CROSS PROJECT DEFECT PREDICTION

A DISSERTATION

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE AWARD OF DEGREE
OF

MASTER OF TECHNOLOGY
IN
SOFTWARE ENGINEERING

Submitted by:

ANAMIKA AGRAWAL
(2K16/SWE/03)

Under the supervision of:

DR. RUCHIKA MALHOTRA
(ASSOCIATE PROFESSOR, CSE)
Department of CSE, DTU

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi – 110042

2016-2018

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
DELHI TECHNOLOGICAL UNIVERSITY
Formerly Delhi College of Engineering)
Bawana Road, Delhi – 110042

CANDIDATE’S DECLARATION

I, ANAMIKA AGRAWAL, 2K16/SWE/03 a student of M.TECH (Software Engineering) declare that the project Dissertation titled “**Cross Project Defect Prediction”** which is submitted by me to Department of Computer Science and Engineering, Delhi Technological University, Delhi in partial fulfilment of the requirement for the award of the degree of Master of Technology, is original and not copied from any source without proper citation. This work has not previously formed the basis for the award of any Degree, Diploma, Fellowship or other similar title or recognition.

Place: DTU, Delhi   ANAMIKA AGRAWAL
Date: M.Tech. (SWE)
                                                                     (2K16/SWE/03)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
DELHI TECHNOLOGICAL UNIVERSITY
(Formerly Delhi College of Engineering)
Bawana Road, Delhi – 110042

CERTIFICATE

This is to certify that the project report entitled "**Cross Project Defect Prediction**" is a bonafide record of the work carried out by **ANAMIKA AGRAWAL (roll no. 2K16/SWE/03)** under my guidance and supervision during the academic session 2016-2018 in the partial fulfilment of the requirement for the award of degree of Master of Technology in Software Engineering from Delhi Technological University, Delhi. To the best of my knowledge, the matter incorporated in the thesis has not been submitted to any other University/Institute for the award of any Degree or Diploma.

Place: DTU, Delhi
Date:

Dr. RUCHIKA MALHOTRA
(Associate Professor, CSE, DTU)
Supervisor

ACKNOWLEDGEMENT

First of all I would like to thank the Almighty, who has always guided me to work on the right path of the life. My greatest thanks are to my parents who bestowed ability and strength in me to complete this work.

I owe a profound gratitude to my project guide **Dr. Ruchika Malhotra** Ma’am who has been a constant source of inspiration to me throughout the period of this project. It was her competent guidance, constant encouragement and critical evaluation that helped me to develop a new insight into my project. Her calm, collected and professionally exemplary style of handling situations not only steered me through every problem, but also helped me to grow as a matured person.

I am also thankful to her for trusting my capabilities to develop this project under her guidance.

ANAMIKA AGRAWAL
M.TECH (SWE)
2K16/SWE/03

ABSTRACT

Cross-project defect prediction (CPDP) recently gained considerable attention. Many studies have provided the success of cross-project defect prediction (CPDP) to predict defects. But, most of the datasets share the same limitations: as the metrics (independent variable) hardly follow a normal distribution. They are mostly skewed data. Hence, these metrics has to be transformed before the training and predicting the defect. Various transformations have been studied in the area of cross-project defect prediction like rank transformation, log transformation and Box-Cox transformation. The yeo-johnson transformations (extended versions of the box-cox transformation) have not been used in the defect prediction area. Since, the metric values contain Zero as a value so most of the transformation did not work for Zero value, so we need to do some pre-computation in data. But yeo-johnson transformation can be used for zero values as well as negative value. This study investigates the effectiveness of yeo-johnson transformation on cross-project defect prediction. we have conducted our experiment on publicly available Promise data sets. Comparing logistic regression model built using with and without yeo-johnson transformation, transformed approach gives better result than raw data in 53% cases and 70% project achieved better result using this transformation. Further, we also investigate which classifier is well suited for this transformation, which is statistically tested by Friedman's test. Naïve bayes outperforms better among all classifiers.

CONTENTS

Titles		Page No.

LIST OF TABLES

Table No.	Title	Page No.

LIST OF FIGURES

Figure No.	Title	Page No.

LIST OF ABBREVIATIONS

CPDP	Cross Project Defect Prediction
LR	Logistic regression
LR+	Logistic regression with transformation
NB	Naïve Bayes
RF	Random Forest
ROC-AUC	Receiver Operating Characteristics-Area Under Curve
MLP	Multi-layer Perceptron
YJ	Yeo-johnson transformation

CHAPTER 1

INTRODUCTION

1.1 Overview

Defect prediction focuses on detecting fault-prone modules precisely and helps to allocate limited resources in testing and maintenance. As we know, most of the software are used in safety purposes, other important work where the severity to fault should be minimum. That's why defect prediction is the important area in the field of research. In defect prediction most of the approaches uses same data for training and predicting. These approaches typically use various features, e.g., process metrics, previous-defect metrics, source code metrics etc., to characterize a class/file/module and employ a classification algorithm to predict if a class/file/module is defective or not. Most defect prediction approaches are trained and applied on classes/files/modules from the same project. But for the new project, such historical data is not always available in practice. So the new projects don't have enough training data. One potential way of predicting defects in the new projects without historical data is to learn predictors from data of other projects. The method which uses other project for training predictors and testing different project data to analyze defect is known as **—cross-project defect prediction"**.

A cross project defect prediction can be with-in company defect prediction or cross-company defect prediction. In with-in company defect prediction we use only same project but different releases (versions) to train the predictor and perform prediction on other release. In cross company defect prediction we use similar project but not the same to train the predictor and perform prediction on other release. Software metrics (e.g. weighted method per class, depth of inheritance and lines of code) are the independent variable which is the major part of the training data used to build a defect prediction model.

Previous studies Louridas et al. [2], Concas et al. [1] and Zhang [3] report that software metrics are heavily skewed and rarely follow a normal distribution, but they constitute the power-law distribution and this skewed behavior is the one of biggest threats to the fitness of classifiers to provide better prediction (Cohen et al. [4]. In the previous studies most widely methods used to normalize the metrics are boxcox, log and rank transformations as shown in the systematic literature review by Hosseini et al. [5]. The log transformation covert the original metric values by their logarithm and the rank transformation covert the original metric values with their ranks. But these transformations do not provide result for the zero values so we first need to first shift the metric values. Here in this thesis, we are first transforming our data using yeo-johnson which work for all values and then use logistic regression to predict the defect. We also use different machine learning algorithm logistic regression, random-forest, multi-layer perceptron and naïve bayes (implemented in R language) for investigating the effect of this transformation on classifier and to generalize the result.

1.2 Motivation

Defect prediction is an important aspect of a software development process. As we know maintaining software takes 67% of the cost of total software. So to reduce the cost we do defect predictions using historical data so that new software have least defect and hence cost reduces. In prior studies only few transformations have been studied but as best of our knowledge yeo-johnson has not been studied so far. We are studying this because this is a power law transformation extended version of BoxCox transformation which can be used for zero and negative values so that we don’t need to shift original values of our data. Hence, originality of data can be maintained.

1.3 Research objective

In this thesis, we try to find empirical evidences to answer the following questions:

RQ1: Is Yeo-johnson transformation effectively increases the normality of software metrices and improves the performance of CPDP?

The yeo-johnson transformation significantly improves the normality of software metrics and reduces both skewness and kurtosis. We use logisitic regression as classifier where 53% of the total investigated pairs give better result with this transformation and 70% of project has best AUC and F-measure with this transformation.

RQ2: Does this transformation approach work well for the other classifier?
We generalize our study by investigating our approach with four classifiers (MLP, Logisitc regression, random forest and naïve bayes). We find that our approach generally out performs the better than model built without using this transformation.

However, we conducted experiments on 10 public data sets obtained from 10 projects. All these data sets are available at the PROMISE Data Repository [6]. We employed yeojohnson transformation and four machine learning algorithms to construct prediction models by using R language and its framework RStudio [7].We transform both training and testing set so that the effect of this normalization is equivalent to both the datasets.

1.4 Thesis Organization

This thesis work is bifurcated into six different chapters. Starting with the abstract, six chapters and references.

Chapter 1 gives the brief introduction about the issues discussed in this study. The chapter explains the need and use of cross project defect prediction. It defines the explaining how they affect the software systems and human life. It also addresses the heavily skewed data problem, how it has been leading to the inaccurate prediction of defect prone classes. The goals of this empirical research are stated in the form of questions at the end of this chapter.

Chapter 2 sums up the related studies with respect to software cross project defect prediction. A lot of research has been carried out in defect prediction area and various transformations had been used in this context. This chapter summarizes the major contributions and findings of the previous studies. Many studies [8], [9], [10], [11], [12] have been investigating in this field by using various transformation techniques to increase the normality of data. Most of the studies use rank transformation and log transformation while other methods are still unexplored in the area of defect prediction. Only few defect prediction study [8] has used boxcox method while the methods like yeo-johnson, lambertWxF are still novel. Furthermore, the related work describes the previous studies which have applied various ML techniques for building models.

Chapter 3 provides the details regarding the experimental design of the study. It describes the dependent, independent variables used to carry out the research. The data collection method, different datasets and the various procedural metrics used in this study are mentioned in detail. The chapter further defines the performance measure used to evaluate the prediction models and discusses the statistical test selection briefly.

Chapter 4 describes the research methodology used in the experiment. It briefly discusses the various prediction models together with the detailed explanation of the algorithms in the RStudio. A proposed transformation method yeo-johnson is also discussed with full details and implementation in RStudio. A detailed discussion is carried out regarding the impact of normalizing the data.

In Chapter 5 the obtained results are stated and analyzed using statistical tests. This chapter answers the above stated questions in chapter 1. We have performed an extensive comparison between various prediction models using non-parametric tests, Friedman. This chapter also states the advantageous use of the proposed method yeo-johnson and describes how it is better to normalize the data.

At last, Chapter 6 concludes the final outcome of the study. It states which method performed the best and guides the researchers to make use of yeo-johnson transformation to further improve the performance of defect prediction models. The chapter also provides the future scope of the research.

CHAPTER 2

LITRATURE STUDY

This section provides the prior related study in the field of defect prediction and use of different transformation technique for the cross project defect prediction.

