A DISSERTATION

ON

DEVELOPMENT OF SOFTWARE CHANGE PREDICTION MODEL USING DEEP LEARNING TECHNIQUE

Submitted in partial fulfilment of the requirements
for the award of the degree of

MASTER OF TECHNOLOGY
In
SOFTWARE TECHNOLOGY

Submitted by
Avanish Shah
University Roll No. 2K14/SWT/507

Under the Esteemed Guidance of
Dr. Ruchika Malhotra
Associate Head & Associate Professor,
Department of Computer Science & Engineering, DTU

2015-2018

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING
DELHI TECHNOLOGICAL UNIVERSITY,
DELHI– 110042, INDIA

DELHI TECHNOLOGICAL UNIVERSITY

DELHI-110042

DECLARATION

I hereby declare that the thesis entitled "**DEVELOPMENT OF SOFTWARE CHANGE PREDICTION MODEL USING DEEP LEARNING TECHNIQUE"** which is being submitted to the **Delhi Technological University**, in partial fulfillment of the requirements for the award of the 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:

Avanish Shah
2K14/SWT/507

DELHI TECHNOLOGICAL UNIVERSITY

DELHI-110042

CERTIFICATE

This is to certify that thesis entitled "**DEVELOPMENT OF SOFTWARE CHANGE PREDICTION MODEL USING DEEP LEARNING TECHNIQUE",** is a bona fide work done by Mr. Avanish Shah (Roll No: 2K14/SWT/507) in partial fulfillment 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 & ASSOCIATE PROFESSOR,
DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING.
DELHI TECHNOLOGICAL UNIVERSITY, DELHI 110042

ACKNOWLEDGEMENT

I am presenting my work on "**DEVELOPMENT OF SOFTWARE CHANGE PREDICTION MODEL USING DEEP LEARNING TECHNIQUE**" with a 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 her 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 has 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 Program and to gain knowledge through this program. This curriculum provided me the knowledge and an opportunity to grow in various domains of computer science.

Avanish Shah
2K14/SWT/507

ABSTRACT

With the increasing demands of timely delivery of software, software testing has become very much important for any software product. Hence, testing has become an inevitable part in software development cycle. Every stakeholder wants product with minimum bugs or defects at runtime. For removing defects as early as possible, we need to provide more optimized approach for finding out defects in any software. Even after spending thousands of dollars on it, we find defects in the software in livable condition, and these defects hamper the brand image of the organization and provide inconvenience to end user as well. Hence we need to develop such products which have minimum or no defect, high in Quality and cost effective in terms of adding new features or modification in the software.

The cost of finding issues in software is directly proportional to the time of its finding in the development cycle. Later found bugs are more expensive w.r.t. previously found defects.

Analysis of statistical analysis of code and software binary together between two consecutive releases of product based on software metrics can be useful in predicting changes in the software product. So focus can be shifted to change prone areas majorly in testing, and hence more chances of finding issues in those areas.

With the help of these software metrics data from Statistical analysis, software change prediction model can be generated that can be useful in predicting issues in later releases of same software. Thus the development of predictive models to predict faulty or defective classes can help & guide the stakeholders in early phase of the software development cycle.

The objective of thesis is to do statistical analysis of code & binary together and then build Deep Learning (DL) based multilayer perceptron model [1] over several Android data sets. The evaluation is performed with an intention to find the effectiveness of the DL based model for prediction of classes' change in software based on software quality metrics. Software quality metrics used in our study are CKJM, McCabe, Halstead [2] that were generated on 7 android module projects i.e. Contacts, DeskClock,

ExactCalculator, Launcher3, ManagedProvisioning, PackageInstaller, and Settings over Android Nougat (7.0) and Android Oreo (8.0) releases.

DL model in our study is 2 step-model that predicts potentially change prone classes within a given set of software project with respect to its metric data. The data set used in our experiment is organized in two forms: one for learning and other for prediction purpose, or the training set & the testing set. We evaluate model developed using DL with Bayes net ML technique [3] over same data and we found that our DL based multilayer perceptron model performs comparable even on medium size data.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ACRONYMS

Chapter 1: Introduction

A lot of surveys are done in past for calculating cost of software, and more than 90% of projects exceeds the cost of their pre-decided budget. Statistics shows importance of predicting potential errors early in software development life cycle & taking the necessary steps before these results come out.

Releasing defect free software projects have always been a difficult task. Especially for larger projects, the task of testing becomes more expensive. The cost of fixing a defect increases exponentially if defects are uncovered towards the end of software development or after product delivery.

On the other hand, evaluating software in a continuous and the disciplined manner bring many benefits like accurate estimation of project cost & schedule, hence improves the quality of product & process. We know that most of defects in two consecutive releases of any software can be found in the delta part of two release. So, software change prediction between two releases can play a very vital role in increasing testing coverage of released software. As it helps in keeping focus of testing limited to those change prone areas of software and thus useful in reducing testing.

Hence, the challenges of effective testing lead to the research area of identifying change prone classes in early phase & aligning the test activity accordingly to increase the maximum coverage in software testing.

In this research, we created change prediction model on medium size OO project using multilayer perceptron based DL technique [1] and calculate its effectiveness by comparing it to Bayes net Machine Learning (ML) technique [3].

Model is developed using OO metrics [2] [4] that are basic characteristics of any OO software. These software metrics which capture various properties (like coupling, cohesion, encapsulation, inheritance, no. of classes, LOC, etc.) of software shall be used for developing models for predicting classes' change proneness in the software. OO metrics' collected from past release of same software (Android subsequent releases Nougat to Oreo) are used for developing the change predicting model. The developed change prediction model can then be subsequently used for classifying the classes of

current projects as containing errors or error free and helping to keep testing efforts only to those areas.

In our work we have developed models for individual projects using Bayes Net ML technique [3] and DL-MLP [5] based technique and check the performance of this technique on subsequent releases of 7 application packages of popular mobile operating system Android. We have used OO metrics for prediction of the change proneness in classes. The results were evaluated & compared based on ROC [6] analysis.

Chapter 2: Literature Review

Several studies were done in the past to relate software metrics with change proneness using ML and DL techniques. Some of the key studies are discussed below.

Malhotra and Khanna [7] deeply studied about relationship between OO metrics & change proneness. Change prediction based model is very helpful in identifying the change prone class which would helpful to focus testing on those areas only and lead to better results. Model developed can be used to decrease the probability of error occurrence and helpful in better maintenance.

Malhotra and Khanna [8] have evaluated the performance of ten ML techniques and searchbased techniques on 3 open source software. Here, author developed change predicting model using two data sets and performs inter-project validations in order to obtain the unbiased results & perfromace of this study yeilding the good result.

Singh, Kaur and Malhotra [9] proposed to find out the relationship of OO metrics & fault proneness in a class. They used seven ML and one logistic regression method so as to predict faulty class categories. The result obtained from this work was based on data set fetched from the open source software. The results show that the predictive accuracy of ML technique Logit Boost is highest with AUC of 0.806. Malhotra [10] also did comparison of different ML techniques to get better performing method.

Li, He and Zhu [11] have proposed framework called Defect Prediction using Convolutional Neural Network (DP-CNN), which allows DL to genetate effective features. It is based on Abstract Syntax Trees (ASTs) of the programs. Here, Author firstly extracted the token vectors, then encoded them to the numerical vectors via mapping & then via words embedding. Then numerical vectors are input to DP-CNN to learn semantic & structural features of programs automatically. These learned features were combined with the program's handcrafted features, for accurate change prediction in software. This method is evaluated on 7 open source projects for checking F-measure in the defect predicting software. The experiment results showed that on an average, performance of defect prediction via convolutional neural network was better then state of art method by tweleve percentage.

Kaur and Kaur [12] proposed SVM based model to find out the relationship between OO metrics given by C&K [2] with change susceptibility. The model proposed was efficiently verified on the KC1 NASA data set using public domain. The performance of SVM based method was then evaluated using analysis of ROC curves [6]. Based on above experiment results, it is efficient to claim that these models could help us in planning & testing part by focusing resources on change-prone parts of the code structure and designing. Thus, the study shows that the SVM method is also useful in the generation of quality model.

Chapter 3: Research Background

Here, we will see the data collection process, tools used in our experiment, OO metrics generation etc.

3.1 Data set generation:

In this study, OO Metrics were obtained using open source mobile OS – Android. 8 Android packages namely Contacts, DeskClock, ExactCalculator, Launcher3, ManagedProvisioning, PackageInstaller, and Settings over Android Nougat and Android Oreo releases" are considered for generating the data sets.

Source code is fetched from Google GIT repository [13] *(https://android.googlesource.com/platform/packages/apps/Contacts/)* for above application packages. Android source code contains java files. First Android code is compiled to generate the class files from the java Files. Our study is focus only on bugs related to functionality, i.e. bugs occurred in java files. We built .class files from downloaded Android Source code through different build methodology [13] of partial building of Android source code. Once .class files are generated for the above packages, Defect Collection and Reporting System (DCRS) tool [14] is used to generate the reports having OO metrics. DCRS tool has integrated CKJM tool which calculates C&K OO metrics [2] by processing the bytecode of the java classes. The program takes input from each class & source code file & generated the OO metrics as mentioned in Table 3.1.1.

Characteristics of different android application package with respect to Android 7.1 and 8.0 releases are mentioned in Table 3.1.1.

Table 3.1.1: Dataset for different android packages for android tag: 7.1.1_r28 & 8.0.0_r4

Packages	Total Classes	Classes having Changes	Change Percentage %
Contacts	172	121	70
DeskClock	112	83	73
ExactCalculator	11	9	82
Launcher3	240	182	76
ManagedProvisioning	58	51	88
PackageInstaller	68	26	38
Settings	580	386	67

GIT is open source versioning control system used for source code management task for Google android code. GIT as a distributed revision control system is aimed for speed, integrity of data and support for non-linear, distributed workflows. Google GIT Repository: https://android.googlesource.com/platform/packages/apps/...

Table 3.1.1 contains android app packages data sets with total class, total number Of classes having changes w.r.t. Android 7.1 and 8.0 release for 7 Application Package. Changes were generated using DCRS Tools developed by the Delhi Technical University (DTU) students.

3.1.1 Downloading Source Code using DCRS Tool:

DCRS Tool [15] is a JAVA based automated tool which collects and reports various changes, defects, bugs or issues which were present in a given version of android Operating System (OS) w.r.t. previous versions of android OS. Matrix generation from DCRS Tool depends on 2 subsequent releases of Java Project over GIT.

Various studies in the past have been done that showed change data collected from the open source operating system i.e. Android is used in research areas of change suspecting. Some of the commonly traversed areas of change prediction include

validation & analysis of the effect of given metrics, on change suspecting, and applicability of such metric suite for the prediction of change suspecting models.

DCRS tool determines the deleted source files, newly added source files, change, etc. It efficiently collects change data from above files that can be used in research areas.

DCRS tool 1st obtain the defect logs of android source files & then filtered them to obtain the defects which were present in a given android OS version & have been fixed in the next released version. The system filters changed logs to extract useful change information like a unique change identifier and change-description, if any.

DCRS tool also associates changes to the relating source files (java files, or simply class files). Then, it performs computation of the total number of changes in every class, i.e., the number of changes that are associated with that class. Finally, the corresponding values of different metric suites are obtained by the system for each class files in the source code of previous version of android OS.

Install & configure GIT first, for extracting the change-logs for source code of each version of the Android OS. Find the path of each android application on Google site: (*https://android.googlesource.com*) for corresponding TAGs i.e. android-7.1.1_r28 and android-8.0.0_r4. Now, download source code of each application for corresponding versions for DCRS tool or directly using tag and application path with command line, source code of both the versions is required to generate the change logs. Versioning can be seen through above GIT Tags. Figure 3.1.1 shown below is the tool UI to download the source code of android application.

