**A DISSERTATION**

**ON**

# SOFTWARE DEFECT PREDICTION USING DEEP NEURAL NETWORKS

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

**MASTER OF TECHNOLOGY**
*In*
**SOFTWARE TECHNOLOGY**


*Submitted by*
**Rohin Kumar**
**University Roll No. 2K15/SWT/513**

*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 "**SOFTWARE DEFECT PREDICTION USING DEEP NEURAL NETWORKS"** 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:

Rohin Kumar
2K15/SWT/513

DELHI TECHNOLOGICAL UNIVERSITY

DELHI-110042

# CERTIFICATE

This is to certify that thesis entitled "**SOFTWARE DEFECT PREDICTION USING DEEP NEURAL NETWORKS",** is a bona fide work done by Mr. Rohin Kumar (Roll No: 2K15/SWT/513) 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 "**SOFTWARE DEFECT PREDICTION USING DEEP NEURAL NETWORKS**" 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 MTech 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.

Rohin Kumar
2K15/SWT/513

# ABSTRACT

Software life cycle is a long series of steps performed to guarantee a reliable, correct and a robust software. Teams of developers and quality assurance specialist are working towards a common goal of providing quality assured software. Traditionally over a period of time quality assurance has drastically improved, this involved not only focusing on what is being developed but also on how it is being developed. With ever increasing demands to deliver a quality-based customer centric product it is essential to devise new ways to achieve greater quality in less time or we say on time.

Today's software systems are made up of large subsystems interlinked together to achieve a common goal. Once devised developers need to spend a huge amount of time in only finding the location of a defect. A defect if goes unnoticed can and will cause organizations to spend not only considerable amount of time and money to drill down to the root cause causing huge delays. Even if the documentations, functions or code snippets are reviewed from early stages there is always a chance that a defect goes unnoticed in development phase. Thus, this is even more important for quality assurance teams to predict the nature of change of software components in time.

With keeping this is mind and to improve software reliability, various defect prediction techniques has been utilized over time by developers and quality assurance teams to assist in finding defects and properly channeling the testing efforts.

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 Android data sets that are generated on Android applications like Bluetooth, Contacts, Gallery, Messaging, Music and Settings. Analyzing code and binary together we will build Deep Learning (DL) based models and fine tune parameters to better understand effects of DL techniques over Defect prediction. 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 [1] on Android Oreo (8.1) and Android Pie (9.0) releases.

Deep learning model was generated by first generating metrics on Android data sets and then building the DL models using Deeplearning4J library. Various models then compared together for performance.

# TABLE OF CONTENTS


Chapter 1 : Introduction ..... 1
Chapter 2 : Literature Review ..... 4
Chapter 3 : Research Background ..... 5

3.1 Android Applications Metrics: ..... 5
3.1.1 Downloading Source Code using DCRS Tool: ..... 6
3.1.2 Examining GIT change logs: ..... 8
3.1.3 Android application compilation: ..... 9
3.1.4 Metrics generation using DCRS: ..... 11

3.2 Relationship of Dependent Variable ..... 16

3.3 Model Architecture ..... 17
3.3.1 Batch Normalization ..... 17
3.3.2 Activation ReLU ..... 18
3.3.3 Dense Layer ..... 18
3.3.4 Softmax Function ..... 18
Chapter 4 : Research Methodlogy ..... 19

4.1 Preprocessing of Data: ..... 19

4.2 Model Creation on DL Feed Forward technique ..... 20

4.3 Predicting Changes between 2 Android Release ..... 21

4.4 Performance Assessment ..... 23
4.4.1 Metrics Assessment ..... 23

4.5 Model Evaluation Results: ..... 25
4.5.1 Contact Package ..... 25
4.5.2 Bluetooth Package ..... 27
4.5.3 Messaging Package ..... 29
4.5.4 Music Package ..... 30
4.5.5 Settings Package ..... 31
4.5.6 Bluetooth Package with TanH Activation Function ..... 33
4.5.7 Baseline Comparison with Deeper ..... 35
Chapter 5: Conclusion & Future Work ..... 36
Bibliography ..... 37

## LIST OF TABLES

Table 3-1: Dataset for different android packages for android tag: 8.1.0_r1 & 9.0.0_r1 ..... 6

Table 3-2: List of commands for building Android packages ..... 10

Table 3-3: CKJM OO Metrics generated from DCRS Tool ..... 13

**Table 4-1: Table Dataset Description** ..... 22

Table 4-2: ROC Values ..... 25

Table 4-3: Performance of DLCS classifier for Contact Package ..... 26

Table 4-4 : Performance of DLCS classifier for Bluetooth Package ..... 27

Table 4-5: Performance of DLCS classifier for Messaging Package ..... 29

Table 4-6: Performance of DLCS Classifier for Music Package ..... 30

Table 4-7: Performance of DLCS Classifier for Settings Package ..... 32

Table 4-8: Performance of TanH Activation for Bluetooth Package ..... 33

Table 4-9 Performance Precision of DLCS and Deeper ..... 35

## LIST OF FIGURES

Figure 3-1: DCRS Tool in download mode ..... 8

Figure 3-2: DCRS Tool - Change logs ..... 9

Figure 3-3: Class files generated for Contact Package ..... 10

Figure 3-4: DCRS Tool [8] for OO Metrics generation ..... 12

Figure 3-5: Independent and Dependent Variables ..... 16

Figure 3-6: DLCS Architecture Overview ..... 17

Figure 4-1: Deeplearning4J and Weka integrated library (University of Waikato, 2018) Specification ..... 20

Figure 4-2: WEKA - Preprocess ..... 21

Figure 4-3: Deep learning classification using DL4J-MLP classifier (University of Waikato, 2018) ..... 22

Figure 4-4: ROC Curve of DLCS Classifier for Contact Package Yes Label ..... 26

Figure 4-5: ROC Curve of DLCS Classifier for Contact Package No Label ..... 27

Figure 4-6: ROC Curve of DLCS Classifier for Bluetooth Package Yes Label ..... 28

Figure 4-7 : ROC Curve of DLCS Classifier for Bluetooth Package No Label ..... 28

Figure 4-8: ROC Curve of DLCS Classifier for Messaging Package ..... 29

Figure 4-9: ROC Curve of DLCS Classifier for Messaging Package ..... 30

Figure 4-10: ROC Curve of DLCS Classifier for Music Package ..... 31

Figure 4-11 : ROC Curve of DLCS Classifier for Music Package ..... 31

Figure 4-12: ROC Curve of DLCS Classifier for Settings Package ..... 32

Figure 4-13: ROC Curve of DLCS Classifier for Settings Package ..... 33

Figure 4-14: ROC Curve DLCS with TanH Activation for Bluetooth ..... 34

Figure 4-15: ROC Curve DLCS with TanH Activation for Bluetooth ..... 34

## ACRONYMS

AUC: Area under Curve .....21

C&K: Chidamber & Kemerer .....4

CKJM: Chidamber & Kemerer Java Metrics.....4

DCRS: Defect Collection and Reporting System .....4

DL: Deep Learning .....3, 16

GUI: Graphical User Interface.....16

LOC: Lines of Code.....2

OO : Object Oriented.....2

OS: Oeprating System.....4

ROC: Receiver operating Characteristics .....2

RQ: Research Questions .....19

WEKA: Waikato Environment for Knowledge Analysis.....17

WMC: Weighted Methods Per Class.....12

ReLU: Rectified Linear Unit……18

# Chapter 1 : Introduction

In the current world there is a continuous demand of software's performing extensively complex activities in a quite simple manner. As an overview these simple looking tasks are complex to design and even more complex to maintain. Various surveys were often conducted to analyze the cost of a software and most the projects usually exceed the cost assumption while in designing or code phase. One of the major reasons to overshoot the budget a software is bug discovery during the later stages of Software Development Life Cycle commonly known as SDLC.

To reduce the cost one of the important activities is Software Quality. Software quality assurance activities play an important role in producing high quality software. A majority of research in defect prediction models are done using the traditional machine learning techniques.

Traditional machine learning techniques perform better when data is less. But in today's world data is increasing at an exponential rate, thus we need systems to analyze and train on data effectively. In recent years Deep learning techniques have been proven quite useful complex activities like Driverless cars, image prediction etc. In recent work, Kamei et al worked on Just-In-Time Quality Assurance using logistic regression algorithm to build a prediction model [2].

Another study motivated the use of Deep learning was performed by Xinli Yang David Lo, Xin Xia, Yun Zhang, and Jianling Sun in their paper on Deep Learning for Just in Time Defect prediction using Deep Belief Networks [3]. The Major problems faced in all the studies as identified by us is the fine tuning of parameters in Deep Neural networks. The number of layers involved in neural networks makes this task even more difficult. In this work we select and fine tune layers of deep neural networks and their effects on Software defect prediction.

We know that most of the software defects found in two side by side or recent releases of any software system can be found in the delta part of two releases or comparing their release notes. 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 [4] and calculate its effectiveness by comparing it to Deeper having DBN networks.

Model is developed using OO metrics [1] [5] 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 Oreo to Pie) 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 DLCS and Deeper technique [3] and checked the performance of this technique on subsequent releases of 5 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.

### 1.1 Thesis Organization

The whole thesis is organized in various chapters having their own objectives. Below is the brief introduction on thesis organization.

**Chapter 1** gives a brief introduction of the DLCS techniques used and how in today's world a traditional machine learning technique is not useful to overcome the challenges posed by complex data.

**Chapter 2** gives a background information and motivation behind this research. Various methods and techniques were studied to formulate a Deep Learning Cost Sensitive classifier. Studies like Malhotra and Khanna [7], Kamei et al. [2] and Xinli Yang, David Lo, Xin Xia and Yun Zhang [3] were one of the major motivators to use Deep Learning techniques in defect prediction.

**Chapter 3** gives an introduction of how data sets were collected for this research. This study is performed on 5 popular android applications like Settings, Bluetooth etc. A popular developed by DTU students was used to collate data from these applications i.e. DCRS [8]. This tool uses log-based techniques and extract changes from git-based systems using two versions for the applications. Source code downloaded is then compiled on Linux machine to generate class files for various metrics like CKJM metrics, Depth of Inheritance etc.