Distribution of data has a very big impact on the defect prediction models. Normal distribution can benefit statistical methods (Osborne [1]), and it also provide better performance of linear models (Kuhn et al. [31]). To measure normal distribution of data we see the skewness and kurtosis value of the data. Most of the software metrices are heavily skewed and doesn't follow the normal distribution [3] and to reduce the skewness and enhance normality, transformation is the one of most common method (Bishara et al. [32]). In prior studies many researcher used different transformation defects natural log transformation and the rank transformation (e.g., Jiang et al. [33], Menzies et al. [34], Song et al. 2011[35] and Cruz et al. [33]) method to normalize the data to predict defects on software metrics. However, use of the log transformation significantly improves the performance of only some Classifier (e.g., Naïve Bayes) but non-significant for other classifier as (e.g., decision tree) [34], [35]. Jiang et al. [33] studies shows that when different transformation applied on ten different classifier then it is concluded that classifiers has different impact of these transformation. Different results were produced on same classifier with different transformations. It has been shown that naïve bayes perform better with rank transforamation and random forest with log transformation. In the CPDP, He et al. [36] has studies distribution characteristics and use different transformation like rank transformation for Naive Bayes, but no transformation for logistic regression and random forest.

As technologies are growing and revolving so fast that we came across various new technologies. Software built using these technologies does not have enough training set to predict defects. As these projects may also differ in terms of software metric to measure all the aspects of the software. This cause the problem of data heterogeneity between the training and testing projects (Zimmermann et al. [37]). To address this problem, Menzies et al. [38] investigated that how can we relate two different based on these metric values. These studies observe that using only the similar instances of both training and testing project can achieves better performance in CPDP than using all instances. Another study Turhan et al. [39] recommend that within project data like for the latest release we can also use the previous release data for CPDP and it yield better result than other project. An alternative solution is to transfer the knowledge from training data to testing projects to improve the performance. Various methods have been used for this and a matrix matching which relate the different instance in training and testing set by finding the type of relation they share and on transfer component analysis (TCA) to transform the training and target projects together (Nam et al. [30]). In Zhang et al. [8], a study has been done which include the simple rank, log and boxcox transformation to transform the metric value and effect of all these transformation have been studies with six classifiers. Here in our study, We perform a different transformation yeo-johnson which is not studied in the field of defect .prediction and CPDP as best of our knowledge. We investigate that does this transformation can help in cross project defect prediction. We thoroughly compare the result of 10 projects with all possible 90 combination of training and testing pairs (i.e. RQ1) and also investigate for which classifier it produced better result among random forest, multilayer perceptron, naïve bayes and logistic regression(i.e. RQ2). We selected these classifier based on previous studies in the area.

CHAPTER 3

EXPERIMENTAL DESIGN

This section provides the details of design setting and tools used in our study.

3.1 Independent And dependent variables

This study uses ‘bug’ as a dependent variable. Bug is a binary variable which indicates the defective nature of the class. A class is said to be defect prone if there is a probability of detecting a fault in the class in future versions otherwise, a class is termed as non-defective. This binary variable is dependent on a number of other variables like Chidamber & Kemerer, Halstead and McCabe metrics. The dependence of defect proneness over static code metrics is considered practical as they have helped in successful detection of the defect prone nature of the class in the past [5], [13], [15], [16]. According to the survey by Malhotra in [19] procedural metrics are widely used metrics in more than 51% of previous defect prediction studies and can be calculated at reasonably low costs for both small and large systems.

These independent variables are the set of value which rarely follow normal distribution and there skewed behavior does not produce better result. So in most of the studies these metric values are first transformed and then used for the prediction.

3.2 Empirical Data collection

We evaluate our models using defect datasets originally collected by Jureczko and Madeyski [23] from the PROMISE data repository [6] which consists of 10 releases from 10 different open source projects. Each instance in the 10 datasets consists of two parts: 20 static code metrics and a label (defective or clean).

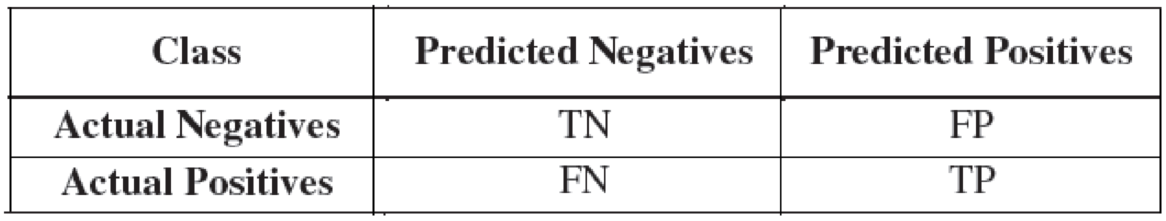
PROJECT	No. of classes	no. of defect classes	% of defect classes
ant1.7	745	166	22.281
camel1.6	965	188	19.481
ivy 2.0	352	40	11.363
jedit4.3	492	11	2.235
log4j1.2	205	189	92.195
lucene2.4	340	203	59.705
poi3.0	442	281	63.574
synapse1.2	255	86	33.725
xalan2.7	909	889	97.799
xerces1.4	573	426	74.345

Table 1 Summary of Projects

3.3 Performance measures

Performance measure is a criteria through which we can evaluate our model. Various performance measure have been used in prior studies of cross project defect prediction like AUC, F-measure, G-mean, precision, recall, balance as shown in the literature study [5]. Here we are evaluating our result on the basis of AUC and f-measure with precision and recall. We are selecting AUC because it shows the tradeoff between correct and incorrect predictions And F-measure because it includes the combined effect of precision and recall so that our study will not be biased to these measures.

These measures are evaluated on the basis of confusion matrix obtained by model's prediction. The confusion matrix contains the number of vales are predicted correctly and incorrectly.



Class	Predicted Negatives	Predicted Positives
Actual Negatives	TN	FP
Actual Positives	FN	TP

Table 2 Confusion matrix

3.3.1 Area under the Curve (ROC)

Area under the ROC Curve (AUC is a combined measure of sensitivity and specificity. The ROC is a curve plotted between sensitivity (Recall) and (1-specificity) on the y and x-coordinate axis respectively. It is a measure of how corrected our defective and non-defective classes are predicted. The larger the area enclosed under the curve the better is the performance of the ML technique.

3.3.2 Precision

Precision is a measure which evaluates that how many values predicted defective are actually defective. So precision for defect prediction is given as:-

$$\text{Precision} = \frac{\text{correctly predicted as defective}}{\text{total predicted as defective}}$$

3.3.3 Recall

Recall is a measure which evaluates that how many values are predicted defective out of actually defective values. So recall for defect prediction is given as:-

$$\text{Recall} = \frac{\text{correctly predicted as defective}}{\text{total actual defective}}$$

3.3.4 F-measure

F-measure is a measure which is the harmonic mean of both precision and recall. We use this measure because this harmonic mean gives the best value when both precision and recall are the best. It gives the combined result of both precision and recall. F-measure is given as:-

$$F = \frac{2 \times \text{Precision} \times \text{recall}}{\text{precision} + \text{recall}}$$

3.4 Statistical test

Statistical test are the test performed for inferential analysis. These are performed to find out that the two datasets are asymptotically significant or not. Here in our study, we applied these test on AUC and F values that are to evaluate our null hypothesis is rejected or not.

3.4.1 Friedman Test

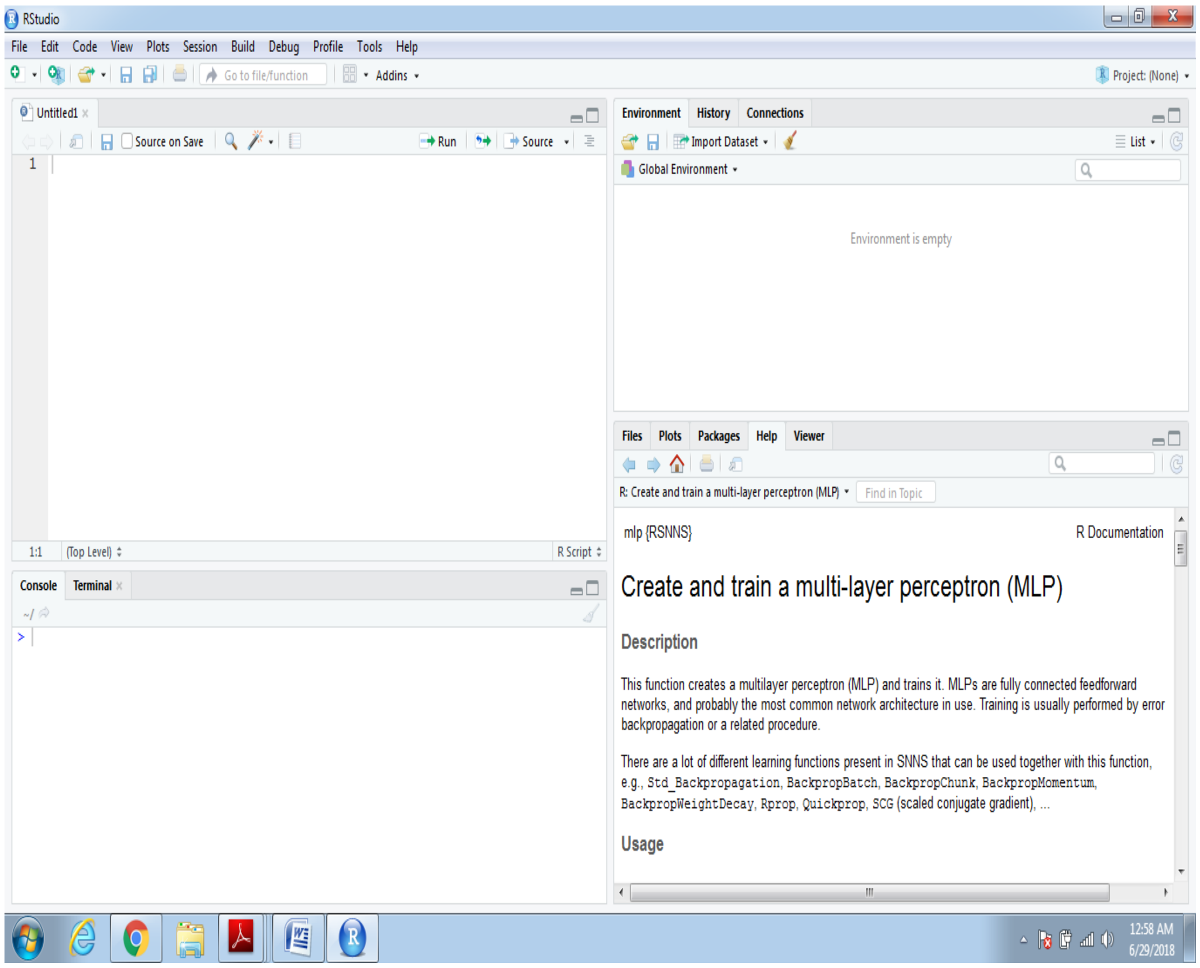
It is non- parametric test which is used to compare the two or more methods with multiple values. It is used when it contains duplicate values. When we have different techniques and multiple values we can’t find out which one is better than other. Here we have four classifier as random forest, multi layer perceptron, naïve bayes and logistic regression. Here we used SPSS tool to perform this test and find that naïve bayes has the best ranking among all the classifiers. We had performed this test on two different performance measure AUC and F- measure , both concluded that naïve bayes is asymptotically significant in both the cases when applied with p value less than 0.05.this test is used to find that our null hypothesis is rejected or not.

