Empirical validation of Object Oriented metrics using Machine Learning Methods

A dissertation submitted in the partial fulfillment for the award of Degree of

Master of Technology

In

Software Engineering

Submitted by

Kapil Sharma (2K11/ST/09)

COMPUTER ENGINEERING DEPARTMENT

DELHI TECHNOLOGICAL UNIVERSITY

BAWANA ROAD, DELHI

July 2014

CERTIFICATE

Date: _____

This is to certify that the Major Project entitled "**Empirical validation of Object Oriented metrics using Machine Learning Methods"** submitted by **KAPIL SHARMA**, Roll Number: **2K11/ST/09**; in partial fulfillment of the requirement for the award of degree Master of Technology in Software Technology to Delhi Technological University, Bawana Road Delhi; is a record of the candidate's own work carried out by him under my supervision. The matter embodied in this thesis is original and has not been submitted for the award of any other Degree.

Dr. Ruchika Malhotra
Asst. Professor, Computer Engineering Dept.
Delhi Technological University
Bawana road, Delhi - 110042

ACKNOWLEDGEMENT

July 2014

I would like to take this opportunity to thank my project guide Dr. RUCHIKA MALHOTRA for her invaluable and consistent guidance throughout this work. I would like to thank her for giving me the opportunity to undertake this topic. I am very appreciative of her generosity with her time, advice, data, and references, to name a few of his contributions. It is her wonderful association that enabled me to achieve the objectives of this work. I humbly extend my grateful appreciation to my friends whose moral support made this study possible.

Lastly, I would like to thank all the people directly and indirectly involved in successfully completion of this project

Kapil Sharma

2K11/ST/09

Master of Technology (Software Technology)

Delhi Technological University

Table of contents

List of Figures

List of Tables

ABSTRACT

Complex and advanced software systems are more prone to faults and result in greater maintenance cost in later stages of software development cycle. With the help of this study we suggest the importance of machine learning algorithms in detection of fault proneness in software systems results in early stages of software development life cycle.

We concentrated on the use of machine learning methods and used them for empirically validating object-oriented design metrics, Chidamber et al. [1], for the purpose of predicting fault proneness. We have used open source project developed in Java language, "MX4J" and "Synapse 1.2", as the base of our empirical study. The defect prediction models developed using machine learning methods are used to compute and evaluate performance of these models.

We evaluated the performance using Receiver Operating Characteristic (ROC) analysis. We used tools such as Weka and SPSS for the purpose of generating data distribution and ROC curve. As per the ROC analysis for both the projects, machine learning methods LogitBoost and Bagging show better performance as compared to other machine learning methods.

Chapter 1

INTRODUCTION

Given the complex nature of software products being developed these days, accompanied by tight deadlines, it is not so unobvious for defects to ship in the final product. Owing to stiff competition between competitors and pressure to quickly launch and hit the market, the defects presence is increasing. It is quiet common to see fault prediction models being applied to predict software faults before the product is delivered to market. These models help managers to align resources to critical areas of the project and also help developers to concentrate on problem areas more prone to critical faults. This in turn will result in high quality and timely delivery of software products.

It requires experts with best in class domain knowledge to predict and assist in fault proneness. Through this study we focus on evaluating the relation of OO metrics, CK metric, with fault proneness using machine learning methods. We also focus on figuring out which of the machine learning methods assist in defect classification. With application on open source Java based projects, we share important insight on usage of machine learning methods and logistic regression for determining fault proneness in object oriented software.

1.1 Goal of the thesis

The goal of the work in this thesis is summarized below:

- We used six machine learning methods for empirical validation of object oriented metrics with the purpose of predicting fault proneness using regression and machine learning methods.
- We applied the study on two projects with different code size and complexities to generalize the behavior.

1.2 Organization of the Thesis

The thesis is organized as follows:

Chapter 2 provides the survey of existing literature in the field of fault proneness.

Chapter 3 discusses the machine learning paradigms, the concepts of machine classification techniques. We also explain the process of data collection and highlight the important aspects of the datasets used in our study. We will also touch upon the concept of metrics and describe the various metrics selected for this study. And finally, we point down the dependent and independent variables.

Chapter 4 explains the research methodology being followed upon in this study. It describes the phases involved in the research methodology. It also introduces the Log Parser tool and finally mentions the machine learning algorithms used to evaluate the results.

Chapter 5 presents a detailed analysis of the results obtained. In this chapter we compare and assess the results after applying the machine learning algorithms on the two datasets. The machine learning methods applied are Bagging, LogitBoost, RandomForest, NaiveBayes, AdaBoost and Logistic.

Chapter 6 presents the conclusions of the thesis and future work.

Chapter 2

RELATED WORK

Several researches have been done targeting relation of OO metrics with fault proneness. This research is directed to assist senior managers in planning resources and helping to focus on fault-prone areas.

Singh et al. [2], in their work, focused developing prediction models using regression and machine learning methods. Through ROC analysis they showed that medium and low severity faults are easier to predict than high severity faults. Through this work they also showed that machine learning methods have better performance as compared to logistic regression methods.

Aggarwal et al. [3], provided empirical evidence to draw the strong conclusion that many metrics capture and provide redundant information and that a subset of metrics can help to identify faulty classes with more than 90% accuracy. The study also confirms results from previous studies and shows that import coupling along with size metrics are related to fault proneness.

Shatnawi & Li [4], empirically investigated class error probability in the post-release evolution process to answer aspects such as if software metrics can predict error post-release system and if it can be related to severity of errors and to identify the classes based on severity of errors. It was concluded that it is desirable to seek alternative methods to locate error-prone classes for higher accuracy.

Zhou & Leung [5], in their study concluded that design metrics and fault proneness are related while considering the severity of faults and low severity faults are easily predicted as compared to high severity faults in fault-prone classes.

The sooner the defects can be predicted, the better. Emam et al. [6] in their study showed that prediction models using design metrics are able to identify faulty classes if used in early stages of development and that export coupling metric has the strongest association with fault proneness.

Fault-Proneness can be predicted using highly iterative development process. It can also be linked to agile software development processes. Olague et al.[8], in their research, empirically validated three OO metrics suites namely CK metrics, MOOD metrics and QMOOD metrics suite. CK and QMOOD suites were able to produce similar statistical models that could efficiently help in identifying error prone classes whereas MOOD metrics suite is not so good in predicting fault proneness in classes.

In modern era, the use of machine learning to predict software fault-proneness is increasing. Iker Gondra [9], in his study, proposed the use of machine learning by training Artificial Neural Network (ANN). Software metrics are identified based on their criticality using ANN. In this study, he used Support Vector Machines (SVM) as the tool to determine the classification whether a module is contains fault or is fault free. The use of sensitivity analysis for selecting software metrics indicates the existence of errors. The experimental results showed that SVMs are a better defect classifier when compared to ANNs.

Khoshgoftaar et. al [10], worked on a software fault prediction model with relation to case-based reasoning (CBR). CBR is a part of the computational intelligence field focusing on

automated reasoning processes. CBR models are seen to have better prediction capability as compared to multiple linear regressions.

Zhou et. al [12], in his study, focused on complexity metrics in order to determine fault-proneness. Complexity metrics were shown as moderate predictors in differentiating whether a class contains error or is error-free. It was shown through experimental results that LOC and WMC exceeded SDMC and AMC in predicting fault proneness.

Chapter 3

RESEARCH BACKGROUND

In this chapter, first we will discuss about the machine learning paradigms. Second, we will discuss the concepts of machine classification techniques. Third, we explain the process of data collection and highlight the important aspects of the two datasets used in this study. Fourth, we will touch upon the concept of metrics and describe the various metrics selected for this study in detail. Fifth and finally, we point down the dependent and independent variables.

3.1 Machine Learning Paradigms

3.1.1 Probabilistic Models

In Probabilistic Models the probability of each class and features are recorded with the help of the training data set. The outcome of the new data or its classification is based on these probabilistic models. One of the examples of Probabilistic Modelling is the Bayesian Model.

3.1.2 Symbolic Learning and Rule Induction

In Symbolic Learning and Rule Induction the algorithms learn by being told and looking at examples. ID3 algorithm developed by Quinlan is one of them.

3.1.3 Neural Networks

In Neural Networks, the data and its output (nodes) are inter-connected in a web like structure through programming constructs which mimic the function of the neurons in the human brain. Based on these, when new data is place in one of the nodes, its output can be predicted or it can be classified accordingly.

3.1.4 Analytic Learning and Fuzzy Logic

Analytic Learning and Fuzzy Logic normally have logical rules which are used to create even more complex rules. Based on these logical rules it looks for truths ranged between 0 and 1.

3.1.5 Evolution Based Models

Evolution based models are based on Darwin's theory of Natural Selection and are divided further into a) Genetic algorithms, b) Evolution strategies and c) Evolutionary programming.

3.2 Classification techniques

3.2.1 Supervised Learners

Algorithms that learn from looking at input/output matches of training data to find results for new data (like the ID3 algorithm).

3.2.2 Unsupervised Learners

There are no training data sets, the algorithms learn by looking at the input patterns and predict the result.

3.2.3 Reinforcement Learners

These algorithms observe the state and make predictions. Each prediction are rewarded or punished according to the accuracy. The algorithm then learns how to make the right decision.

3.3 Empirical Data Collection

We focused on data sets from different projects MX4J [13] and Synapse 1.2 [14]. The data was taken from the public domain projects available on sourceforge.net. In this section we will explain the data source MX4J and the process for data collection which is common to both the projects. MX4J is an implementation of the following Java specification requests (JSR),

- JSR 3 - Java Management Extensions technology (JMX), and
- JSR 160 - Java Management Extensions technology Remote API

JMX is not a mandatory package. Detailed information about this project can be checked at the following URL: http://sourceforge.net/projects/mx4j/

The MX4J project adheres to the specification and provides its reference implementation. The JMX specification aims to support developers by providing APIs and interfaces that assist in designing loosely coupled modules and robust codebase. It finds it usage for monitoring applications and management of network services.