**Chapter 4** explains the use of Deep Learning model using Cost Sensitive classifier on the generated data set. Explains various activation functions and their usage and effect on defect prediction modelling. Preprocessing of data and selection of various hidden nodes along with best possible activation function for defect prediction technique is described in this chapter. This chapter also evaluates our model on five applications and compared with baseline to predict performance.

**Chapter 5** gives the final overview, conclusion and future research motivation to continue in Deep Learning for defect prediction. Various techniques still need to be investigated including but not limited to CNN's and LSTM. These techniques are widely popular in field like Image Processing and Audio Processing. Effects of these techniques still needs to be looked into for defect prediction.

# 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.

Xinli Yang, David Lo, Xin Xia and Yun Zhang [3] talked about the unexplored realm of deep learning in defect prediction. With their research to leverage the deep learning techniques were studied using deep belief networks. Several restricted Boltzmann's Machines were used in a two-layer network for feature detection.

Malhotra and Khanna [9] 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.

C. Manjula and Lilly Florence [10] helped us understand the use of software metrics in defect prediction techniques using deep neural networks. The combination of Genetic algorithm with DNN's proved a useful approach in defect prediction.

Jindal, Jain and Malhotra [11] studied model prediction using Radial Basis function of Neural Network. Activation functions as the name suggested uses a radial function. This helped us explore the use of other activation function and their impact on defect prediction. This laid important foundation in our study of deep neural networks using different hidden layer composing of multiple activation functions.

Kamei et al. [2] Also proposed a just in time prediction defect prediction using the logistic regression function to build a model. We also used the metrics which were proposed by Kamei et al. [2] cost effectiveness. This gives us a certain degree of performance to predict defects in certain percentage of code which is being inspected.

For deep learning to work efficiently we need metrics which proved useful in past and shown results in defect prediction. One of the studies conducted by Malhotra and Khanna [12] which uses the evolution-based metrics along with object-oriented metrics encourages us to use and study different metrics for our study of deep neural networks.

# Chapter 3 : Research Background

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

### 3.1 Android Applications Metrics:

For the Deep Learning model to work, input has to be fed in the feed forward network. This input will be generated from metrics. Object Oriented Metrics were generated from open source android application packages. These packages include popular applications like Bluetooth, Messaging, Music, Contacts and Settings. The above packages were studied in two different version namely Android Oreo and Android Pie.

Android is available as an open source in the Google GIT repository [13]. This GIT repository is hosted by Google. Android source code comprises of java and C/C++ files. Since we are majorly focused on application layer, we will take into account only the Java files. To feed data into our DLCS model, metrics need to be generated from Java as well as Class files from compiled Java files. Class files are generated from compiling java files for every application.

Java files were compiled by partially building the Android Source code. Generating class files only accomplishes ten percent of our work, since metrics are yet to be generated. To generate metrics, we needed a system efficient enough to study and generate defect logs from GIT based systems.

Defect Collection and Reporting System (DCRS) tool [8] is used to generate the reports having OO metrics. DCRS tool has an inbuilt CKJM tool which calculates C&K OO metrics [1] 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.

Characteristics of different android application package with respect to Android 8.1 and 9.0 releases are mentioned in Table 3-1.

**Table 3-1: Dataset for different android packages for android tag: 8.1.0_r1 & 9.0.0_r1**

| Packages | Total Classes | Classes having Changes | Change Percentage % |
|---|---|---|---|
| Contacts | 172 | 121 | 70 |
| Bluetooth | 112 | 83 | 73 |
| Music | 11 | 9 | 82 |
| Messaging | 240 | 182 | 76 |
| Settings | 580 | 386 | 67 |

GIT is open source versioning regulator system used for source code organization 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 contains android app packages data sets with total class, total number of classes having changes for Android 8.1 and 9.0 release for five Application Packages. Change logs for above packages were generated using DCRS Tools developed by the Delhi Technical University (DTU) students.

### *3.1.1 Downloading Source Code using DCRS Tool:*

DCRS Tool [8] is a JAVA based GUI tool which allows user to download, collect and reports various changes, defects, bugs or issues which were present in a given version of android Operating System (OS) in comparison to previous versions of android OS. Metrics will then be generated by using change logs, the generated metrics is to fed to the deep learning model.

As per previous researches use of open source systems have been widely popular among researchers due to the fact that they are easily available and most importantly it can easily be validated by other. The use of Android open source and other systems and applications are more popular. Defect prediction is all about traversing areas of code to predict weather the system is faulty or not. This allows

software quality analyst to focus testing efforts on important systems rather than concentrating on whole project.

DCRS tool help us find changes in source files by studying change logs from two versions. It also 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.

This will obtain the logs of android source files & then search 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.

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. This tool also links changes to the relating source files.

In order to use this tool, we require to configure GIT to extract the source and change logs from the two Android versions. To download sources for different packages, tags of each application packages can be found in Google android site. 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 sit for corresponding TAGs i.e. android-8.1.1_r1 and android-9.0.0_r1. 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.

Figure 3-1 shown below is the tool UI to download the source code of android application.


**Figure 3-1: DCRS Tool in download mode**

Git change logs contains all the information regarding every changes done on the file versions. Examining those logs can give us important information for the modifications done on the files. DCRS reads the change logs files from Git repository and generate change logs. The bug-ids and description can be retrieved. Based on the information processed following is generated:

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 Examining GIT change logs:*

We can obtain change logs using DCRS tool which processes the Git repository and obtains change logs of two predetermined consecutive releases. 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 5 android application projects between their 2 consecutive releases (Android-8.1.1_r1 & Android-9.0.0_r1). Figure 3-2 is DCRS tool UI displaying the GIT change logs.

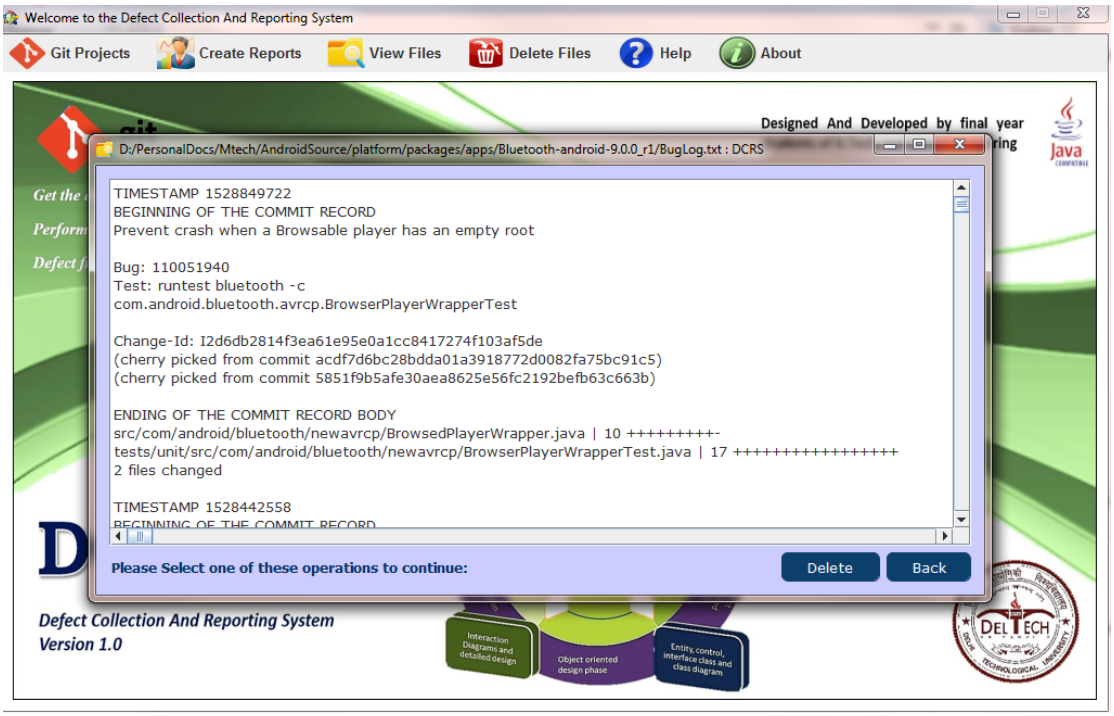


**Figure 3-2: DCRS Tool - Change logs**

### *3.1.3 Android application compilation:*

We downloaded the complete android source code separately for Tag android-8.1.0_r1 and android-9.0.0_r1 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 3-2: 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 |

Figure 3-3 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.

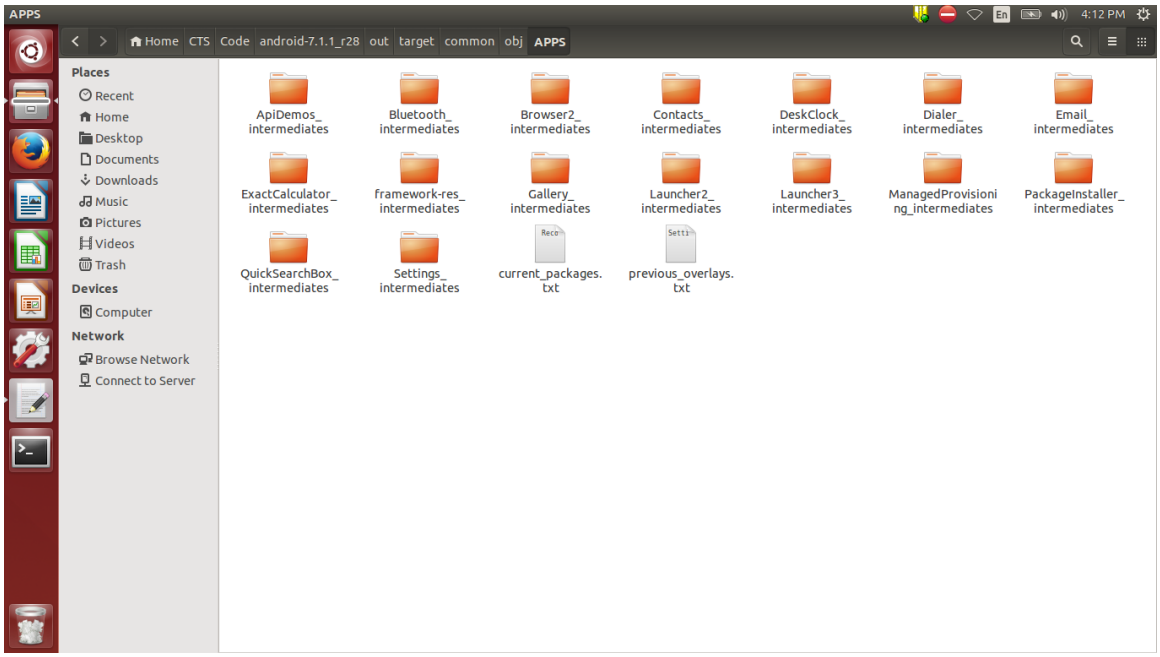


**Figure 3-3: Class files generated for Contact Package**

### *3.1.4 Metrics generation using DCRS:*

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 [1] [14] [15].