3.5 Tools used

This includes the tools which are used to perform our experiment in the study. We broadly used two tools.

3.5.1 RStudio

RStudio is an IDE of R language, for programming and statistical analysis. RStudio is an open source which is freely available. It is available in two editions which are RStudio Desktop and RStudio server. We have used RStudio desktop application but with the use of RStudio server we can access our IDE with the help of a browser also. Here, we can import our data in various formats like CSV and various libraries are there which have the implementation of various method. We have used these libraries and modifies according to the work. We have chosen this IDE because it’s not only a tool but had a methods which can be modifies according to the situation and made our work simple and efficient.



mlp {RSNNS} R Documentation

Create and train a multi-layer perceptron (MLP)

Description

This function creates a multilayer perceptron (MLP) and trains it. MLPs are fully connected feedforward networks, and probably the most common network architecture in use. Training is usually performed by error backpropagation or a related procedure.

There are a lot of different learning functions present in SNNS that can be used together with this function, e.g., `Std_Backpropagation`, `BackpropBatch`, `BackpropChunk`, `BackpropMomentum`, `BackpropWeightDecay`, `Rprop`, `Quickprop`, `SCG` (scaled conjugate gradient), ...

Usage

Fig 1 RStudio

3.5.2 SPSS

Statistical Package for the Social Sciences is a tool used for statistical analysis. This tool has various features which can be directly accessed by drop down menu. It is the most widely used tool for the market research. We have used this tool to validate our result by applying Friedman's.

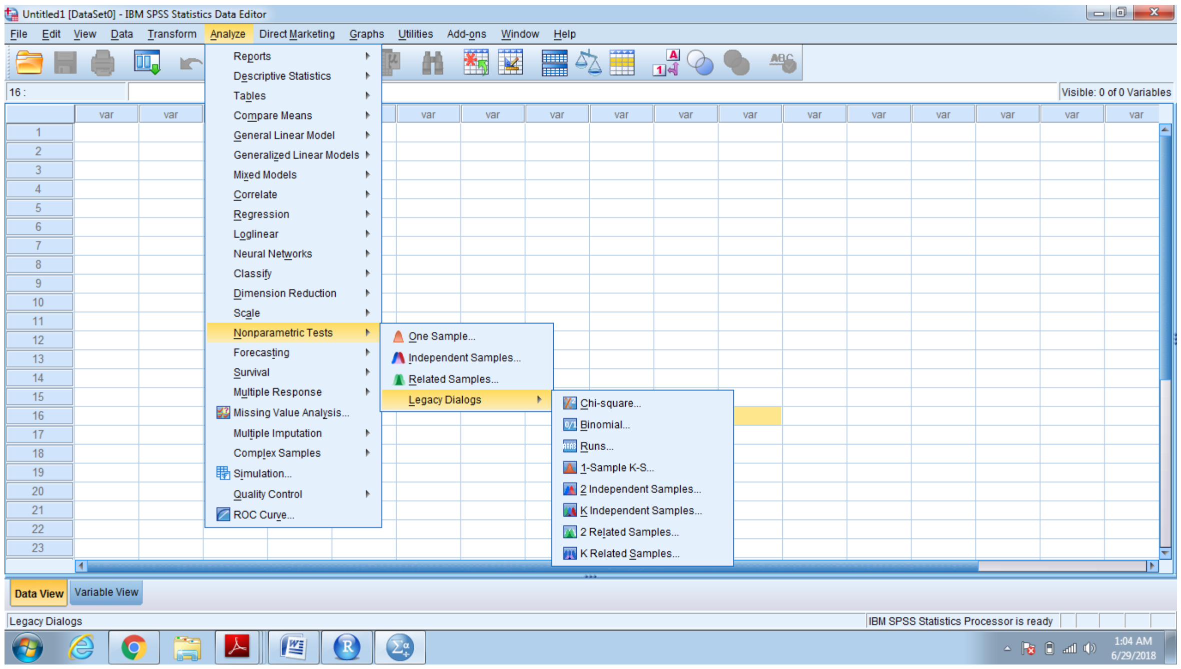

Untitled1 [DataSet0] - IBM SPSS Statistics Data Editor
File Edit View Data Transform Analyze Direct Marketing Graphs Utilities Add-ons Window Help
Reports
Descriptive Statistics
Tables
Compare Means
General Linear Model
Generalized Linear Models
Mixed Models
Correlate
Regression
Loglinear
Neural Networks
Classify
Dimension Reduction
Scale
Nonparametric Tests
Forecasting
Survival
Multiple Response
Missing Value Analysis...
Multiple Imputation
Complex Samples
Simulation...
Quality Control
ROC Curve...
One Sample...
Independent Samples...
Related Samples...
Legacy Dialogs
Chi-square...
Binomial...
Runs...
1-Sample K-S...
2 Independent Samples...
K Independent Samples...
2 Related Samples...
K Related Samples...
Visible: 0 of 0 Variables
Data View
Variable View
Legacy Dialogs
IBM SPSS Statistics Processor is ready

Fig 2 SPSS

CHAPTER 4

RESEARCH METHODOLOGIES

The Fig 3(a) and Fig 3(b) describes the flow of the events occurred in this study.

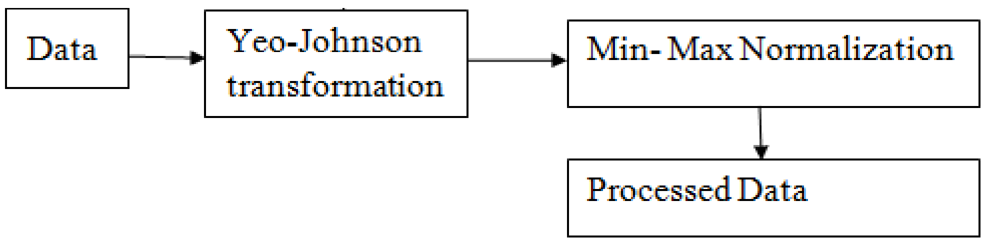


Fig 3(a) Data Preprocessing

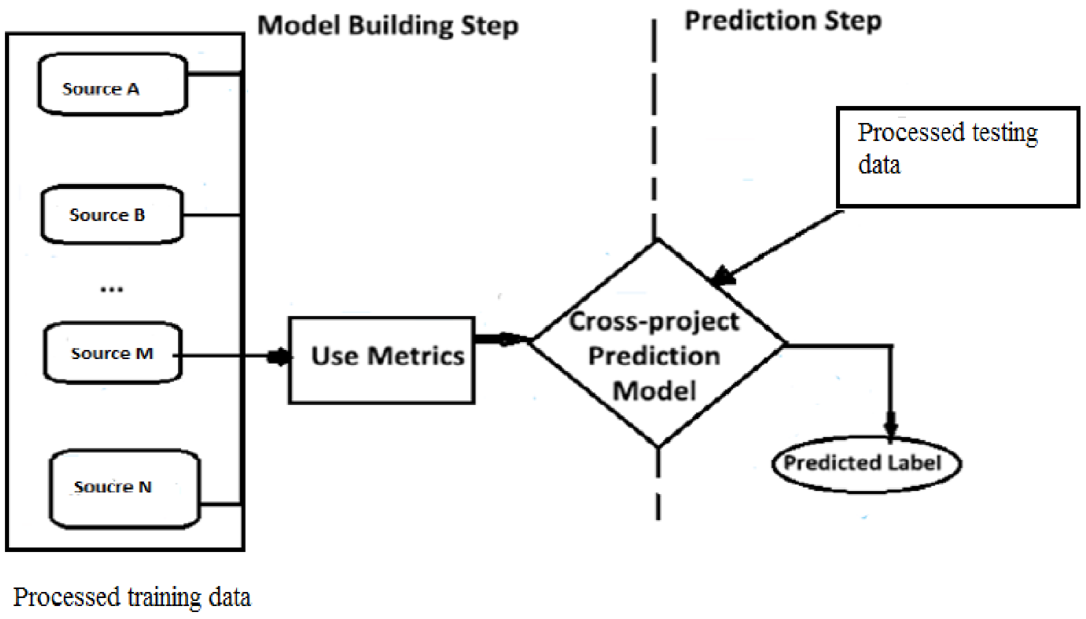


Fig 3(b) Cross Project Defect Prediction

Firstly, we transformed our data using yeo-johnson transformation [24] and normalized to the scale of 0 to 1. This transformed data is used for further study. We preprocessed both training and testing set with each metrics independently so that the impact is equivalent to both the sets.

Finally, we have a model having two steps a model building and prediction steps. Where model building includes the formation of all the possible combination of training and testing set. Here we have total 90 combinations of 10 projects. And prediction step chooses a classifier and apply the prediction to predict the label.

4.1 Normality measurements

Widely used measures of data normality are kurtosis and skewness. Here, we computed the skewness and kurtosis of these software metrics, using the R functions skewness [25] and kurtosis [25] in the R package e1071 [27].

4.1.1 Skewness

It measures the degree of asymmetry in the probability distribution of the values of software metric. We find skewness of all the metrics and try to normalize the data so that skewness approaches to zero because normalized curve has zero skewness. Skewness can be either positive, negative or zero as illustrated in Fig. 4(a). If the values are greater than 1 and less than -1 means data is heavily skewed.

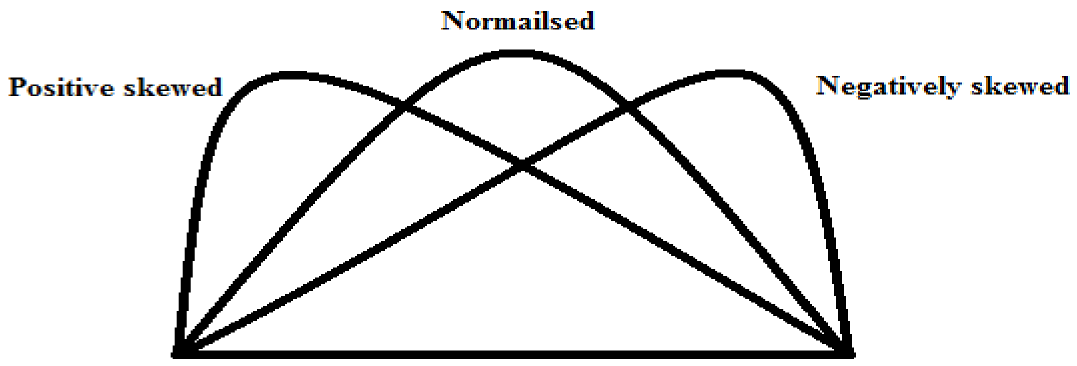


Fig 4(a) Skewness

4.1.2 Kurtosis

It measures the peakness or flatness (e.g., the width of the peak) in the probability distribution of the values of software metric. The value of kurtosis should be zeros for normal distribution. It can either be positive, negative or zero. Positive and negative kurtosis are illustrated in Fig. 4(b).

