

**A DISSERTATION**  
**ON**  
**DEVELOPMENT OF SOFTWARE PREDICTION MODELS**  
**USING VARIOUS MACHINE LEARNING TECHNIQUES**

*Submitted in partial fulfilment of the requirements*

*For the award of the degree of*

**MASTER OF TECHNOLOGY**

*in*

**SOFTWARE TECHNOLOGY**

*Submitted by*

**Abhishek Sharma**

**University Roll No. 2K13/SWT/02**

*Under the Esteemed Guidance of*

**Dr. Ruchika Malhotra**

**Associate Head & Assistant Professor, Department of Computer Science &  
Engineering, DTU**



2013-2016

**DEPARTMENT COMPUTER ENGINEERING & ENGINEERING**  
**DELHI TECHNOLOGICAL UNIVERSITY,**  
**DELHI- 110042, INDIA**

---



DELHI TECHNOLOGICAL UNIVERSITY

DELHI-110042

## DECLARATION

I hereby declare that the thesis entitled “**DEVELOPMENT OF SOFTWARE PREDICTION MODELS USING VARIOUS MACHINE LEARNING TECHNIQUES**” which is being submitted to the **Delhi Technological University**, in partial fulfilment of the requirements for the award of degree of **Master of Technology in Software Technology** is an authentic work carried out by me. The material contained in this thesis has not been submitted to any university or institution for the award of any degree.

DATE:

SIGNATURE:

ABHISHEK SHARMA  
2K13/SWT/02



DELHI TECHNOLOGICAL UNIVERSITY

DELHI-110042

## CERTIFICATE

This is to certify that thesis entitled “**DEVELOPMENT OF SOFTWARE PREDICTION MODELS USING VARIOUS MACHINE LEARNING TECHNIQUES**”, is a bonafide work done by Mr. Abhishek Sharma (Roll No: 2K13/SWT/02) in partial fulfilment of the requirements for the award of **Master of Technology Degree in Software Technology** at Delhi Technological University, Delhi, is an authentic work carried out by him under my supervision and guidance. The content embodied in this thesis has not been submitted by him earlier to any University or Institution for the award of any Degree or Diploma to the best of my knowledge and belief.

DATE:

SIGNATURE:

Dr. RUCHIKA MALHOTRA  
ASSOCIATE HEAD & ASSISTANT PROFESSOR,  
DEPARTMENT OF COMPUTRE SCIENCE & ENGINEERING.  
DELHI TECHNOLOGICAL UNIVERSITY, DELHI 110042

## ACKNOWLEDGEMENT

I am presenting my work on “**DEVELOPMENT OF SOFTWARE PREDICTION MODELS USING VARIOUS MACHINE LEARNING TECHNIQUES**” with lot of pleasure and satisfaction. I take this opportunity to express my deep sense of gratitude and respect towards my guide **Dr. Ruchika Malhotra**. I am very much indebted to her for his generosity, expertise and guidance I have received from her while working on this project. Without her support and timely guidance the completion of the project would have seemed a far –fetched dream. In this respect I find myself lucky to have my guide. She have guided not only with the subject matter, but also taught the proper style and techniques of documentation and presentation. Besides my guides, I would like to thank entire teaching and non-teaching staff in the Department of Computer Science & Engineering, DTU for all their help during my tenure at DTU. Kudos to all my friends at DTU for thought provoking discussion and making stay very pleasant. I am also thankful to the SAMSUNG who has provided me opportunity to enroll in the M.Tech Programme and to gain knowledge through this programme. This curriculum provided me knowledge and opportunity to grow in various domains of computer science.

ABHISHEK SHARMA  
2K13/SWT/02

## ABSTRACT

In current world, all information systems development follows well defined development process based on requirements & business needs. Every stake holder's expectation is to get best product with zero or minimum defects also cost effective. So, In order to achieve best product in given time, it is necessary to uncover most defects as early as possible in development life cycle.

In spite of thorough planning, well documentation and proper process control during software development, occurrences of certain defects are inevitable. These defects may leads to poor software quality which may hamper the brand image & lead to business failure. In today's competitive world it's necessary to make world class product with minimum defects, high in quality & cost effective.

Cost of defects finding is directly proportional to time of its finding in development cycle. Later are defects found, more is the cost of fixing them leads higher development cost. Hence it's necessary to identify defective classes in early phase of software development to reduce the testing cost. This may guide the product planning team for efficient resource planning for testing. Software metrics have been used to describe the complexity of the program and, to estimate software development time.

Software metrics can be used in simultaneity with defect data to develop models for predicting defective classes. The development of predictive models to predict fault classes can help & guide the stakeholders in predicting faulty classes in early phase of software development. Hence, it is vital to analyse and compare the predictive accuracy of machine learning classifiers. Various Machine Learning Techniques were used to understand & analyse the core relationships of classes and fetching useful information from problems.

The objective of this thesis is to evaluate the performance & comparison of Machine Learning Techniques over unpopular data sets. The evaluation is performed with an intention to identify which algorithm suits best for prediction of defect prone classes in software based on software quality metrics. Chidamber & Kemerer Java matrices [4] were generated over 4 subsequent releases of Android 'Contact' Module. Jelly Bean to KitKat to Lollypop to Marshmallow. 7 Machine Learning techniques were

compared to evaluate the relationship of Chidamber & Kemerer Java matrices on defective classes.

The result shows the predictive capability of Machine Learning Techniques & suggested model. The results of work were based on data sets obtained from popular open source mobile software Android “Contacts” module.

# TABLE OF CONTENTS

DECLARATION.....	..ii
CERTIFICATE .....	..iii
ACKNOWLEDGEMENT.....	iv
ABSTRACT.....	v
TABLE OF CONTENTS.....	vii
CHAPTER 1. INTRODUCTION .....	2
CHAPTER 2. LITERATURE REVIEW .....	4
CHAPTER 3. RESEARCH BACKGROUND.....	6
3.1 Data Set Generation: .....	6
3.1.1 Defect Collection and Reporting System (DCRS) Tool: .....	7
3.1.2 Generating Change Logs:.....	10
3.1.3 Generating Metrics:.....	11
3.2 Chidamber and Kemerer Java Metrics:.....	13
3.2.1 Weighted Methods Per Class (WMC): .....	13
3.2.2 Depth of Inheritance Tree (DIT):.....	13
3.2.3 Number of Children (NOC): .....	13
3.2.4 Coupling between Object Classes (CBO).....	14
3.2.5 Response for a Class (RFC) .....	14
3.2.6 Lack of Cohesion of Methods (LCOM).....	14
3.3 Dependent & Independent Variables: .....	15
CHAPTER 4. RESEARCH METHODOLOGY.....	16
4.1 Preprocessing of Data: .....	16
4.2 Machine Learning Techniques Selection.....	17
4.2.1 Bayes Net .....	17
4.2.2 Naive Bayes .....	18
4.2.3 Logistics Regression .....	19
4.2.4 KStar .....	20
4.2.5 Bagging .....	21
4.2.6 Logit Boost.....	21

4.2.7 Random Forest .....	22
4.3 Application of Algorithms .....	23
4.4 Performance Evaluation.....	25
4.5 Model Evaluation Results:.....	26
4.5.1 Bayes Net .....	27
4.5.2 Naïve Bayes .....	27
4.5.3 Logistic Regression.....	27
4.5.4 KStar .....	28
4.5.5 Bagging .....	28
4.5.6 Logit Boost.....	28
4.5.7 Random Forest .....	29
CHAPTER 5. CONCLUSION.....	30
BIBLIOGRAPHY.....	31