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 of Android 8.1 & 9.0 release. Figure 3-4 illustrates the OO metrics generation process.

In the Figure 3-4 different metrics can be selected for the corresponding packages. Metrics such as LOC, WMC, DIT, NOC, LCOM etc. can be generated. We have selected the consolidated defect and change report containing all the metrics. The metrics generated will be saved in the form of CSV files which can later be included in the deep learning model.

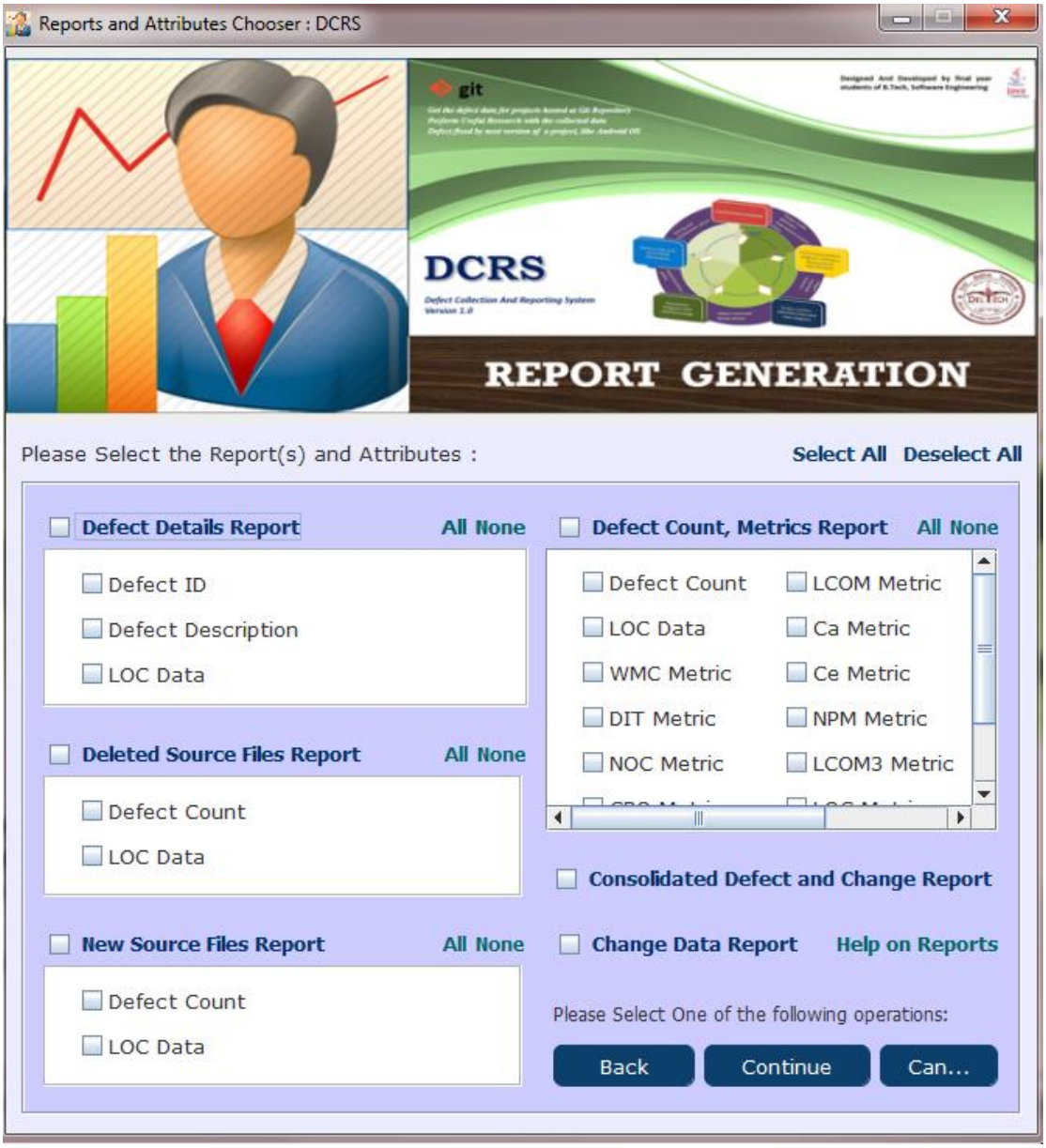


**Figure 3-4: DCRS Tool [8] for OO Metrics generation**

OO metrics generated using DCRS Tool is displayed below in Table 3-3:

**Table 3-3: 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 [1] 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 methods 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 Relationship of Dependent Variable

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. In Figure 3-5, change is the dependent variable which dependent on independent variables i.e. WMC & NOC, CBO, RFC, LCOM & Ca, NPM and DIT.

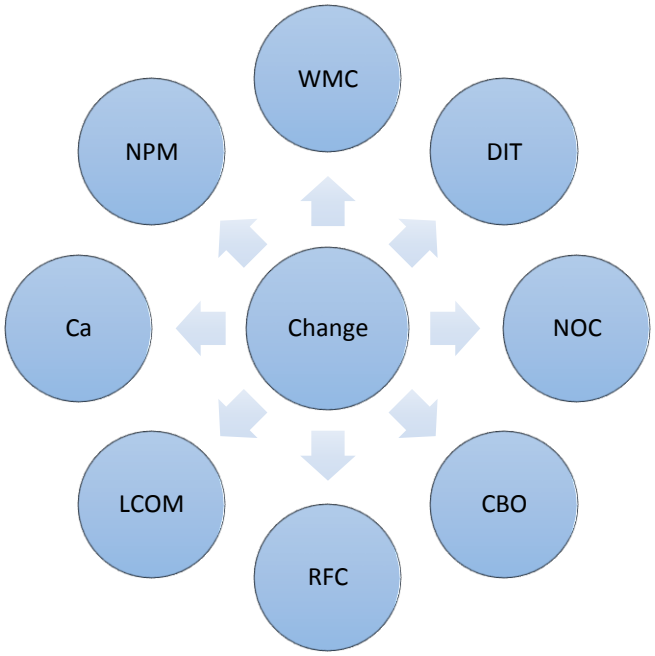


**Figure 3-5: Independent and Dependent Variables**

### 3.3 Model Architecture

In this section we will give a formal introduction of our feed forward DLCS technique. Our technique uses a basic 4 layer architecture. First being the input layer followed by Batch Normalization, Activation ReLU, Dense Layer and output layer using the softmax function.

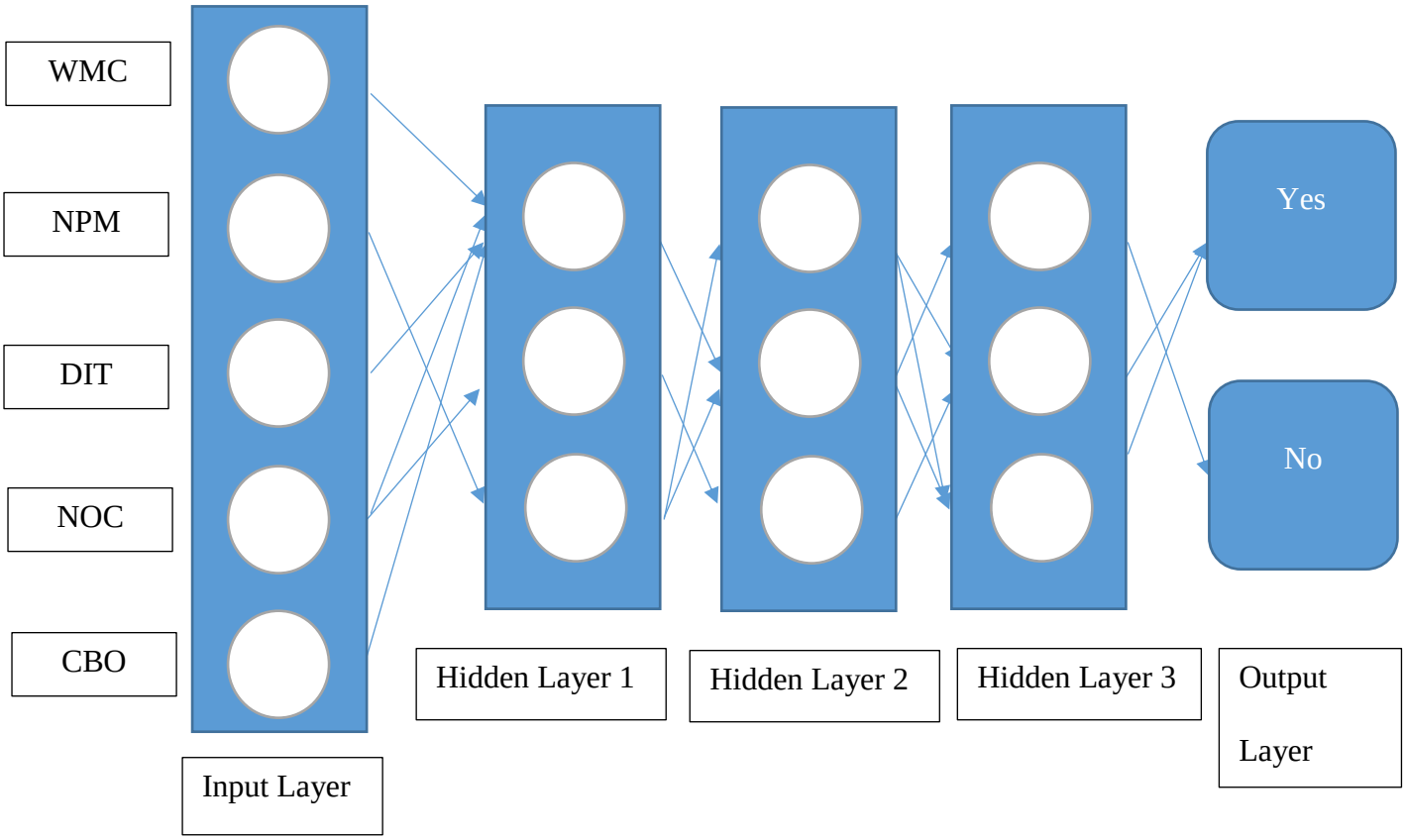


**Figure 3-6: DLCS Architecture Overview**

#### *3.3.1 Batch Normalization*

Input layer consist of input range of any values. These values needs be normalized and adjusted to scale to activations. Consider an example, suppose an input sets contains only defective entries corresponding to No.