Figure 3.1.1: DCRS Tool in download mode

We can fetch information from DCRS as per below method. It processes two versions of source code to generate change logs using GIT. Change log provides description regarding the modifications that have been made in the source code. These change-logs are further processed to get bug-logs. We can retrieve bug-ids and description from the bug-logs. These bug-ids are mapped to the classes in source code. Based on the above gathered information, DCRS generates the following reports:

a. Bug-Report –Contains details of each bug data, class-wise (bug-id and description)

b. Bug-Count report - Contains bug-count (class-wise), CKJM and other metrics data for each class

c. Change Report – contains total inserted and deleted LOC class-wise, for all incurred changes

We can collect change data from android OS change logs as per below steps:

3.1.2 Generating change logs from DCRS tool:

We can obtain change logs using DCRS tool which processes the Git repository and obtains change logs of two predetermined consecutive releases (like Android-7.1.1_r28 and Android-8.0.0_r4). The change is due to errors, 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 and a list of changed lines of the source code. Here, we obtained change log for 7 android application projects between their 2 consecutive releases (Android-7.1.1_r28 & Android-8.0.0_r4). Figure 3.1.2 is DCRS tool UI displaying the GIT change logs.

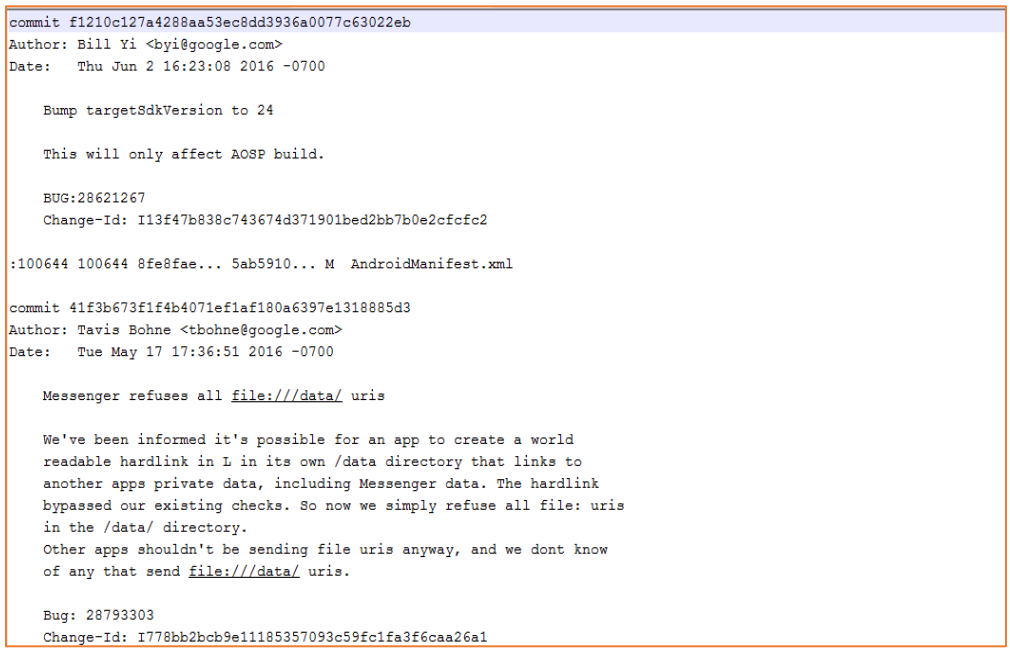

Figure 3.1.2: DCRS Tool - Change logs

3.1.3 Generating class files from android application:

We downloaded the complete android source code separately for Tag android-7.1.1_r28 and android-8.0.0_r4 for generating the class files that was used for generating OO metrics.

Then, we built code on the Linux server machine with the below set of commands [13] to generate binary (.class) files:

Table 3.1.2: List of commands for building Android packages

Serial No	Commands
1.	>curl https://storage.googleapis.com/git-repo-downloads/repo > ~/bin/repo
2.	chmod a+x ~/bin/repo
3.	repo init -u https://android.googlesource.com/platform/packages/apps/Contacts/ -b android-7.1.1_r28
4.	repo sync
5.	source build/envsetup.sh
6.	lunch aosp_arm-eng
7.	make javac-check-Contacts

It will create class files in following folder: /out/target/obj/APPS/Contacts_Intermediates/classes/. Figure 3.1.3 shows the created class files at above specified folder location in the system.

Source code along with generated class files combined will be input in DCRS Tool.

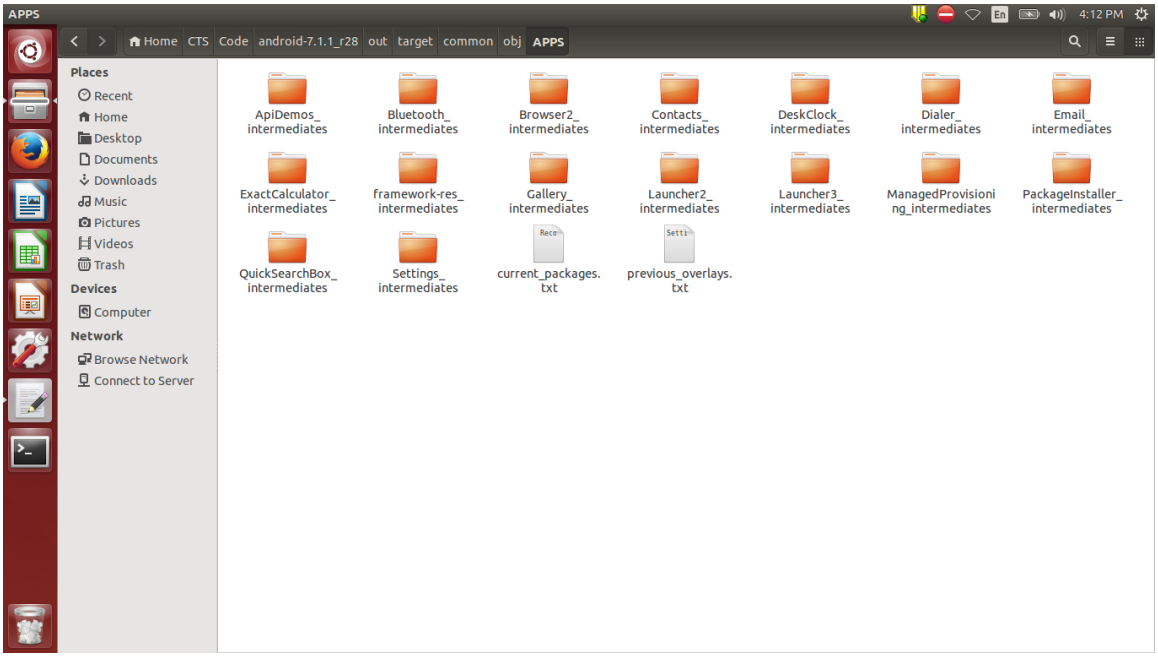

Figure 3.1.3: Class files generated for Contact Package

3.1.4 OO Metrics generation using DCRS Tool:

OO metrics is used to predict & evaluate the software's quality. OO metrics generated is used for change prediction & as an early indicator of externally visible attributes (like cohesion, coupling, Encapsulation, inheritance etc.) CKJM metrics is the most popular used as OO Metrics. Other metrics that is also used is Mood metrics [2] [16] [12].

OO Metrics were generated using DCRS tool on each Java file. We provided the path of generated class files and downloaded source code to tool, and tool generated OO metrics for each of the classes of android application packages w.r.t Android 7.1 & 8.0 release. Figure 3.1.4 illustrates the OO metrics generation process.

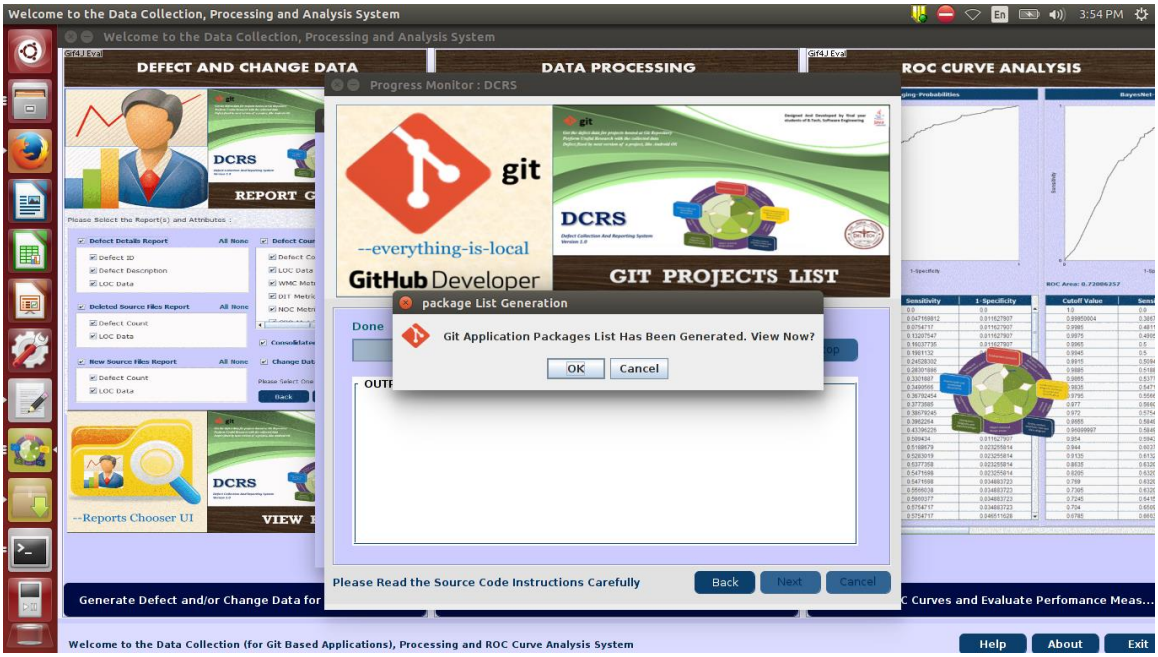

Figure 3.1.4: DCRS Tool- OO Metrics generation

OO metrics generated using DCRS Tool is displayed below in Table 3.1.3:

Table 3.1.3: CKJM OO Metrics generated from DCRS Tool

Abbreviation	Attribute Name	Description
WMC	Weighted Methods Per Class	Count of sum of complexities of number of methods in a class.
NOC	Number Of Children	Number of sub classes of a given class.
DIT	Depth of Inheritance	Tree Provides the maximum steps from the root to the leaf node.
LCOM	Lack of Cohesion	Among Methods of a Class Null pairs not having common attributes.
CBO	Coupling Between Objects	Number of classes to which a class is coupled.
NPM	Number of Public Methods	Number of public methods in a given class.
Ca	Afferent Couplings	Number of classes calling a given class.

3.1.4.1 CKJM Metrics:

C&K [2] define the so called C&K metric suite. This metric suite offers informative insight whether developers are following OO principles in their design & development. This metrics helps managers to create higher style selections. C&K metrics is incredibly standard among the researchers conjointly also and it's the most well-known suite of measurements for OO software quality. C&K had projected six metrics. Following discussion describes its attributes:

3.1.4.2 Weighted Methods Per Class (WMC):

WMC represents total number of the methods defined in any class. It calculates the complexity of any class and it is can be checked by the cyclomatic complexity of the methods. More is the value of WMC shows class is more complex than less values. Hence, class with low WMC value is better. As WMC is quality mensuration metric and it provide a plan of needed effort in maintenance of a particular class.

3.1.4.3 Depth of Inheritance Tree (DIT):

DIT shows maximum inheritance distance from the class to its base class. It is the length of the maximum distance from the child node to the base of the tree. Hence, this metric calculates how far a class is present in the inheritance hierarchy. It is used to check number of ancestor classes that can potentially impact this class. DIT shows the complexity of the behaviour of any class, the design complexity of any class and its potential reuse. The deeper is the class in its hierarchy, more methods and variables it will likely to inherit, making it more complex. A high DIT indicates increase errors in the project and recommended value of DIT is 5 or less.

3.1.4.4 Number of Children (NOC):

NOC shows total number of immediate sub-class of any class. It measures sub classes' number that is inheriting the methods of its parent class. NOC size indicates the reuse of code in any application. If NOC value increases then it means more reuse of

code. On the other hand, if NOC value increase, then it means more checking of code will be needed because more children in a class which indicate greater responsibility of class. Hence, NOC displays total efforts required to test the class & its reuse.