## LIST OF TABLES

Table 3.1 .....	7
Table 3.2 Android - Contacts .....	11
Table 3.3 Object Oriented Metrics.....	12
Table 4.1 ROC Value.....	26
Table 4.2 Evaluation Results for Bayes Net .....	27
Table 4.3 Evaluation Results for Naïve Bayes .....	27
Table 4.4 Evaluation Results for Logistic Regression.....	28
Table 4.5 Evaluation Results for KStar .....	28
Table 4.6 Evaluation Results for Bagging .....	28
Table 4.7 Evaluation Results for Logit Boost.....	29
Table 4.8 Evaluation Results for Random Forest .....	29
Table 5.1 Result Summary.....	30

## LIST OF FIGURES

Figure 3.1 DCRS Tool .....	9
Figure 3.2 DCRS – Change Logs .....	10
Figure 4.1 WEKA - Preprocess .....	24
Figure 4.2 WEKA - Classify.....	24



# Chapter 1. Introduction

In current time software development have reached to another level from standalone systems to multiple interactive systems like Internet of Things, Robotics, Spacecraft's etc. Hence the complexity of software system is increasing day by day leading to more defects prone. It's been almost impossible to produce software systems without defects. Therefore Software testing is considered as one of the key phase of software development. It consumes at least half the development resources & still error free or 100% correctness of software system is a dream. Testing phase should be considered as critical area while software development to minimise the cost & maintain the market trust. The cost of finding defects is directly proportional to time of its occurrence. The cost of fixing a defect increases exponentially if defects are uncovered towards the end of software development or after product delivery. Market reputation is also hits badly if product is not correct or having defects. Hence finding & fixing most software defects as early as possible is always recommended.

Software defect prediction can play a very vital role in this regards. Software defect prediction is good practise in reducing testing efforts also help in proper test planning. Early detection of defects may results in delivering the high quality & cost effective software product. The challenges of effective testing leads to research area of identifying faulty classes in early phase & aligning the test activity accordingly. This predictive model helps to guide & produce defect free/cost effective software using object oriented metrics for predicting fault classes. These software matrices which capture various properties (like Coupling, Cohesion, Encapsulation, Inheritance, No. of classes etc.) Of software shall be used for developing models for predicting defective classes in software. Often these metrics have been used as an early indicator of these externally visible attributes, because the externally visible attributes could not be measures until too late in the software development process. The software metrics collected from a similar project or past release (Android subsequent releases Jellybean to Marshmallow) can be used for developing defect prediction model. The developed defect

Prediction model can then be subsequently used for classifying classes of current projects as defective or not defective.

Various machine learning techniques are available which may be used to predict faulty classes. We have used 7 machine learning techniques over the object oriented matrices generated from open source software Android subsequent releases. Machine learning techniques used are Bayes Net, Naive Bayes, Logistic, Kstar, Bagging, Logit Boost and Random Forest. The performance of these techniques varies with different datasets and it's difficult to determine which technique is superior to another.

In this work we have compared the performance of 7 machines learning techniques on 5 releases of 'Contacts' application package of popular mobile operating system Android. This enables the investigation whether one technique outperforms others and also provides insights on the selection of a particular ML technique. We have used object oriented metrics for predicting the defect prone classes. The results were evaluated based on Receiver operating Characteristics.

## Chapter 2. Literature Review

Several studies were done in the past to relate software metrics with defect proneness. Some of the key studies are discussed below.

Malhotra R, Raje R [1] addressed four issues (i) comparison of the machine learning techniques over unpopular used data sets (ii) use of inappropriate performance measures for measuring the performance of defect prediction models (iii) less use of statistical tests and (iv) validation of models from the same data set from which they are trained. To resolve these issues, we have compared 18 machine learning techniques for investigating the effect of Object-Oriented metrics on defective classes. The results are validated on six releases of the open source android operating system ‘MMS’ application package. The overall results of the study indicate the predictive capability of the machine learning techniques and an endorsement of one particular ML technique to predict defects.

Malhotra R, Singh Y [2], proposed to find the relation of object oriented metrics and fault proneness of a class. They used seven machine learning and one logistic regression method in order to predict faulty classes. The results of work are based on data set obtained from open source software. The results show that the predictive accuracy of machine learning technique Logit Boost is highest with Area under Curve of 0.806.

Kaur A., Malhotra R, Singh Y [3], proposed Support Vector Machine (SVM) model to find the relationship between object-oriented metrics given by Chidamber and Kemerer [4] and fault proneness. The proposed model is empirically evaluated using public domain KC1 NASA data set. The performance of the SVM method was evaluated by Receiver Operating Characteristic (ROC) analysis. Based on these results, it is reasonable to claim that such models could help for planning and performing testing by focusing resources on fault-prone parts of the design and code. Thus, the study shows that SVM method may also be used in constructing software quality models.

Kaur A, Kaur I [5], Used six machine learning models for software quality prediction on five open source software. Varieties of metrics have been evaluated for the software including Chidamber and Kemerer [4], Henderson & Sellers, McCabe etc. Results show that Random Forest and Bagging produce good results while Naïve Bayes is least preferable for prediction

Gyimothy T, Ference R, Siket I [6] showed how to calculate the object-oriented metrics given by Chidamber and Kemerer [4] to illustrate how fault-proneness detection of the source code of the open source Web and e-mail suite called Mozilla can be carried out. He checked the values obtained against the number of bugs found in its bug database - called Bugzilla - using regression and machine learning methods to validate the usefulness of these metrics for fault-proneness prediction. He also compared the metrics of several versions of Mozilla to see how the predicted fault-proneness of the software system changed during its development cycle.

Zhou, Y., Leung, H. [7], distinguish among faults according to the severity of impact. It would be valuable to know how object-oriented design metrics and class fault-proneness are related when fault severity is taken into account. In this paper, we use logistic regression and machine learning methods to empirically investigate the usefulness of object-oriented design metrics, specifically, a subset of the Chidamber and Kemerer suite [4], in predicting fault-proneness when taking fault severity into account. Our results, based on a public domain NASA data set, indicate that 1) most of these design metrics are statistically related to fault-proneness of classes across fault severity, and 2) the prediction capabilities of the investigated metrics greatly depend on the severity of faults. More specifically, these design metrics are able to predict low severity faults in fault-prone classes better than high severity faults in fault-prone classes

## Chapter 3. Research Background

In this section we will see the Data collection process, tools used, Object Oriented Metrics generation etc.

### 3.1 Data Set Generation:

In this study, Object Oriented Metrics were obtained using open source mobile Operating System – Android. “Contact” package is considered for generating the data sets. We have downloaded the 4 latest released of Android Operating System from KitKat (version 4.4 to) to Marshmallow (version 6.0.0). Android is most widely used & leading phone Operating system.

Source code is fetched from Google GIT\* Repository [8] (<https://android.googlesource.com/platform/packages/apps/Contacts/>) for Contact Application package. Android source code contains JAVA files. First Android code is compiled to generate the Class files from the JAVA Files. For Compiling the Android code we require multiple .jar files (firmware etc.), Resource files to compile Source code successfully. Once Class files are generated, Defect Collection and Reporting System (DCRS) tool [6] is used to generate the reports having Object Oriented Metrics. DCRS have CKJM tool integrated which calculates Chidamber and Kemerer object-oriented metrics [4] by processing the bytecode of compiled Java files. The program calculates for each class & generated the Object Oriented Metrics mentioned in Table 3.1. And displays them in a standard output format.

To generates the reports. We give the path of App package compiled code path having class files & run the tool to generate Object Oriented metrics with respect to each Class files.

Characteristics of Contact Application package with respect to different releases are mentioned in Table 3.1.