When the model will be trained, No value can easily be predicted but for a model to be efficient both No and Yes should be predicted correctly despite of the entries in the model. To scale this and normalize the input values Batch Normalization process is applied on the input data.

### *3.3.2 Activation ReLU*

Deep learning as in feed forward type networks contains multiple of activation functions. Activation functions are what gives the network a non linear property.

Can we move without Activation Functions? The answer to this question is yes but without activation functions the network would just be linear which is nothing else but polynomial of a degree 1. This polynomial is good for solving non complex problems. In order to map more complex data we require our network to move into non linearity. The most popular function is ReLU which is Rectified Linear Unit. ReLU is defined as below

$R(x) = \max(0, x)$

If $x < 0$, $R(x) = 0$ and If $x > 0$ $R(x) = x$.

### *3.3.3 Dense Layer*

Dense layer is a fully connected layer in which every neuron is connected to every neuron in a linear manner. This is generally followed by non linear activation function. In this model we have used a dense layer with twenty outputs followed by non linear ReLU activation function.

### *3.3.4 Softmax Function*

ReLU functions have some disadvantages as well. ReLU functions are not suitable for output layers. During a ReLU functions some neurons can never activate causing them to die. Thus this is not suitable for output layer.

The output layer generally uses either Softmax function for a classification problem or a simple linear function for regression. In our model we need to predict out classes to be either defective or not. This enables us to use the Softmax function.

# 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 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 DLCS based model and Deeper [3] based model.
5. Model evaluation results.

### 4.1 Preprocessing of Data:

We have used twelve metrics as input data for change prediction. Uncorrelated and the best attributes are selected out of a set of OO metrics using correlation-based feature selection [16] technique. An attribute is selected if the correlation with the dependent variable is higher than the highest correlation amongst the attributes. This technique is simple, widely used and very fast method in for sub selecting attributes using the DL technique.

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.

The main purpose to preprocess data is remove redundancy and select appropriate input data features to map to main model in feed forward neural networks. So the feature selection technique takes care for both the irrelevant attributes.

### 4.2 Model Creation on DL Feed Forward technique

Our model is based on Feed Forward neural network methodology. In this technique multiple layers are connected in forward mechanism. This model consists of four layer mechanism having Batch Normalization, Activation ReLU and Dense layer being the major ones. The DL [4] models learn several data abstracted representations.

It finds detailed structure in large data sets by using the back-propagation algorithm to using the Stochastic Gradient Descent algorithm which is easier to train saving the train time. This is essential in the concept of Just in Time defect prediction as we do not want larger times in training.

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 [17] which is open source library. We implemented our work using deeplearning4j library and Weka tool [18]. DL [19] 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 4-1 illustrates about deeplearning4j library.

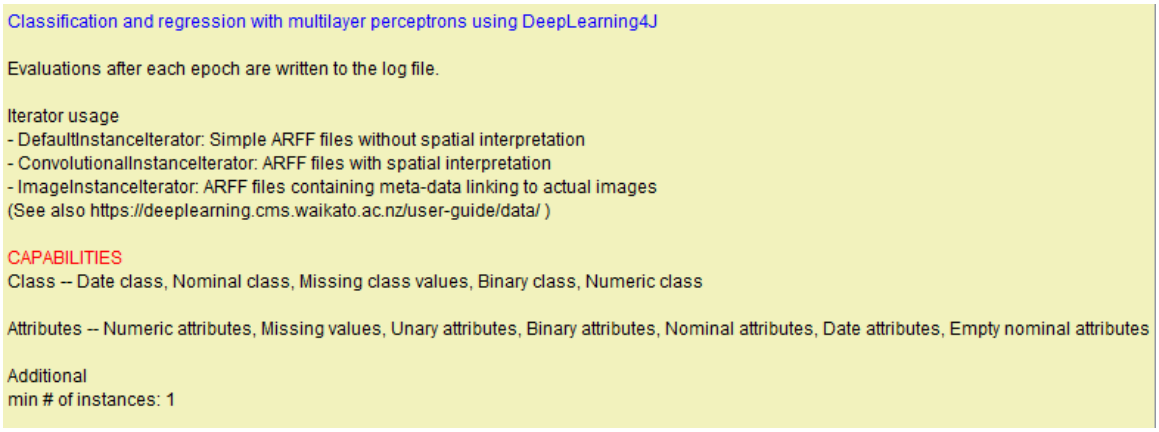


**Figure 4-1: Deeplearning4J and Weka integrated library (University of Waikato, 2018) Specification**

### 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 4-2, 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.

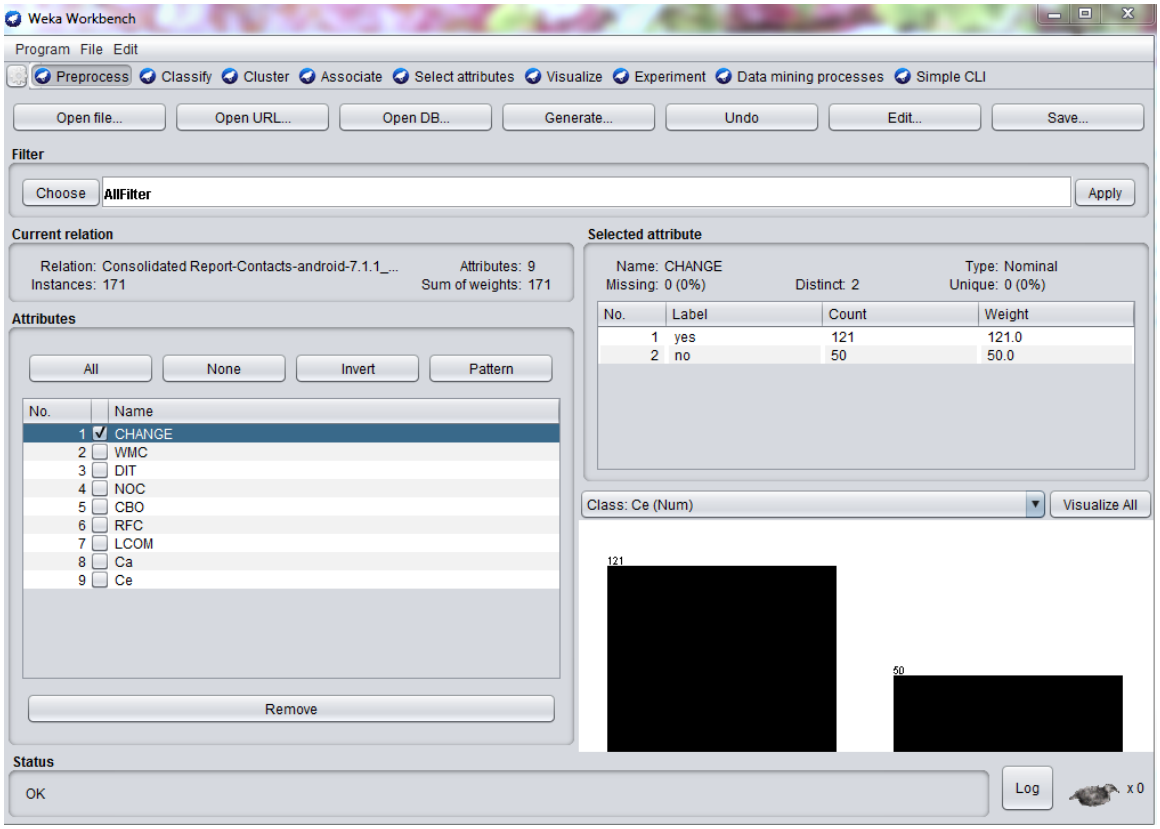


**Figure 4-2: WEKA - Preprocess**

In Figure 4-3, we have used WEKA for executing DLCS 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.

