J Inno Sci Eng, 2020, 4(1): 11-21
https://doi.org/10.38088/jise.693098

Research Article

A Comparison of Software Defect Prediction Metrics Using Data Mining Algorithms

Zeynep Behrin GUVEN AYDIN [1,2*], Ruya SAMLI[2]

[1*] *Department of Software Engineering, Maltepe University, Istanbul, Turkey*
[2] *Department of Computer Engineering, Istanbul University-Cerrahpasa, Avcilar, Istanbul, Turkey*

Abstract

Data mining is an interdisciplinary field that uses methods such as machine learning, artificial intelligence, statistics, and deep learning. Classification is an important data mining technique as it is widely used by researchers. Generally, statistical methods or machine learning algorithms such as Decision Trees, Fuzzy Logic, Genetic Programming, Random Forest, Artificial Neural Networks and Logistic Regression have been used in software defect prediction in the literature. Performance measures such as Accuracy, Precision, Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) are used to examine the performance of these classifiers. In this paper, 4 data sets entitled JM1, KC1, CM1, PC1 in the PROMISE repository, which are created within the scope of the publicly available NASA institution's Metric Data Program, are examined as in the other software defect prediction studies in the literature. These datasets include Halstead, McCabe method-level, and some other class-level metrics. Data sets are used with Wakiato Environment for Knowledge Analysis (WEKA) data mining software tool. By this tool, some classification algorithms such as Naive Bayes, SMO, K *, AdaBoost1, J48 and Random Forest were applied on NASA error datasets in PROMISE repository and their accuracy rates were compared. The best value among the accuracy rates was obtained in the Bagging algorithm in the PC1 data set with the values of %94.13.

Keywords: Software Defect Prediction, McCabe, Halstead, Data Mining, Accuracy, Random Forest

Cite this paper as:
GÜVEN AYDIN, Z.B., SAMLI, R. (2020). *A Comparison of Software Defect Prediction Metrics Using Data Mining Algorithms*. Journal of Innovative Science and Engineering. 4(1): 11-21

*Corresponding author: Zeynep Behrin GÜVEN AYDIN
E-mail: zeynepguven@maltepe.edu.tr

Received Date: 24/02/2020
Accepted Date: 05/05/2020

1. Introduction

Nowadays, software is very important for everyone who uses computer systems. For this reason, software errors cause large or serious problems. With the development of technology in recent years, a lot of data is produced in every field. Defects in software are usually caused by source code. Some of this data needs to be cleaned daily. Defect prediction is a rapidly growing subfield of data mining.

Some coding standards have been established in order to develop software with better quality and reliability. Software quality depends on various factors such as accuracy, availability, maintenanceability, testability and so on. A software is subjected to various tests throughout the development process. Errors detected are eliminated and the reliability of the software is increased. In every phase of software, an error can occur. However, detecting errors in the earlier stages of testing will reduce the test costs. For a software developer, defect prediction is a very important process to provide the quality and reliability of a software [1]. Data mining techniques and machine learning algorithms are useful in prediction of software defects. Public software defect prediction dataset repositories are increasing day by day. One of them is the PROMISE dataset, owned by NASA, which conducts space exploration.

So in this paper a software defect prediction study was presented. The aim of this paper is to evaluate and compare the performance of the above mentioned classification algorithms and perform a comparative study with other research works. The implementation of these algorithms is carried out in WEKA data mining tool on JM1, CM1, KC1 and PC1 data sets freely available from the PROMISE data repository.

The rest of the paper is organized as follows: In the second section, the studies on defect prediction using machine learning methods are shown. In the third section metric sets and data sets are given in detail. The results and discussion can be found in Section 4 and Section 5 provides a conclusion.

2. Related Works

In this section, the literature studies about software defect predictions by data mining methods were summarized in table. As seen from the table, there are many different algorithms such as Support Vector Machine, Logistic Regression, Multilayer Perceptron, Radial Basis Function, Naive Bayes, Bayes Belief Network, Random Forest, Decision Tree, Logistic Model Trees, Linear Discriminant Analysis, Quadratic Discriminant Analysis, Least Angle Regression, Classification and Regression Tree, Alternating Decision Tree, Augmented Naive Bayes, Artificial Neural Networks and Discriminant Analysis were used for defect prediction. Also, it can be easily seen that there is not a definite “best algorithm” in this process and the algorithm which gives the best results change according to the dataset. In table-1 SVM is Support Vector Machine, LR is Logistic Regression, MP is Multilayer Perceptron, RBF is Radial Basis Function, NB is Naive Bayes, BBN is Bayes Belief Network, RF is Random Forest, DT is Decision Tree, LMT is Logistic Model Trees, LDA is Linear Discriminant Analysis, QDA is Quadratic Discriminant Analysis, LARS is Least Angle Regression, CART is Classification and Regression Tree (CART), ADT is Alternating Decision Tree, ANB is Augmented Naive Bayes, ANN is Artificial Neural Networks, and DA is Discriminant Analysis.