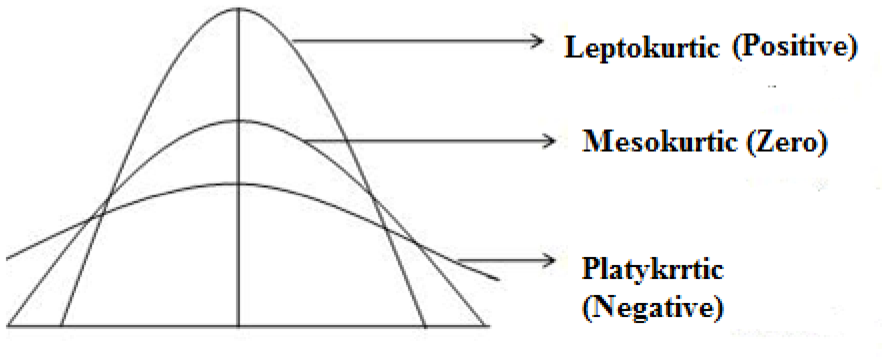


Fig. 4(b) Kurtosis

4.2 Yeo-Johnson transformation

YJ transformation is the extended version of boxcox power law transformation. Boxcox is used for strictly positive numbers. So it has a dependency on the value of values to be transformed but YJ does not have any restriction on the input value. YJ transformation is same as Boxcox for positive values and also works well for Zero and negative value without shifting the values.

As per best of our knowledge YJ transformation has not been studied in the field of CPDP.

BoxCox transformation is given as:-

$$BoxCox(x,\lambda) = \begin{cases} \frac{x^{\lambda}-1}{\lambda} & \text{if } \lambda \neq 0 \\ ln(x) & \text{if } \lambda = 0 \end{cases}$$

EQ(1) BoxCox Transformation

YJ transformation is given as:-

$$\mathrm{YJ}(\lambda, y) = \begin{cases} ((y+1)^{\lambda} - 1)/\lambda & \text{if } \lambda \neq 0, y \geq 0 \\ \log(y+1) & \text{if } \lambda = 0, y \geq 0 \\ -[(-y+1)^{2-\lambda} - 1)]/(2-\lambda) & \text{if } \lambda \neq 2, y < 0 \\ -\log(-y+1) & \text{if } \lambda = 2, y < 0 \end{cases}$$

EQ(2) Yeo-Johnson Transformation

Here, y is the values to be transformed. As we can see in above equations BoxCox is only valid for the strictly positive values as when the value of ʎ=0 then there is log-transformation and the log transformation of zero is infinity. So to use the BoxCox we first need to shift the data values. But in YJ transformation it does not depend on the value of y also where ʎ=0. YJ contains all the good properties of the BoxCox transformation that's why we want to explore this in our thesis.

4.2.1 Estimating lambda value

The value of lambda plays a very important role in this transformation.

Values of Lambda(ʎ)	Type of power-law transformation
0.0	Log transformation
1.0	No transformation
0.5	Square-root transformation
-1.0	Inverse transformation

Table-3 Power-law transformation based on lambda value

Since the different metrics rarely follow the same distribution so we estimate the lambda parameter for each metrics using "yeojohnson" method of "BestNormalize" [28] library in R, which gives the best estimated value of lambda. This lambda values gives approximately minimum skewness to the metric value and transform values according to the estimated lambda. Here in our work, we transform each software metric values independently for both training and testing datasets.

4.2.2 Min Max transformation

Most of the transformed values are in a wide range. So min-max transformation scales our values to the scale of 0 and 1, which is better for classification models (Han et al. [29], Nam et al. [30]).

$$X = \frac{y - \min}{max - \min} + 1$$

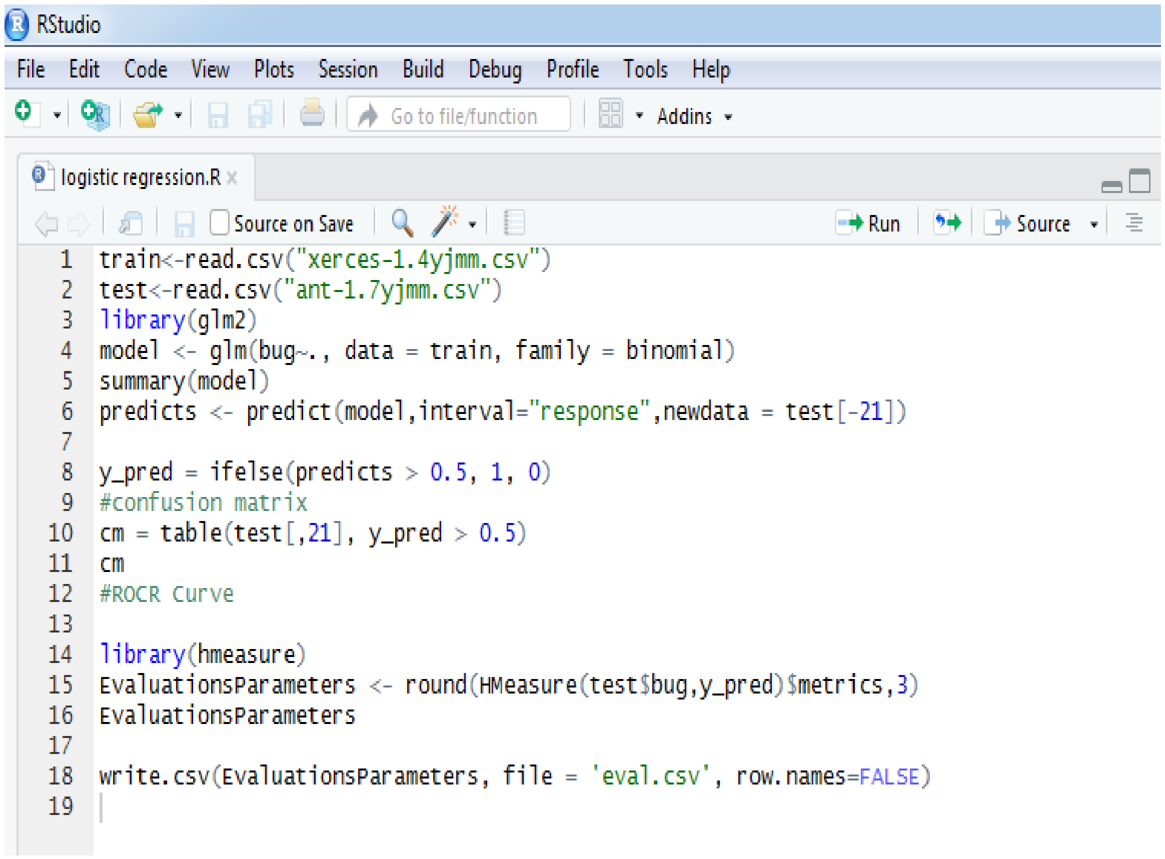
EQ(3) Min-Max Transformation

4.3 Machine learning techniques

We have used logistic regression to find the impact of yeo-johnson transformation to the software metrics for predicting defect. We also used four machine learning classifier (logistic regression, random forest, multi layer perceptron and naïve bayes) to generalize the result of our study.

4.3.1 Logistic Regression

Logistic regression is the classification algorithm used when dependent variable is binary. Here bug is the binary dependent variable. This dependent variable has class as either zero (non-defective) or one (defective). Logistic regression model where the log-odds of the probability of an event is a linear combination of independent or predictor variables. The binary logistic regression is where the dependent variable is in two classes and multi logistic regression model is where dependent variable is in multiclass. These values are ordinal values which classify the data which describe according to the severity of the defects. Here we have implemented logistic regression in Rstudio using GLM library.

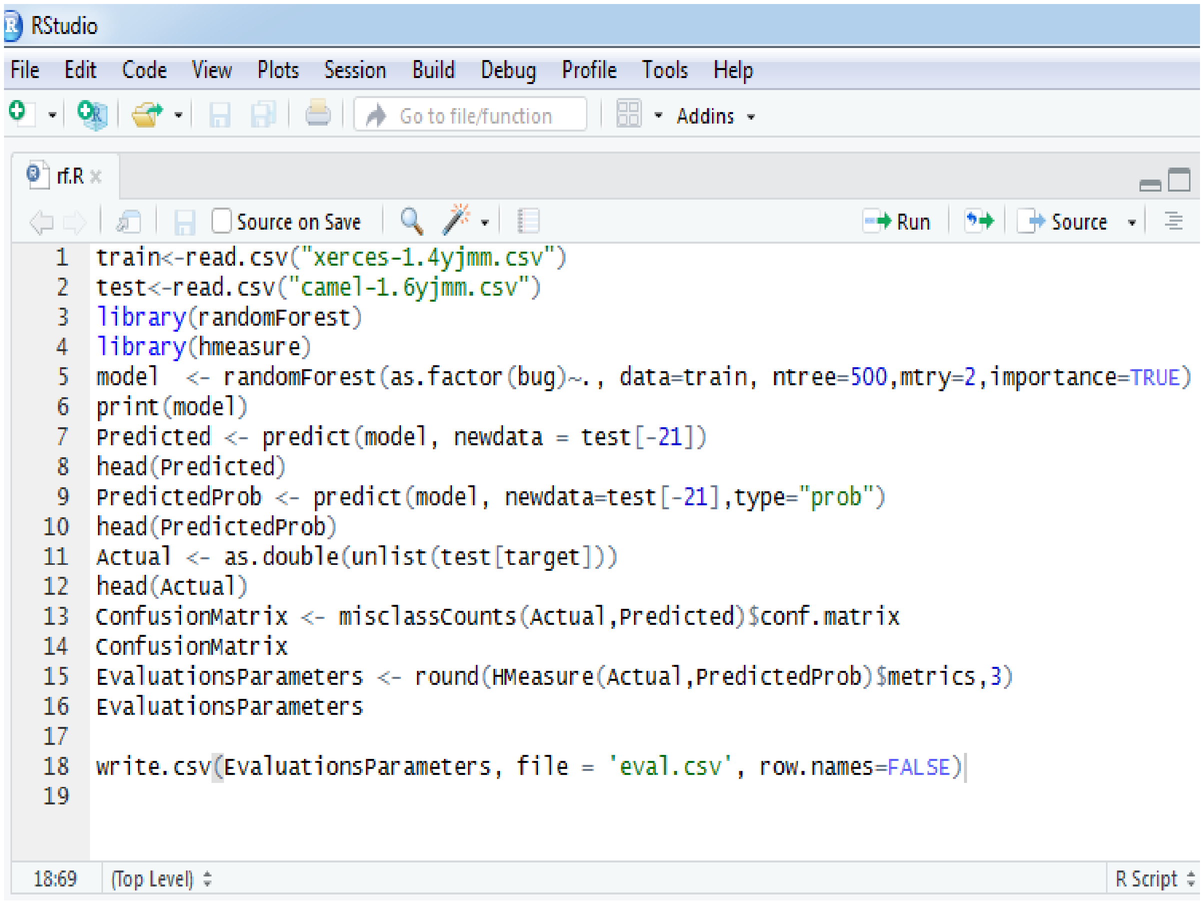


```
train<-read.csv("xerces-1.4yjmm.csv")
test<-read.csv("ant-1.7yjmm.csv")
library(glm2)
model <- glm(bug~., data = train, family = binomial)
summary(model)
predicts <- predict(model,interval="response",newdata = test[-21])

y_pred = ifelse(predicts > 0.5, 1, 0)
#confusion matrix
cm = table(test[,21], y_pred > 0.5)
cm
#ROCR Curve

library(hmeasure)
EvaluationsParameters <- round(HMeasure(test$bug,y_pred)$metrics,3)
EvaluationsParameters

write.csv(EvaluationsParameters, file = 'eval.csv', row.names=FALSE)
```