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

1. High reuse of a base-class. Inheritance is reusing of code.
2. Base class might require more test.
3. Improper use of abstract for parent class.
4. Improper of sub-classes.
5. High NOC indicates lesser bugs in code.

3.1.4.5 Coupling between Object Classes (CBO)

CBO shows coupling between the classes. If any object is using other object then it is said to be coupled. A class is coupled with another class if the methods of one class is using the methods of second class. An increase in CBO value shows decrease in class reusability. Hence, the CBO for each class must be as less as possible.

3.1.4.6 Response for a Class (RFC)

For any response to message, RFC is the number of methods that are called. As RFC value increases, testing efforts also get increases as testing sequence grows. Design complexity of a class increases with increase in RFC value and it becomes harder to understand. On other side, its lower value represents more polymorphism. RFC values lies between 0 and 50 for any class, it can increase up to 100 for some cases depending on project.

3.1.4.7 Lack of Cohesion of Methods (LCOM)

LCOM metric represents degree of equality between the methods. It shows the degree of cohesiveness in the software, i.e. way of designing of the system and amount of complexity of the class. LCOM is subtraction of the number of method pairs whose

likeness is zero and count of method pairs whose similarity is not zero. So, LCOM value should be kept Low and cohesion high.

3.2 Independent and Dependent Variables:

In our study, the dependent variable is the change that occurred in the class & the OO metrics of the class is the independent variables. The objective of our study is to establish the relation of OO metrics and the change in a class. We have used CKJM metrics with other OO metrics as independent variables. We use DL method to predict change in a class. Our dependent variable will be forecasted based on the change found during SDLC. It is also calculated using DCRS tool along with OO metrics generation. The metrics given by C&K [2] are summarized in Table 3.2. In figure 3.2.1, change is the dependent variable which dependent on independent variables i.e. WMC & NOC, CBO, RFC, LCOM & Ca, NPM and DIT.

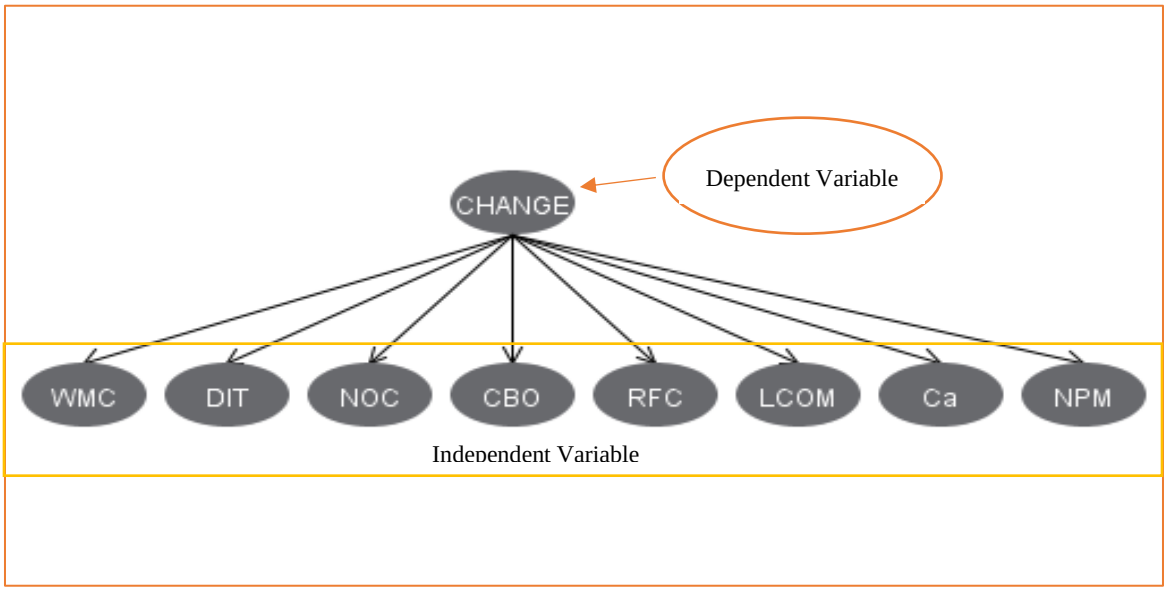

Figure 3.2.1: Independent and Dependent Variables

Chapter 4: Research Methodlogy

To answers our research questions, we have conducted an empirical validation of various ways on two releases of the android OS given in Table 4.3.1 using the following steps.

1. Pre-processing of android data-sets.
2. Building DL based model for the change prediction.
3. Predicting changes between two android releases.
4. Performance evaluation based on comparison between ML based model and DL based model.
5. Model evaluation results.

4.1 Preprocessing of Data:

We have used eight OO metrics for change prediction. Uncorrelated and the best attributes are selected out of a set of OO metrics using correlation based feature selection [11] technique. This technique is simple, widely used and very fast method in for sub selecting attributes using the DL technique. In order to predict models using DL technique, 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. Feature selection technique based on correlation searches all the combinations of attributes to find the best combination of the independent variables. The feature selection technique based on correlation is a heuristic technique that computes the correlation between the independent & dependent variable. The feature selection technique based on correlation is based on the principle that good attributes are those that are highly correlated among the dependent variables and that are less correlated amongst them. An attribute is selected if the correlation with the dependent variable is higher than the highest correlation amongst the attributes. The aim in our study is to get individual variables that are correlated with the dependent variables and uncorrelated with other independent variables. Thus, the

correlation based feature selection technique handles both redundant and irrelevant attributes.

4.2 Building Model based on DL Based technique

In DL [1] based computational models made of multiple processing layers that learn multi-level abstracted data representations.

It finds detailed structure in large data sets by using the back-propagation algorithm to give idea how any machine should change its internal prams which are used to differentiate the representation of particular layer from the representation in the previous layer.

It has been very impressive in state of art in the visual object, speech recognition and many other domains. With such high effectiveness in other domains, we applied it in predicting change in two consecutive versions of software.

In this study, we have used DL based technique. After creating the dataset, the next step is to build a neural network model based on DL. As we are building the model in JAVA, there is a library called deeplearning4j [5] which is open source library. We implemented our work using deeplearning4j library and Weka tool [17]. DL [18] can be implemented using this library alone, but Weka provides GUI platform to input various tuning parameters used in it, that is useful in reducing time of coding. Figure 4.2.1 illustrates about deeplearning4j library.

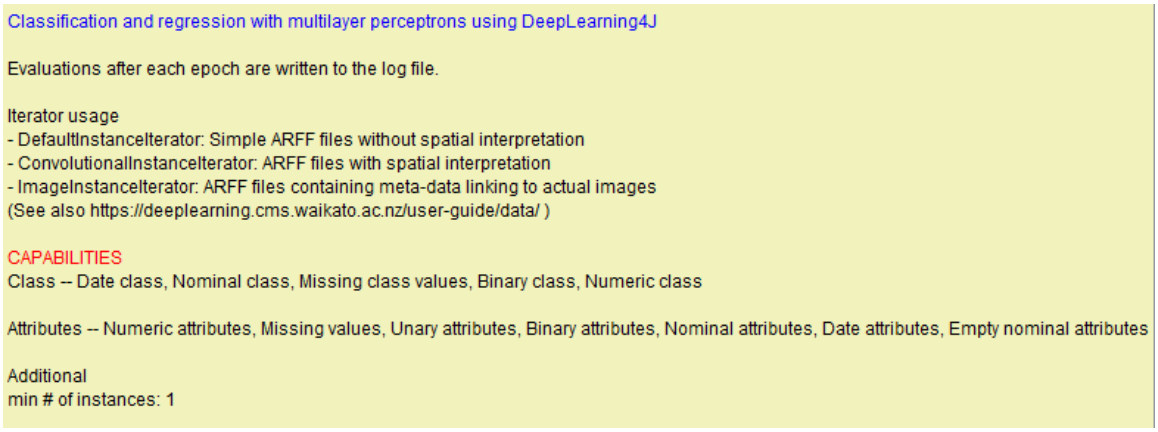

Figure 4.2.1: Deeplearning4J and Weka integrated library (University of Waikato, 2018) Specification

4.3 Predicting Changes between 2 Android Release

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

In Figure 4.3.1, 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.

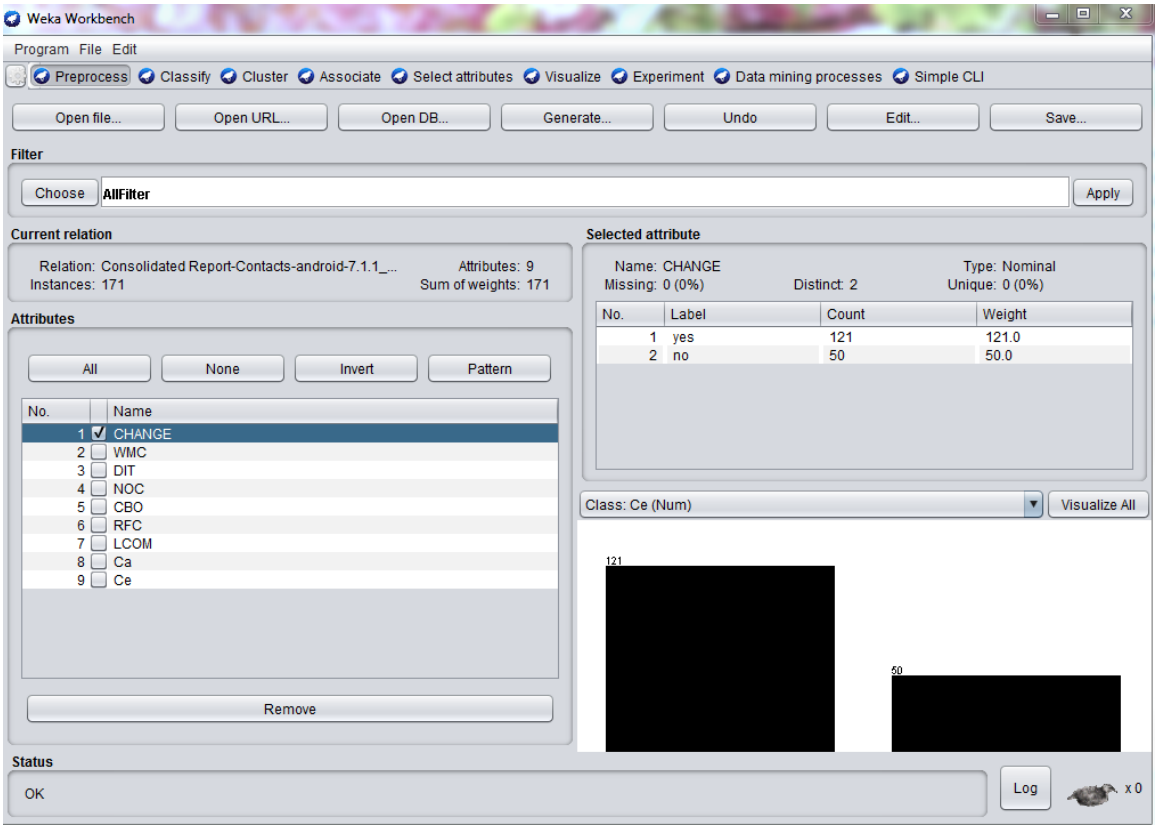

Figure 4.3.1: WEKA - Preprocess

In Figure 4.3.2, we have used WEKA for executing DL based algorithm & generating results with respect to each android release for different applications. Results shows performance measures like confusion matrix, sensitivity, precision, F-Measure, ROC etc.