3.3.1 Login Command:

cvs -d:pserver:anonymous@mx4j.cvs.sourceforge.net:/cvsroot/mx4j login

This generates the login for our repository on the mentioned site. Press enter when the command prompt asks for password.

3.3.2 Local Repo Command:

Following command downloads and creates local repository of the project.

cvs -z3 –d : pserver:anonymous@mx4j.cvs.sourceforge.net: /cvsroot/mx4j co -P modulename

Here *modulename* refers to the module you want to create the local repository.

3.3.3 Generate Log files Command:

cvs log –d : pserver:anonymous@mx4j.cvs.sourceforge.net: /cvsroot/mx4j co -P modulename > commitlog.txt

This command generates the log file commitlog.txt

3.3.4 Project Statistics

The Understand tool generated detailed statistics for the open source project MX4J. Some of the statistics of interest are as follows:

Table 3.1: Open source project (MX4J) statistics

Project Statistics	
Total records:	8808
Files Count:	770
Public Class:	595
Class:	193
Public Abstract Class:	49

The table below contains instance count for each of the class types considered for this study.

Table 3.2 Table for instance count of class types for MX4J project

No.	**Label**	**Count**
1	Class	32
2	Public_Class	587
3	Public_Abstract_Class	49

3.4 Metrics Used

For the purpose of this study we have chosen CK metrics which are defined at the class level. These metrics are known to incorporate the object oriented behavior and can be easily measured using some common and well known tools, such as Understand. We have also used SLOC, number of lines of code as a metric in this study. Table 3.3 below provides definitions of these metrics.

Table 3.3 Metrics Used

Metrics	Definition
Coupling between objects (CBO)	CBO defines the coupling between the object and denotes the count of couple classes.
Response for a class (RFC)	This metric denotes the count of functions in a class and also the count of functions available for access to sub classes in class inheritance.
Lack of cohesion (LCOM)	This metric denotes the usage of member data in a class by the member functions in the class.
Number of children (NOC)	This denotes the concept of inheritance and stands for the total number of child classes of a given class.
Depth of inheritance (DIT)	This metric denotes the level count from the root to the given node level. The root is generally considered at level 0.

Weighted methods per class (WMC)	This denotes the aggregation of complexities of all functions in the given class.
Source lines of code (SLOC)	This denotes the count of lines of code in the source code.

3.5 Dependent and independent variables

Independent variable is defined as the variable that is manipulated and dependent variable is defined as the response that is measured. Independent variable can also be defined as the presumed cause and dependent variable can also be defined as the presumed effect.

In broader terms, independent variable is the antecedent whereas dependant variable is the consequent.

With respect to experimental research, independent variables is the variable that is to be manipulated in the research, and dependent variable is observed or measured for variation as a presumed result of the variation in the independent variable.

In non-experimental research, where there is no experimental manipulation, the independent variable is the variable that 'logically' has some effect on a dependent variable.

For the purpose of our study, defect presence is the dependent variable whereas all other metrics form the independent variables. We would check the effect of object oriented metrics and fault proneness after applying machine learning methods.

Chapter 4

RESEARCH METHODOLOGY

In this chapter we explain the research methodology followed in this study. We describe the phases involved in the research methodology. We will introduce the Log Parser tool we developed to assist this study. Finally we brief on the machine learning algorithms we used to evaluate the results.

4.1 Methodology

Figure 4.1 provides an outline of the methodology used in this study.

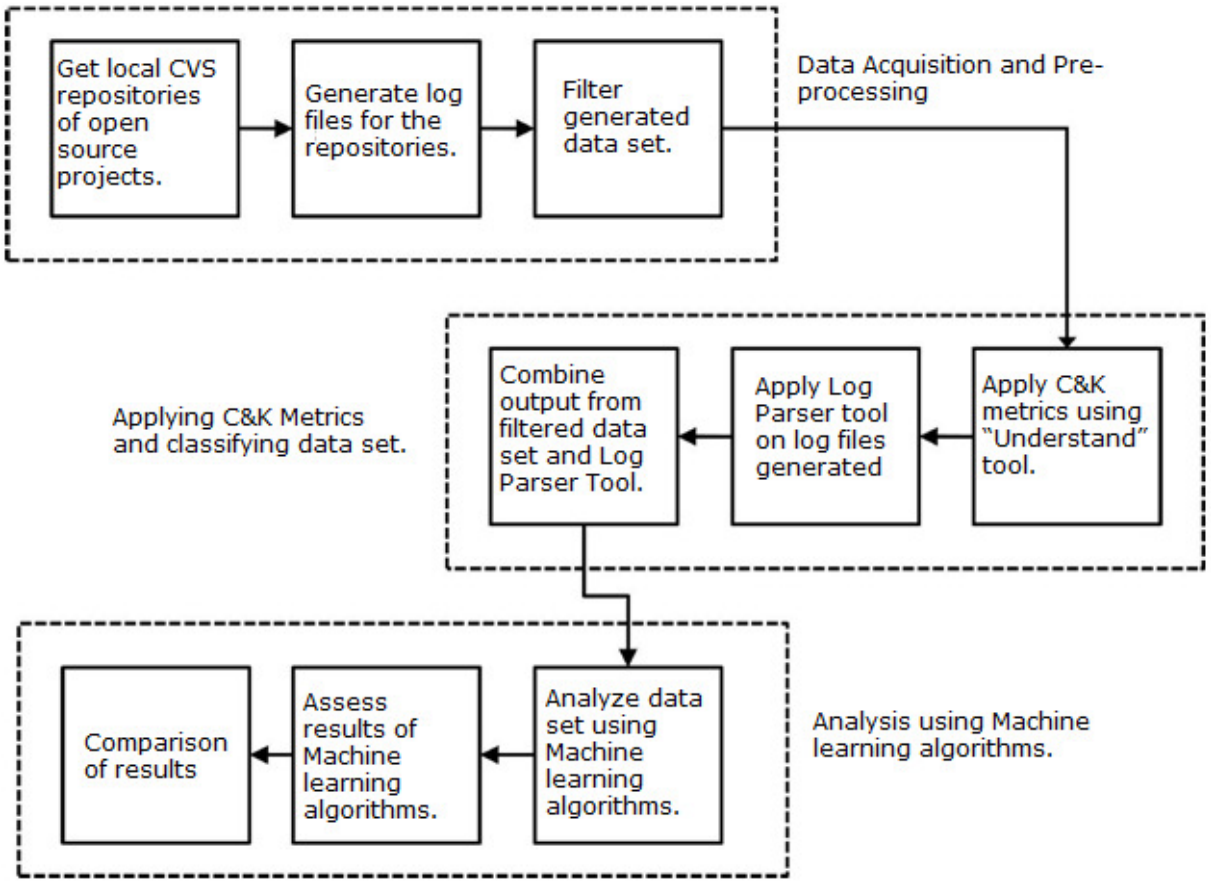


Figure 4.1: Outline of Research Methodology

As shown in the figure above, the entire process can be divided into three parts,

4.1.1 Data acquisition and pre-processing.

In this step the empirical data is collected. Data is fetched from online CVS repositories of open source projects. From these local repositories, we generate log files. Some filtering is also performed on the generated data set.

4.1.2 Applying C&K Metrics and classifying data set.

We apply C&K metrics using the Understand tool on the local repositories. Also we apply Log Parser tool on log files generated in first step. As a last step of the phase, we combine output from filtered data set and Log Parser Tool.

4.1.3 Analysis using Machine learning algorithms.

We apply the Machine learning algorithms to the filtered data set and perform analysis. As a last step of this phase, we perform comparison of results of different machine learning algorithms.

4.1.4 Filtering Data set

Filtering data set is common to phase I and II of our research methodology. We used the Understand tool to apply C&K Metrics to the open source code. It produces a detailed output metrics which is further filtered to match our requirements. For the scope of our study we have concentrated on following classes only:

- Class
- Public Class
- Public Abstract Class

Other class types were excluded as they didn't find their presence in the log files generated using CVS Log command.

Following attributes are included in the CSV file generated by the Understand tool. The CK [1] metrics are selected from the Understand tool.

Table 4.1 Mapping between Metrics used and Understand tool

Undertand Fields	**Metrics Used**
Kind	Not applicable
Name	Not applicable
CountLineCode	Source lines of code (SLOC)
MaxInheritanceTree	Depth of Inheritance Tree (DIT)
CountDeclMethod	Weighted Methods Per Class (WMC)
CountClassCoupled	Coupling between Object Classes (CBO)
CountClassDerived	Number of Children (NOC)
PercentLackOfCohesion	Lack of Cohesion in Methods (LCOM)
CountDeclMethodAll	Response for a Class (RFC)

The DefectPresent info is captured using the Log Parser tool. This helps to specify whether a given class contains an issue or not.

4.2 Log Parser Tool

We have developed “Log Parser” tool using Visual studio 2008, and is composed of two projects.

- a C# based project – responsible for the UI part of the Tool, and
- a C++ DLL project – responsible for the business logic of the tool

The tool takes log file, generated using CVS Log command, as input. The tool then parses the log file and stores the parsed information in separate files. The information stored is Class Name, Defect present info, Defect count and Bugs Information.

The files created are namely,

- LogFile_BugInfo.txt
- LogFile_DefectCount.txt
- LogFile_DefectInfo.txt

The tool is capable of parsing CVS log files, and generate information for the defect count and bugs ID. It can arrange classes as per ascending/descending order of number of defects.

For the scope of this study, we have used the defect classifying property of this tool to classify whether a given class contains issues or not.

A snapshot of the tool used is given below.