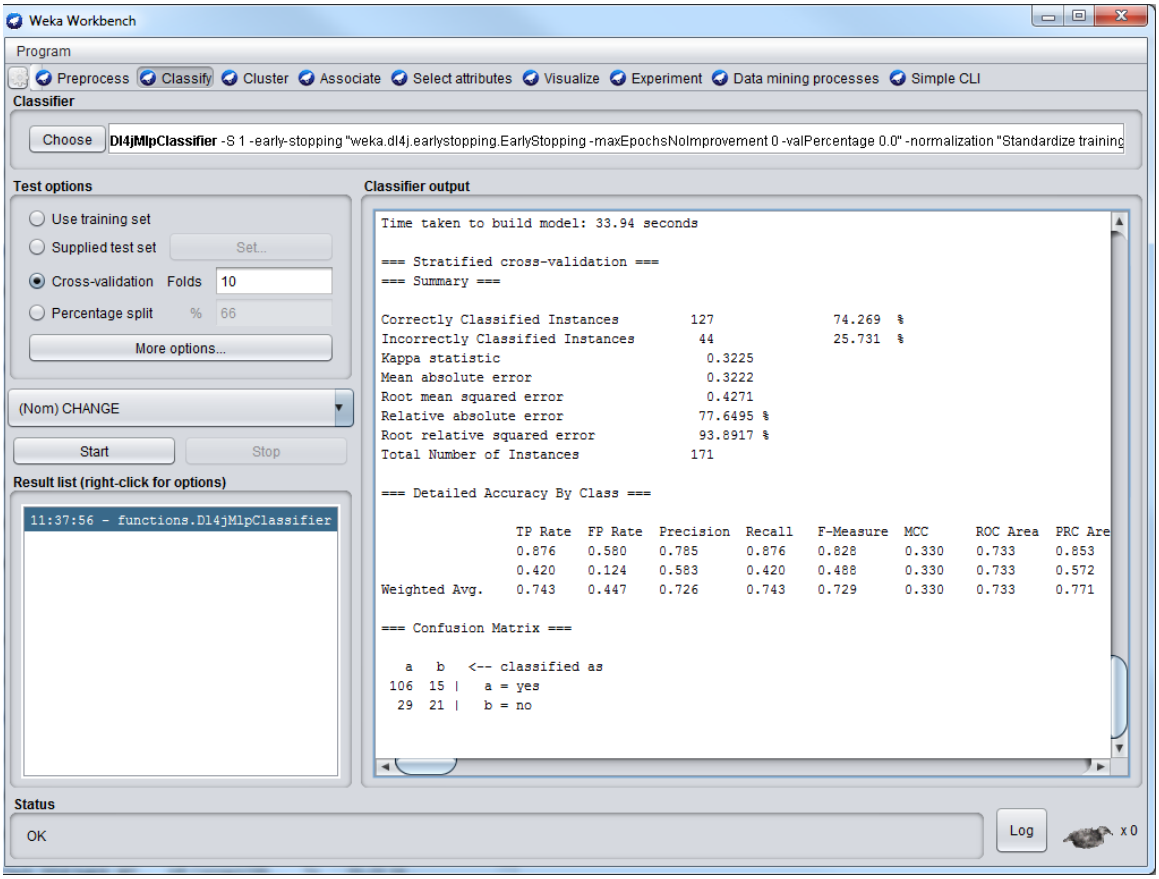


**Figure 4-3: Deep learning classification using DL4J-MLP classifier (University of Waikato, 2018)**

**Table 4-1: Table Dataset Description**

| Project Description | Versions | Total Files | Change Rates (%) |
|---|---|---|---|
| Contacts | android-8.1.1_r1, android-9.0.0_r1 | 172 | 70 |
| Bluetooth | android-8.1.1_r1, android-9.0.0_r1 | 112 | 73 |
| Messaging | android-8.1.1_r1, android-9.0.0_r1 | 240 | 76 |
| Music | android-8.1.1_r1, android-9.0.0_r1 | 11 | 82 |
| Settings | android-8.1.1_r1, android-9.0.0_r1 | 580 | 67 |

Table 4-1 states about dataset description of android applications that, we obtained from calculation from DCRS Tool developed by Malhotra and Nagpal [8].

### 4.4 Performance Assessment

In this section, we evaluate effectiveness of our DLCS 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: Evaluation of DLCS using ROC curve.

RQ2: Do the DLCS based methods outperform traditional Deeper [3] methods.

#### *4.4.1 Metrics Assessment*

To evaluate the prediction accurateness, we use a widely adopted following metrics [20], [21]:

F-measure (or F1 score), which is harmonic mean of recall & precision [14]. 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 number 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 4-2: 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 DLCS 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 4-3 to Table 4-8 shows results for different performance parameters TP rate & FP Rate, Precision & Recall, F Measure, & ROC Area with respect to DLCS Techniques.

### *4.5.1 Contact Package*

Table 4.5.1 shows comparison between DL and cost sensitive classifier with the below parameters and configurations. Figure 4.5.1 displays the ROC curves of the above configuration for DLCS classifier with 4 Layer configuration.

**Table 4-3: Performance of DLCS classifier for Contact Package**

| Method | TP Rate | FP Rate | Precision | Recall | F-Measure | MCC | ROC Area | PRC Area |
|---|---|---|---|---|---|---|---|---|
| DLCS – 4 Layer | 0.653 | 0.403 | 0.891 | 0.653 | 0.735 | 0.137 | 0.755 | 0.888 |

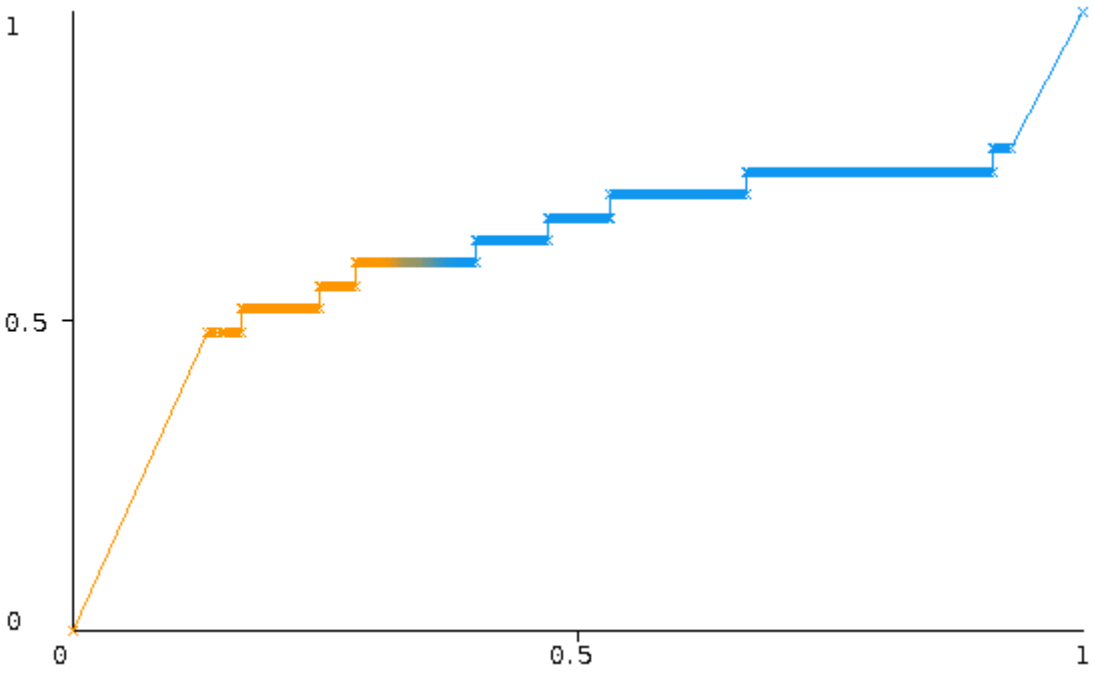


**Figure 4-4: ROC Curve of DLCS Classifier for Contact Package Yes Label**

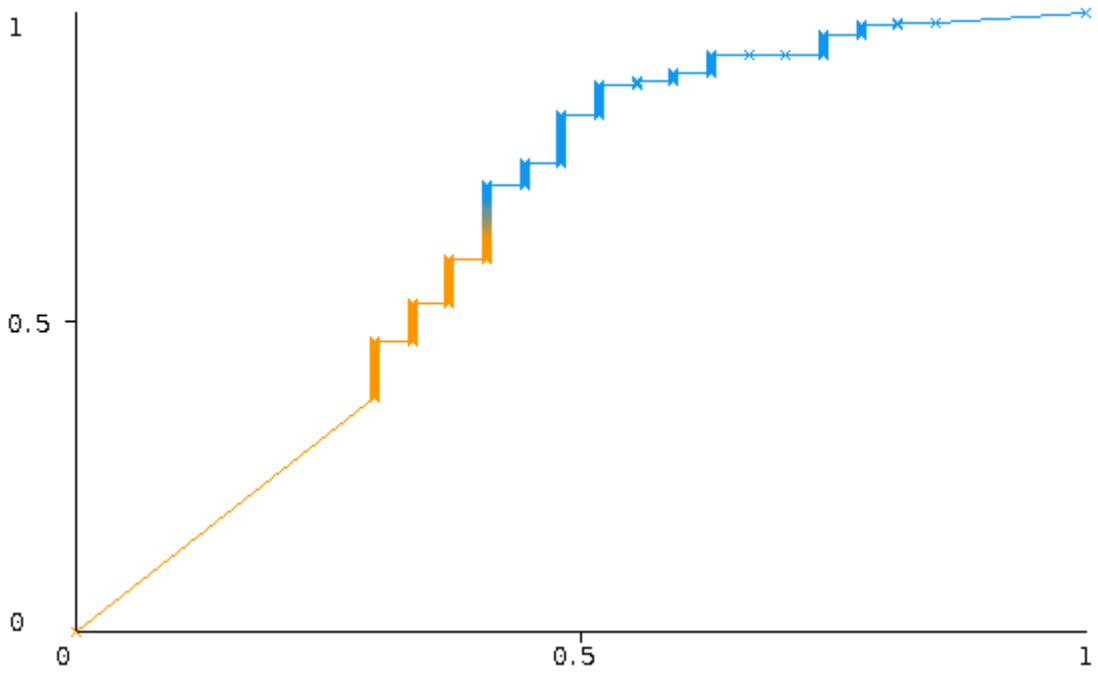


**Figure 4-5: ROC Curve of DLCS Classifier for Contact Package No Label**

### *4.5.2 Bluetooth Package*

Table 4.5.2 shows comparison between DL and cost sensitive classifier with the below parameters and configurations. Figure 4.5.3 displays the ROC curves of the above configuration for DLCS classifier with 4 Layer configuration.

.