**Table 3.1**

<b>Data Set : Android App Package - "Contacts"</b>					
<i>Google GIT Repository :</i> <a href="https://android.googlesource.com/platform/packages/apps/Contacts/">https://android.googlesource.com/platform/packages/apps/Contacts/</a>					
<b>Android OS Release</b>	<b>Code Name</b>	<b>Total LoC</b>	<b>Total Class</b>	<b>Defective Count</b>	<b>Defective %</b>
<b>4.4</b>	<b>KitKat ( KK )</b>	49,040	210	28	13%
<b>5.0.0</b>	<b>Lollipop ( LL )</b>	36,969	177	115	65%
<b>5.1.1</b>	<b>Lollipop ( LL )</b>	37,257	139	48	35%
<b>6.0.0</b>	<b>Marshmallow (M)</b>	41,012	138	49	36%

*\*GIT is a version control system used for software development & version control task for Google Android source code. GIT as a distributed revision control system is aimed at speed, data integrity and support for distributed, non-linear workflows.*

Table 3.1 contains Android App Packages – Contacts Data Sets with Total LoC, Total Class, Defective/Non Defective Classes w.r.t. each Android release for Contact Application Package. Defects were generated using DCRS Tools (Defect Collection & Reporting System) developed by Delhi Technical University (DTU) Students. LoC is generated using LoC Metrics freeware tool.

### **3.1.1 Defect Collection and Reporting System (DCRS) Tool:**

The Defect Collection and Reporting System (DCRS) [6] is a JAVA based automated tool which collects and reports various defects, bugs or issues which were present in a given Version of Android OS, and have been fixed in the subsequent Version. The System caters to only two consecutive versions of Android OS.

Studies have shown that defect data collected from open source projects (like Android) can be used in research areas such as defect prediction. Some commonly



traversed areas of defect prediction include Analysis and Validation of the effect of given metric suite, (Like Metrics) on fault proneness; and applicability of such metric suites for the prediction of fault proneness models.

DCRS determining the deleted source files, newly added source files, defects fixed, etc. It efficiently collects defect data and can be used research areas.

DCRS first obtain the defect logs of android source code and filter them to obtain the defects which were present in a given version of Android Operating System and have been fixed in the subsequent version. Then, the system process the filtered defect logs to extract useful defect information such as unique defect identifier and defect description, if any.

DCRS also associates defects to their corresponding source files (java code files, or simply class files in the android OS source code). Then, it performs the computation of total number of fixed defects for each class, i.e., the number of defects which have been associated with that class. Finally, the corresponding values of different metric suites are obtained by the system for each class file in the source code of previous version of Android OS.

Install & Configure Git software installed, for extracting the change-logs for source code of each version of Android OS. Download 'Android Manifest File' from Git repository to get the list of Android Source Code components/ projects available at 'Git' repository (<https://android.google.com>). Source code of both the versions is required to generate the change logs. The system will guide user to download the same.



**Figure 3.1 DCRS Tool**

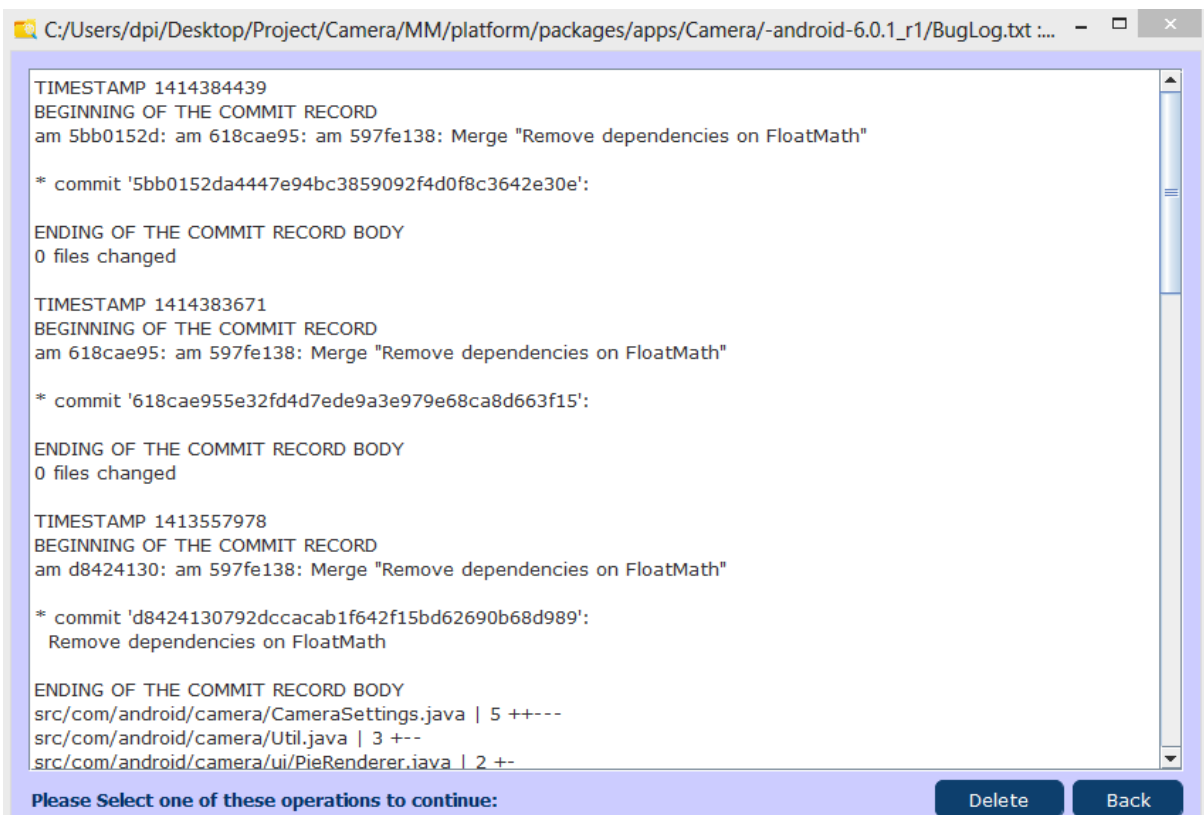
We can fetch information using DCRS as per below procedure. It process two versions of source code to retrieve Change-Logs. Change log provides information regarding the modifications that have been made in the source code. These Change Logs are further processed to get Bug-Logs (i.e. changes w.r.t. bug-fixes only). We can retrieve Bug-Ids and Description from the Bug-Logs. These Bug-Ids were mapped to Classes in the source code. Based on the above gathered information, the DCRS generates the following Reports:

- a. *Bug-Report –Contains details of each bug, class-wise (Bug-ID and description)*
- b. *Bug-Count report - Contains Bug-count (class-wise), Chidamber and Kemerer Java Matrics and other Matrics data for each class*
- c. *Change Report – Contains total LOC inserted and deleted, class-wise, for all the incurred changes*

We can collect defects from the Android OS defects logs as per below steps:

### 3.1.2 Generating Change Logs:

We can obtain Change logs using DCRS tool which processes the Git repository and obtains change logs of two predetermined consecutive releases (like Android-4.4\_r1, Android-5.0.0\_r1). The change is due to defect fixation, addition of new functionality, refactoring or other related enhancements. Each change constitutes a single change record. A change logs consists of various information like timestamp of committing, unique identifier, change description (optional) and list of changed lines of source code. The change logs of four releases of Android Contact App Package were obtained.



```
C:/Users/dpi/Desktop/Project/Camera/MM/platform/packages/apps/Camera/-android-6.0.1_r1/BugLog.txt ...  
TIMESTAMP 1414384439  
BEGINNING OF THE COMMIT RECORD  
am 5bb0152d: am 618cae95: am 597fe138: Merge "Remove dependencies on FloatMath"  
  
* commit '5bb0152da4447e94bc3859092f4d0f8c3642e30e':  
  
ENDING OF THE COMMIT RECORD BODY  
0 files changed  
  
TIMESTAMP 1414383671  
BEGINNING OF THE COMMIT RECORD  
am 618cae95: am 597fe138: Merge "Remove dependencies on FloatMath"  
  
* commit '618cae955e32fd4d7ede9a3e979e68ca8d663f15':  
  
ENDING OF THE COMMIT RECORD BODY  
0 files changed  
  
TIMESTAMP 1413557978  
BEGINNING OF THE COMMIT RECORD  
am d8424130: am 597fe138: Merge "Remove dependencies on FloatMath"  
  
* commit 'd8424130792dccacab1f642f15bd62690b68d989':  
  Remove dependencies on FloatMath  
  
ENDING OF THE COMMIT RECORD BODY  
src/com/android/camera/CameraSettings.java | 5 +----  
src/com/android/camera/Util.java | 3 +--  
src/com/android/camera/ui/PieRenderer.iava | 2 +-  
  
Please Select one of these operations to continue: [Delete] [Back]
```