Fig. 5 Implementation of Logistic Regression in RStudio

4.3.2 Random Forest

Random Forest is an ensemble machine learning algorithm used for both regression and classification problems. It is the extended version of of decision tress algorithm where multiple decision tree are formed, this forests of trees splitting with oblique hyperplanes can gain accuracy as they grow without suffering from overtraining, as long as the forests are randomly restricted to be sensitive to only selected feature dimensions. Here we have implemented random forest in Rstudio using rpart library.



```
train<-read.csv("xerces-1.4yjmm.csv")
test<-read.csv("camel-1.6yjmm.csv")
library(randomForest)
library(hmeasure)
model  <- randomForest(as.factor(bug)~., data=train, ntree=500,mtry=2,importance=TRUE)
print(model)
Predicted <- predict(model, newdata = test[-21])
head(Predicted)
PredictedProb <- predict(model, newdata=test[-21],type="prob")
head(PredictedProb)
Actual <- as.double(unlist(test[target]))
head(Actual)
ConfusionMatrix <- misclassCounts(Actual,Predicted)$conf.matrix
ConfusionMatrix
EvaluationsParameters <- round(HMeasure(Actual,PredictedProb)$metrics,3)
EvaluationsParameters

write.csv(EvaluationsParameters, file = 'eval.csv', row.names=FALSE)

```

Fig. 6 Implementation of Random Forest in RStudio

4.3.3 Multilayer perceptron

Basically, a multilayer perceptron (MLP) is a feed-forward artificial neural model. There is single hidden layer lies between input-output layers. Its functionality is mapping the set of inputs to a particular set of outputs means data row in one direction from input to the output layer. Each join is a neuron leaving the input joins and each join has an activation function (Y) corresponding to it. MLP is applying a back propagation algorithm for training the neural network. MLP is a revision of the standard linear perceptron and can solve the problem which is not linear-separable.

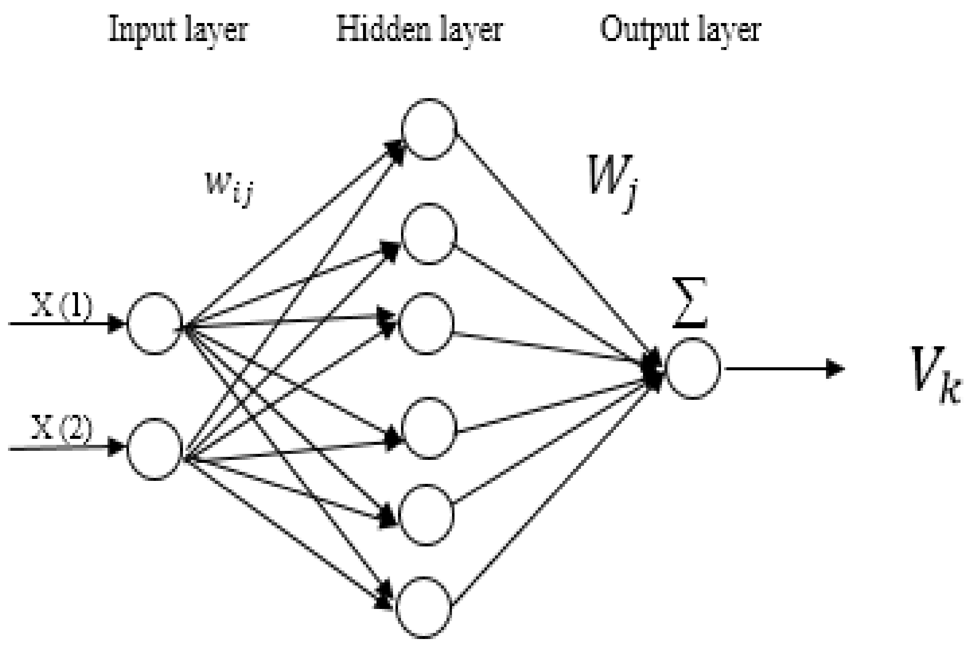
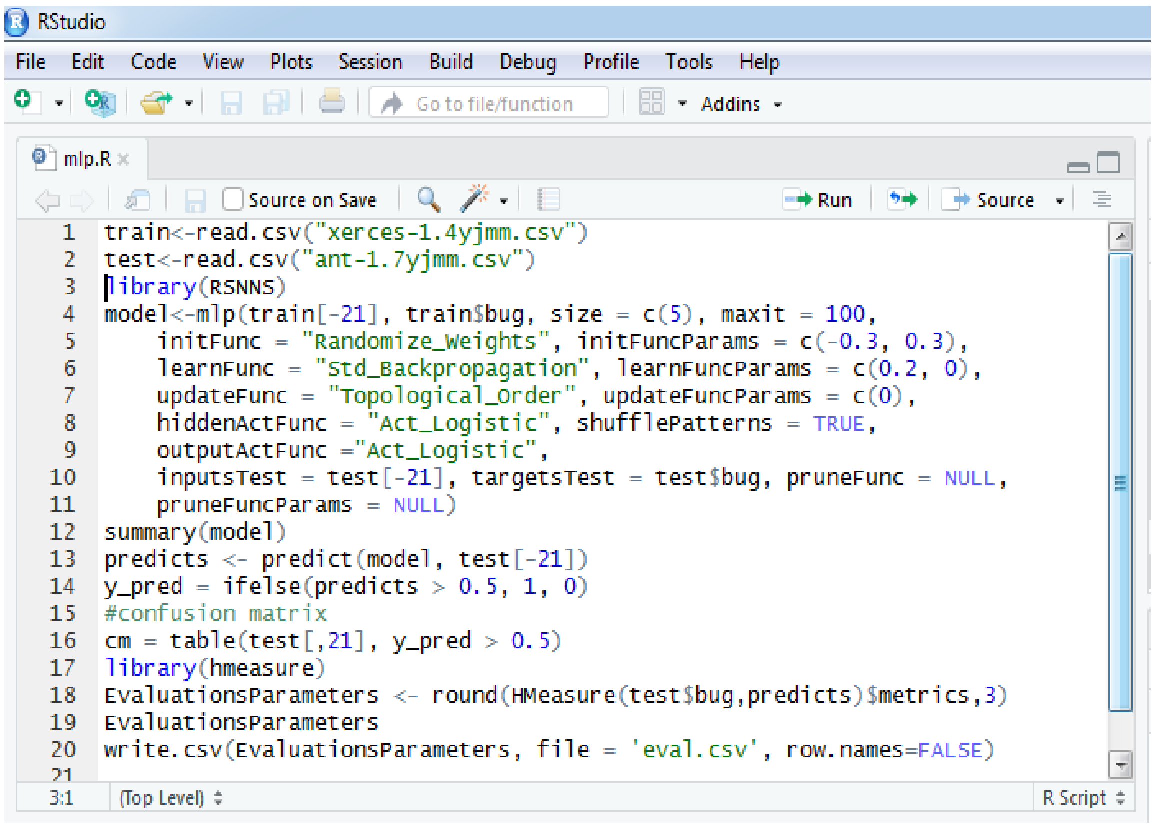