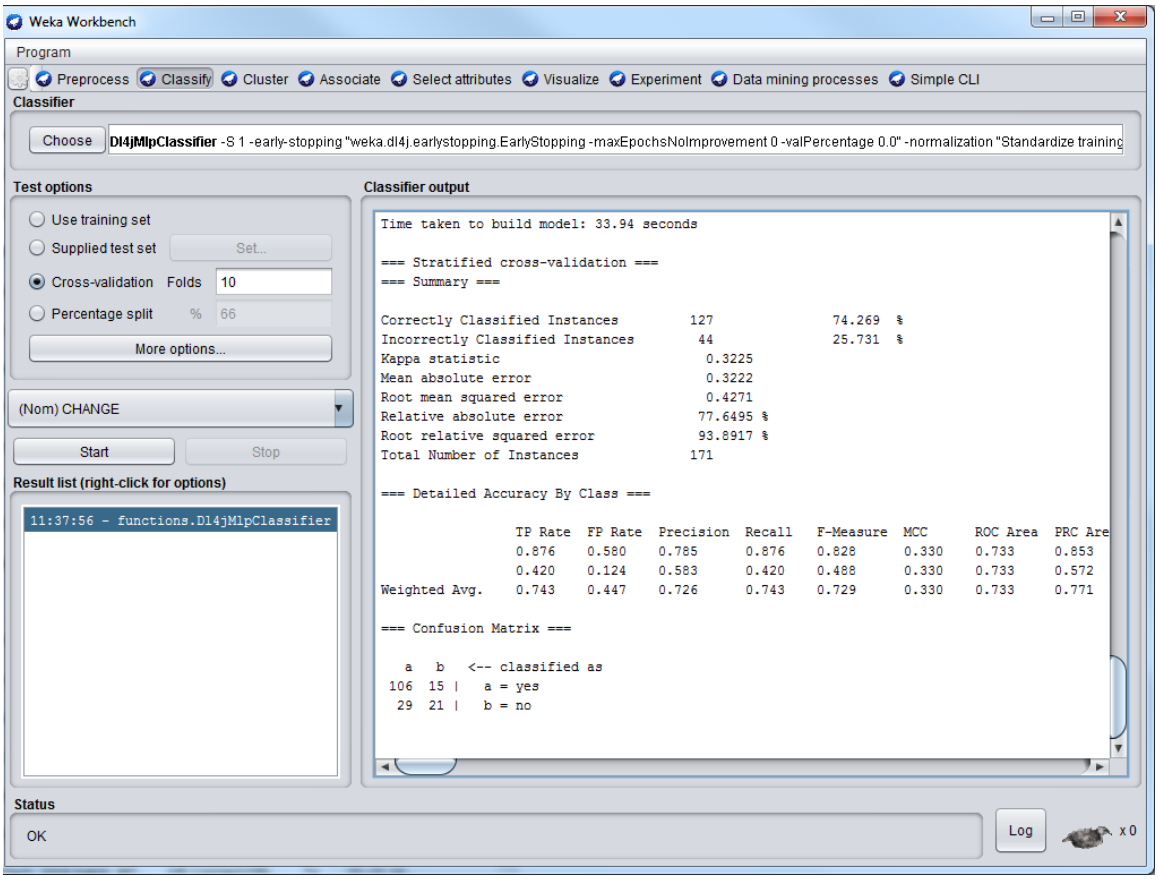

Figure 4.3.2: Deep learning classification using DL4J-MLP classifier (University of Waikato, 2018)

Table 4.3.1: Table Dataset Description

Project Description	Versions	Total Files	Change Rates (%)
Contacts	android-7.1.1_r28, android-8.0.0_r4	171	71.8
DeskClock	android-7.1.1_r28, android-8.0.0_r4	112	74.1
Dialer	android-7.1.1_r28, android-8.0.0_r4	395	99.0
ExactCalculator	android-7.1.1_r28, android-8.0.0_r4	11	81.8
Launcher3	android-7.1.1_r28, android-8.0.0_r4	240	75.8
ManagedProvisioning	android-7.1.1_r28, android-8.0.0_r4	58	87.9
PackageInstaller	android-7.1.1_r28, android-8.0.0_r4	68	38.2
Settings	android-7.1.1_r28, android-8.0.0_r4	581	66.4

Table 4.3.1 states about dataset description of android applications that, we obtained from calculation from DCRS Tool developed by Malhotra and Nagpal [15]. Basically, it calculates changes between files of 2 versions by parsing GIT reference logs file.

4.4 Performance Evaluation

In this section, we evaluate effectiveness of our DL model on comparing accuracy of change prediction method with other state of art methods. In particular, our evaluation resolves the following Research Questions (RQ):

RQ1: ROC Analysis for the evaluation of change suspecting / prediction model based on DL.

RQ2: Do the DL based methods outperform traditional ML methods.

All experiments here were executed on Linux-Ubuntu machine. Unless otherwise stated, every experiment was run for at least 5 times and average result was reported.

4.4.1 Evaluation Metrics

To evaluate the prediction accurateness, we use a widely adopted following metrics [19], [20]:

F-measure (or F1 score), which is harmonic mean of recall & precision [16]. We first represent some notations here in displaying recall, precision and F-measure:

(a) Predict the changed-file as change-file (c → c);

(b) Predict the changed-file as clean-file (c → c1); and

(c) Predict the cleaned-file as changed-file (c1 → c).

N denotes the amount of files in every above definition, e.g., Nc→c for the 1st case. Then, our metrics can be defined as follows:

Precision: The ratio of total files really buggy to the total files classified as buggy.

$$Precision; P = \frac{Nc \rightarrow c}{Nc1 \rightarrow c}$$

Recall: The ratio of the total files correctly classified as buggy to the total number of truly buggy files.

$$Recall;\ R\ = \frac{Nc \rightarrow c}{(Nc \rightarrow c\ +\ Nc \rightarrow c1)}$$

F-measure: The traditional F-measure (F1 score) is the harmonic mean of total precision value P and the recall R value.

$$F - measure;\ F\ = \frac{2\ *\ P\ *\ R}{(P\ +\ R)}$$

TP Rate: True Positive (TP) is positive tuples correctly labeled by the classifier.TP Rate is the ratio of TP and TP plus False Negative (FN)

$$TP\ Rate;\ TP\ = \frac{TP}{(TP + FN)}$$

FP Rate: False Positive (FP) is the false alarms. There are the negative tuples that are incorrectly labeled as positive. FP Rate is ratio of FN and FN plus True Negative (TN).

$$FP\ Rate: FP\ = \frac{FP}{(FP\ +\ TN)}$$

ROC analysis [6]: The output of the evaluated models can be analyzed using analysis of ROC curves. ROC curve is a graph 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 the measure obtained using ROC analysis lies between 0 and 1 and higher the AUC value means good is the prediction capacity of the developed model. This gives us optimal cut off point that maximizes both as well as sensitivity & the specificity. This measure is very effective in measuring the quality of the predicted models and is popularly being used in ML research. The following rules can be used to categorize AUC: Table 4.2 illustrates the validation of outputs from ROC analysis.

Table 4.4.1: ROC Values

ROC Value	**Remarks**
ROC=< 0.5	No Discrimination
0.7 =< ROC < 0.8	Acceptable Discrimination
0.8 =< ROC < 0.9	Excellent Discrimination
ROC => 0.9	Outstanding Discrimination

4.5 Model Evaluation Results:

In this section, we will discuss about evaluation of performances of various DL techniques for model based on change prediction for 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 DL Algorithms with respect to 2 Android OS release

on 8 android modules. The results of models predicted using DL techniques were predicted using WEKA tool with the help of deeplearning4j library. The predicted models are verified using 10-fold cross validation technique in weka tool.

After this, we empirically compared the ML techniques and the results were evaluated on basis of the AUC. The AUC is widely accepted by researchers as a primary indicator of performance comparison of the various predicted models as AUC is helpful in dealing with unbalanced and noisy data also and it doesn't get impacted by the changes in the class distributions. The deep technique yielding best AUC for a given release will be highlighted.

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

4.5.1 Contact Package

Table 4.5.1 shows comparison between DL classification with multilayer perceptron (DL-MLP) and ML classification with Bayes Net technique. Figure 4.5.1 displays the ROC curves of Bayes Net ML technique and figure 4.5.2 shows the ROC curves of the MLP based on DL technique.

Table 4.5.1: Comparison for ML and DL Techniques for Contact Package

Technique	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area
DL-MLP	0.754	0.383	0.747	0.754	0.750	**0.763**
ML-Bayes Net	0.754	0.301	0.768	0.754	0.759	0.744