Figure 3.2 DCRS – Change Logs

**Table 3.2 Android - Contacts**

<b>Android OS Release</b>	<b>Code Name</b>
<b>4.4</b>	KitKat ( KK )
<b>5.0.0</b>	Lollipop ( LL )
<b>5.1.1</b>	Lollipop ( LL )
<b>6.0.0</b>	Marshmallow (M)

**3.1.3 Generating Metrics:**

Object Oriented Metrics are being used to evaluate & predict the Software Quality. Object Oriented Metrics are usually used for defect prediction & as an early indicator of externally visible attributes (like coupling, cohesion, inheritance, Encapsulation etc.) as these cannot be measures until too late in software development lifecycle. Chidamber & Kemerer metrics are the most popular used Object Oriented Metrics. Another comprehensive set of metrics is Mood metrics [13, 14, 17].

Object Oriented Metrics were generated using DCRS tool on each Java file. Downloaded Android Packages is first compiled to generate Class files used to generate Object Oriented Metrics. Android code is compiled using the ADT (Android Development Tools). Object Oriented Metrics are collected for each of the classes mentioned in Table 3.1 w.r.t each Android release. Object Oriented Metrics generated using DCRS are described below in Table 3.3:

**Table 3.3 Object Oriented Metrics**

<b>WMC</b>	<b>Weighted Methods Per Class</b>	<b>Count of sum of complexities of number of methods in a class.</b>
<b>NOC</b>	Number Of Children	Number of sub classes of a given class.
<b>DIT</b>	Depth of Inheritance	Tree Provides the maximum steps from the root to the leaf node.
<b>LCOM</b>	Lack of Cohesion	Among Methods of a Class Null pairs not having common attributes.
<b>CBO</b>	Coupling Between Objects	Number of classes to which a class is coupled.
<b>RFC</b>	Response For a Class	Number of external and internal methods in a class.
<b>DAM</b>	Data Access Metric	Ratio of the number of private (and/or protected) attributes to the total number of attributes of a class.
<b>MOA</b>	Measure Of Aggression	Percentage of data declarations (user defined) in a class.
<b>MFA</b>	Method of Functional Abstraction	Ratio of total number of inherited methods to the number of methods in a class.
<b>CAM</b>	Cohesion Among the Methods of a Class	Computes method similarity based on their signatures.
<b>AMC</b>	Average Method Complexity	Computed using McCabe's Cyclomatic Complexity method.
<b>LCOM3</b>	Lack Of Cohesion Among Methods of a Class	Revision of LCOM metric given by Henderson sellers
<b>LOC</b>	Line Of Code	Number of lines of source code of a given class.
<b>NPM</b>	Number of Public Methods	Number of public methods in a given class.
<b>Ca</b>	Afferent Couplings	Number of classes calling a given class.
<b>Ce</b>	Efferent Couplings	Number of other classes called by a class.
<b>IC</b>	Inheritance Coupling	Number of parent classes to which a class is coupled.
<b>Defects</b>	Defect Count	Binary variable indicating the presence or absence of the defects.

We have used six CKJM metrics in our research mentioned below :

### **3.2 Chidamber and Kemerer Java Metrics:**

Chidamber and Kemerer [4] define the so called C&K metric suite. This metric suite offers informative insight into whether developers are following object oriented principles in their design. These metrics collectively helps managers and designers to make better design decision. C&K metrics have generated a significant amount of interest and are currently the most well-known suite of measurements for Object Oriented software. Chidamber and Kemerer proposed six metrics; the following discussion shows their metrics.

#### ***3.2.1 Weighted Methods Per Class (WMC):***

This metric represents number of methods defined in class. It measures the complexity of a class. Complexity of a class can for example be calculated by the cyclomatic complexities of its methods. High value of WMC indicates the class is more complex than that of low values. So class with less WMC is better. As WMC is complexity measurement metric, we can get an idea of required effort to maintain a particular class.

#### ***3.2.2 Depth of Inheritance Tree (DIT):***

This metric shows maximum inheritance path from the class to the root class. DIT metric is the length of the maximum path from the node to the root of the tree. So this metric calculates how far down a class is declared in the inheritance hierarchy. This metric also measures how many ancestor classes can potentially affect this class. DIT represents the complexity of the behaviour of a class, the complexity of design of a class and potential reuse. The deeper a class is in the hierarchy. The more methods & variables it is likely to inherit, making it more complex. A high DIT has been found to increase faults. A recommended DIT is 5 or less.

#### ***3.2.3 Number of Children (NOC):***

This metric shows total Number of immediate sub-classes of a class. This metric measures how many sub-classes are going to inherit the methods of the parent class. The

size of NOC approximately indicates the level of reuse in an application. If NOC grows it means reuse increases. On the other hand, as NOC increases, the amount of testing will also increase because more children in a class indicate more responsibility. So, NOC represents the effort required to test the class and reuse.

A high NOC, a large no. of Child class, indicates following:

1. High reuse of base class. Inheritance is a form of reuse.
2. Base class may require more testing
3. Improper abstraction of parent class.
4. Misuse of sub-classing.
5. High NOC indicates fewer faults.

### ***3.2.4 Coupling between Object Classes (CBO)***

This metric shows number of classes to which a class is coupled. The idea of this metrics is that an object is coupled to another object if two object act upon each other. A class is coupled with another if the methods of one class use the methods or attributes of the other class. An increase of CBO indicates the reusability of a class will decrease. Thus, the CBO values for each class should be kept as low as possible

### ***3.2.5 Response for a Class (RFC)***

RFC is the number of methods that can be invoked in response to a message in a class. Since RFC increases, the effort required for testing also increases because the test sequence grows. If RFC increases, the overall design complexity of the class increases and becomes hard to understand. On the other hand lower values indicate greater polymorphism. The value of RFC can be from 0 to 50 for a class<sup>12</sup>, some cases the higher value can be 100- it depends on project to project.

### ***3.2.6 Lack of Cohesion of Methods (LCOM)***

This metric uses the notion of degree of similarity of methods. LCOM measures the amount of cohesiveness present, how well a system has been designed and how

complex a class is. LCOM is a count of the number of method pairs whose similarity is zero, minus the count of method pairs whose similarity is not zero

### **3.3 Dependent & Independent Variables:**

The binary dependent variable in our study is fault proneness and the independent variables are object oriented metrics. The objective of our study is to explore empirically the relationship between Object Oriented metrics and fault proneness. We have used CKJM Object Oriented metrics as independent variables. Fault proneness is defined as the probability of fault detection in a class. We use machine learning methods to predict the probability of fault proneness. Our dependent variable will be predicted based on the faults found during software development life cycle. The metrics given by [4] are summarized in Table 3.3.



## Chapter 4. Research Methodology

In a quest for answers for our research questions, we have conducted an empirical validation of various techniques on the four releases of the Android Mobile OS given in Table 3.2 using the following steps.

1. Pre-processing of collected data sets.
2. Selection of various ML Techniques for defect prediction.
3. Selection of performance measures and model validation techniques for analysing the Performance of the models developed using Android data sets.
4. Selection of relevant Object Oriented metrics
5. Model development for defect prediction using ML techniques in step 2.