Fig. 7(a) Multi-layer Perceptron

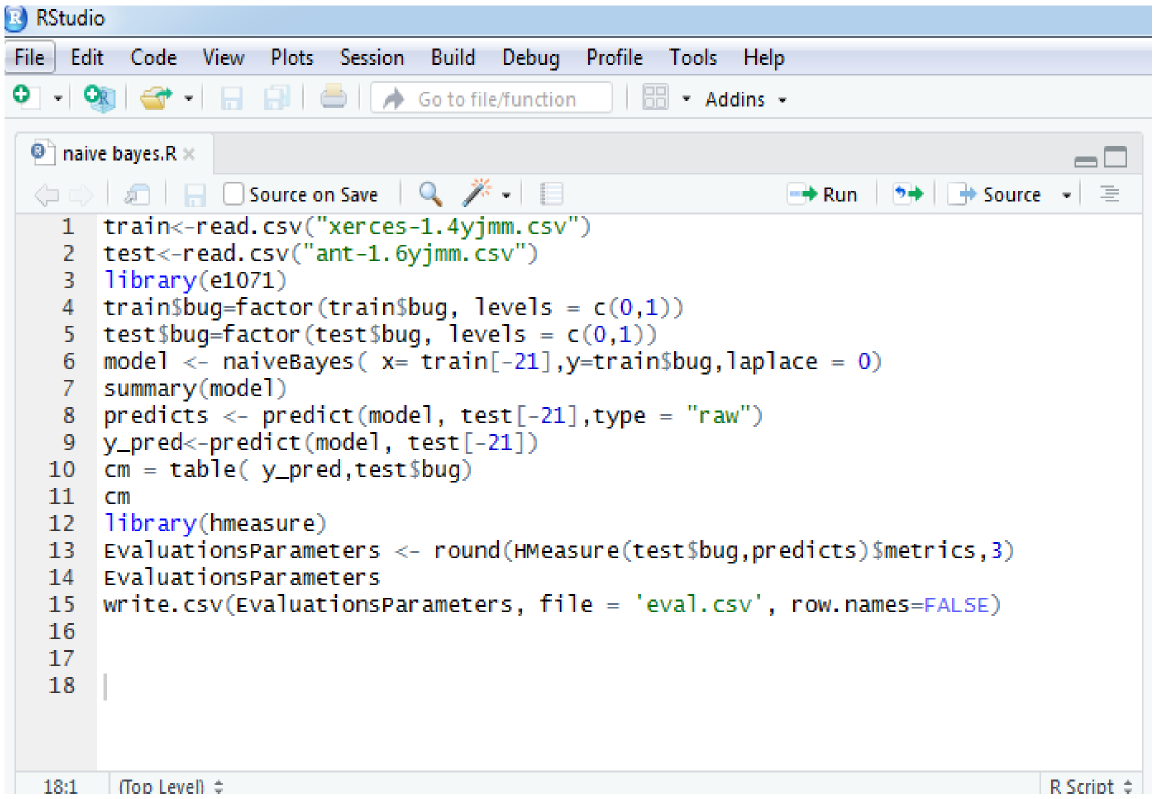


```r
train<-read.csv("xerces-1.4yjmm.csv")
test<-read.csv("ant-1.7yjmm.csv")
library(RSNNS)
model<-mlp(train[-21], train$bug, size = c(5), maxit = 100,
    initFunc = "Randomize_Weights", initFuncParams = c(-0.3, 0.3),
    learnFunc = "Std_Backpropagation", learnFuncParams = c(0.2, 0),
    updateFunc = "Topological_Order", updateFuncParams = c(0),
    hiddenActFunc = "Act_Logistic", shufflePatterns = TRUE,
    outputActFunc ="Act_Logistic",
    inputsTest = test[-21], targetsTest = test$bug, pruneFunc = NULL,
    pruneFuncParams = NULL)
summary(model)
predicts <- predict(model, test[-21])
y_pred = ifelse(predicts > 0.5, 1, 0)
#confusion matrix
cm = table(test[,21], y_pred > 0.5)
library(hmeasure)
EvaluationsParameters <- round(HMeasure(test$bug,predicts)$metrics,3)
EvaluationsParameters
write.csv(EvaluationsParameters, file = 'eval.csv', row.names=FALSE)
```

Fig. 7(b) Implementation of Multi-layer Perceptron in RStudio

4.3.4 Naïve Bayes

Naïve Bayes belongs to family of classifier based on the probabilistic model. It assumes that the independent variables of a datasets has its independent impact on the classifier build and then find the correlation among the independent variables. It uses the bayes theorem to find the conditional probability of the model. This has been proved very efficient in supervised learning. Here we have implemented this in RStudio e1071 library.



```
train<-read.csv("xerces-1.4yjmm.csv")
test<-read.csv("ant-1.6yjmm.csv")
library(e1071)
train$bug=factor(train$bug, levels = c(0,1))
test$bug=factor(test$bug, levels = c(0,1))
model <- naiveBayes( x= train[-21],y=train$bug,laplace = 0)
summary(model)
predicts <- predict(model, test[-21],type = "raw")
y_pred<-predict(model, test[-21])
cm = table( y_pred,test$bug)
cm
library(hmeasure)
EvaluationsParameters <- round(HMeasure(test$bug,predicts)$metrics,3)
EvaluationsParameters
write.csv(EvaluationsParameters, file = 'eval.csv', row.names=FALSE)
```

Fig. 8 Implementation of Naïve Bayes in RStudio

CHAPTER 5

EMPERICAL RESULT & ANALYSIS

In this section, we discuss and analyse the results obtained by applying logistic regression technique on yeo-johnson transformed as well as original datasets. We also try to find out which among four ML technique outperform the other when used with this transformation. In order to examine and equate the performance of yeojohnson transformation as well as MC learners, we use five performance metrics: sensitivity (recall), precision, AUC, Flase positive rate and F-measure. The results are assessed by using two non-parametric statistical tests: Friedman and Wilcoxon signed rank test. The investigation of results is carried out systematically by sequentially answering the research questions mentioned in chapter 1.

5.1 RQ1: Is Yeo-johnson transformation effectively increases the normality of software metrices and improves the performance of CPDP?

Table 4, provide the performance measure like AUC, F-measrue, precision and recall for the logistic regression without transformation as LR and with transformation as LR+. To empirical study shows that in more than 54% cases LR+ outperform than LR. To statistically validate our result we perform the Friedman test for the below hypothesis.

		LR+		LR	
Training	Testing	AUC	F	AUC	F
ant1.7	camel1.6	0.52	0.087	0.519	0.095
	ivy2.0	0.59	0.308	0.677	0.43
	jedit4.3	0.67	0.276	0.628	0.119
	log4j1.2	0.5	0.011	0.505	0.021
	lucene2.4	0.51	0.02	0.527	0.128
	poi3.0	0.5	0.014	0.521	0.113
	synapse1.2	0.55	0.186	0.622	0.42

	xalan2.7	0.53	0.107	0.556	0.2
	xerces1.4	0.5	0.009	0.538	0.14
camel1.6	ant1.7	0.51	0.086	0.548	0.211
	ivy2.0	0.65	0.347	0.663	0.386
	jedit4.3	0.64	0.138	0.546	0.071
	log4j1.2	0.54	0.48	0.523	0.923
	lucene2.4	0.58	0.362	0.517	0.965
	poi3.0	0.52	0.247	0.509	0.111
	synapse1.2	0.62	0.436	0.552	0.243
	xalan2.7	0.53	0.111	0.517	0.965
	xerces1.4	0.59	0.357	0.533	0.147
jedit4.3	ant1.7	0.53	0.13	0.51	0.047
	camel1.6	0.51	0.051	0.501	0.03
	ivy2.0	0.5	NA	0.57	0.245
	log4j1.2	0.5	NA	0.5	NA
	lucene2.4	0.5	NA	0.505	0.02
	poi3.0	0.5	0.007	0.52	0.088
	synapse1.2	0.51	0.023	0.512	0.045
	xalan2.7	0.5	0.013	0.511	0.041
	xerces1.4	0.5	NA	0.5	0.028
log4j1.2	ant1.7	0.51	0.367	0.516	0.37
	camel1.6	0.51	0.321	0.506	0.327
	ivy2.0	0.51	0.206	0.504	0.205
	jedit4.3	0.51	0.044	0.53	0.046
	lucene2.4	0.59	0.406	0.507	0.742
	poi3.0	0.53	0.757	0.508	0.774
	synapse1.2	0.5	0.504	0.512	0.509
	xalan2.7	0.58	0.263	0.541	0.296
	xerces1.4	0.5	0.063	0.61	0.872
lucene2.4	ant1.7	0.63	0.43	0.661	0.46
	camel1.6	0.59	0.354	0.589	0.359
	ivy2.0	0.72	0.333	0.679	0.308
	jedit4.3	0.66	0.082	0.544	0.05
	log4j1.2	0.62	0.65	0.583	0.517
	poi3.0	0.7	0.714	0.7	0.719
	synapse1.2	0.64	0.555	0.647	0.57
	xalan2.7	0.63	0.418	0.813	0.77
	xerces1.4	0.71	0.63	0.613	0.561
poi3.0	ant1.7	0.71	0.508	0.641	0.443

	camel1.6	0.56	0.317	0.55	0.32
	ivy2.0	0.7	0.308	0.716	0.348
	jedit4.3	0.57	0.054	0.532	0.048
	log4j1.2	0.63	0.664	0.524	0.519
	lucene2.4	0.57	0.574	0.557	0.589
	synapse1.2	0.66	0.591	0.617	0.548
	xalan2.7	0.77	0.766	0.735	0.718
	xerces1.4	0.73	0.672	0.651	0.555
ivy2.0	ant1.7	0.5	NA	0.576	0.271
	camel1.6	0.5	NA	0.5	0.326
	jedit4.3	0.5	NA	0.61	0.154
	log4j1.2	0.51	0.031	0.505	0.021
	lucene2.4	0.53	0.111	0.52	0.076
	poi3.0	0.51	0.042	0.518	0.082
	synapse1.2	0.51	0.023	0.541	0.151
	xalan2.7	0.5	0.015	0.517	0.065
	xerces1.4	0.52	0.064	0.52	0.077
synapse1.2	ant1.7	0.66	0.47	0.702	0.538
	camel1.6	0.52	0.197	0.509	0.169
	ivy2.0	0.76	0.474	0.756	0.458
	jedit4.3	0.62	0.067	0.579	0.063
	log4j1.2	0.54	0.147	0.557	0.381
	lucene2.4	0.53	0.283	0.541	0.401
	poi3.0	0.54	0.185	0.772	0.754
	xalan2.7	0.7	0.663	0.62	0.498
	xerces1.4	0.61	0.38	0.636	0.444
xalan2.7	ant1.7	0.52	0.374	0.501	0.365
	camel1.6	0.5	0.323	0.513	0.052
	ivy2.0	0.56	0.224	0.554	0.203
	jedit4.3	0.55	0.048	0.504	0.044
	log4j1.2	0.56	0.296	0.5	0.959
	lucene2.4	0.6	0.698	0.5	0.748
	poi3.0	0.51	0.288	0.507	0.028
	synapse1.2	0.54	0.491	0.503	0.023
	xerces1.4	0.67	0.843	0.506	0.023
xerces1.4	ant1.7	0.54	0.381	0.536	0.379
	camel1.6	0.55	0.346	0.55	0.348
	ivy2.0	0.53	0.213	0.508	0.167
	jedit4.3	0.53	0.047	0.552	0.049

	log4j1.2	0.54	0.156	0.514	0.264
	lucene2.4	0.5	0.736	0.525	0.736
	poi3.0	0.54	0.783	0.578	0.79
	synapse1.2	0.55	0.533	0.553	0.531
	xalan2.7	0.52	0.968	0.541	0.89

Table 4 AUC and F value of LR and LR+

Null Hypothesis (H0): The (AUC, F-measure) results of CPDP models of the Logistic regression without yeo-johnson transformation (LR) and with transformation (LR+) approach are same when applied to 10 projects (PROMISE repository datasets) with all possible 90 combinations.

Alternate Hypothesis (H0a): The (AUC, F-measure) results of CPDP models of the Logistic regression without yeo-johnson transformation (LR) and with transformation (LR+) approach are different when applied to 10 projects (PROMISE repository datasets) with all possible 90 combinations.

Friedman Test analysis

According to the test performed on the AUC and F values it is shown that LR+ result are non significant but in 53% of the results are better in LR+ than LR with p-value as 0.05.