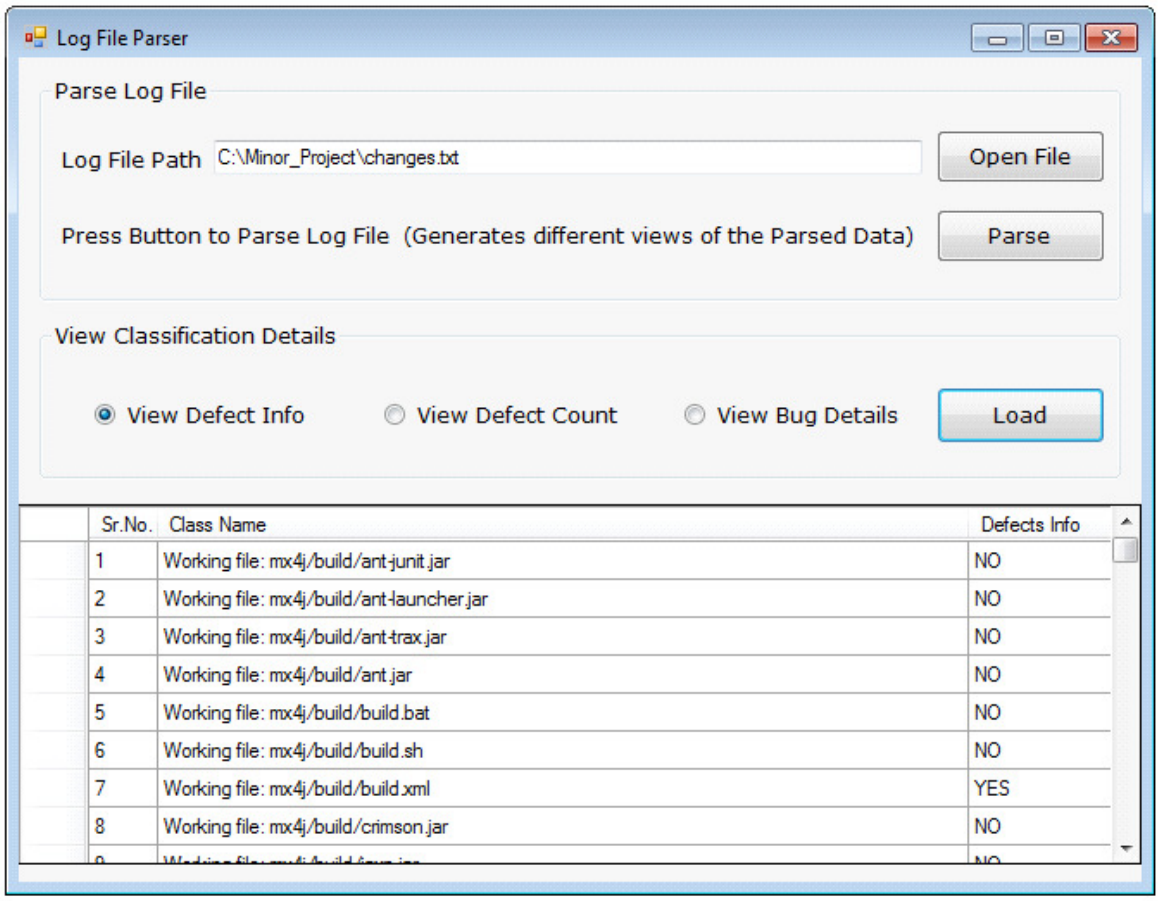


Figure 4.2 Log File Parser

4.3 Machine Learning Algorithms for Analysis

4.3.1 Naive Bayes Classifier

The Naïve Bayes Classification is named after Thomas Bayes, who proposed the Bayes Theorem. The Bayesian Classification is known as a supervised learning method and as a statistical method that can be used for classification.

This classification technique is based on the principle of probability and helps in determining the uncertainty involved. Bayesian Classification provides a base for several machine learning algorithms and helps in evaluation of a number of algorithms.

Some of the most common uses of Naive Bayes classification are as under:

- Text classification

  As mentioned above, this classification is a probability base model and supports learning. Naive Bayes classification is fundamental in classifying and well known successful algorithms for text classification.

- Spam filtering

  Spam filtering can be considered as a sub task of the test classification method discussed above. This again uses the well known classification technique for identifying spam e-mail. Due to its increasing usage and advantages, this is used as a component in several email systems, both corporate and personal email systems. This also comes as a side utility to support filtering of email. The utility can be embedded both in client side as well as server side in an email system.

- Recommender Systems

This application of Naive Bayes Classifier is very common these days, especially on e-commerce websites. It is used in combination with data mining methods to form Recommender Systems. Both machine learning methods and data mining techniques are applied to remember and focus on user's preference and help in predicting the probable choice of selection for an end user. Despite its simplicity, it is a well performing model in real world applications.

Naïve Bayes classification is easy to apply and provides quick first results. It considers every attribute in each class as a separate entity and helps in evaluating the probabilities with normal distributions. Naïve Bayes has many successful applications, e.g., email filtering etc.

4.3.2 LogitBoost

LogitBoost is a boosting scheme which was proposed by Jerome Friedman, Trevor Hastie and Robert Tibshirani. Boosting is a process of applying a classification algorithm to the training instances, reweighting them again and again, and then taking a majority vote of the number of classifiers thus produced. LogitBoost algorithm takes AdaBoost algorithm as a additive model and applies the cost functional of logistic regression. LogitBoost is suitable for problems involving two class situations.

As mentioned above, LogitBoost is a boosting algorithm generally used for predictive classification and is considered to be sensitive to outliers. It is also considered to be agood ensemble learning methods. Ensemble learning denotes the concept where several models combine to produce better results as compared to individual results. When compared with

AdaBoost, LogitBoost proves to be more robust against noisy data. Reason being LogistBoost uses the binomial log which has the tendency to grow in linear manner whereas AdaBoost uses an exponential function.

4.3.3 Bagging

Bagging, an acronym for Bootstrap Aggregating, was introduced by Breiman. Idea was to combine classification techniques and to improve the overall classification results. Another example for machine learning ensemble method, it aims at combining multiple predictors and improving classification. Being a meta-algorithm, it uses aggregation to average out the results of several bootstrap samples. Thus, Bagging can be viewed as a combination of Bootstrapping and aggregation.

Bagging = Bootstrapping + Aggregation

Discussing the Predictors, Predictors can be classifiers such as Decision Trees or estimators such as Regression trees or some other parsers.

The bagging algorithm can then be briefed as follows:

Let the training data be defined as T,

Repeat following process N times:

- Fetch a bootstrap sample T_k from T.
- Now train the predictor using T_k.

As a second step, we need to combine/aggregate N predictors by

- Voting, generally used in case of classification problems, and/or
- Averaging, which is generally used in case of estimation problems

4.3.4 Random Forest

The term Random forest, “randomized decision forests”, was introduced by Tin in 1995 and the algorithm was induced by Breiman in 1999. The idea is to combine randomness and the concept of bagging to produce a forest of trees but with a controlled variant. RandomForest is considered as modification of bagging with trees designed to reduce correlation as during the course of training, it optimizes split over dimensions by choosing different subset of dimensions. It then chooses the best split amongst the possible splits. Here the important point is the selection of subset which is chosen randomly from the given dimensions. Lets discuss the algorithm in brief below:

- Decide on the number of input variables that would be used to determine the selection criteria at the node. Let this count be n.
- Create bootstrap samples from the data set.
- Now comes the training part. Let N denote the total number of features or total number of input variables. Idea is to train the bootstrap sample by randomly selecting n<<N input features at each node and then decide on the best spilt among the selected n input features.
- For the purpose of prediction, the best bootstrap needs to be appended at the bottom of the tree as a leaf node and the response would be leaf value. Follow the same process with other bootstrap samples with the response with higher number of votes resulting as our prediction value.

4.3.5 AdaBoostM1

AdaBoost, an acronymt for "Adaptive Boosting", is a meta-algorithm that combines several other

machine learning algorithms with the aim of producing better performance. AdaBoost is considered to be sensitive to outliers. Adaboost helps in reducing variance of classifiers and focus on feature selection that assists in decision making.

4.3.6 Logistic

Logistic regression is a methods commonly used for depicting and analyzing relationship between independent and dependent variables. Logistic regression helps to predict the occurrence of an event and its division amongst the groups that are depicted by the dependent variable.

The value predicted by logistic regression is within the range of 0.0 and 1.0. Depending upon the value of the cut-off point the subject is classified for any of the relevant groups. Logistic regression is used as a base for several machine learning algorithms as it provides quick and easy statistics for easy reference.

Chapter 5

RESULTS

In order to evaluate the results of the thesis following measures have been used in this work.

Sensitivity: Sensitivity is defined as the ratio of the number of correctly predicted defective classes to the total number of actual defective classes.

Specificity: Specificity is defined as the ratio of the number of correctly predicted non-defective classes to the total number of actual non-defective classes.

Receiver Operating Characteristic Curves (ROC): ROC is constructed between 1-Specificity as X-axis and sensitivity as Y-axis. Area Under the Curve (AUC) is used to analyze the performance of a given model. More the AUC value better is the performance of the model.

Following section discusses the results obtained after applying machine learning methods. The models predicted were applied to a total of 668 classes. We have used Weka[11] to apply different machine learning algorithms on the data set obtained by applying the data cleaning methods above.