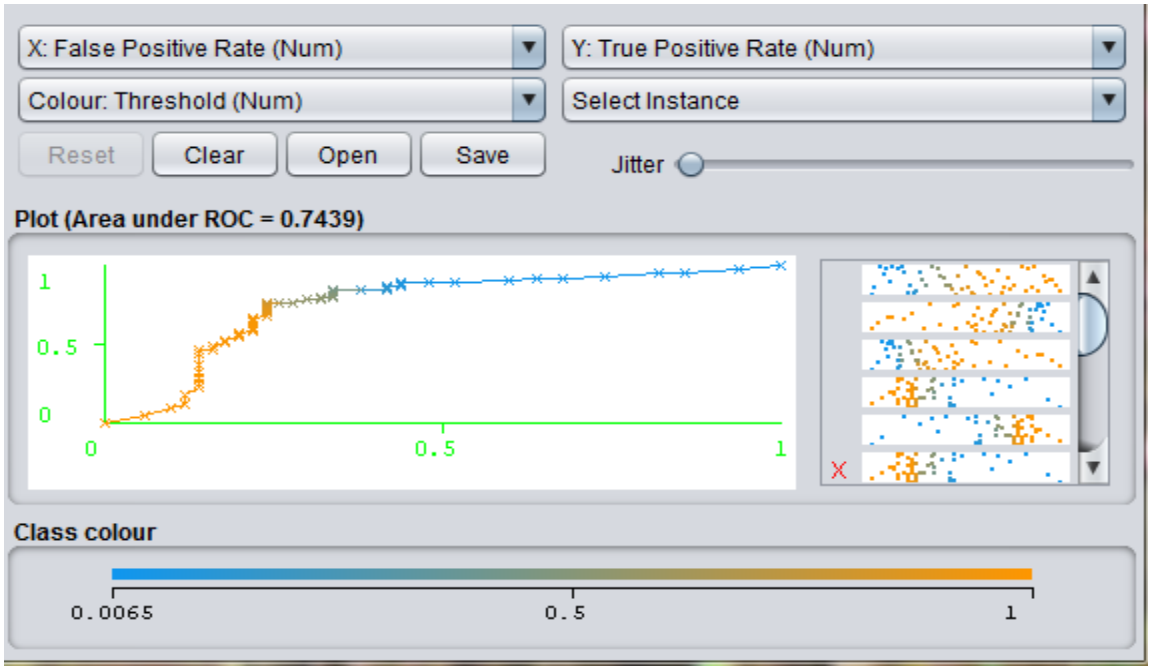

Figure 4.5.1: ROC Curve of ML-Bayes Net Classification for Contact Package

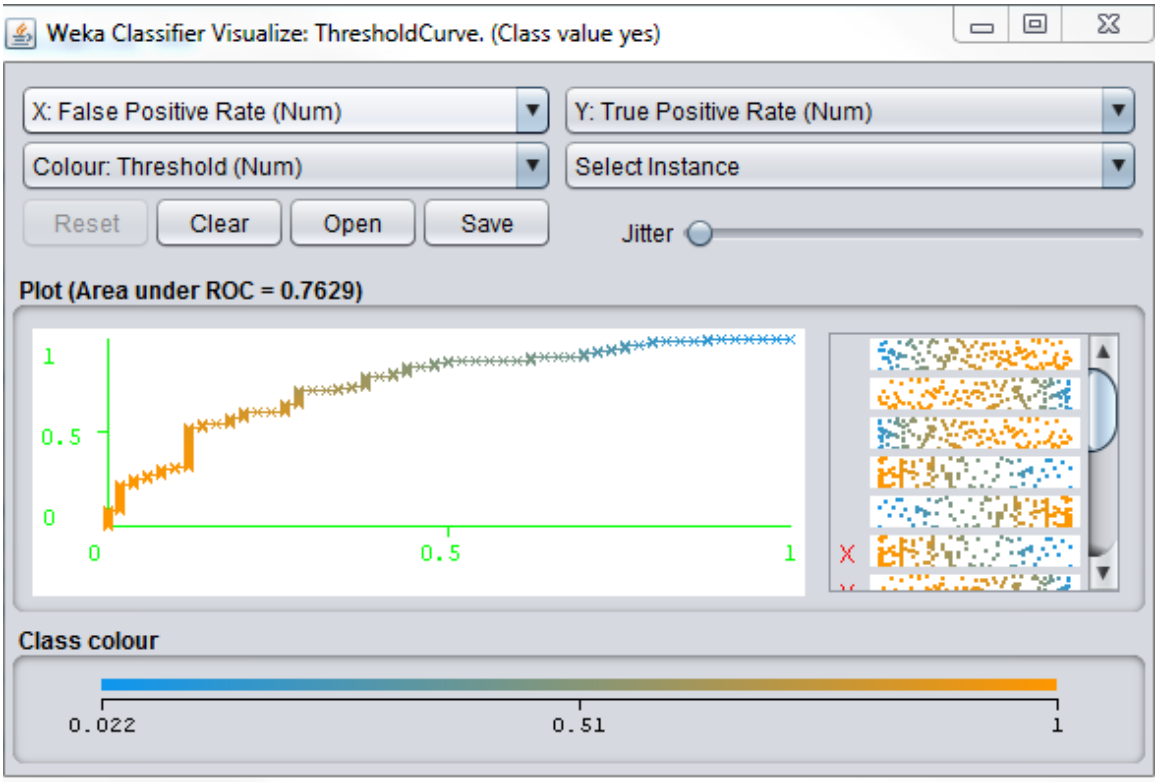

Figure 4.5.2: ROC Curve of DL-MLP Classifier for Contact Package

4.5.2 DeskClock Package

Table 4.5.2 shows comparison between DL-MLP and ML classification with Bayes Net technique. Figure 4.5.3 shows the ROC curves of Bayes Net ML technique & figure 4.5.4 shows the ROC curves of the MLP based on DL technique. Here, ML based Bayes net technique outperforms DL-MLP technique.

Table 4.5.2 : Comparison for ML and DL Techniques for DeskClock Package

Technique	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area
DL-MLP	0.723	0.545	0.696	0.723	0.705	0.702
ML-Bayes Net	0.786	0.254	0.809	0.786	0.793	**0.803**

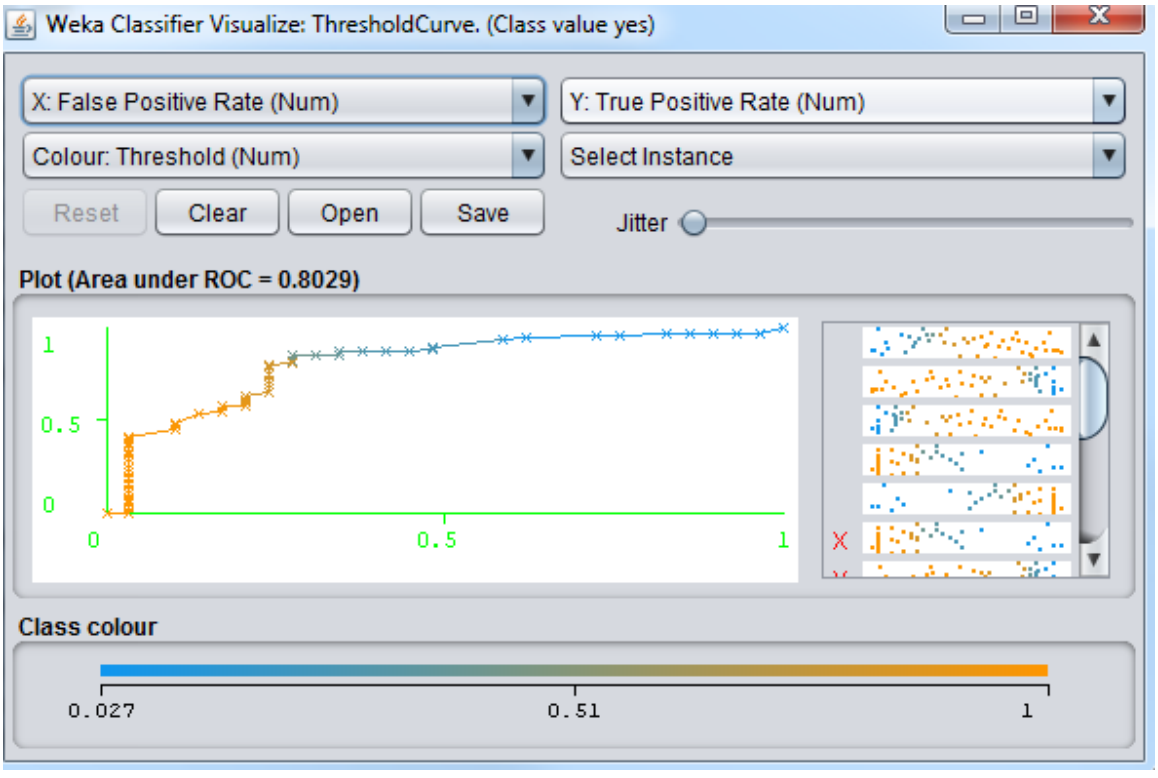

Figure 4.5.3: ROC Curve of ML-Bayes Net Classification for DeskClock Package

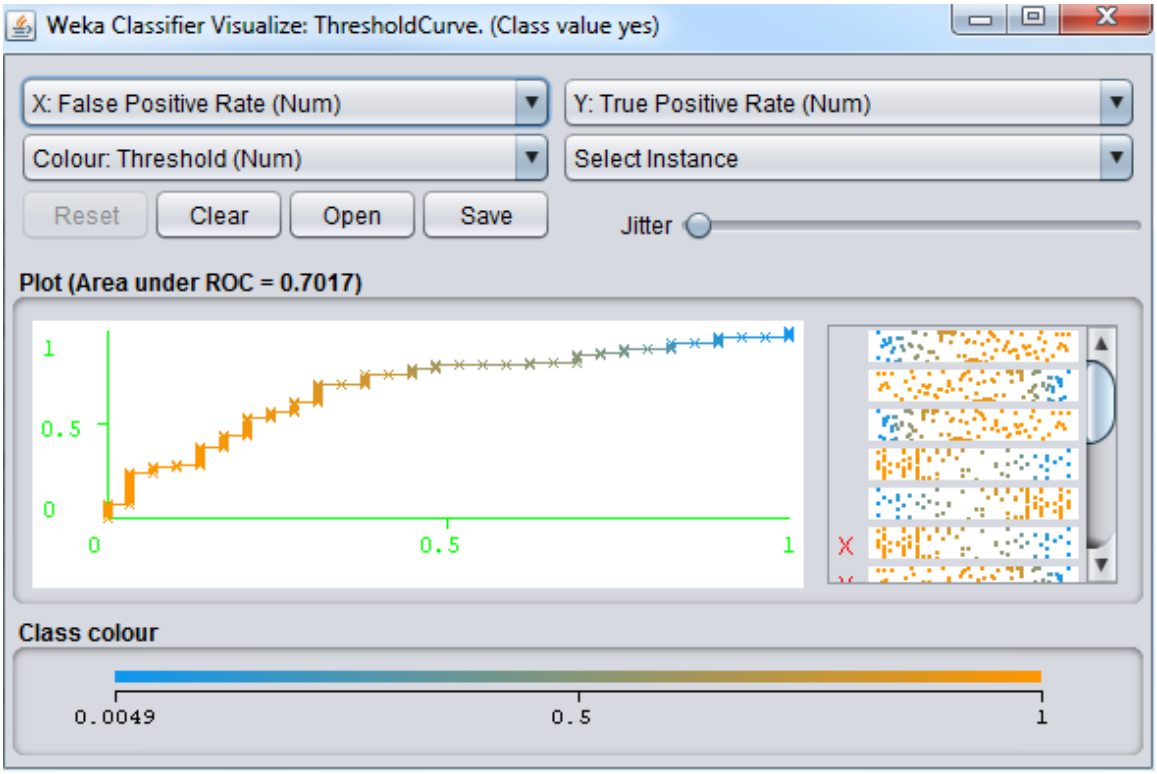

Figure 4.5.4 : ROC Curve of DL-MLP Classifier for DeskClock Package

4.5.3 ExactCalculator Package

Table 4.5.3 shows comparison between DL-MLP and ML classification with Bayes Net technique. Figure 4.5.5 shows the ROC curves of Bayes Net ML technique & figure 4.5.6 displays the ROC curves of the MLP based on DL technique. Here, DL-MLP technique outperforms ML-Bayes Net technique.

Table 4.5.3: Comparison for ML and DL Techniques for ExactCalculator Package

Technique	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area
DL-MLP	0.727	0.838	0.655	0.727	0.689	0.778
ML-Bayes Net	0.818	0.818	0.669	0.818	0.736	**0.917**

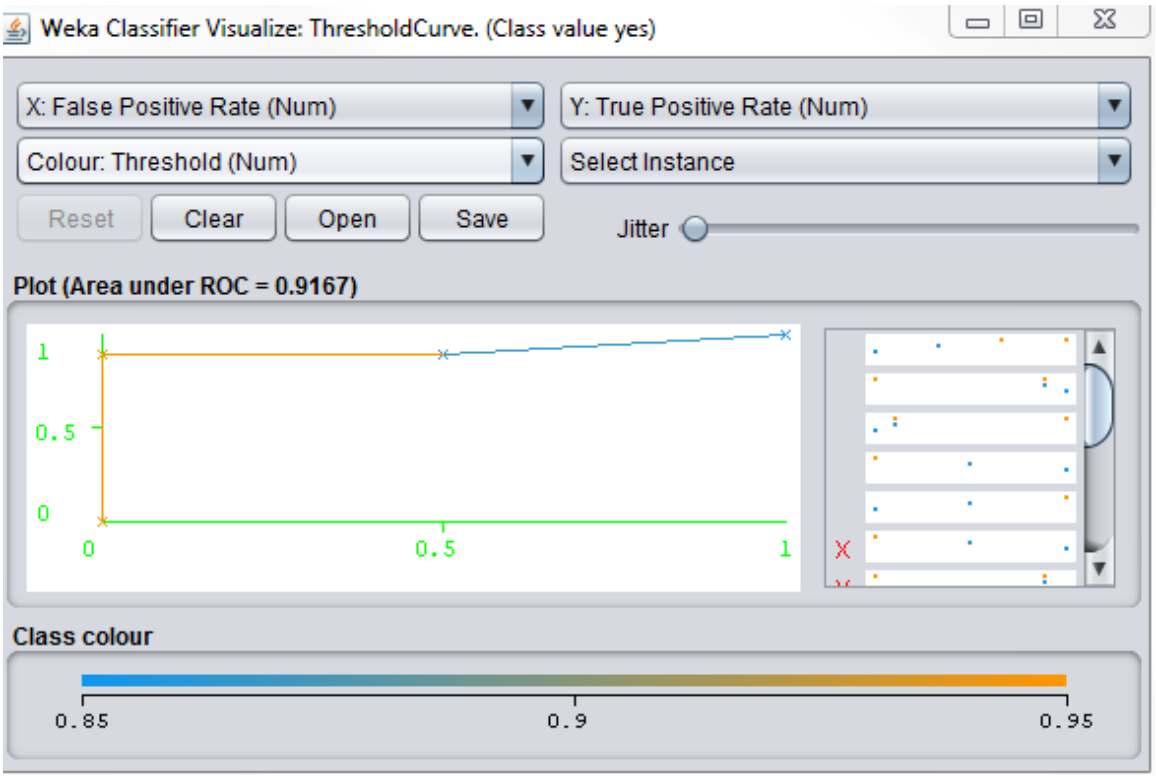

Figure 4.5.5: ROC Curve of ML-Bayes Net Classification for ExactCalculator Package