**Table 4-4 : Performance of DLCS classifier for Bluetooth Package**

| Method | TP Rate | FP Rate | Precision | Recall | F-Measure | MCC | ROC Area | PRC Area |
|---|---|---|---|---|---|---|---|---|
| DLCS – 4 Layer | 0.839 | 0.810 | 0.892 | 0.839 | 0.865 | 0.025 | 0.642 | 0.896 |

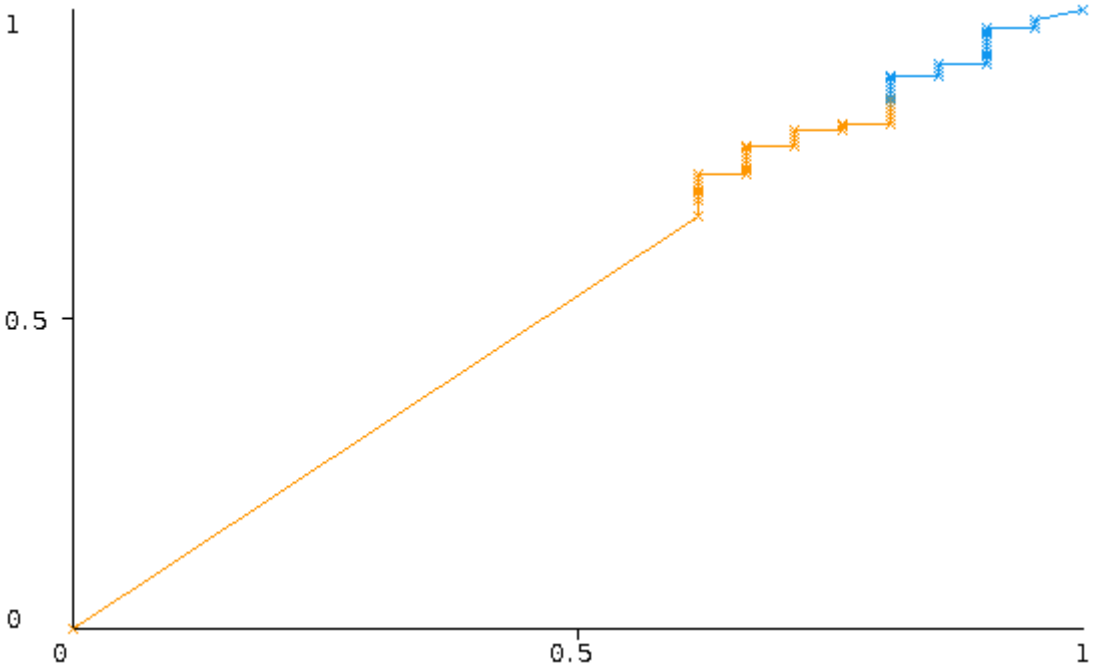


**Figure 4-6: ROC Curve of DLCS Classifier for Bluetooth Package Yes Label**

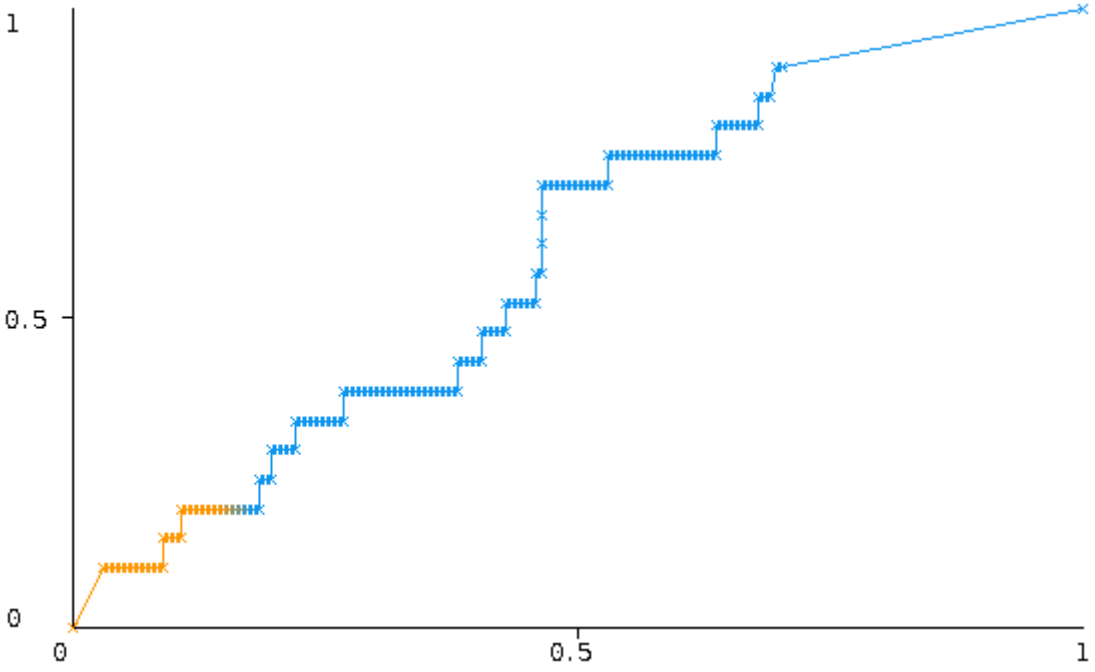


**Figure 4-7 : ROC Curve of DLCS Classifier for Bluetooth Package No Label**

### *4.5.3 Messaging Package*

Table 4-5 shows comparison between DL and cost sensitive classifier with the below parameters and configurations. Figure 4-8 displays the ROC curves of the above configuration for DLCS classifier with 4 Layer configuration.

**Table 4-5: Performance of DLCS classifier for Messaging Package**

| Method | TP Rate | FP Rate | Precision | Recall | F-Measure | MCC | ROC Area | PRC Area |
|---|---|---|---|---|---|---|---|---|
| DLCS – 4 Layer | 0.718 | 0.498 | 0.986 | 0.718 | 0.828 | 0.045 | 0.804 | 0.987 |

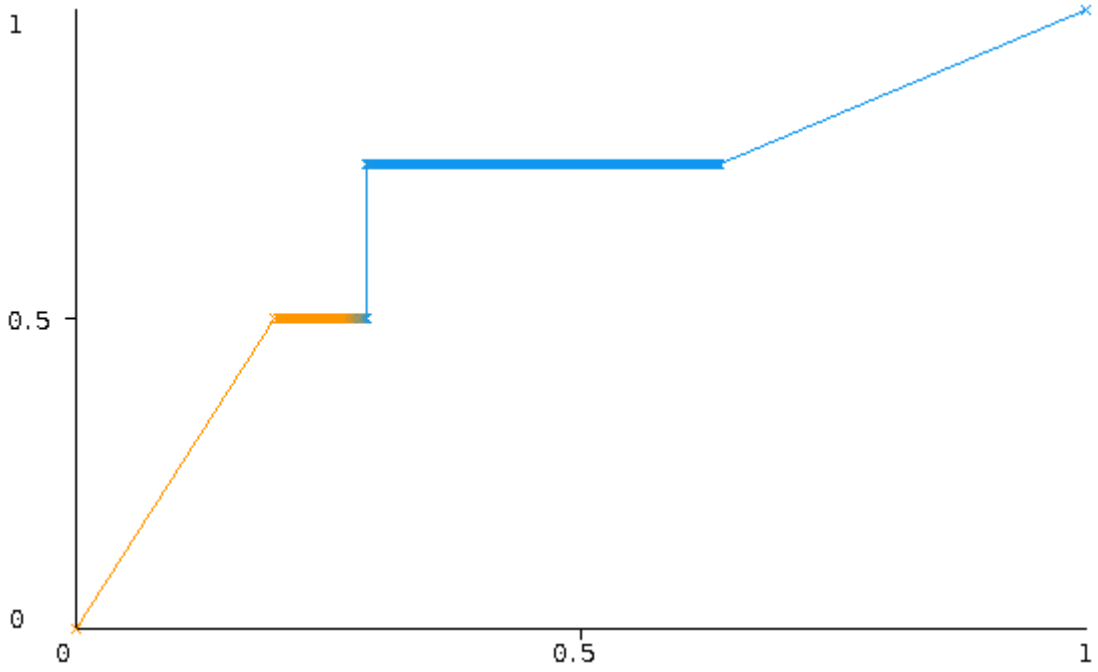


**Figure 4-8: ROC Curve of DLCS Classifier for Messaging Package**

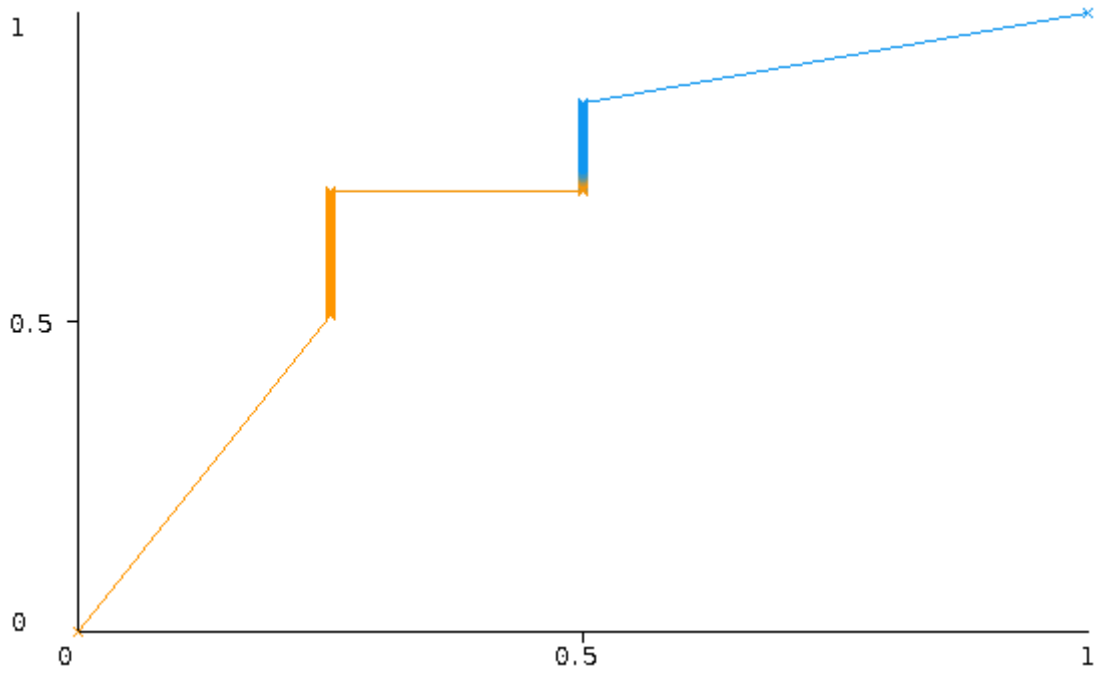


**Figure 4-9: ROC Curve of DLCS Classifier for Messaging Package**

### *4.5.4 Music Package*

Table 4-6 shows comparison between DL and cost sensitive classifier with the below parameters and configurations. Figure 4-10 displays the ROC curves of the above configuration for DLCS classifier with 4 Layer configuration.

