault content and fault proneness using bayesian methods. *Software Engineering, IEEE Transactions on*, 33(10):675–686, 2007.

[24] Giovanni Denaro, Sandro Morasca, and Mauro Pezzè. Deriving models of software fault-proneness. In *Proceedings of the 14th international conference on Software engineering and knowledge engineering*, pages 361–368. ACM, 2002.

[25] Ling-Feng Zhang and Zhao-Wei Shang. Classifying feature description for software defect prediction. In *Wavelet Analysis and Pattern Recognition (ICWAPR), 2011 International Conference on*, pages 138–143. IEEE, 2011.

[26] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. The weka data mining software: an update. *ACM SIGKDD Explorations Newsletter*, 11(1):10–18, 2009.

[27] DMW Powers. Evaluation: From precision, recall and f-measure to roc., informedness, markedness & correlation. *Journal of Machine Learning Technologies*, 2(1):37–63, 2011.

[28] Mark H Zweig and Gregory Campbell. Receiver-operating characteristic (roc) plots: a fundamental evaluation tool in clinical medicine. *Clinical chemistry*, 39(4):561–577, 1993.