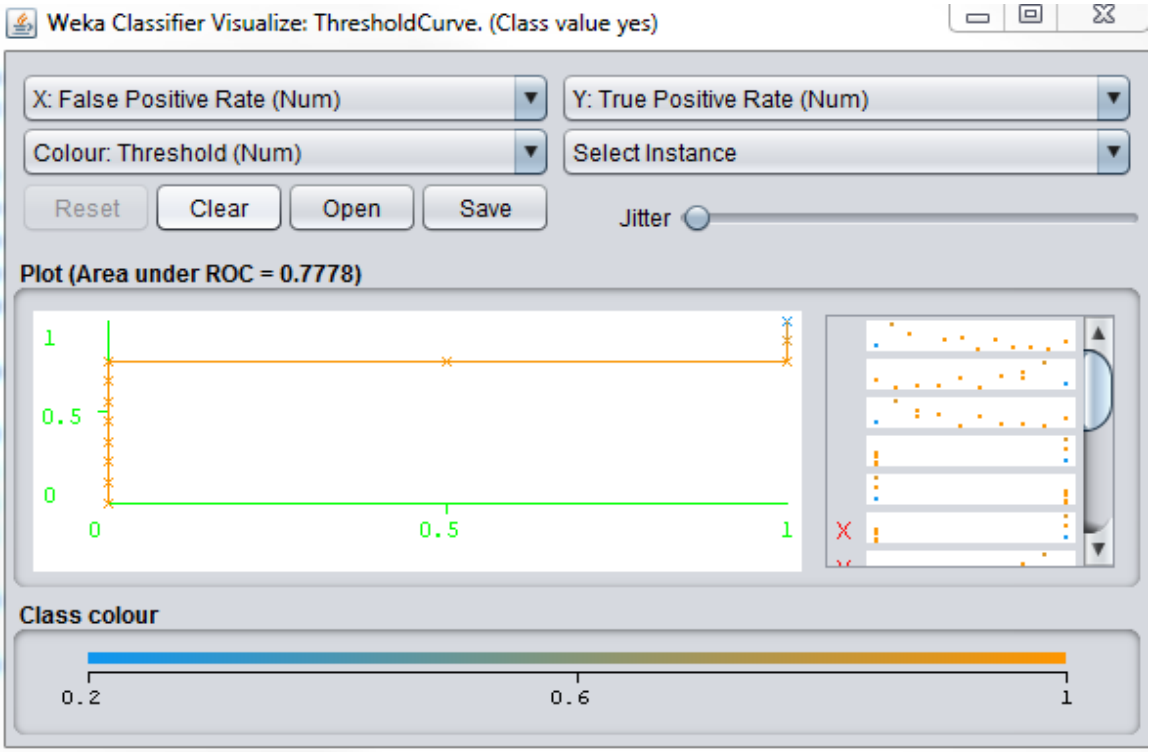

Figure 4.5.6: ROC Curve of DL-MLP Classifier for ExactCalculator Package

4.5.4 Launcher3 Package

Table 4.5.4 shows comparison between DL-MLP and ML classification with Bayes Net technique. Figure 4.5.7 illustrates the ROC curves of Bayes Net ML technique & figure 4.5.8 shows the ROC curves of the MLP based on DL technique. Here, performance of both the techniques is comparable.

Table 4.5.4: Comparison for ML and DL Techniques for Launcher3 Package

Technique	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area
DL-MLP	0.704	0.588	0.681	0.704	0.691	0.656
ML-Bayes Net	0.646	0.477	0.691	0.646	0.663	**0.665**

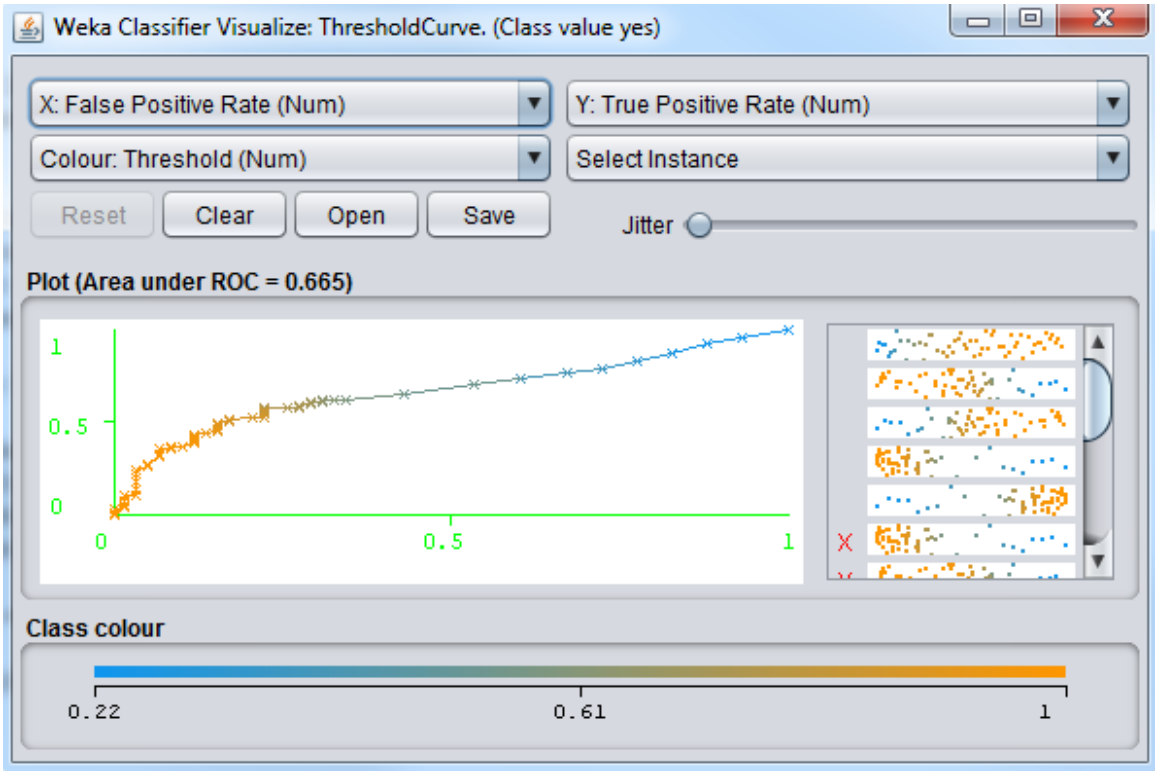

Figure 4.5.7: ROC Curve of ML-Bayes Net Classification for Launcher3 Package

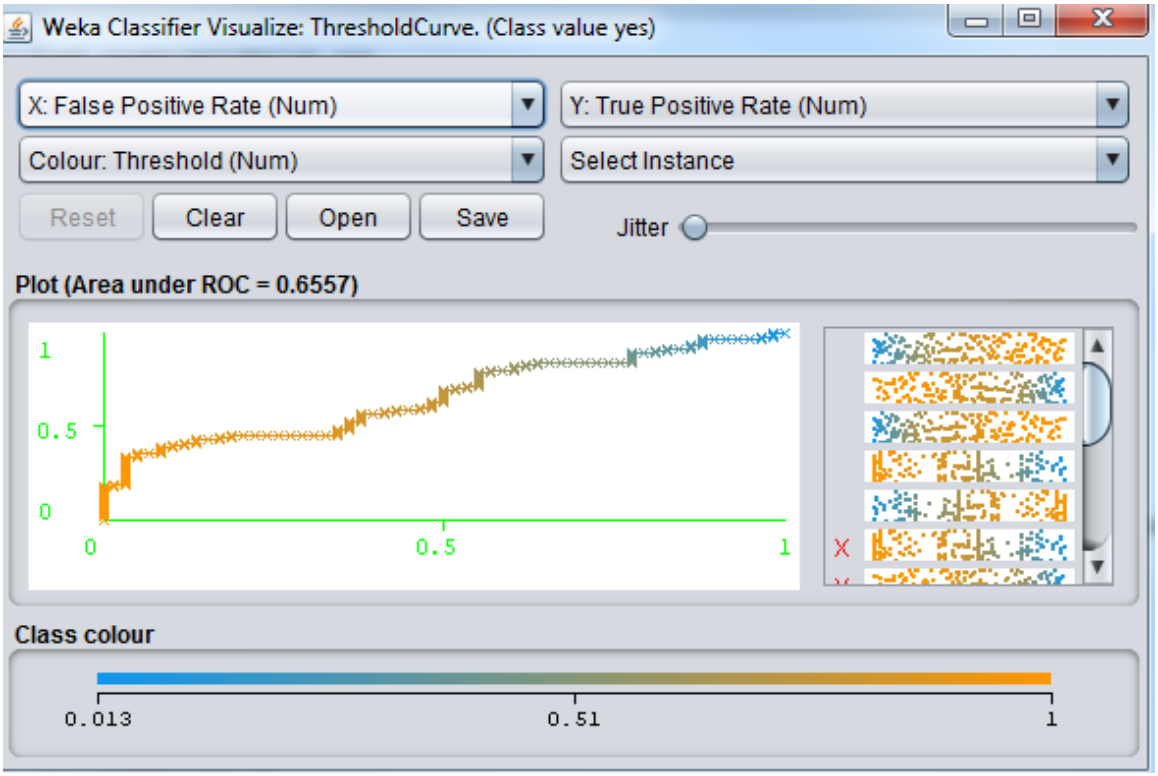

Figure 4.5.8 : ROC Curve of DL-MLP Classifier for Launcher3 Package

4.5.5 ManagedProvisioning Package

Table 4.5.5 shows comparison between DL-MLP and ML classification with Bayes Net Technique. Figure 4.5.9 displays the ROC curves of Bayes Net ML Technique & figure 4.5.10 shows the ROC curves of the MLP based on DL technique. Here, DL-MLP technique outperforms ML-Bayes Net technique.

Table 4.5.5: Comparison for ML and DL Techniques for ManagedProvisioning Package

Technique	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area
DL-MLP	0.914	0.505	0.905	0.914	0.903	**0.692**
ML-Bayes Net	0.828	0.886	0.767	0.828	0.796	0.367

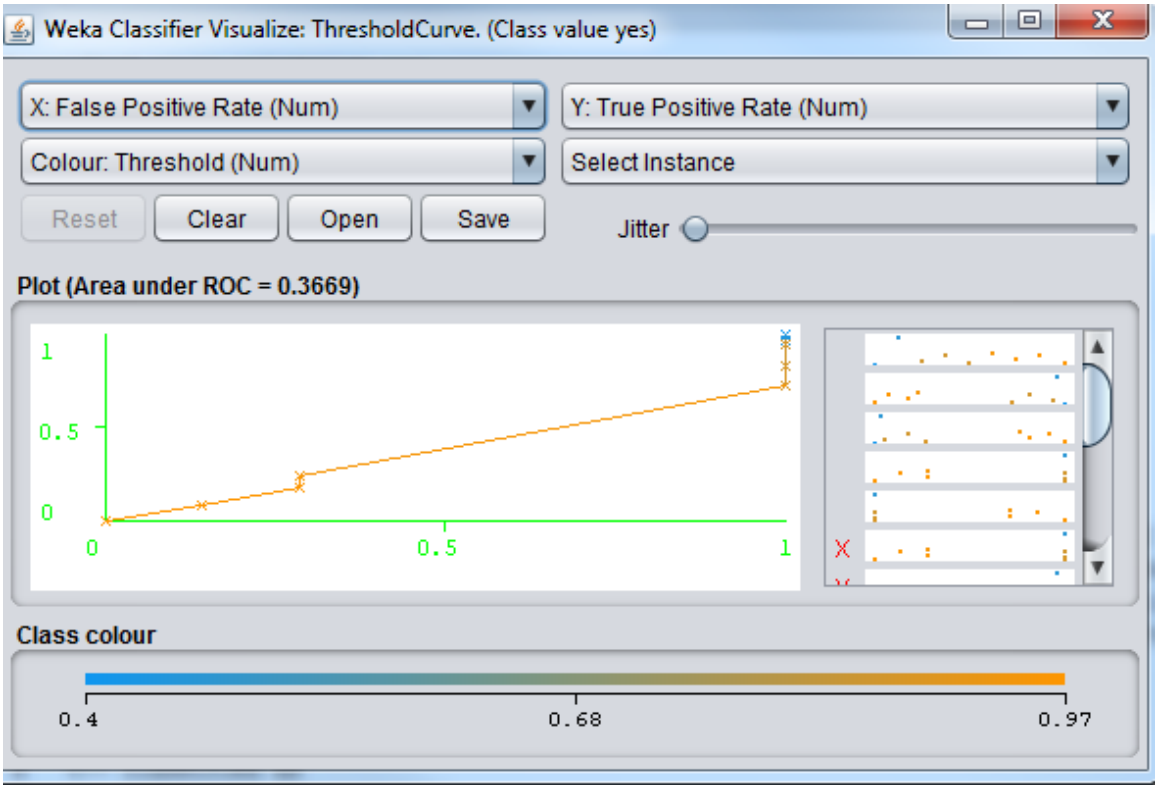

Figure 4.5.9: ROC Curve of ML-Bayes Net Classification for ManagedProvisioning Package