A total of 4 attributes were used for processing of data sets. The attributes included were:

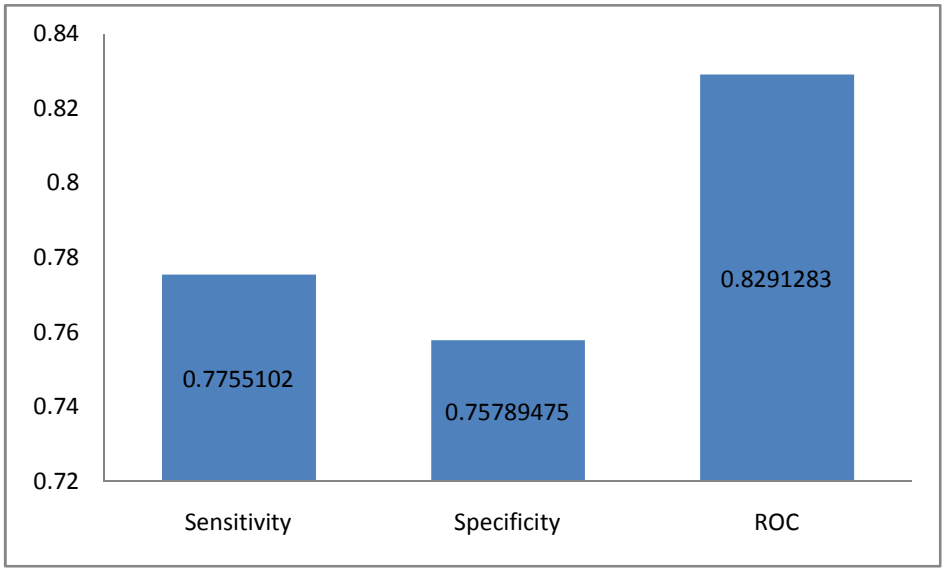
- CountDeclMethod
- CountClassCoupled

- CountDeclMethodAll
- DefectPresent

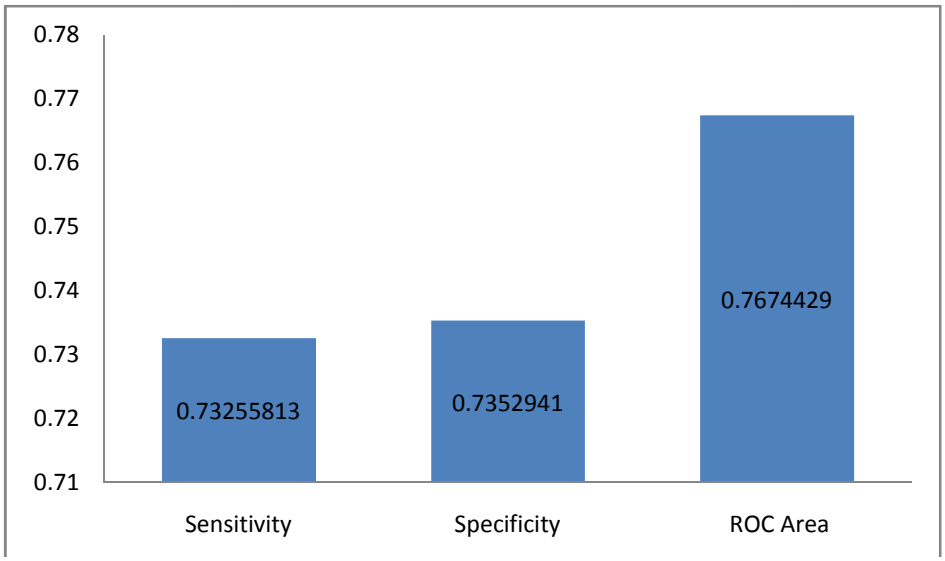
When checked through Weka[11] tool, the value of standard deviation and mean for DIT and NOC metrics are low for the chosen dataset, which shows that inheritance is not much used in the given dataset. Also the metrics LCOM and SLOC had very high values for standard deviation and mean. As a result DIT, NOC, LCOM and SLOC were not considered while processing the data sets.

5.1 Bagging:

The above data set was tried in Weka using the Bagging algorithm. Following graph shows the results using Bagging algorithm.

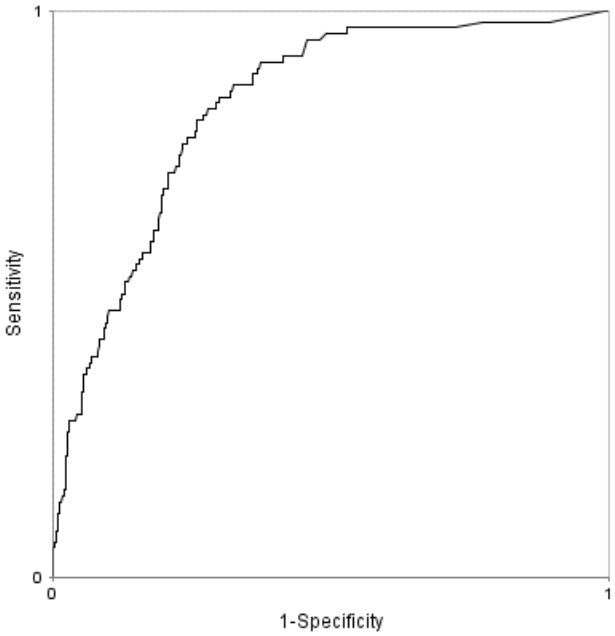


(a) MX4J

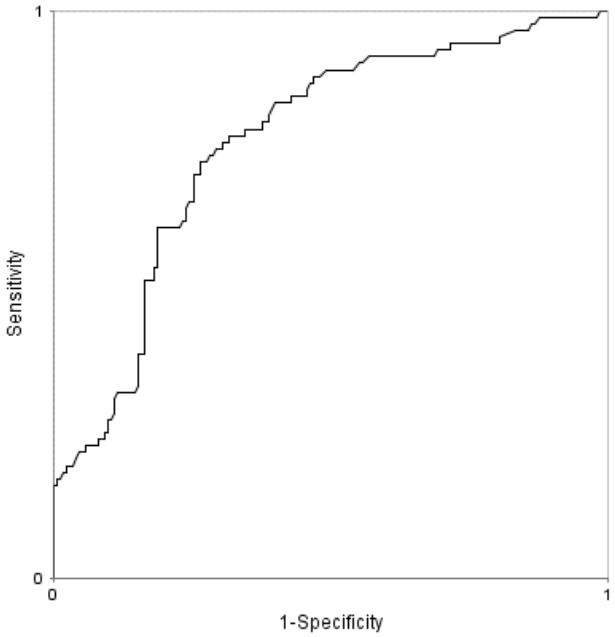


(b) Synapse 1.2

Figure 5.1 Sensitivity, Specificity and ROC graph for (a) MX4J and (b) Synapse 1.2 using Bagging Algorithm



(a) MX4J

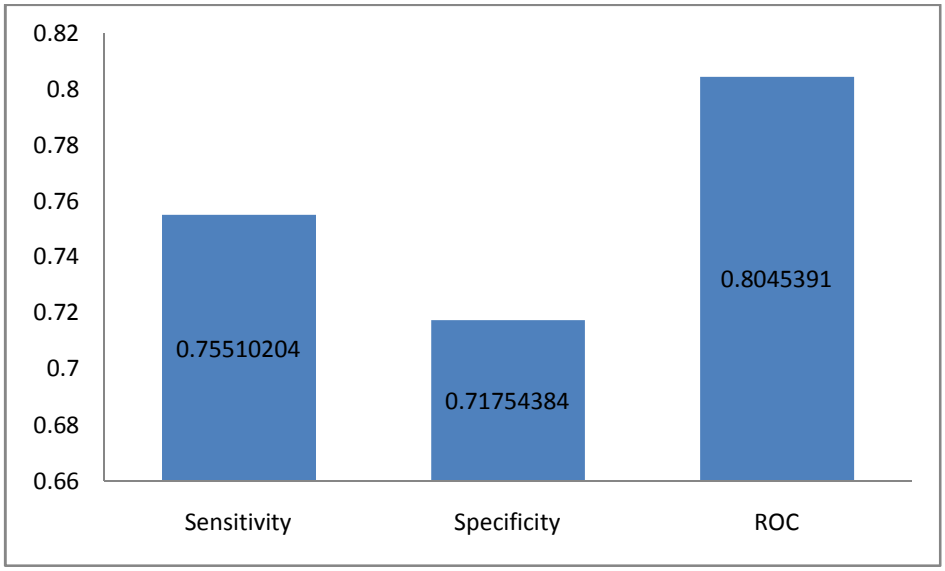


(b) Synapse 1.2

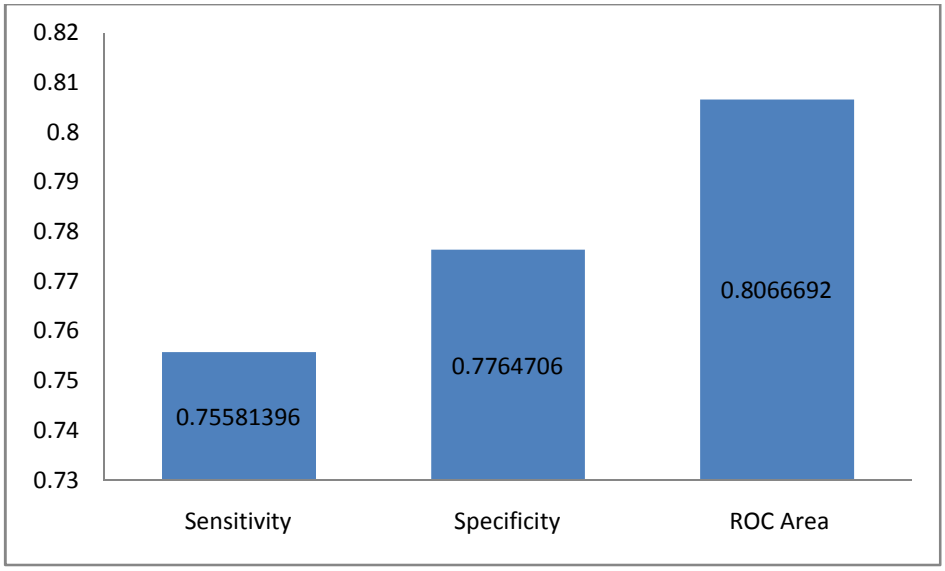
Figure 5.2 ROC curve for (a) MX4J and (b) Synapse 1.2 using Bagging Algorithm

5.2 LogitBoost:

The above data set was tried in Weka[11] using the LogitBoost algorithm. Following graph shows the results using LogitBoost algorithm.