**Table 4-6: Performance of DLCS Classifier for Music Package**

| Method | TP Rate | FP Rate | Precision | Recall | F-Measure | MCC | ROC Area | PRC Area |
|---|---|---|---|---|---|---|---|---|
| DLCS – 4 Layer | 0.864 | 1.000 | 0.792 | 0.864 | 0.826 | 0.169 | 0.682 | 0.809 |

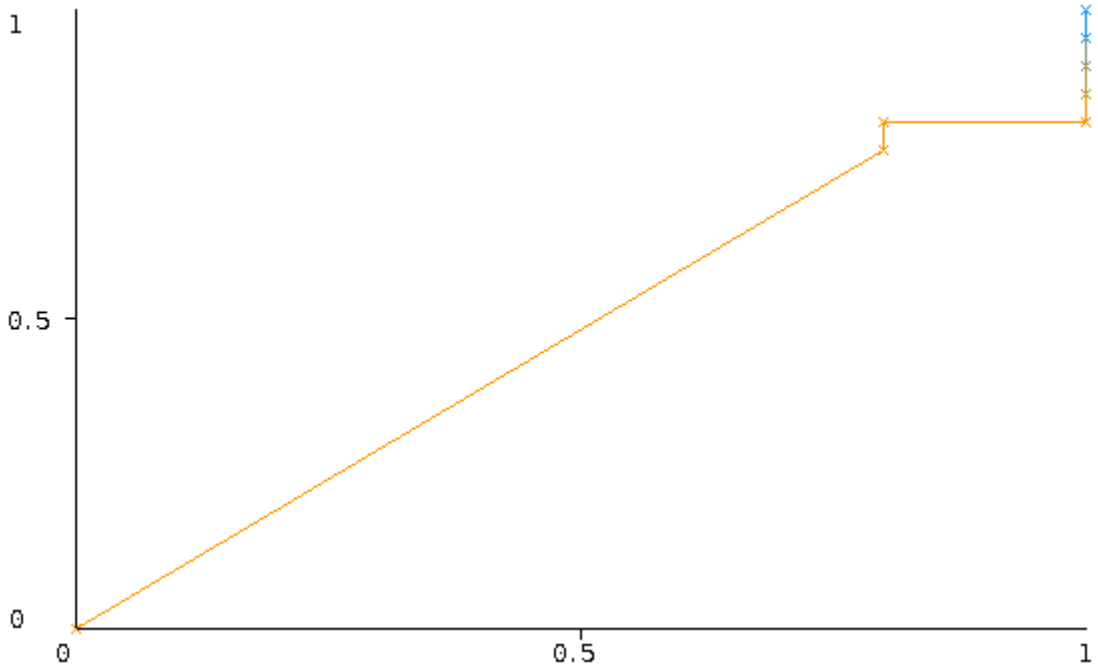


**Figure 4-10: ROC Curve of DLCS Classifier for Music Package**

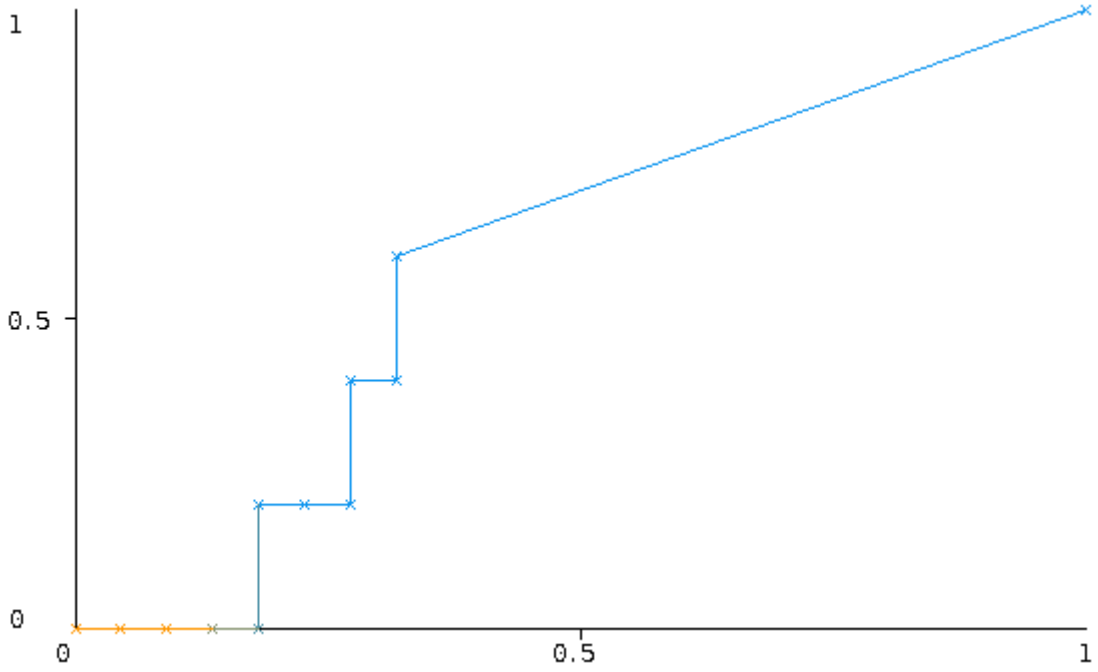


**Figure 4-11 : ROC Curve of DLCS Classifier for Music Package**

### *4.5.5 Settings Package*

Table 4.5.5 shows comparison between DL and cost sensitive classifier with the below parameters and configurations. Figure 4.5.9 displays the ROC curves of the above configuration for DLCS classifier with 4 Layer configuration.

**Table 4-7: Performance of DLCS Classifier for Settings Package**

| Method | TP Rate | FP Rate | Precision | Recall | F-Measure | MCC | ROC Area | PRC Area |
|---|---|---|---|---|---|---|---|---|
| DLCS – 4 Layer | 0.765 | 0.713 | 0.695 | 0.765 | 0.728 | 0.056 | 0.762 | 0.667 |

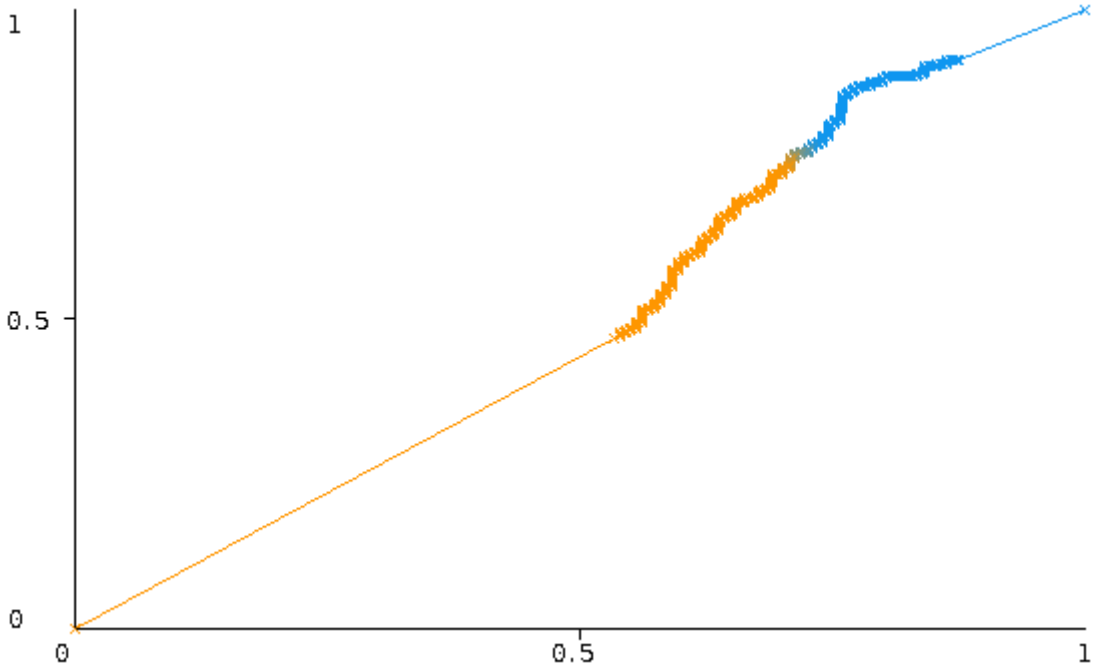


**Figure 4-12: ROC Curve of DLCS Classifier for Settings Package**

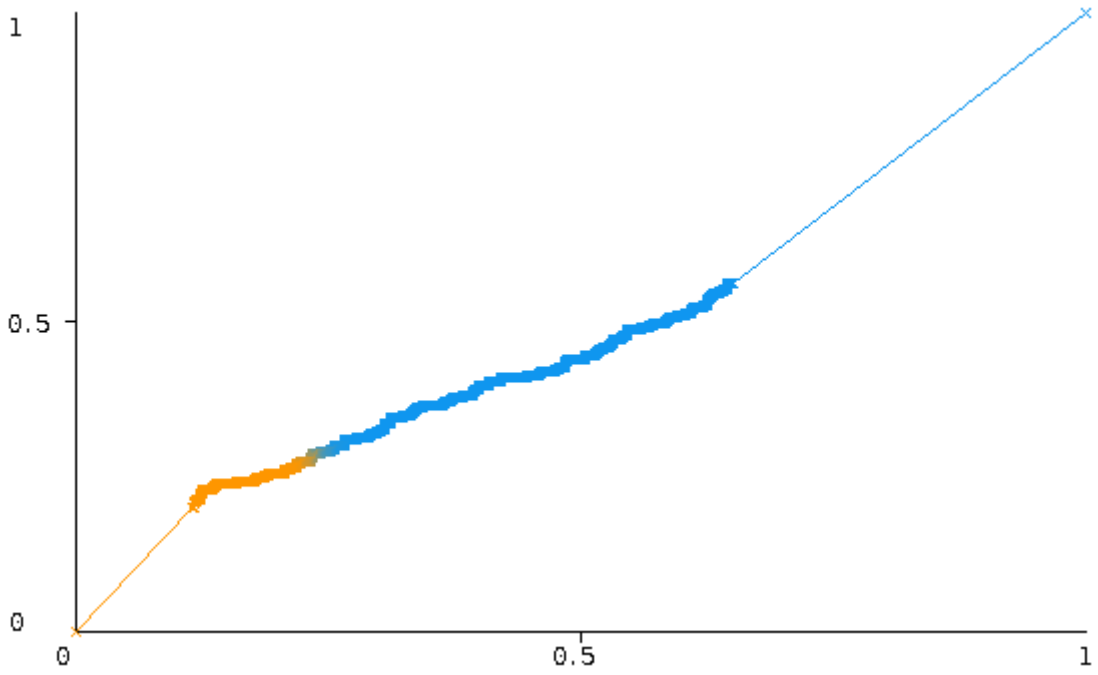


**Figure 4-13: ROC Curve of DLCS Classifier for Settings Package**

### *4.5.6 Bluetooth Package with TanH Activation Function*

Table 4.5.6 shows comparison between DL and cost sensitive classifier with the below parameters and configurations. Figure 4.5.9 displays the ROC curves of the above configuration for DLCS classifier with 4 Layer configuration.