5.2 RQ2: Does this transformation approach work well for the other classifier?

Table 5 and table 6 provide the performance measure like AUC and F-measure all classifier LR+, random forest, MLP and naïve respectively. To empirical study shows that in more than 54% cases LR+ outperform than LR. To statistically validate our result we perform the Friedman test for the below hypothesis.

Null Hypothesis (H1): The (AUC, F-measure) results of CPDP models of all the classifier are same when applied to 10 projects (PROMISE repository datasets) with all possible 90 combinations.

Alternate Hypothesis (H1a): The (AUC, F-measure) results of CPDP models of all the classifier are different. When applied to 10 projects (PROMISE repository datasets) with all possible 90 combinations.

		AUC			
Training	Testing	NB	RF	MLP	LR
ant1.7	camel1.6	0.621	0.592	0.602	0.518
	ivy2.0	0.79	0.834	0.808	0.594
	jedit4.3	0.636	0.616	0.617	0.667
	log4j1.2	0.671	0.556	0.51	0.503
	lucene2.4	0.705	0.686	0.631	0.505
	poi3.0	0.836	0.784	0.8	0.504
	synapse1.2	0.745	0.749	0.751	0.546
	xalan2.7	0.737	0.887	0.757	0.528
	xerces1.4	0.788	0.598	0.706	0.502
camel1.6	ant1.7	0.782	0.737	0.68	0.514
	ivy2.0	0.756	0.733	0.716	0.647
	jedit4.3	0.592	0.517	0.549	0.637
	log4j1.2	0.643	0.618	0.567	0.536
	lucene2.4	0.644	0.687	0.6	0.579
	poi3.0	0.705	0.72	0.541	0.522
	synapse1.2	0.681	0.689	0.676	0.615
	xalan2.7	0.651	0.819	0.539	0.53
	xerces1.4	0.725	0.702	0.823	0.59
jedit4.3	ant1.7	0.749	0.768	0.632	0.531
	camel1.6	0.515	0.562	0.602	0.511
	ivy2.0	0.739	0.8	0.71	0.5
	log4j1.2	0.614	0.663	0.618	0.5
	lucene2.4	0.605	0.71	0.598	0.5
	poi3.0	0.751	0.69	0.776	0.502
	synapse1.2	0.671	0.694	0.531	0.506
	xalan2.7	0.681	0.83	0.833	0.503
	xerces1.4	0.525	0.564	0.871	0.5
log4j1.2	ant1.7	0.756	0.564	0.698	0.508
	camel1.6	0.611	0.55	0.603	0.507
	ivy2.0	0.74	0.628	0.631	0.505
	jedit4.3	0.576	0.584	0.546	0.506
	lucene2.4	0.675	0.551	0.69	0.588
	poi3.0	0.758	0.618	0.674	0.525
	synapse1.2	0.742	0.563	0.59	0.5
	xalan2.7	0.802	0.664	0.601	0.576
	xerces1.4	0.659	0.527	0.529	0.503
lucene2.4	ant1.7	0.785	0.767	0.744	0.634

	camel1.6	0.632	0.637	0.644	0.594
	ivy2.0	0.81	0.78	0.756	0.717
	jedit4.3	0.621	0.567	0.621	0.659
	log4j1.2	0.65	0.594	0.716	0.621
	poi3.0	0.831	0.766	0.776	0.701
	synapse1.2	0.755	0.728	0.715	0.636
	xalan2.7	0.822	0.902	0.836	0.632
	xerces1.4	0.724	0.703	0.818	0.71
poi3.0	ant1.7	0.797	0.802	0.781	0.713
	camel1.6	0.612	0.639	0.578	0.559
	ivy2.0	0.8	0.796	0.784	0.697
	jedit4.3	0.614	0.637	0.613	0.566
	log4j1.2	0.624	0.581	0.647	0.629
	lucene2.4	0.69	0.739	0.683	0.574
	synapse1.2	0.76	0.763	0.741	0.658
	xalan2.7	0.772	0.664	0.738	0.765
	xerces1.4	0.794	0.734	0.723	0.728
ivy2.0	ant1.7	0.818	0.791	0.81	0.5
	camel1.6	0.605	0.551	0.614	0.5
	jedit4.3	0.623	0.636	0.617	0.5
	log4j1.2	0.627	0.677	0.525	0.508
	lucene2.4	0.699	0.679	0.619	0.526
	poi3.0	0.841	0.741	0.765	0.511
	synapse1.2	0.753	0.742	0.727	0.506
	xalan2.7	0.792	0.842	0.706	0.504
	xerces1.4	0.753	0.516	0.663	0.516
synapse1.2	ant1.7	0.798	0.8	0.777	0.658
	camel1.6	0.615	0.608	0.602	0.521
	ivy2.0	0.797	0.818	0.777	0.762
	jedit4.3	0.61	0.603	0.597	0.619
	log4j1.2	0.622	0.547	0.534	0.54
	lucene2.4	0.726	0.691	0.672	0.534
	poi3.0	0.84	0.806	0.806	0.539
	xalan2.7	0.799	0.873	0.781	0.703
	xerces1.4	0.824	0.817	0.847	0.614
xalan2.7	ant1.7	0.743	0.76	0.791	0.52
	camel1.6	0.598	0.586	0.588	0.5
	ivy2.0	0.754	0.738	0.776	0.557
	jedit4.3	0.512	0.679	0.62	0.547

	log4j1.2	0.644	0.537	0.625	0.556
	lucene2.4	0.751	0.656	0.693	0.598
	poi3.0	0.811	0.716	0.772	0.506
	synapse1.2	0.754	0.689	0.732	0.535
	xerces1.4	0.629	0.537	0.769	0.673
xerces1.4	ant1.7	0.67	0.726	0.713	0.535
	camel1.6	0.576	0.589	0.607	0.545
	ivy2.0	0.678	0.619	0.723	0.528
	jedit4.3	0.514	0.529	0.682	0.531
	log4j1.2	0.554	0.53	0.514	0.542
	lucene2.4	0.674	0.629	0.669	0.503
	poi3.0	0.753	0.792	0.668	0.536
	synapse1.2	0.7	0.686	0.693	0.553
	xalan2.7	0.816	0.646	0.699	0.519

Table 5 AUC value of all classifiers

		F			
Training	Testing	NB	RF	MLP	LR
ant1.7	camel1.6	0.306	0.276	0.135	0.087
	ivy2.0	0.374	0.479	0.455	0.308
	jedit4.3	0.097	0.111	0.132	0.276
	log4j1.2	0.421	0.225	0.256	0.011
	lucene2.4	0.517	0.323	0.16	0.02
	poi3.0	0.556	0.287	0.042	0.014
	synapse1.2	0.573	0.537	0.566	0.186
	xalan2.7	0.42	0.366	0.115	0.107
	xerces1.4	0.372	0.31	0.309	0.009
camel1.6	ant1.7	0.517	0.207	0.144	0.086
	ivy2.0	0.307	0.326	0.344	0.347
	jedit4.3	0.078	0.15	0.085	0.138
	log4j1.2	0.612	0.225	0.234	0.48
	lucene2.4	0.505	0.217	0.34	0.362
	poi3.0	0.41	0.12	0.75	0.247
	synapse1.2	0.556	0.367	0.557	0.436
	xalan2.7	0.428	0.121	0.101	0.111
	xerces1.4	0.516	0.206	0.225	0.357
jedit4.3	ant1.7	0.463	NA	0.364	0.13
	camel1.6	0.3	NA	NA	0.051
	ivy2.0	0.337	NA	NA	NA

	log4j1.2	0.225	NA	0.959	NA
	lucene2.4	0.179	NA	0.748	NA
	poi3.0	0.214	NA	0.777	0.007
	synapse1.2	0.384	NA	0.504	0.023
	xalan2.7	0.184	NA	0.994	0.013
	xerces1.4	0.265	NA	0.853	NA
log4j1.2	ant1.7	0.466	0.363	0.364	0.367
	camel1.6	0.34	0.326	0.326	0.321
	ivy2.0	0.269	0.205	0.204	0.206
	jedit4.3	0.051	0	0.044	0.044
	lucene2.4	0.714	0.748	0.748	0.406
	poi3.0	0.807	NA	0.777	0.757
	synapse1.2	0.6	0.504	0.504	0.504
	xalan2.7	0.706	0.069	0.994	0.263
	xerces1.4	0.728	NA	NA	0.063
lucene2.4	ant1.7	0.498	0.443	0.418	0.43
	camel1.6	0.349	0.361	0.364	0.354
	ivy2.0	0.329	0.27	0.333	0.333
	jedit4.3	0.061	0.044	0.039	0.082
	log4j1.2	0.616	0.693	0.669	0.65
	poi3.0	0.839	0.75	0.801	0.714
	synapse1.2	0.606	0.576	0.566	0.555
	xalan2.7	0.645	0.802	0.785	0.418
	xerces1.4	0.692	0.78	0.611	0.63
poi3.0	ant1.7	0.498	0.493	0.55	0.508
	camel1.6	0.339	0.364	0.321	0.317
	ivy2.0	0.296	0.254	0.315	0.308
	jedit4.3	0.056	0.047	0.053	0.054
	log4j1.2	0.713	0.723	0.687	0.664
	lucene2.4	0.693	0.733	0.418	0.574
	synapse1.2	0.585	0.55	0.563	0.591
	xalan2.7	0.664	0.718	0.744	0.766
	xerces1.4	0.638	0.69	0.545	0.672
ivy2.0	ant1.7	0.558	0.124	0.012	NA
	camel1.6	0.24	0.011	0.107	NA
	jedit4.3	0.101	0.133	0	NA
	log4j1.2	0.346	NA	NA	0.031
	lucene2.4	0.373	0.039	0.01	0.111
	poi3.0	0.374	0.014	0.007	0.042

	synapse1.2	0.536	0.129	0.023	0.023
	xalan2.7	0.32	0.013	0.009	0.015
	xerces1.4	0.298	0.068	0.037	0.064
synapse1.2	ant1.7	0.525	0.503	0.522	0.47
	camel1.6	0.29	0.246	0.214	0.197
	ivy2.0	0.368	0.444	0.314	0.474
	jedit4.3	0.074	0.077	0.061	0.067
	log4j1.2	0.506	0.428	0.519	0.147
	lucene2.4	0.634	0.42	0.556	0.283
	poi3.0	0.674	0.208	0.427	0.185
	xalan2.7	0.54	0.42	0.355	0.663
	xerces1.4	0.557	0.452	0.63	0.38
xalan2.7	ant1.7	0.433	0.364	0.364	0.374
	camel1.6	0.342	0.326	0.326	0.323
	ivy2.0	0.244	0.204	0.204	0.224
	jedit4.3	0.04	0.044	0.044	0.048
	log4j1.2	0.855	0.959	0.959	0.296
	lucene2.4	0.776	0.748	0.748	0.698
	poi3.0	0.816	0.777	0.777	0.288
	synapse1.2	0.547	0.504	0.504	0.491
	xerces1.4	0.805	0.853	0.853	0.843
xerces1.4	ant1.7	0.41	0.364	0.369	0.381
	camel1.6	0.363	0.34	0.34	0.346
	ivy2.0	0.238	0.205	0.211	0.213
	jedit4.3	0.037	0.044	0.045	0.047
	log4j1.2	0.837	0.948	0.948	0.156
	lucene2.4	0.756	0.747	0.745	0.736
	poi3.0	0.818	0.783	0.777	0.783
	synapse1.2	0.551	0.51	0.517	0.533
	xalan2.7	0.865	0.956	0.927	0.968

Table 6 F-measure of all classifiers

Friedman Test analysis

According to the test performed on the AUC and F values it is shown that naïve bayes result are asymptotically significant when compared with all other classifier with p-value as less than 0.01.Hence, we reject null hypothesis and conclude that Naïve bayes performed better than other classifier when applied on transformed data.