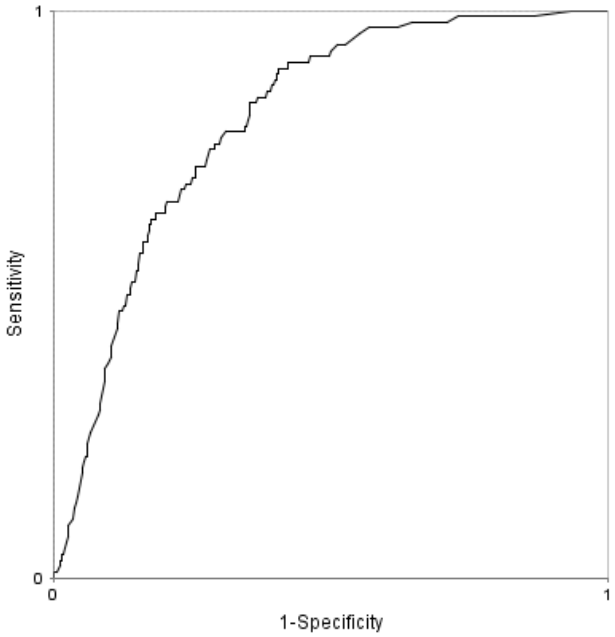


(a) MX4J

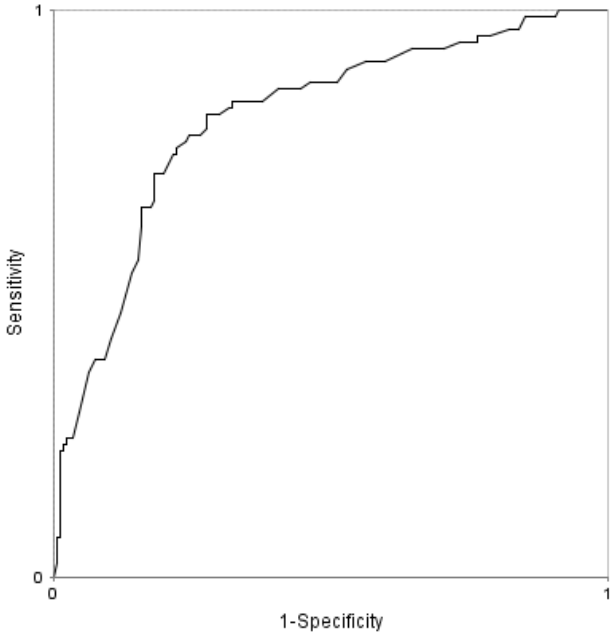


(b) Synapse 1.2

Figure 5.3 Sensitivity, Specificity and ROC graph for (a) MX4J and (b) Synapse 1.2 using LogitBoost Algorithm



(a) MX4J

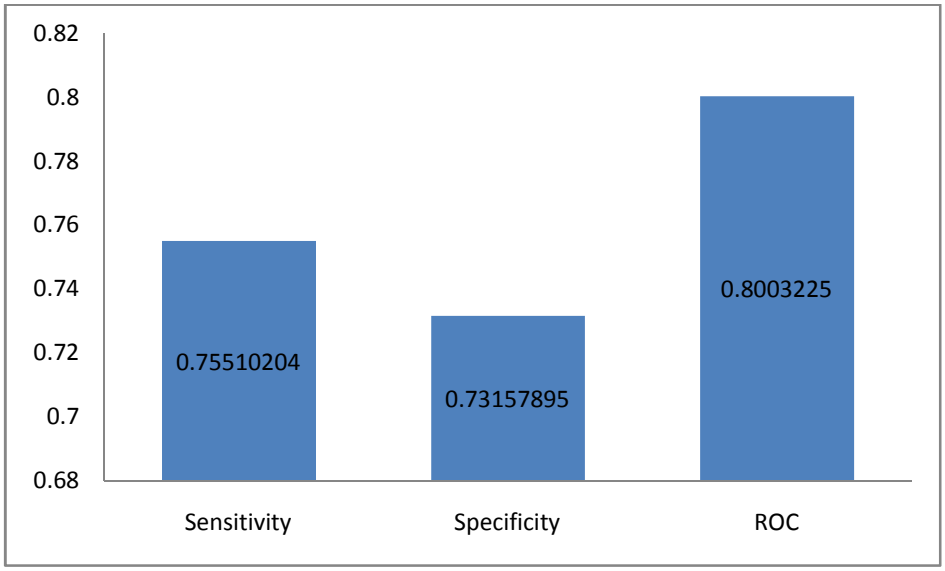


(a) Synapse 1.2

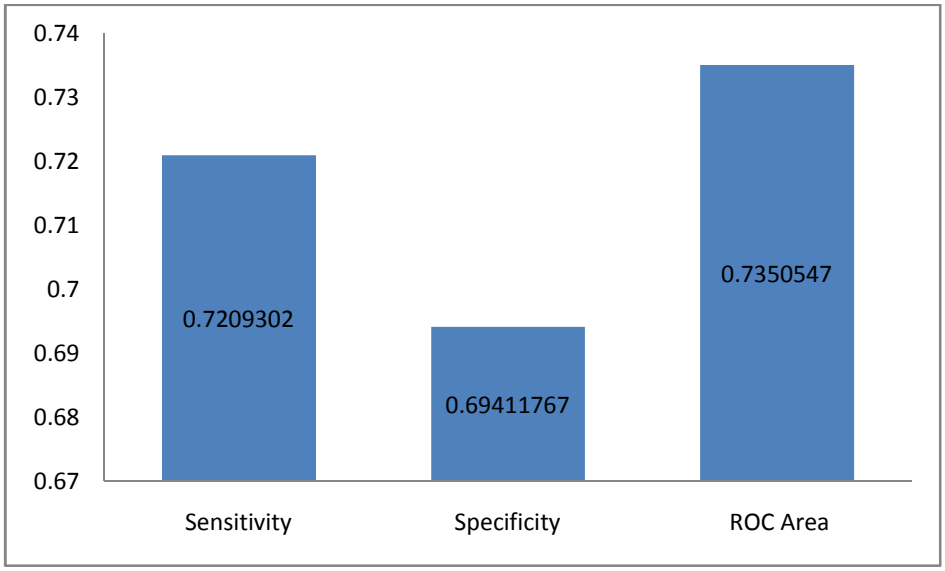
Figure 5.4 ROC curve for (a) MX4J and (b) Synapse 1.2 using LogitBoost Algorithm

5.3 RandomForest:

The above data set was tried in Weka using the RandomForest algorithm. Following graph shows the results using RandomForest algorithm.

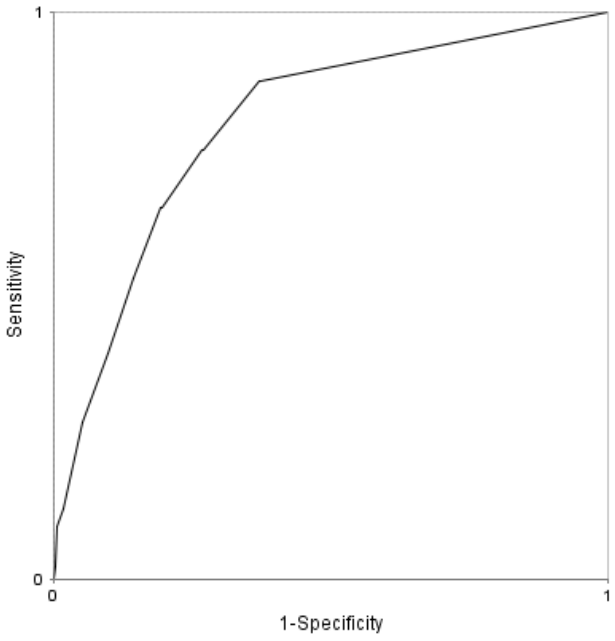


(a) MX4J

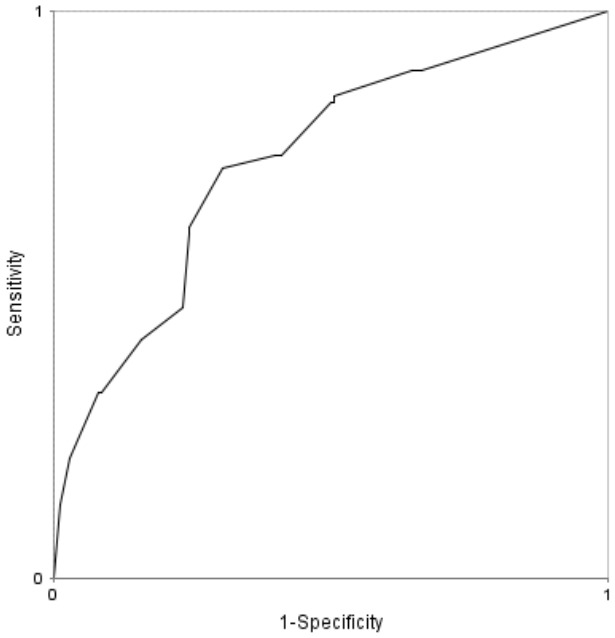


(b) Synapse 1.2

Figure 5.5 Sensitivity, Specificity and ROC graph for (a) MX4J and (b) Synapse 1.2 using RandomForest Algorithm