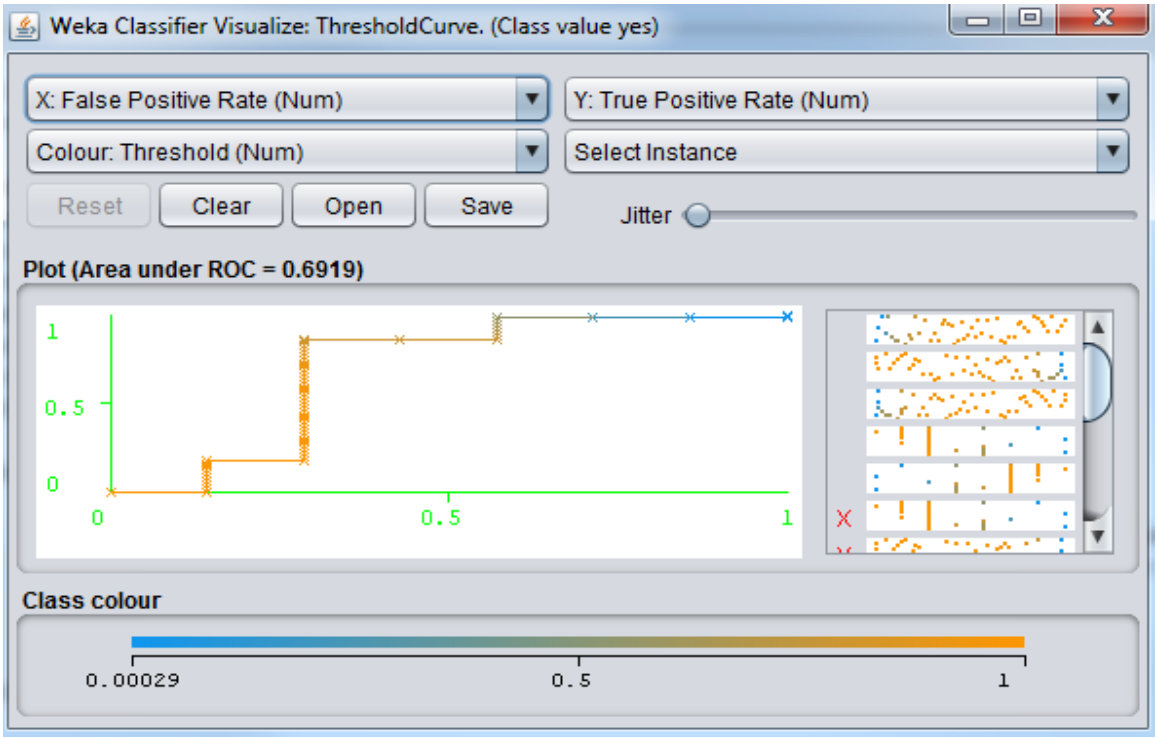

Figure 4.5.10: ROC Curve of DL-MLP Classifier for ManagedProvisioning Package

4.5.6 PackageInstaller Package

Table 4.5.6 shows comparison between DL-MLP and ML classification with Bayes Net technique. Figure 4.5.11 shows the ROC curves of Bayes Net ML Technique & figure 4.5.12 displays the ROC curves of the MLP based on DL technique. Here, DL-MLP technique outperforms ML-Bayes Net technique.

Table 4.5.6: Comparison for ML and DL Techniques for PackageInstaller Package

Technique	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area
DL-MLP	0.750	0.272	0.752	0.750	0.751	**0.798**
ML-Bayes Net	0.662	0.356	0.670	0.662	0.665	0.681

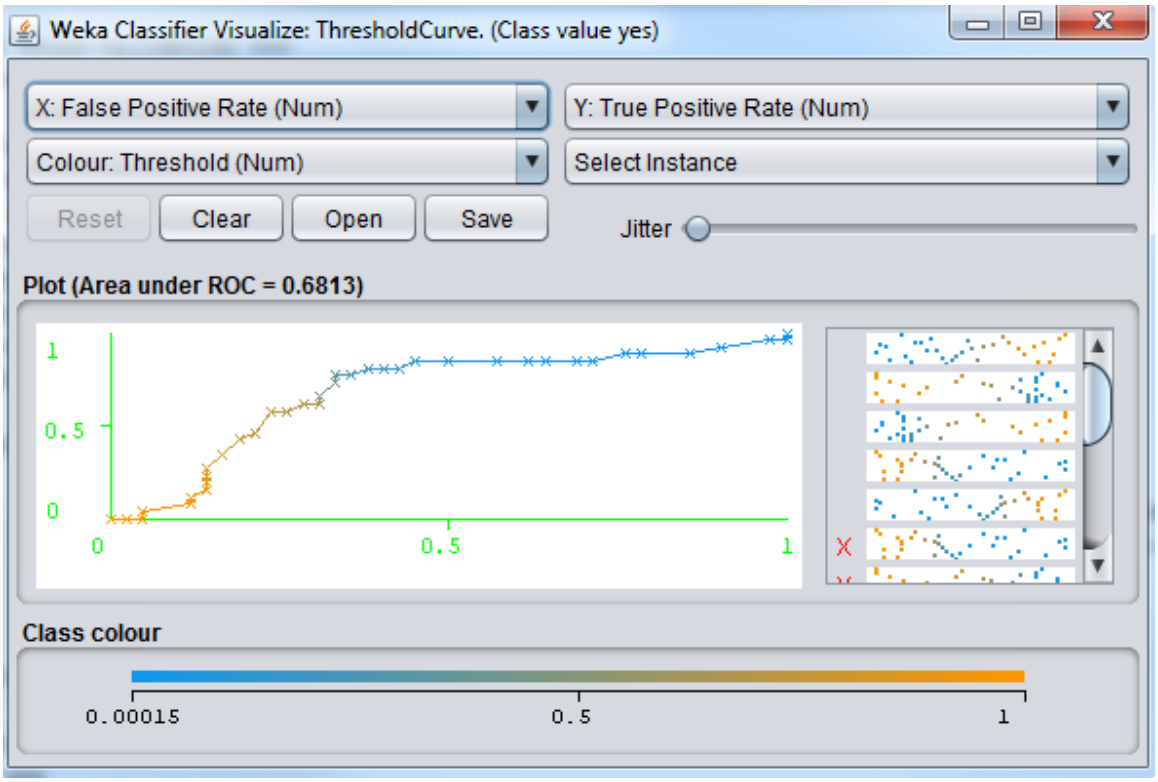

Figure 4.5.11: ROC Curve of ML-Bayes Net Classification for PackageInstaller Package

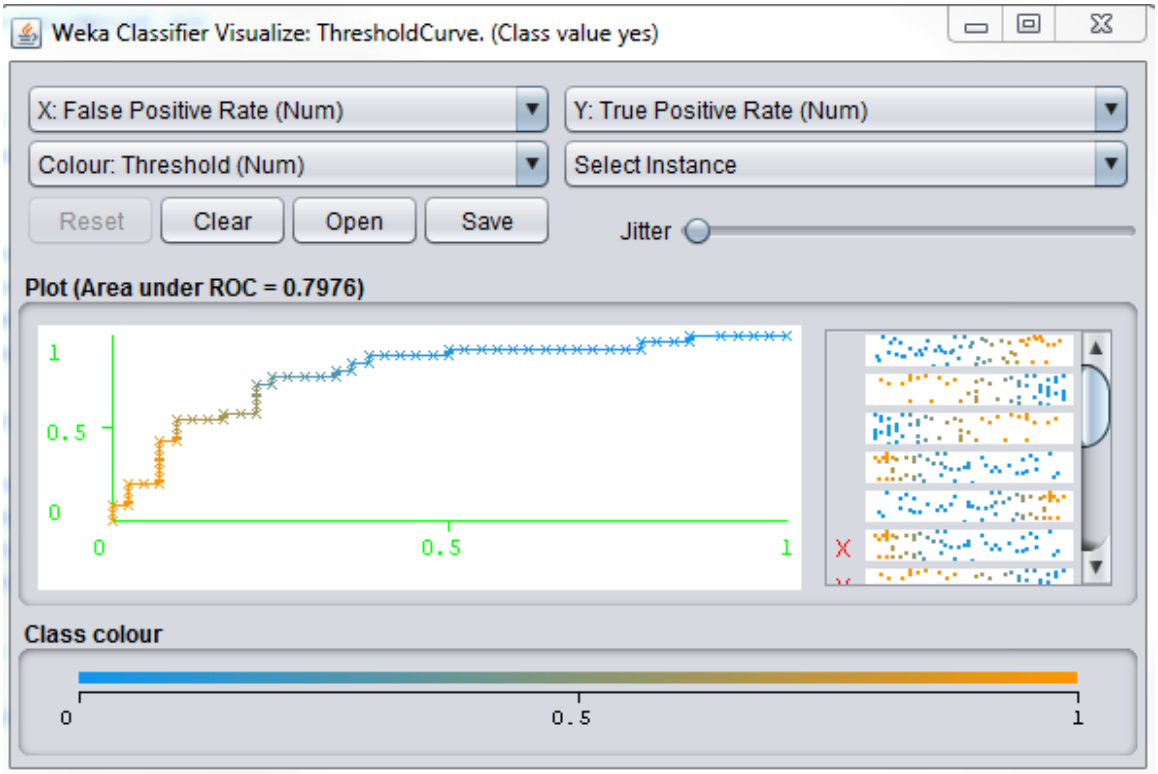

Figure 4.5.12: ROC Curve of DL-MLP Classifier for PackageInstaller Package

4.5.7 Settings Package

Table 4.5.7 shows comparison between DL-MLP and ML classification with Bayes Net technique. Figure 4.5.13 illustrates the ROC curves of Bayes Net ML technique & figure 4.5.14 illustrates the ROC curves of the MLP based on DL technique. Here, ML based Bayes net technique outperforms DL-MLP technique.