**Table 4-8: Performance of TanH Activation for Bluetooth Package**

| Method | TP Rate | FP Rate | Precision | Recall | F-Measure | MCC | ROC Area | PRC Area |
|---|---|---|---|---|---|---|---|---|
| DLCS – 4 Layer | 0.994 | 1.000 | 0.888 | 0.994 | 0.938 | -0.026 | 0.397 | 0.878 |

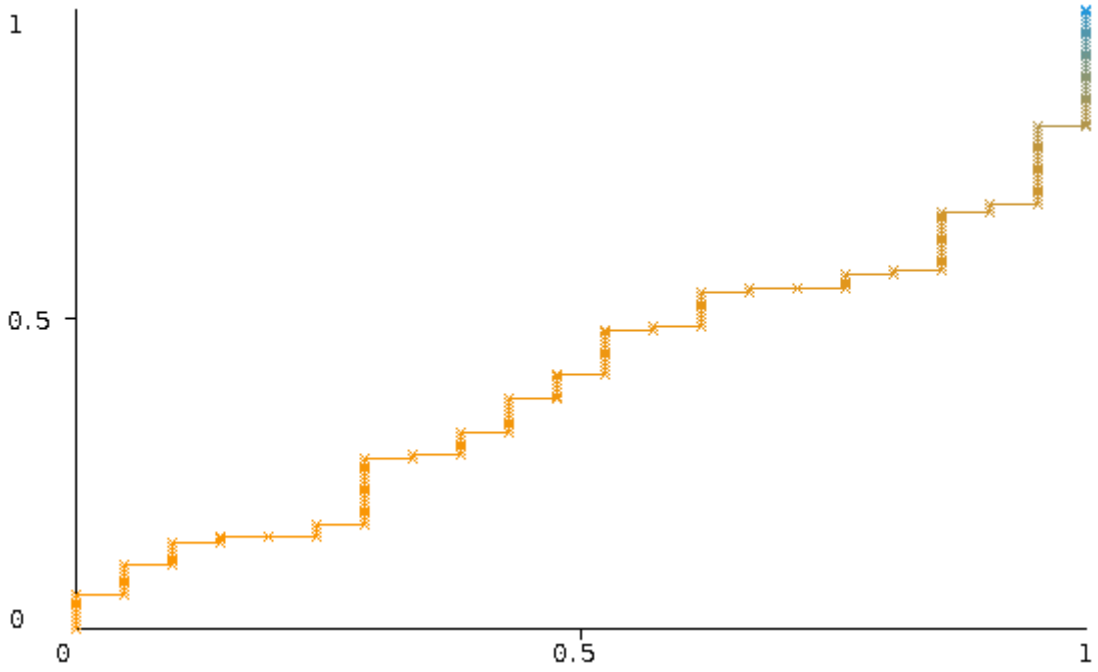


**Figure 4-14: ROC Curve DLCS with TanH Activation for Bluetooth**

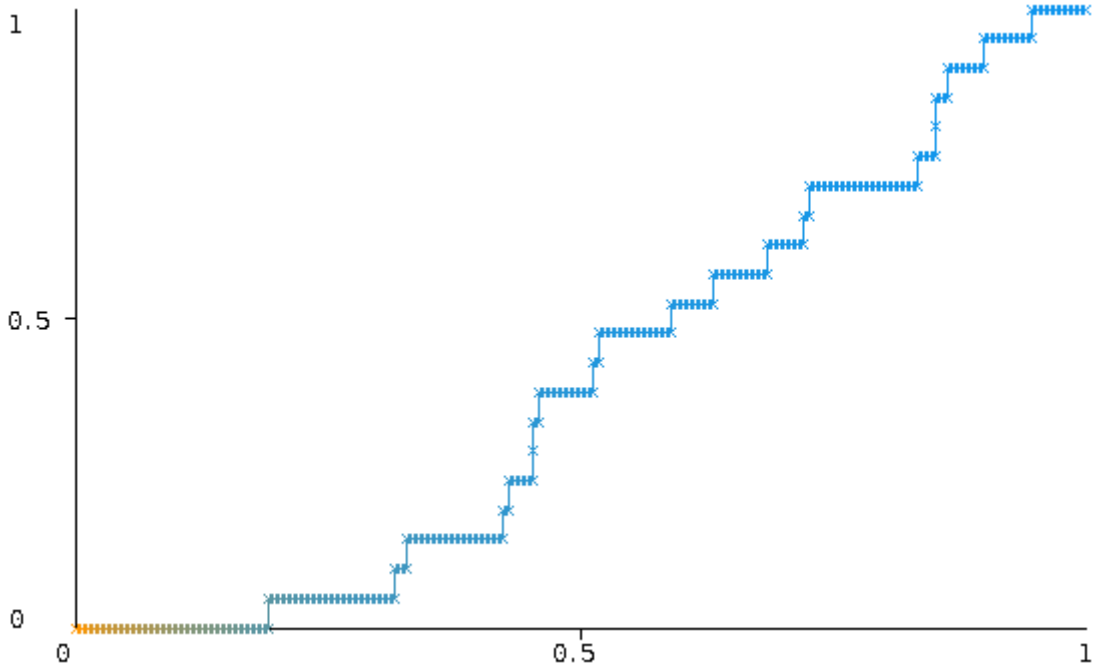


**Figure 4-15: ROC Curve DLCS with TanH Activation for Bluetooth**

### *4.5.7 Baseline Comparison with Deeper*

Table 4-9 shows comparison between DLCS and Deeper techniques. Our motivation was based on Deeper [3] which performed test on promise dataset using DBN. Our techniques use DLCS using Stochastic Gradient Descent algorithm with 4 Layer configuration. We also ran our configuration on the PROMISE dataset and observed the below results.

**Table 4-9 Performance Precision of DLCS and Deeper**

| Project | DLCS | Deeper |
|---|---|---|
| Bugzilla | 0.771 | 0.557 |
| Columba | 0.654 | 0.469 |
| JDT | 0.682 | 0.259 |
| Platform | 0.756 | 0.264 |
| Mozilla | 0.611 | 0.132 |
| PostgreSQL | 0.567 | 0.457 |
| Average | 0.673 | 0.356 |

# Chapter 5: Conclusion & Future Work

In Our work we have found relationship between CKJM Metrics suite & change proneness of any class using Feed Forward Neural Networks. From our experiment, we found that ROC and Precision values outperforms with baseline technique experimented in Deeper [3]. But, overall both the techniques are comparable on above projects.

The datasets selected in the above experiments were moderate in size but widely used applications in Android system. Predicting defects could prove useful in early stages of a life cycle of a project. The above experiments were conducted on Intel Core I7 7$^{th}$ generation CPU with 8 GB RAM and 2 GB dedicated GPU.

Performance of any classifier is critical in determining correct instances and learning from incorrect responses. Due to this we selected and combined Cost Sensitive and Feed Forward Neural Networks naming it as DLCS (Deep Learning Cost Sensitive classifier).

Since, due to system limitation, we selected the moderate data size for our project and under such small data, performance of DLCS classifier is very promising and motivating, as DLCS classifier gives competition to DBN based technique described in Deeper [3]. Hence, we can conclude our work on DLCS based model for change prediction developed can be used for forecasting change prone classes in subsequent releases of Android OS Data sets (like Android Oreo to Pie Release).

In future, we can improve the performance of DLCS based model including not only Feed Forward Networks but also Convolutional and LSTM based famous techniques. Different layers composing for CNN's and LSTM can be used and experimented on to improve accuracy in Defect Prediction. 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 DLCS combining CNN's and LSTM techniques.

# Bibliography


[1] 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.

[2] Y. K. e. al., "A large-scale empirical study of just-in-time quality assurance," *IEEE Transactions on Software Engineering,* vol. 39, no. 6, pp. 757-773, 2013.

[3] D. L. X. X. Y. Z. a. J. S. X. Yang, "Deep Learning for Just-in-Time Defect Prediction," in *IEEE International Conference on Software Quality, Reliability and Security*, Vancouver, BC, 2015.

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

[5] 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.

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

[7] 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.

[8] N. P. K. N. a. P. U. R. Malhotra, "Defect Collection and Reporting System for Git based Open Source Software," in *International Conference on Data Mining and Intelligent Computing (ICDMIC)*, New Delhi, 2014.

[9] 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.

[10] L. F. C. Manjula, "Deep neural network based hybrid approach for software defect prediction using software metrics," *Cluster Computing*, vol. III, pp. 1-17, 2018.

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

[12] M. K. Ruchika Malhotra, "Prediction of change prone classes using," *Journal of Intelligent & Fuzzy Systems*, vol. 34, no. 3, pp. 1755-1766, 2018.

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

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

[15] 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.

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

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

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

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

[20] 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.

[21] 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.