(a) MX4J

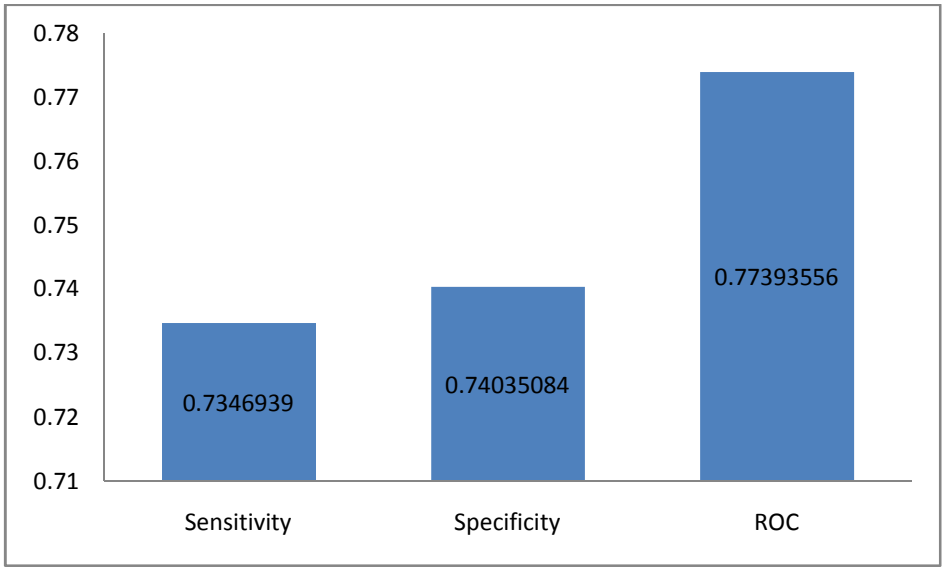


(b) Synapse 1.2

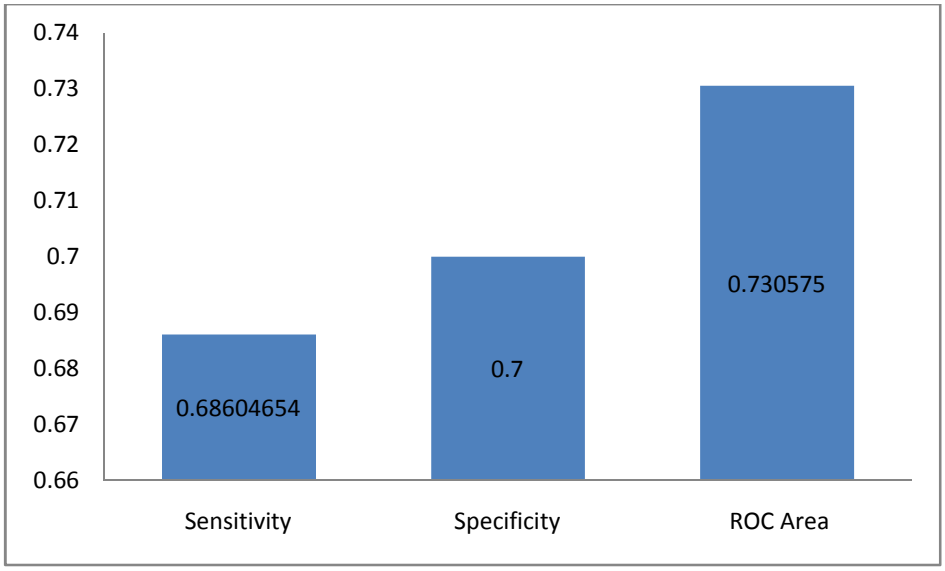
Figure 5.6 ROC curve for (a) MX4J and (b) Synapse 1.2 using RandomForest Algorithm

5.4 NaiveBayes:

The above data set was tried in Weka using the NaiveBayes algorithm. Following graph shows the results using NaiveBayes algorithm.

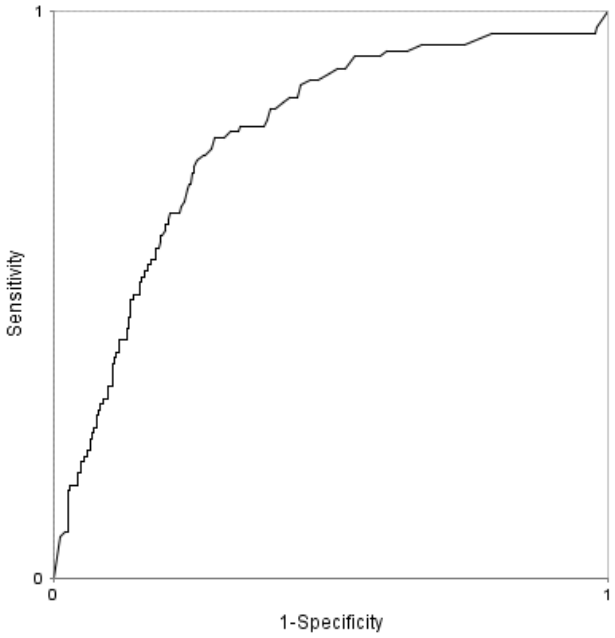


(a) MX4J

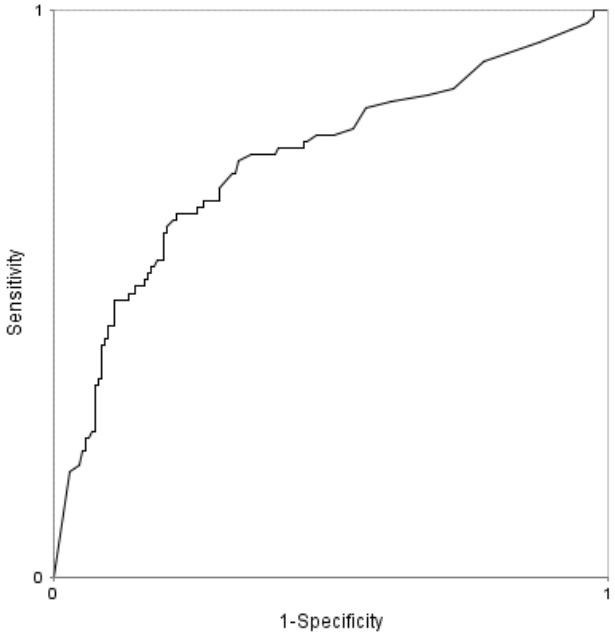


(b) Synapse 1.2

Figure 5.7 Sensitivity, Specificity and ROC graph for (a) MX4J and (b) Synapse 1.2 using NaiveBayes Algorithm



(a) MX4J

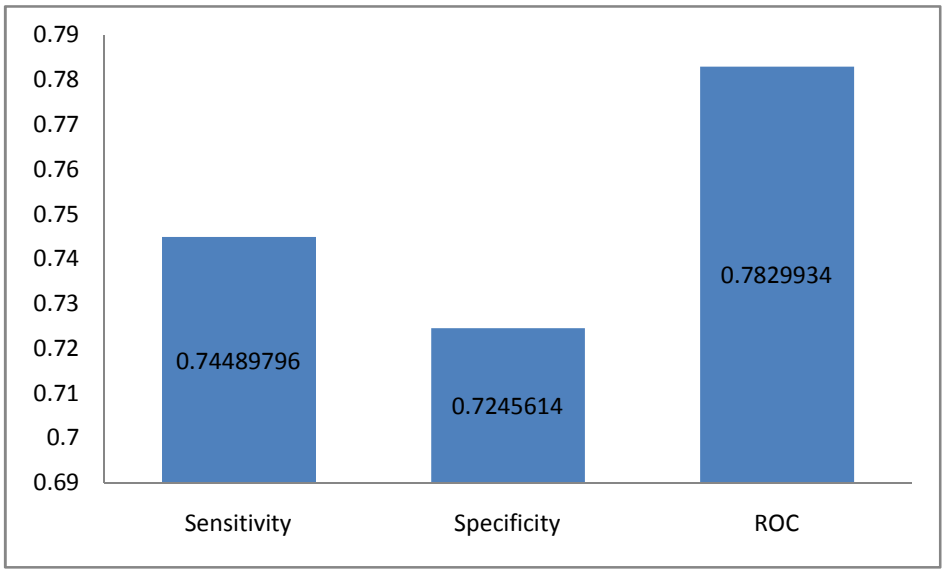


(b) Synapse 1.2

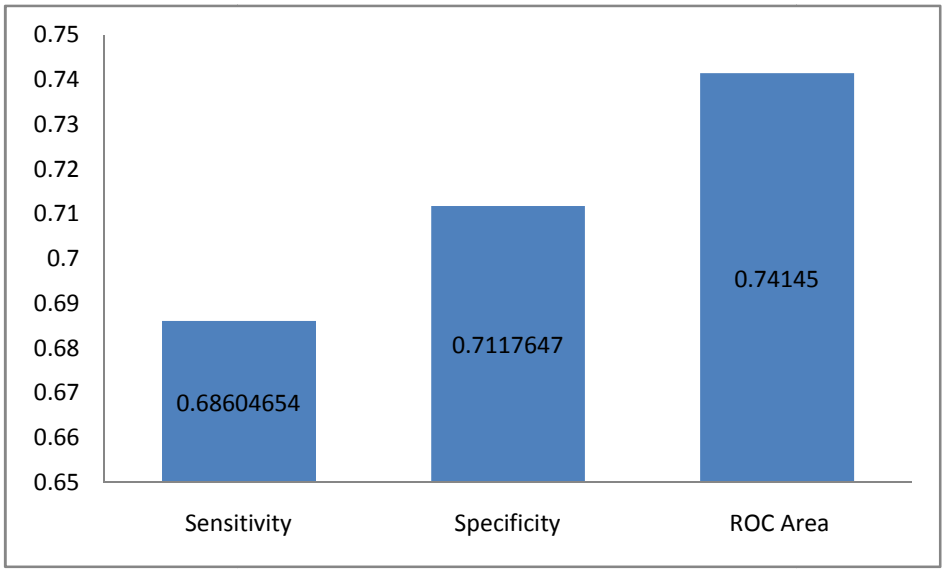
Figure 5.8 ROC curve for (a) MX4J and (b) Synapse 1.2 using NaiveBayes Algorithm

5.5 AdaBoostM1:

The above data set was tried in Weka using the AdaBoostM1 algorithm. Following graph shows the results using AdaBoostM1 algorithm.