### 4.1 Preprocessing of Data:

We have used six Object Oriented Metrics for defect prediction. The uncorrelated and best attributes are selected out of a set of Object Oriented metrics using correlation based feature selection [24] technique. This technique is simple, fast and widely used method in for sub selecting attributes using the ML techniques. In order to predict models using machine learning techniques, it is important to identify relevant and important features. A relevant feature is one that is correlated to the class and is less related to other features. The correlation based feature selection technique searches all the combinations of attributes to find the best combination of the independent variables. The correlation based feature selection is a heuristic technique that evaluates the correlation between the independent variables and the dependent variable. The correlation based feature selection technique is based on the principle that good attributes are highly correlated with the dependent variables and less correlated amongst themselves. An attribute is selected if the correlation with the dependent variable is higher than the highest correlation amongst the attributes. The aim is to select individual variables that are correlated with the dependent variable and uncorrelated with other independent variables. Thus, the correlation based feature selection technique handles both redundant and irrelevant attributes.

## **4.2 Machine Learning Techniques Selection**

In this study, we have used seven Machine Learning Techniques namely Bayes Net, Naive Bayes, Logistic Regression, KStar, Bagging, Logit Boost, Random Forest.

### ***4.2.1 Bayes Net***

A Bayesian network [9] is computer technology that deals with probabilities in Artificial Intelligence; Bayesian networks (BNs) are graphical models for reasoning under uncertainty, where the nodes represent variables (discrete or continuous) and arcs represent direct connections between them. These direct connections are often causal connections. In addition, BNs model the quantitative strength of the connections between variables, allowing probabilistic beliefs about them to be updated automatically as new information becomes available

Bayesian networks (BNs), also known as belief networks (or Bayes nets for short), belong to the family of probabilistic graphical models (GMs). These graphical structures are used to represent knowledge about an uncertain domain. In particular, each node in the graph represents a random variable, while the edges between the nodes represent probabilistic dependencies among the corresponding random variables. These conditional dependencies in the graph are often estimated by using known statistical and computational methods. Hence, BNs combine principles from graph theory, probability theory, computer science, and statistics.

A Bayesian network is a graphical structure that allows us to represent and reason about an uncertain domain. BNs correspond to directed acyclic graph that is popular in the statistics, the machine learning, and the artificial intelligence societies. BNs are both mathematically rigorous and intuitively understandable. They enable an effective representation and computation of the joint probability distribution over a set of random variables Bayes network or probabilistic directed acyclic graphical model is a probabilistic graphical model (a type of statistical model) that represents a set of random variables and their conditional dependencies via a directed acyclic graph.

Formally, Bayesian networks are directed acyclic graphical whose nodes represent random variables in the Bayesian sense: they may be observable quantities, latent variables, unknown parameters or hypotheses. Edges represent conditional dependencies; nodes that are not connected represent variables that are conditionally independent of each other. Each node is associated with a probability function that takes as input a particular set of values for the node's parent variables and gives the probability of the variable represented by the node.

Efficient algorithms exist that perform inference and learning in Bayesian networks. Bayesian networks that model sequences of variables (e.g. speech signals or protein sequences) are called dynamic Bayesian networks. Generalizations of Bayesian networks that can represent and solve decision problems under uncertainty are called influence diagrams.

#### ***4.2.2 Naive Bayes***

The Naive Bayes [10] Classifier technique is based on the so-called Bayesian theorem and is particularly suited when the dimensionality of the inputs is high. Despite its simplicity, Naive Bayes can often outperform more sophisticated classification methods. Naive Bayes is a simple but important probabilistic model.

The Bayesian Classification represents a supervised learning method as well as a statistical method for classification. Assumes an underlying probabilistic model and it allows us to capture uncertainty about the model in a principled way by determining probabilities of the outcomes. It can solve diagnostic and predictive problems.

This Classification is named after Thomas Bayes [10], who proposed the Bayes Theorem. Bayesian classification provides practical learning algorithms and prior knowledge and observed data can be combined. Bayesian Classification provides a useful perspective for understanding and evaluating many learning algorithms. It calculates explicit probabilities for hypothesis and it is robust to noise in input data

In machine learning, naive Bayes classifiers are a family of simple probabilistic classifiers based on applying Bayes' theorem with strong (naive) independence assumptions between the features.

Naive Bayes has been studied extensively since the 1950s. It was introduced under a different name into the text retrieval community in the early 1960s and remains a popular (baseline) method for text categorization, the problem of judging documents as belonging to one category or the other (such as spam or legitimate, sports or politics, etc.) with word frequencies as the features. With appropriate pre-processing, it is competitive in this domain with more advanced methods including support vector machines. It also finds application in automatic medical diagnosis.

Naive Bayes classifiers are highly scalable, requiring a number of parameters linear in the number of variables (features/predictors) in a learning problem. Maximum-likelihood training can be done by evaluating a closed-form expression which takes linear time, rather than by expensive iterative approximation as used for many other types of classifiers.

### ***4.2.3 Logistic Regression***

Logistics Regression [20] is a regression model where the dependent variable is categorical. Logistic Regression is a type of regression model where the dependent variable (target) has just two values, such as (1/0, Y/N, F/T). Involves a more probabilistic view of classification. The binary logistic model is used to estimate the probability of a binary response based on one or more predictor (or independent) variables (features). As such it is not a classification method. It could be called a qualitative response/discrete choice model in the terminology of economics.

Logistic regression measures the relationship between the categorical dependent variable and one or more independent variables by estimating probabilities using a logistic function, which is the cumulative logistic distribution. Thus, it treats the same set of problems as probit regression using similar techniques, with the latter using a cumulative normal distribution curve instead. Equivalently, in the latent variable interpretations of these two methods, the first assumes a standard logistic distribution of errors and the second a standard normal distribution of errors.

Logistic regression can be seen as a special case of the generalized linear model and thus analogous to linear regression. The model of logistic regression, however, is based on quite different assumptions (about the relationship between dependent and independent variables) from those of linear regression. In particular the key differences of these two models can be seen in the following two features of logistic regression. First, the conditional distribution is a Bernoulli distribution rather than a Gaussian distribution, because the dependent variable is binary. Second, the predicted values are probabilities and are therefore restricted to (0, 1) through the logistic distribution function because logistic regression predicts the probability of particular outcomes.

Logistic regression is an alternative to linear discriminate analysis. If the assumptions of linear discriminant analysis hold, the conditioning can be reversed to produce logistic regression. The converse is not true, however, because logistic regression does not require the multivariate normal assumption of discriminant analysis

#### ***4.2.4 KStar***

KStar [11] is an instance based learning algorithm that uses entropy distance measure. The use of entropy as a distance measure has several benefits. Amongst other things it provides a consistent approach to handling of symbolic attributes, real valued attributes and missing values. We describe K\* [12], an instance-based learner which uses such a measure. The algorithm has a loose relationship to the k-nearest neighbor classifier, a popular machine learning technique for classification that is often confused with k-means because of the k in the name. One can apply the 1-nearest neighbor classifier on the cluster centers obtained by k-means to classify new data into the existing clusters. K\* is a simple, instance based classifier, similar to K-Nearest Neighbor (K-NN). New data instances,  $x$ , are assigned to the class that occurs most frequently amongst the k-nearest data points,  $y_j$ , where  $j = 1, 2, \dots, k$  (Hart, 1968). Entropic distance is then used to retrieve the most similar instances from the data set. Using entropic distance as a metric has a number of benefits including handling of real valued attributes and missing values.