CHAPTER-6

CONCLUSION AND FUTURE WORK

CPDP is one of the most challenging areas. Here, we use different training and testing set having software metrics as independent variable. These metrices exhibit power law distribution. Data normality is one of the main causes behind inaccurate predictions. Many transformations (log, rank and boxcox) have been studied but these transformations have some inappropriate behavior to values close to zero so we first need to shift our data. To remove this problem we have studied yeo-johnoson transformation. This transformation can handle zero and negative data also. Here in this thesis, we investigated the effect of this transformation in CPDP. To generalize our study we also explored whether this transformation work well for other classifier or not.

In this study we have seen that yeojohnson transformed data provide asymptotically better result than the without transformed data. We have used logistic regression for this experiment and by using Firedman's test on AUC and F-measure we have statistically validated our results. We further explored our studied with other classifiers such as random forest, naïve bayes and multilayer perceptron to generalize our result. Here naïve bayes out perform well than other classifier. We also validate our result with Friedman test. A pair-wise comparison with wilcoxon signed rank test and it proves that naïve bayes outperform significantly with AUC and F-measure.

For Future work, we will compare the other transformation like log, rank and boxcox with this transformation. We recommend future studies to experiment with this transformation for data normalization for potential gains in cross project defect prediction. We are also interested to apply more advanced ensemble learners with our approach.

REFRENCES

[1] G. Concas , M. Marchesi , S. Pinna, N. Serra, “Power-laws in a large object-oriented software system”, *IEEE Trans Softw Eng*, pp. 687–708, 2007.

[2] P. Louridas, D. Spinellis and V. Vlachos, “Power laws in software”, *ACM Trans Softw Eng Methodology*, pp. 1-26, 2008.

[3] H. Zhang , “Discovering power laws in computer programs”, *Inforcessing Process Manag*, pp. 477–483, 2009.

[4] J. Cohen, P. Cohen, S. West, L. Aiken, “Applied multiple Regression/Correlation analysis for the behavioral sciences”, *3rd edition. Lawrence Erlbaum*, Mahwah, NY, USA, 2003.

[5] S. Hosseini, B. Turhan, D. Gunarathna, “A systematic literature review and meta analysis on cross project defect prediction” , *IEEE trans. of soft Eng*, 2016.

[6] T. Menzies, R. Krishna and D. Pryor, “The promise repository of empirical software engineering data”, Dept Computer science, North Carolina State University, 2015, http://openscience.us/repo.

[7] RStudio Team, “RStudio: Integrated Development for R. RStudio”, Inc.,Boston, MA URL, 2016, http://www.rstudio.com/.

[8] F. Zhang , I. Keivanloo, Y. Zou, “Data transformation in cross project defect prediction”, in *Empir Soft Eng*, 2017.

[9] F. Zhang, A. Mockus, Y. Zou, F. Khomh, A.E. Hassan, “How does context affect the distribution of software maintainability metrics?” , *Proceedings of the 29th IEEE international conference on software maintainability, ICSM ’13*, pp 350–359, 2013.

[10] F. Zhang , A. Mockus, I. Keivanloo, Y. Zou, “Towards building a universal defect prediction model”, *Proceedings of the 11th working conference on mining software repositories, MSR ’14*, pp 41–50, 2014.

[11] F. Zhang, A. Mockus, I. Keivanloo, Y. Zou, “Towards building a universal defect prediction model with rank transformed predictors” , *Empir Soft Eng*, pp 1–39, 2015.

[12] F. Zhang, A. Mockus, Y. Zou, F. Khomh, A.E. Hassan, “Cross-project defect prediction using a connectivity-based unsupervised classifier” , *Proceedings of the 38th international conference on software engineering, ICSE ’16*, pp 309–320, 2016.

[13] R. Malhotra and A. Jain, "Fault Prediction Using Statistical and machine learning Methods for Improving Software Quality," *Journal of Information Processing Systems*, vol. 8, no. 2, pp. 241-262, 2012.

[14] M. Tan, L. Tan, S. Dara and C. Mayeux, “Online Defect Prediction for Imbalanced Data”, *IEEE/ACM 37th IEEE International Conference on Software Engineering*, 2015.

[15] C. Catal, B. Diri, “Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem”, *Information Sciences 179*, 1040–1058, 2009.

[16] C. Catal, “Software fault prediction: a literature review and current trends”, *Expert Syst. Appl. 38*, 4626–4636, 2011.

[17] I. Gondra, "Applying machine learning to software fault-proneness prediction," *The Journal of Systems and Software*, vol. 81, pp. 186-195, 2008.

[18] T. Menzies, A. Dekhtyar, J. Distefance, J. Greenwald, “Problems with precision:a response to comments on ‘data mining static code attributes to learn defectpredictors”, *IEEE Trans. Softw. Eng*. 33 637–640, 2007.

[19] R. Malhotra, “A systematic review of machine learning techniques for software fault prediction”, *Applied Soft Computing* 27, 504-518, 2015.

[20] A. Shanthini and R. M. Chandrasekaran, “Applying machine learning for Fault Prediction Using Software Metrics”, *International Journal of Advanced Research in Computer Science and Software Engineering,* Volume 2, Issue 6, June 2012.

[21] Y. Singh, R. Malhotra, A. Kaur, “Empirical validation of object-oriented metrics for predicting fault proneness at different severity levels using support vector machines”, *Int. Journal of System Assur. Eng. Management*, 1(3):269–281, July-Sept, 2010.

[22] S. Lessmann, B. Baesans, C. Mues, S. Pietsch, “Benchmarking classification models for software defect prediction: a proposed framework and novel finding,” *IEEE Trans on Softw Eng,*Vol 34, 485–496, july/august 2008.

[23] M. Jureczko and L. Madeyski, “Towards identifying software project clusters with regard to defect prediction”, *Proc. 6th Int. Conf. Predictive Models Software Engineering*, pp. 9, 2010.

[24] Yeo, In-Kwon, Johnson, R.A, “A new family of power transformations to improve normality or symmetry” , *Biometrika 87 (4)*, 954–959, 2000.

[25] L. Komsta and F. Novomestky, “moments: Moments, cumulants, skewness, kurtosis and related tests”, R package version 0.14, 2015. https://CRAN.R-project.org/package=moments

[26] C. Catal and B. Diri, "A systematic review of software fault prediction studies*," Expert Systems with Applications*, vol. 36, pp. 7346-7354, 2009.

[27] D. Meyer, E. Dimitriadou, K. Hornik, A. Weingessel and F. Leisch, “e1071: Misc Functions of the Department of Statistics, Probability Theory Group (Formerly: E1071)”, TU Wien. R package version 1.6-8, 2017. https://CRAN.R-project.org/package=e1071

[28] R.A. Peterson, “Estimating normalization transformations with bestNormalize”, 2017. URL https://github.com/petersonR/bestNormalize

[29] J. Han, M. Kamber, J. Pei, “Data Mining: concepts and techniques”, *3rd edn. Morgan Kaufmann*, Boston, 2012.

[30] J. Nam , S. Kim, “Heterogeneous defect prediction”, *Proceedings of the 2015 10th joint meeting on foundations of software engineering, ACM*, New York, NY, USA, ESEC/FSE, pp 508–519, 2015.

[31] M. Kuhn, K. Johnson, “Data pre-processing” , *In the Applied predictive modeling. Springer*, New York, pp 27–59, 2013.

[32] A.J. Bishara, J.B. Hittner, “Reducing bias and error in the correlation coefficient due to non normality”, *Educational and Psychological Measurement*, 2014. http://epm.sagepub.com/content/early/2014/11/10/0013164414557639.full.pdf+html

[33] Y. Jiang, B. Cukic, T. Menzies, “Can data transformation help in the detection of fault-prone modules?”, *Proceedings of the 2008 workshop on defects in large software systems, DEFECTS ’08*, pp 16–20, 2008.

[34] T. Menzies , J. Greenwald , A. Frank, “Data mining static code attributes to learn defect predictors” , *IEEE Trans Softw Eng (TSE) 33(1):2–13*, 2012.

[35] Q. Song, Z. Jia, M. Shepperd, S. Ying, J. Liu, “A general software defect-proneness prediction framework”, *IEEE Trans Softw Eng 37(3):356–370*, 2011.

[36] Z. He, F. Peters, T. Menzies, Y. Yang, “Learning from open-source projects: an empirical study on defect prediction.” , *ACM/IEEE international symposium on empirical software engineering and measurement*, pp 45–54, 2013.

[37] T. Zimmermann, N. Nagappan, H. Gall, E. Giger, B. Murphy, “Cross-project defect prediction: a large scale experiment on data vs. domain vs. process”, *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on the foundations of software engineering, ESEC/FSE ’09*, pp 91–10, 2009.

[38] T. Menzies, A. Butcher, D. Cok, A. Marcus, L. Layman, F. Shull, B. Turhan, T. Zimmermann, “Local versus global lessons for defect prediction and effort estimation.”, I*EEE Trans Softw Eng 39(6):822–834*, 2013.

[39] B. Turhan, A.T. Misirli, A.B. Bener, “Empirical evaluation of the effects of mixed project data on learning defect predictors”, *Inf Softw Technol 55(6):1101–1118*, 2013.

[40] Y. Ma, G. Luo, X. Zeng, A. Chen, “Transfer learning for cross-company software defect prediction”, *Inf Softw Technol 54(3):248–256*, 2012.