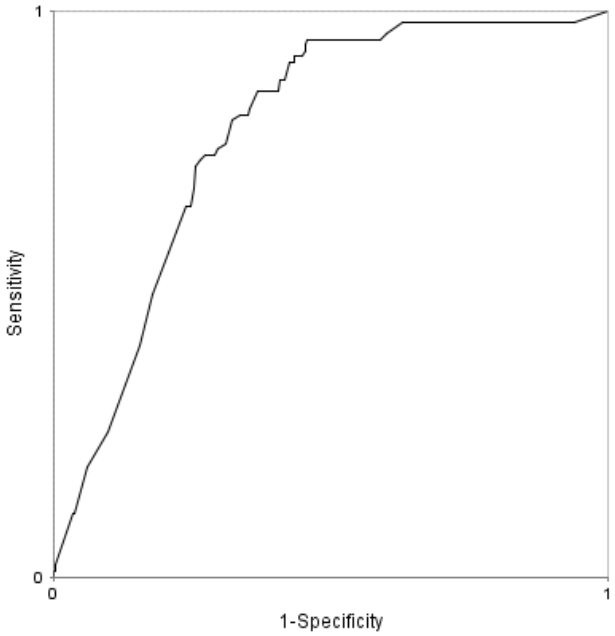


(a) MX4J

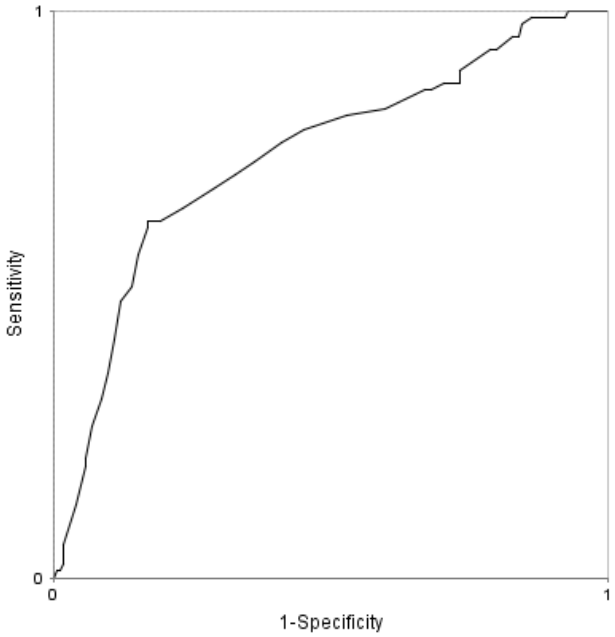


(b) Synapse 1.2

Figure 5.9 Sensitivity, Specificity and ROC graph for (a) MX4J and (b) Synapse 1.2 using AdaBoostM1 Algorithm



(a) MX4J

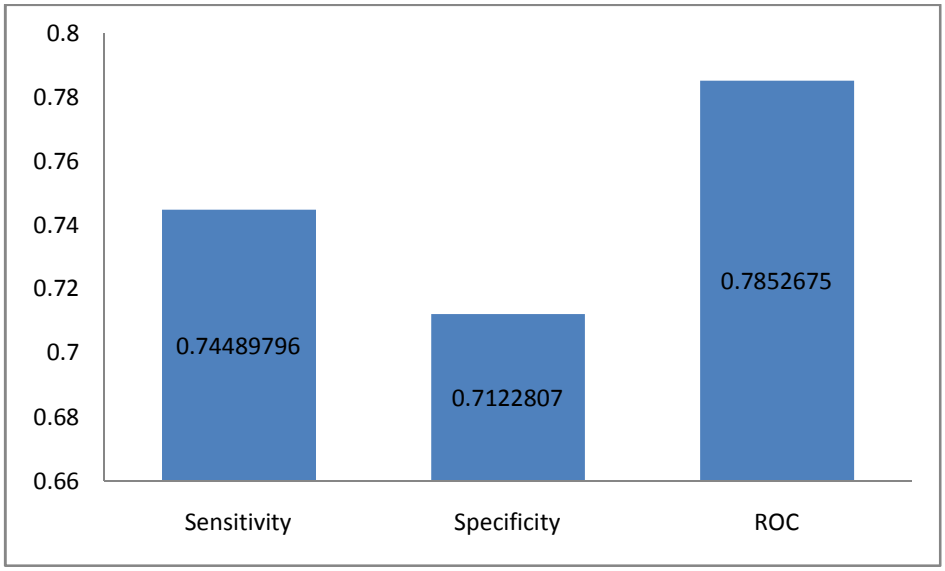


(b) Synapse 1.2

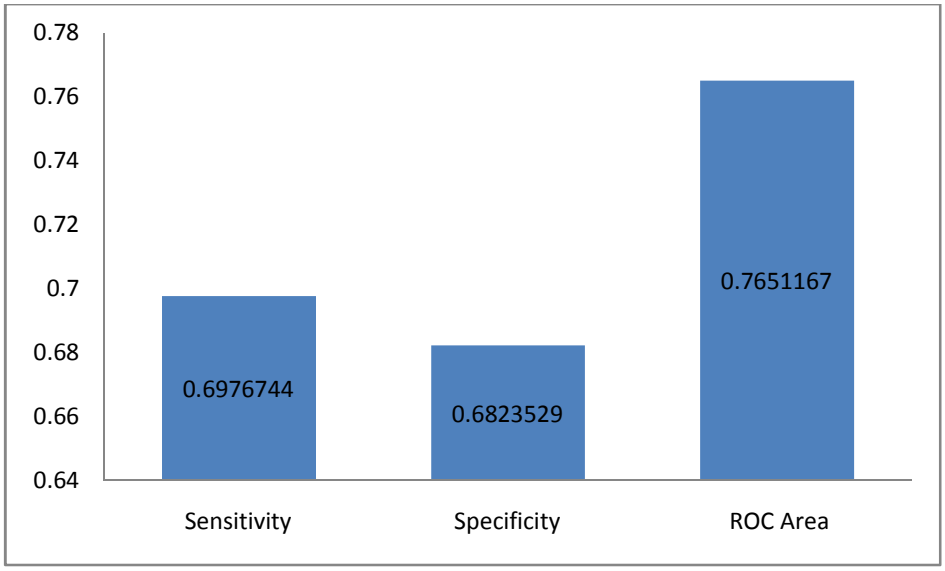
Figure 5.10 ROC curve for (a) MX4J and (b) Synapse 1.2 using AdaBoostM1 Algorithm

5.6 Logistic:

The above data set was tried in Weka using the Logistic algorithm. Following graph shows the results using Logistic algorithm.

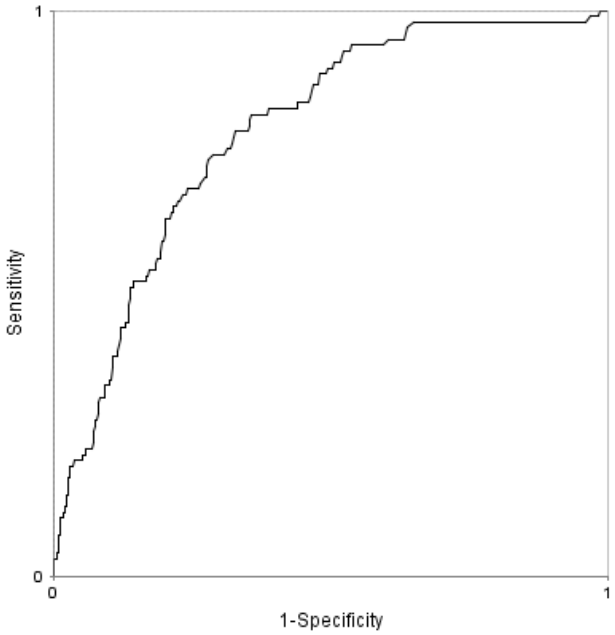


(a) MX4J

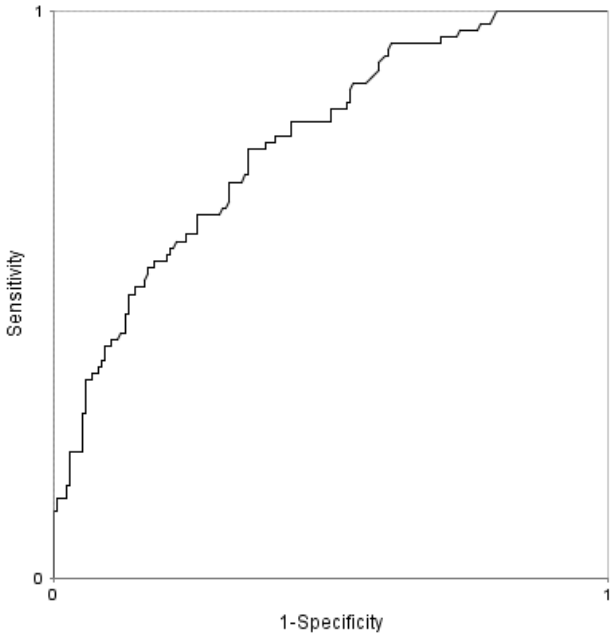


(b) Synapse 1.2

Figure 5.11 Sensitivity, Specificity and ROC graph for (a) MX4J and (b) Synapse 1.2 using Logistic Algorithm



(a) MX4J



(b) Synapse 1.2

Figure 5.12 ROC curve for (a) MX4J and (b) Synapse 1.2 using Logistic Algorithm

For the purpose of our work we have compared the ROC values for different methods applied. The comparison results showed that Bagging and Logistic methods provided greater ROC area as compared to RandomForest and NaiveBayes.