### **4.2.5 Bagging**

Bagging or Bootstrap [21] Aggregating is a machine learning ensemble meta-algorithm designed to improve the stability and accuracy of machine learning algorithms used in statistical classification and regression. Leo Breiman proposed this technique in 1996. It also reduces variance and helps to avoid overfitting. Although it is usually applied to decision tree methods, it can be used with any type of method. Bagging is a special case of the model averaging approach. It improves the performance of classification models by creating various sets of the training sets. The aim is to create numerous similar training sets and train a new function for each of these sets. In the case of class prediction the result of majority voting is considered. In order to construct multiple versions we create bootstrap duplicates of the learning set. These sets are then used as the new learning set. Bagging has the following advantages:

1. It can improve classification accuracy over other classification models.
2. It reduces variance.
3. It avoids overfitting

If we are given a training set  $R_n$ , with  $N$  samples from a population  $P$ , a bootstrap set  $B_i$  would also contain  $N$  samples. For each sample of  $B_i$  we can draw any sample from  $R$  independently and with replacement from  $R$ . Hence we would obtain some samples being repeated in  $B_i$  and some samples being present in  $R$  but not in  $B_i$ . Hence the dataset  $B_i$  is as plausible as  $R_n$  but is drawn from  $R$  rather than  $P$ . The bootstrap datasets are combined by taking the average output and hence we get an aggregated prediction result. In this work we use Bag Size Percent of 100, 10 iterations and REP tree as classifier. These are the default setting provided by the WEKA tool

### **4.2.6 Logit Boost**

In machine learning and computational learning theory, LogitBoost is boosting algorithm formulated by Jerome Friedman, Trevor Hastie, and Robert Tibshirani [18, 19]. The original paper casts the AdaBoost algorithm into a statistical framework. Specifically, if one considers AdaBoost as a generalized additive model and

then applies the cost functional of logistic regression, one can derive the LogitBoost algorithm. Logit Boost performs additive logistic regression. It use decision stump (creates binary one-level decision tree algorithm) classification algorithm designed to be used for boosting algorithms

#### ***4.2.7 Random Forest***

Random forests are an ensemble learning method for classification (and regression) that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes output by individual trees. The algorithm for inducing a random forest was developed by Leo Breiman and Adele Cutler [15, 16], and "Random Forests" is their trademark. The term came from random decision forests that were first proposed by Ho of Bell Labs in 1995 [15, 16]. The method combines Breiman's "bagging" idea and the random selection of features, introduced first by Ho and later by Amit and Geman [25] in order to construct a collection of decision trees with controlled variance.

Random Forest constructs a forest of multiple trees and each tree depends on the value of a random vector. For each of the tree in the forest, this random vector is sampled with the same distribution and independently. Hence, random forest is a classifier that consists of a number of decision trees. The resultant output class is the mode of classes output by the individual trees.

The algorithm for constructing the tree is as follows: For  $M$  training sets,  $N$  variables in the classifier and the variable  $n$  ( $n \ll N$ ) where  $n$  indicates the number of independent

Variables that determine the decision at the terminal node of the tree. A bootstrap training sample is selected. The best split is based on these  $n$  variables in the training set. Each tree is allowing growing fully and is not pruned. RFs have the following benefits:

1. RFs are simple and Runs efficiently on large data bases. It can handle thousands of input variables without variable deletion
2. RFs are comparatively robust to outliers and noise.

3. RFs provide give useful internal estimates of error, strength, correlation and Variable importance. It gives estimates of what variables are important in the classification.
4. RFs may produce a highly accurate classifier for various data sets. It has an effective method for estimating missing data and maintains accuracy when a large proportion of the data are missing
5. RFs provide fast learning. Generated forests can be saved for future use on other data.
6. It computes proximities between pairs of cases that can be used in clustering, locating outliers or (by scaling) give interesting views of the data.
7. It offers an experimental method for detecting variable interactions.

### **4.3 Application of Algorithms**

WEKA tool is for implementing algorithms. Correlation based feature selection technique is applied as preprocessing technique using the Object Oriented Metrics attributes- WMC, DIT, NOC, CBO, RFC, LCOM, LOC.

In Figure 4.1, WEKA is used to pre-process the selected data set. WEKA is capable of reading '.csv' format files. Data is loaded into WEKA, We have performed a series of operations using WEKA's attribute. We have used the GUI interface for WEKA Explorer.

In Figure 4.2, we have used WEKA for executing different Machine Learning Techniques & generating results with respect to each Android release. Results shows performance measures like Sensitivity, Precision, F Measure, ROC etc.