Table 4.5.7: Comparison for ML and DL Techniques for Settings Package

Technique	TP Rate	FP Rate	Precision	Recall	F-Measure	ROC Area
DL-MLP	0.653	0.484	0.637	0.653	0.642	0.676
ML-Bayes Net	0.650	0.335	0.693	0.650	0.660	**0.690**

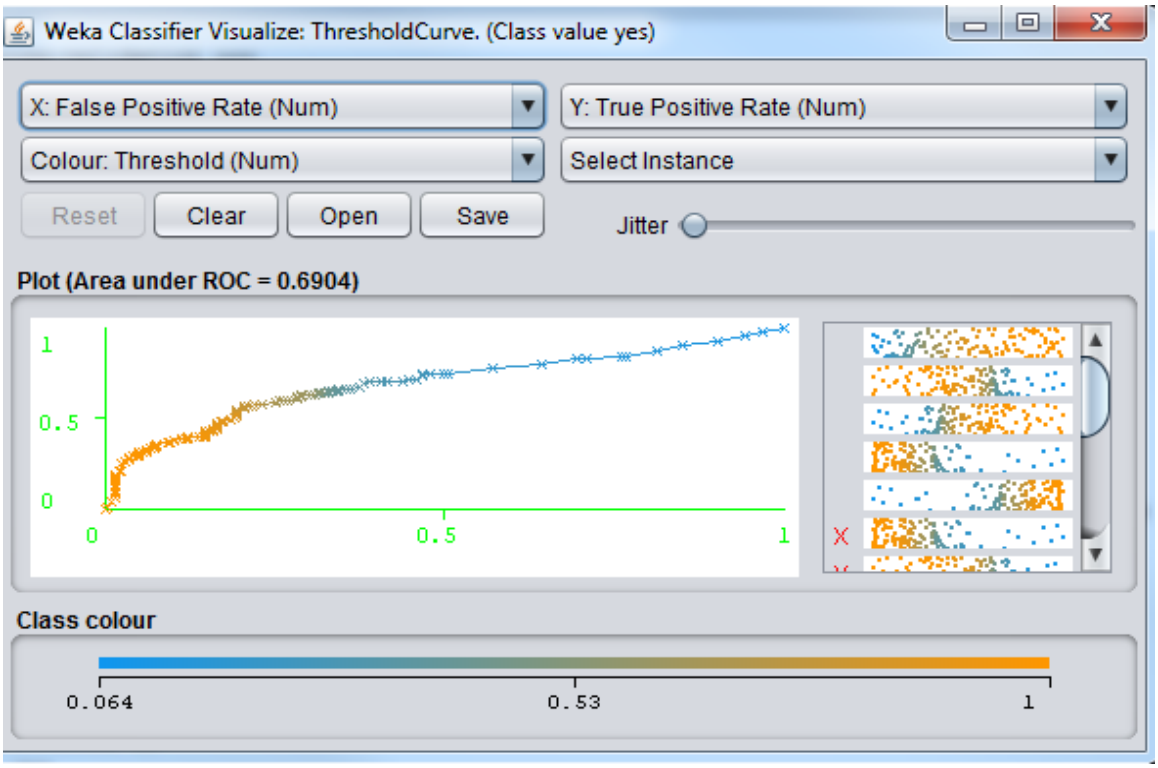

Figure 4.5.13: ROC Curve of ML-Bayes Net Classification for Settings Package

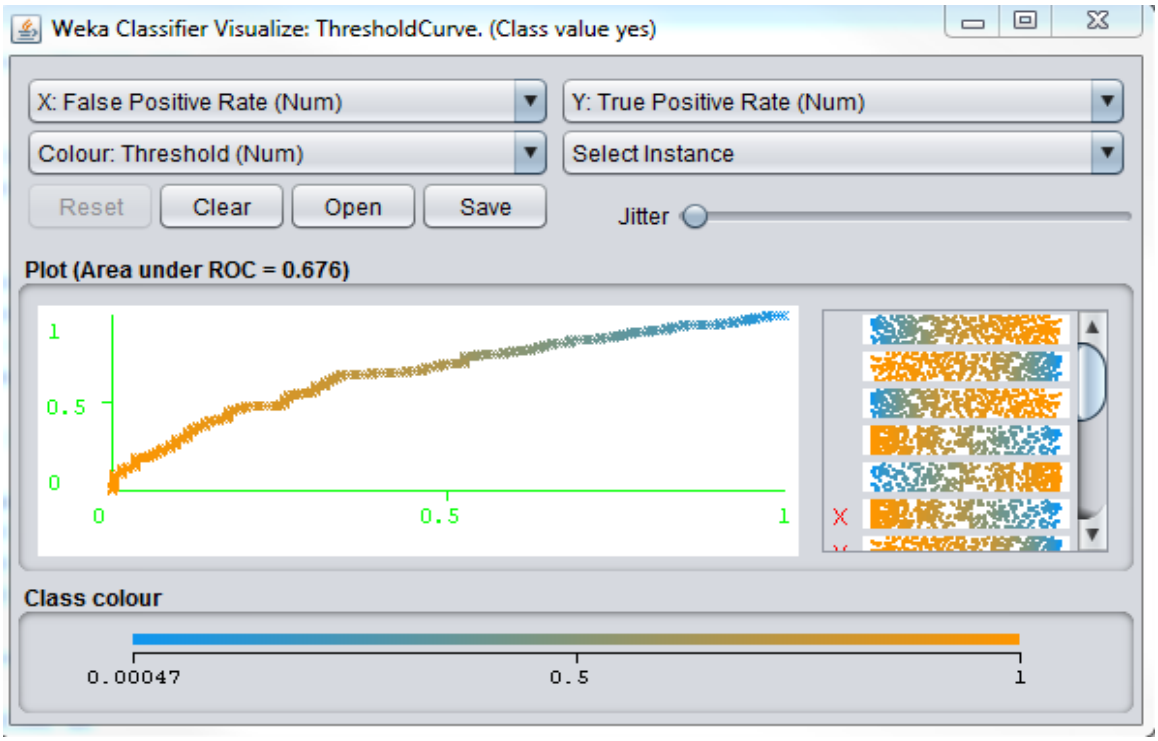

Figure 4.5.14: ROC Curve of DL-MLP Classifier for Settings Package

Chapter 5: Conclusion & Future Work

In Our work we have found relationship between CKJM Metrics suite & change proneness of any class. From our experiment, we found that for some projects ROC values of DL-MLP technique is better and for some projects, ML-Bayes Net technique outperforms. But, overall both the techniques are comparable on above project. On the basis of above experiment, we created package wise performance and it is visible in Figure 5.1.

Since, due to system limitation, we selected the moderate data size for our project and under such small data, performance of DL-MLP classifier is very promising and motivating, as DL-MLP classifier gives competition to ML based technique which works well on moderate size data. Hence, we can conclude our work on DL-MLP based model for change prediction developed can be used for forecasting change prone classes in subsequent releases of Android OS Data sets (like Android Nougat MR1 to Oreo Release).

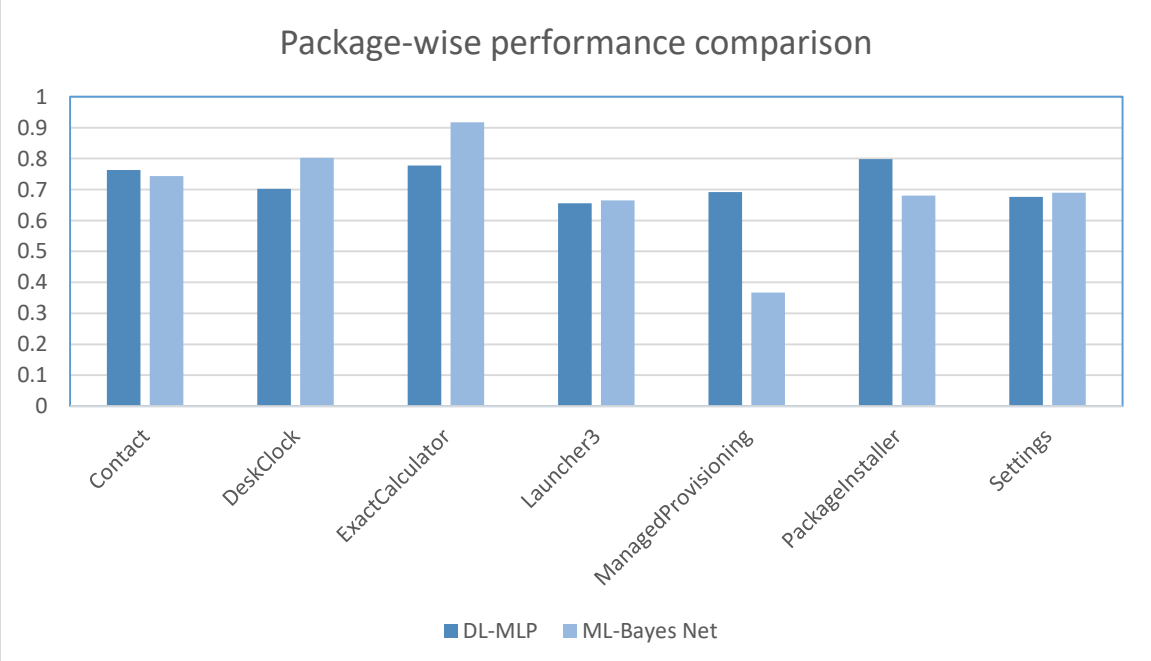

Figure 4.5.1: Above comparison chart shows 10-fold cross validation results for 7 Android Packages w.r.t. DL-MLP classifier and ML-Bayes Net technique

In future, we can compare the performance of DL-MLP based model to different ML based technique. Also, we can apply developed models to different projects that are similar in nature. We will check performance of above developed models on cross projects. We have planned to enhance scope of our work to large data sets & more DL techniques. Our future scope of work includes comparison of various ML technique to DL technique and conclude that which one is most efficient to use industrially and provide solution of change prediction that can be helpful to corporate world that can be helpful in cost reduction of many of the product release.

Bibliography

[1] Y. LeCun, Y. Bengio and G. Hinton, "Deep learning," *Nature,* vol. 521, no. 436, p. 436–444, 2015.

[2] B. Curtis, S. B. Sheppard and P. Milliman, "Measuring the Psychological Complexity of Software Maintenance Tasks with the Halstead and McCabe Metrics," in *IEEE Transactions on Software Engineering*, 1979.

[3] N. FriedmanDan, G. and G. , "Bayesian Network Classifiers," *Machine Learning,* p. 131–163, November 1997.

[4] T. Gyimothy, R. Ferenc and I. Siket, "Empirical validation of object-oriented metrics on open source software for fault prediction," in *IEEE Transactions on Software Engineering*, 2005.

[5] University of Waikato, "WekaDeeplearning4J: Deep Learning using Weka," 31 January 2018. [Online]. Available: https://deeplearning.cms.waikato.ac.nz/. [Accessed 21 January 2018].

[6] TomFawcett, "An introduction to ROC analysis," pp. 861-874, Pattern Recognition Letters.

[7] R. Malhotra and M. Khanna, "Investigation of relationship between object-oriented metrics and change proneness," *International Journal of Machine Learning and Cybernetics*, pp. Volume 4, Issue 4, pp 273–286, 2013.

[8] R. Malhotra and M. Khanna, "Mining the impact of object oriented metrics for change prediction using Machine Learning and Search-based techniques," in

Advances in Computing, Communications and Informatics (ICACCI), 2015 International Conference on, Kochi, India, 2015.

[9] Y. Singh, A. Kaur and R. Malhotra, "Software Fault Proneness Prediction Using Support Vector Machines," in *World Congress on Engineering*, London, U.K, 2009.

[10] R. Malhotra and R. Raje, "An Empirical Comparison of Machine Learning Techniques for Software Defect Prediction," *Proceedings of the 8th International Conference on Bioinspired Information and Communications Technologies*, pp. 320-327, 01 December 2014.

[11] J. Li, P. He and J. Zhu, "Software Defect Prediction via Convolutional Neural Network," in *IEEE International Conference*, Prague, 2017.

[12] A. Kaur and I. Kaur, "An empirical evaluation of classification algorithms for fault prediction in open source projects," *Journal of King Saud University - Computer and Information Sciences*, vol. 30, no. 1, pp. 2-17, 2018.

[13] Google, "Android Open Source Project," 31 December 2017. [Online]. Available: https://source.android.com/setup/initializing.

[14] R. Malhotra, N. Pritam and K. Nagpal, "Defect Collection and Reporting System for Git based Open Source Software," in *2014 International Conference on Data Mining and Intelligent Computing*, New Delhi, 2014.

[15] R. Malhotra, K. Nagpal and P. Upmanyu, Defect Collection and Reporting System for Git based Open Source Software, New Delhi, India: IEEE, 2014.

[16] C. . D. Manning, P. Raghavan and H. Schutze, Introduction to Information Retrieval, london: Cambridge University Press, 2008.

[17] University of Waikato, "Weka 3: Data Mining Software in Java," 22 December 2017. [Online]. Available: https://www.cs.waikato.ac.nz/ml/weka/.

[18] A. Gibson, C. Nicholson and J. Patterson, "Deep Learning for Java," 13 August 2017. [Online]. Available: https://deeplearning4j.org.

[19] J. Nam, "Survey on software defect prediction," in *Department of Compter Science and Engineerning, The Hong Kong University of Science and Technology*, Hong Kong, 2014.

[20] T. Menzies, J. Greenwald and A. Frank, "Data mining static code attributes to learn defect predictors," *IEEE Transactions on Software Engineering,* p. 2–13, 11 December 2006.