The comparison results are consolidated in the table below:

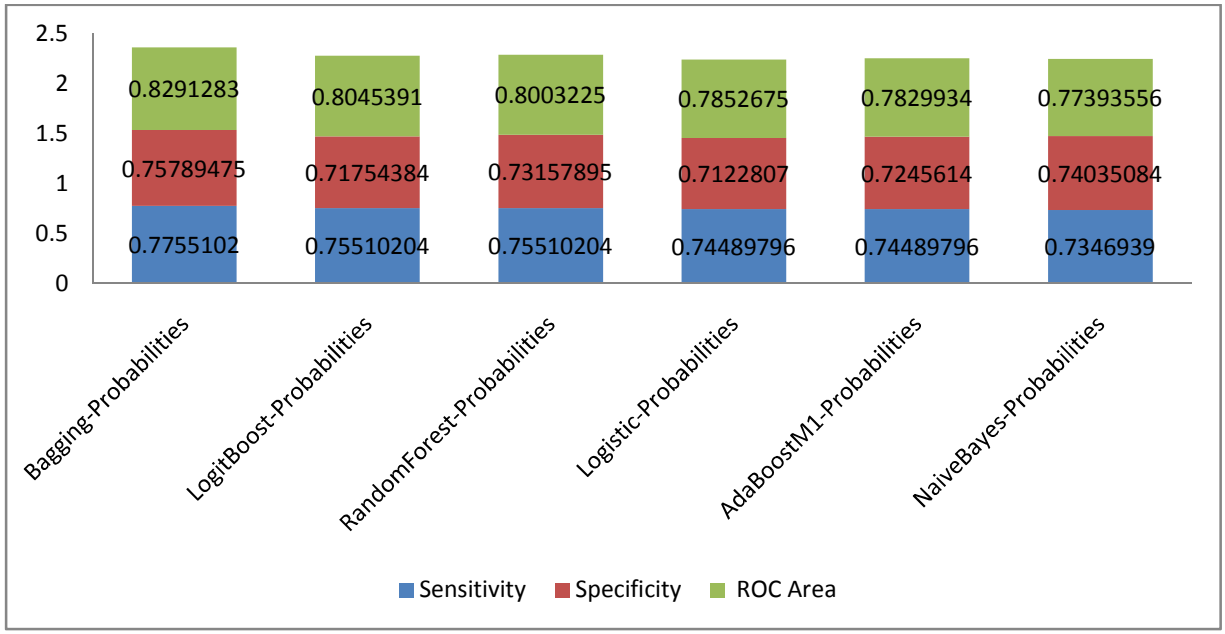
Table 5.1 Comparison of Sensitivity, Specificity and ROC Area for (a) MX4J and (b) Synapse 1.2 for machine learning methods

Classifier	**Sensitivity**	**Specificity**	**ROC Area**
Bagging-Probabilities	0.7755102	0.75789475	0.8291283
LogitBoost-Probabilities	0.75510204	0.71754384	0.8045391
RandomForest-Probabilities	0.75510204	0.73157895	0.8003225
Logistic-Probabilities	0.74489796	0.7122807	0.7852675
AdaBoostM1-Probabilities	0.74489796	0.7245614	0.7829934
NaiveBayes-Probabilities	0.7346939	0.74035084	0.77393556

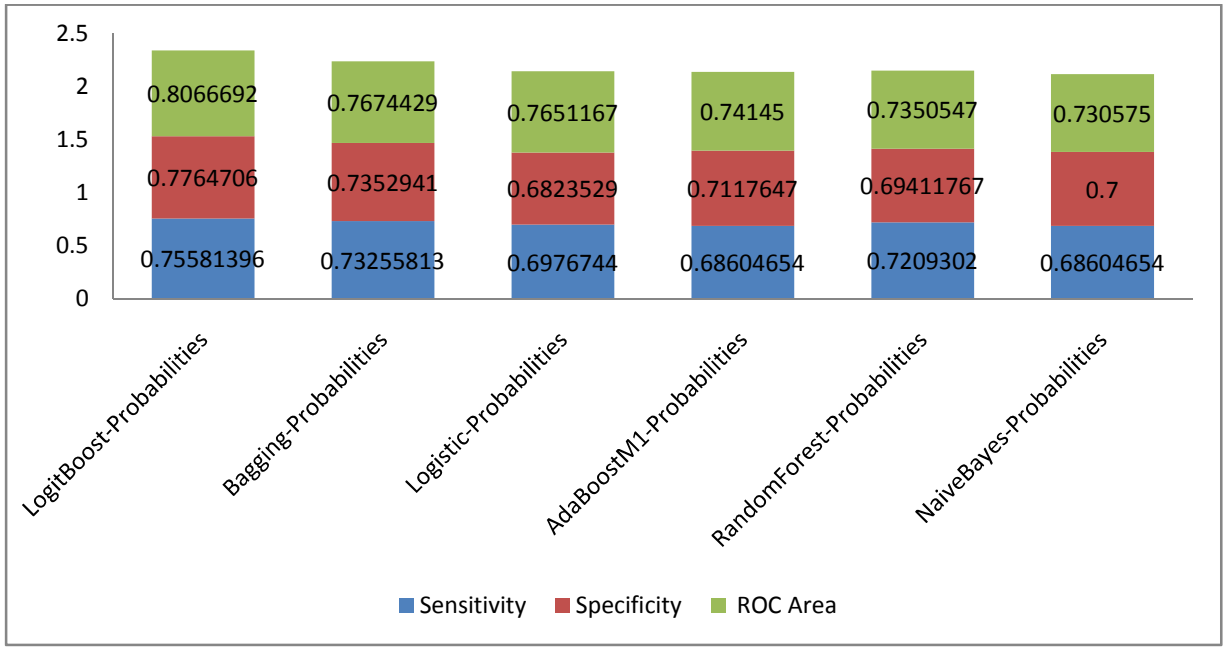
(a) MX4J

Classifier	**Sensitivity**	**Specificity**	**ROC Area**
LogitBoost-Probabilities	0.75581396	0.7764706	0.8066692
Bagging-Probabilities	0.73255813	0.7352941	0.7674429
Logistic-Probabilities	0.6976744	0.6823529	0.7651167
AdaBoostM1-Probabilities	0.68604654	0.7117647	0.74145
RandomForest-Probabilities	0.7209302	0.69411767	0.7350547
NaiveBayes-Probabilities	0.68604654	0.7	0.730575

(b) Synapse 1.2



(a) MX4J



(b) Synapse 1.2

Figure 5.13: Graphs for comparison of Sensitivity, Specificity and ROC Area of machine learning methods for (a) MX4J and (b) Synapse 1.2

Chapter 6

CONCLUSION & FUTURE WORK

In this chapter we empirically analyzed the performance of machine learning methods. We applied machine learning methods on two open source Java based varying datasets, MX4J[13] and Synapse 1.2[14]. We first measured the sensitivity and specificity using the Weka[11] tool and then used the SPSS[12] tool to generate the ROC curve and perform analysis. We evaluated the relation between OO metrics, CK metrics, and fault proneness after applying machine-learning methods. We used Receiver Operating Characteristic (ROC) as a measure to check the effectiveness of each of the prediction models. The results show that the area under the curve (measured from the ROC analysis) of models predicted using Bagging and Logistic methods are than area under the curve of other machine learning methods hence providing better performance in predicting fault proneness. Also the classification capability of Bagging process was better than other Methods followed by AdaBoost1 and Logistic Methods, whereas NaiveBayes being the last in the comparison results.

This study provides a cost-effective way by assisting managers to align resources on fault prone modules. The developers can be asked to focus upon modules that are predicted to have higher fault proneness. The process of applying fault prediction models based on object oriented (OO) design metrics for the purpose of detecting fault prone classes would help in reducing in software defects in final products launched in the market or shared with the end user.

Since we conducted our analysis on two datasets, we would replicate the study on larger number of datasets in order to generalize our findings. We plan to perform extended studies with different data sets to confirm our findings over a larger range of domain. Also we plan to use the number of defects to be used as a major area of focus along with the severity of faults. We plan to extend our study by confirming the prediction performance by applying genetic algorithms.

References

[1] Chidamber, S., & Kamerer, C. (1994). A metrics suite for object-oriented design. IEEE Transactions on Software Engineering, 20(6), 476–493. doi:10.1109/32.295895.

[2] Singh, Y., Kaur, A., & Malhotra, R. (2010). Empirical validation of object-oriented metrics for predicting fault proneness models, Software Qual J (2010) 18:3–35, DOI 10.1007/s11219-009-9079-6

[3] Aggarwal, K. K., Singh, Y., Kaur, A., & Malhotra, R. (2009). Empirical analysis for investigating the effect of object-oriented metrics on fault proneness: A replicated case study. Software Process: Improvement
and Practice, 16(1), 39–62. doi:10.1002/spip.389s.

[4] Shatnawi, R., Li, W., 2008. The effectiveness of software metrics in identifying errorprone classes in post-release software evolution process. Journal of Systems and Software 81 (11), 1868–1882.

[5] Zhou, Y., Leung, H., 2006. Empirical analysis of object-oriented design metrics for predicting high and low severity faults. IEEE Transactions on Software Engineering 32 (10), 771–789.

[6] El-Emam, K., Melo, 1999. The prediction of faulty classes using object-oriented design metrics. published as NRC/ERB-1064. November 1999. 24 pages. NRC 43609.

[7] El-Emam, K., Melo, W., Machado, J.C., 2001b. The prediction of faulty classes using object-oriented design metrics. Journal of Systems and Software 56 (1), 63–75.

[8] Olague, H.M., Etzkorn, L.H., Gholston, S., Quattlebaum, S., 2007. Empirical validation of three software metrics suites to predict fault-proneness of object-oriented classes developed using highly iterative or agile software development processes. IEEE Transactions on Software Engineering 33 (6), 402–419.

[9] Gondra, Iker , 2007. Applying machine learning to software fault-proneness prediction. The Journal of Systems and Software 81 (2008) 186–195.

[10] Taghi M. Khoshgoftaar, Naeem Seliya, Nandini Sundaresh (2006) .An empirical study of predicting software faults with case-based reasoning. Software Qual J (2006) 14: 85–111 DOI 10.1007/s11219-006-7597-z

[11] Weka official site:
http://www.cs.waikato.ac.nz/ml/weka/

[12] SPSS official site:

http://www-01.ibm.com/software/in/analytics/spss/

[13] SourceForge official site. Source code for MX4J dataset

http://sourceforge.net/projects/mx4j/

[14] Synapse official site:

http://synapse.apache.org/source-repository.html