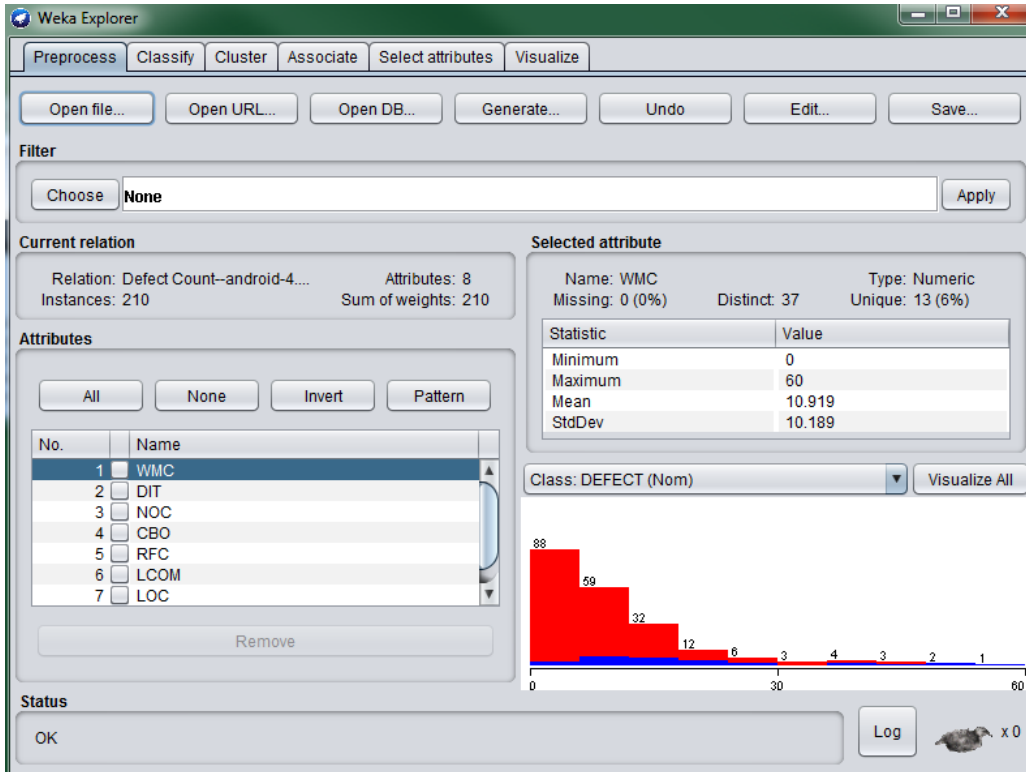


Figure 4.1 WEKA - Preprocess

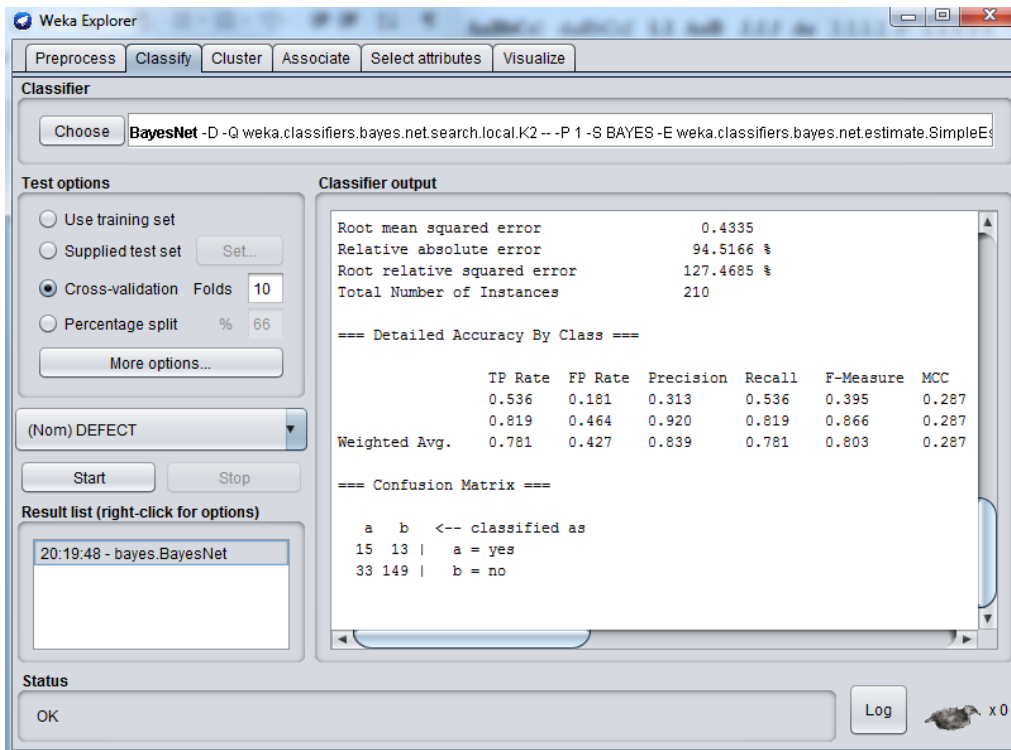


Figure 4.2 WEKA - Classify

#### 4.4 Performance Evaluation

A variety of performance measures have been used in the literature to examine the strength of the developed models using different ML techniques. The data having disproportionate ratio of defective and not defective classes is often known as unbalanced data. Given the imbalanced nature of the datasets, the ROC analysis is a commonly used performance measure. The ROC curve depicts the percentage of correctly predicted defective classes (sensitivity) on the y-axis versus the one minus the percentage of correctly predicted non-defective classes (1- specificity) on the x-axis at various cut-off points. Hence, in ROC analysis the sensitivity and 1-specificity of the developed model is calculated at each cut-off point. To compare various ML techniques the ROC curves are drawn for each ML technique. The Area Under Curve (AUC) computed using ROC analysis lies between 0 and 1 and higher the value of the AUC better is the predictive capability of the developed model. The advantage of using AUC for evaluating performance of the models is that it can deal with noisy and unbalanced data as it is insensitive to changes in class distribution.

In order to validate the predicted models we use the following performance measures:

1. Sensitivity: Also known as True Positive Rate (TP Rate) or Recall measures the proportion of positives that are correctly identified. It is defined as the ratio of classes predicted as faulty to the total number of classes actually faulty.
2. Specificity: It Measures the proportion of negatives that are correctly identified. It is defined as the ratio of classes predicted as non-faulty to the total number of classes actually non faulty.
3. Precision: It is defined as the ratio of classes predicted correctly as faulty and non-faulty to the total number of classes.
4. ROC analysis: The output of the predicted models can be analyzed using ROC analysis. ROC curve is a plot of sensitivity (on the y-axis) and 1-specificity (on the x-axis). Many cut off points are selected between 0 and 1 while the construction of ROC curves. AUC is a measure obtained using ROC analysis. This gives optimal cut off point that maximizes both sensitivity and specificity.

This measure is very effective in measuring the quality of the predicted models and is popularly being used in machine learning research. The following rules can be used to categorize AUC:

**Table 4.1 ROC Value**

ROC Value	Remarks
$\leq 0.5$	No Discrimination
$0.7 \leq \text{ROC} < 0.8$	Acceptable Discrimination
$0.8 \leq \text{ROC} < 0.9$	Excellent Discrimination
$\text{ROC} \geq 0.9$	Outstanding Discrimination

We have used AUC as a measure to evaluate and assess the models predicted using machine learning techniques.

#### **4.5 Model Evaluation Results:**

In this section we will discuss about performance evaluation of various ML Techniques for defect prediction on generated data set OO metrics indicated above and the outcome of the prediction model based on our work. Below are the evaluation parameters for used Machine Learning Algorithms with respect to four Android OS release. The results of models predicted using machine learning techniques were predicted using WEKA tool. The predicted models are validated using 10-fold cross validation.

After this we empirically compared the ML techniques and the results were evaluated in terms of the AUC. The AUC has been advocated as a primary indicator of comparative performance of the predicted models. The AUC measure can deal with noisy and unbalanced data and is insensitive to the changes in the class distributions. The ML technique yielding best AUC for a given release was highlighted.

Table 4.2 to Table 4.8 shows results for different performance parameters TP rate, FP Rate, Precision, Recall, F Measure, and ROC Area with respect to various Machine Learning Techniques.

#### 4.5.1 Bayes Net

Table 4.2 shows the evaluation results for Bayes Net ML technique:

**Table 4.2 Evaluation Results for Bayes Net**

ML Technique	Android OS Version	TP Rate	FP Rate	Precision	Recall	F Measure	ROC Area
Bayes Net	4.4	0.78	0.43	0.84	0.78	0.8	0.71
	5.0.0	0.68	0.31	0.71	0.68	0.69	0.68
	5.1.1	0.75	0.41	0.75	0.75	0.73	0.71
	6.0.0	0.59	0.43	0.59	0.59	0.59	0.58

#### 4.5.2 Naïve Bayes

Table 4.3 shows the evaluation results for Naïve Bayes ML technique:

**Table 4.3 Evaluation Results for Naïve Bayes**

Attributes	Android	TP Rate	FP Rate	Precision	Recall	F Measure	ROC Area
Naïve Bayes	4.4	0.85	0.57	0.84	0.85	0.85	0.75
	5.0.0	0.61	0.27	0.74	0.61	0.6	0.68
	5.1.1	0.75	0.39	0.74	0.74	0.73	0.76
	6.0.0	0.7	0.48	0.69	0.7	0.66	0.66

#### 4.5.3 Logistic Regression

Table 4.4 shows the evaluation results for Logistic ML technique:

**Table 4.4 Evaluation Results for Logistic Regression**

Attributes	Android	TP Rate	FP Rate	Precision	Recall	F Measure	ROC Area
<b>Logistic</b>	4.4	0.86	0.75	0.82	0.86	0.83	0.74
	5.0.0	0.65	0.45	0.64	0.65	0.64	0.7
	5.1.1	0.78	0.34	0.77	0.77	0.77	0.76
	6.0.0	0.7	0.41	0.69	0.7	0.69	0.72

**4.5.4 KStar**

Table 4.5 shows the evaluation results for KStar ML technique:

**Table 4.5 Evaluation Results for KStar**

Attributes	Android	TP Rate	FP Rate	Precision	Recall	F Measure	ROC Area
<b>K Star</b>	4.4	0.82	0.6	0.82	0.82	0.82	0.68
	5.0.0	0.66	0.34	0.66	0.66	0.66	0.69
	5.1.1	0.7	0.42	0.69	0.7	0.69	0.72
	6.0.0	0.66	0.4	0.66	0.66	0.66	0.67

**4.5.5 Bagging**

Table 4.6 shows the evaluation results for Bagging ML technique:

**Table 4.6 Evaluation Results for Bagging**

Attributes	Android	TP Rate	FP Rate	Precision	Recall	F Measure	ROC Area
<b>Bagging</b>	4.4	0.87	0.78	0.84	0.87	0.83	0.78
	5.0.0	0.71	0.37	0.7	0.7	0.7	0.75
	5.1.1	0.7	0.43	0.7	0.7	0.7	0.7
	6.0.0	0.72	0.39	0.71	0.72	0.71	0.7

**4.5.6 Logit Boost**

Table 4.7 shows the evaluation results for Logit Boost ML technique:

**Table 4.7 Evaluation Results for Logit Boost**

Attributes	Android	TP Rate	FP Rate	Precision	Recall	F Measure	ROC Area
<b>Logit Boost</b>	4.4	0.85	0.78	0.8	0.85	0.82	0.75
	5.0.0	0.72	0.31	0.73	0.72	0.73	0.73
	5.1.1	0.76	0.37	0.76	0.76	0.75	0.71
	6.0.0	0.72	0.43	0.71	0.72	0.69	0.7

#### *4.5.7 Random Forest*

Table 4.8 shows the evaluation results for Random Forest ML technique:

**Table 4.8 Evaluation Results for Random Forest**

Attributes	Android	TP Rate	FP Rate	Precision	Recall	F Measure	ROC Area
<b>Random Forest</b>	4.4	0.84	0.69	0.81	0.84	0.82	0.68
	5.0.0	0.68	0.38	0.68	0.68	0.68	0.73
	5.1.1	0.73	0.4	0.72	0.73	0.71	0.71
	6.0.0	0.66	0.43	0.65	0.65	0.66	0.7

## Chapter 5. Conclusion

In Our work we have found relationship between CKJM Metrics & Fault Proneness of a class. In Table 5.1, AUC results for most of the models predicted using various Machine Learning techniques on Android App Package ‘Contacts’ is 0.7 which depicts the predictive capability of ML techniques. Bagging, Naïve Bayes & Random Forest shows best prediction with AUC value 0.75. ML Techniques with AUC 0.7 & above is highlighted in bold.

Hence, we can conclude our work as ML models for defect prediction developed can be used for identifying defective classes in subsequent releases of Android OS Data Sets( like Android KitKat to Lollipop to Marshmallow ). Developed Models can be also applied in future to different projects that are similar in nature.

Table 5.1 shows 10-fold Cross Validation Results of 7 ML Techniques with respect to AUC.

**Table 5.1 Result Summary**

<b>Android OS Release</b>	Bayes Net	Naïve Bayes	Logistic	Kstar	Bagging	Logit Boost	Random Forest
<b>4.4</b>	<b>0.71</b>	<b>0.75</b>	<b>0.74</b>	0.68	<b>0.78</b>	<b>0.75</b>	<b>0.74</b>
<b>5.0.0</b>	0.68	0.68	<b>0.7</b>	0.69	<b>0.75</b>	<b>0.73</b>	<b>0.7</b>
<b>5.1.1</b>	<b>0.71</b>	<b>0.76</b>	<b>0.76</b>	<b>0.72</b>	<b>0.7</b>	<b>0.71</b>	<b>0.76</b>
<b>6.0.0</b>	0.58	0.66	<b>0.72</b>	0.67	<b>0.7</b>	<b>0.7</b>	<b>0.72</b>

In future we have planned to enhance scope of our work to larger data sets & more Machine Learning techniques. Current work is focused only on one Android module ‘Contacts’. In future we can consider more than one modules to understand the relationship of Chidamber & Kemerer Java matrices on defective classes.

## Bibliography

- [1] R. Malhotra and R. Raje, "An Empirical Comparison of Machine Learning Techniques for Software Defect Prediction," in *ICST*, Brussels, Belgium, 2014.
- [2] Y. Singh, A. Kaur and R. Malhotra, "Software Fault Proneness Prediction Using Support Vector Machines," in *World Congress on Engineering, WCE 2009*, London, U.K, 2009.
- [3] A. Kaur and I. Kaur, "An empirical evaluation of classification algorithms for fault prediction in open source projects," in *Journal of King Saud University - Computer and Information Sciences*, 2016.
- [4] T. Gyimothy, R. Ferenc and I. Siket, "Empirical validation of object-oriented metrics on open source software for fault prediction," *IEEE Transactions on Software Engineering*, vol. 31, no. 10, pp. 897-910, 2005.
- [5] Y. Zhou and H. Leung, "Empirical analysis of object oriented design metrics for predicting high severity faults" *IEEE Transactions on Software Engineering*, vol. 32, no. 10, pp. 771-784, 2006.
- [6] R. Malhotra, K. Nagpal, P. Upmanyu and N. Pritam, "Defect Collection and Reporting System for Git Based Open Source Software" *International Conferenc on Data Mining & Intellegent Computing(ICDMIC)New Delhi,India, 2014*. pp. 1-7
- [7] S. R. Chidamber and C. F. Kammerer, "A Metrics Suite for Object Oriented Design," *IEEE Transactions on Software Engineering*, vol. 20, no.6, pp. 20, 6, 476-493., 1994.
- [8] R. Malhotra and Y. Singh, "On the Applicability of Machine Learning Techniques for Object Oriented Software Defect Prediction," *An International Journal (SEIJ)*, vol. 1, no. 1, pp. 24-37, 2011.
- [9] T. M. Mitchell, "Machine Learning," McGraw Hill, 2015, pp. 1-17.
- [10] J. G. Cleary and L. E. Trigg, "K\*: An Instance-based Learner Using an Entropic



- Distance Measure," University of Waikato, New Zealand, 2001.
- [11] S. Vijayarani and M. Muthulakshmi, "Comparative Analysis of Bayes and Lazy Classification Algorithms," *International Journal of Advanced Research in Computer and Communication Engineering*, vol. 2, no. 8, pp. 3118-3124, 2013.
- [12] M. JURECZKO and D. D. SPINELLIS, "Using Object-Oriented Design Metrics to Predict Software Defects," *Proceedings of RELCOMEX 2010: Fifth International Conference on Dependability of Computer Systems DepCoS*, p. 69–81, 2010.
- [13] J. Bansiy and C. G. Davis, "A Hierarchical Model for Object Oriented Design Quality Assessment," *IEEE*, vol. 28, no. 1, pp. 4-16, 2002.
- [14] F. Ruggeri, F. Faltin and K. R. "Bayesian Networks," in *Encyclopedia of Statistics in Quality & Reliability*, Wiley & Sons, 2007.
- [15] G. Biau, "Analysis of a Random Forests Model," *Journal of Machine Learning Research*, pp. 1063-1095, 2012.
- [16] L. Breiman, "Random forests," in *Machine Learning*, 2001, vol. 45, no. 1, pp. 5–32.
- [17] S. Chawla, "Review of MOOD and QMOOD metric sets," *International Journal of Advanced Research in Computer Science and Software Engineering*, vol. 3, no. 3, pp. 448-451, 2013.
- [18] R. E. Schapire, "The Boosting Approach to Machine Learning," *Nonlinear Estimation and Classification*, pp. 1-16, 19 December 2001.
- [19] S. B. Kotsiantis and P. E. Pintelas, "Logitboost of Simple Bayesian Classifier," *Informatica 29*, pp. 53-59, 30 November 2004.
- [20] J. Yan, M. Koç and J. Lee, "A prognostic algorithm for machine performance assessment and its application," *Taylor & Francis Online*, vol. 15, no. 8, pp. 796-801, 2007.
- [21] L. Breiman, "Bagging predictors," *Machine Learning*, vol. 24, pp. 123-140, 1996.
- [22] R. Malhotra, "Comparative Analysis of statistical and machine learning methods for predicting faulty modules," *Appl. Soft Computing*, vol. 21, pp. 286-297, 2014.
- [23] R. Malhotra, A. Kaur and Y. Singh, "Empirical validation of object-oriented metrics

for predicting fault proneness at different severity levels using support vector machines," *Int. J. Systems Assurance Eng. and Management*, vol. 1, no. 3, pp. 269-281, 2010.

[24] Hall, M. 2007. Correlation-based feature selection for discrete and numeric class machine learning. In *Proceedings of the 17th Int. Conference on Machine Learning*. pp. 359–366.

[25] Amit, Yali and Donald, Geman , "Shape quantization and recognition with randomized trees" *Neural Computation*, MIT Press, vol. 9, no. 7, pp. 1545-1588, 1997.