Table 1. Related Works

Reference	Data Sets	Data Mining Algorithms	Best
[2]	NASA	J48, NB	NB
[3]	NASA	SVM, LR,k-NN, MP, RBF, NB, BBN, RF, DT	SVM
[4]	NASA	RF, LMT, LDA, QDA, NB, Bayes Net, LARS, k-NN, ANN, SVM, C4.5, CART, ADT	RF
[5]	NASA, Eclipse	NB, LR, RF, ANB	RF
[6]	Open source web and Mozilla e-mail suite	Logistic Regression, LR, DT, ANN	LR
[7]	NASA	Multi-Layered Perceptron, Bayesian Network, NB, ANN	NN
[8]	NASA	J48 and K*	J48
[9]	NASA	LR, DA, Classification Tree, Boosting, Kernel Density, NB, J48, IBk, Voted Perceptron, VF1, Hyperpipes, ROCKY, RF	RF
[10]	NASA	NB, DT	NB
[11]	Telecommunication system,Eclipse project, NASA	NB, K-NN, SVM, LR	SVM
[12]	Eclipse project	DT, K-NN, RF	K-NN and J-48

Predicting the fault-prone software modules is of a great interest among the software quality researchers and industry professionals [13]. As a result of this, various efforts have been made for software defect prediction using methods such as Desicion Trees [3], Artificial Neural Networks [4], Support Vector Machines [5], Bayesian Methods [6], Naive Bayes [7], Fuzzy Logic [8, 9], Dempster–Shafer Belief Networks [10], Genetic Programming [11], Casebased Reasoning [12, 13], and Logistic Regression [14, 15]. In the literature, there are many classification techniques, some of the most commonly used ID3, C4.5, logistic regression, linear and quadratic discriminant analysis, k-nearest neighbor, ANN and SVM [16].

In the literature, generally the fault concept is investigated. A fault is a defect in source code that causes failures when executed [17]. Data mining techniques and machine learning algorithms are useful in prediction of software defects. Data mining technique comprises of classification, regression, clustering and association [18]. It has been seen in the literature that there are many studies on data mining methods and software error estimation. In these studies, Halstead and McCabe software metrics and NASA datasets were used, and more machine learning methods were successful in predicting errors. Table 1 shows which data sets are used in the literature, which data mining algorithms are preferred and which algorithm has the most successful results.

3. Material and Methods

There are many studies in the literature with WEKA. The study was conducted on these data using clustering algorithms presented in WEKA. In this study the version of WEKA is used WEKA 3.8 [19]. WEKA software can calculate such as mean absolute error, root mean squares mean, relative absolute error and root relative squared error. This section consists of two main parts: (i) data sets and (ii) confusion matrix

3.1. Data Sets

In this study, four data sets entitled JM1, PC1, KC1, CM1 belonging to NASA in the PROMISE database were studied. The data sets contain information from some NASA softwares. These datasets have an .arff extension so the file can be easily used in WEKA. Table 2 gives the names of the data sets, the number of properties they have, the number of registers, the programming language developed, and the content of the data set.

Table 2. Data Sets Properties

Data Set	Number of Feature	Number of Record	Programming Language	Data Sets Content
JM1	22	10885	C	Ground System
KC1	22	2109	C++	Location Warehouse Management
CM1	38	498	C	Spacecraft Tools
PC1	22	1109	C	Earth orbit in flight

The data sets include metric measurement values and variables. Each of the records in the dataset has a class label, which means that there is either a reported or not reported error of the software module to which it is connected [20]. The Table 3 describes the properties of the McCabe and Halstead metrics used in the data sets. These data sets are primarily located in the PROMISE data warehouse, with the .arff extension, consisting of some metrics such as the complexity of the software from the software included in the datasets, the number of lines of code, basic and derived measurements. Detailed information about the definitions and calculations of McCabe and Halstead Metrics is as follows.

Table 3. Metrics Properties

Attribute	**Explanation**
loc	McCabe “line count of code”
v(g)	McCabe "cyclomatic complexity"
ev(g)	McCabe "essential complexity"
iv(g)	McCabe "design complexity"
n	Halstead total operators + operands
v	Halstead "volume"
l	Halstead "program length"
d	Halstead "difficulty"
i	Halstead "intelligence"
e	Halstead "effort"
b	Halstead
t	Halstead time estimator
lOCode	Halstead line count
lOComment	Halstead count of lines of comments
lOBlank	Halstead count of blank lines
lOCodeAndComent	
uniq_Op	unique operators
uniq_Opnd	unique operands
total_Op	total operators
total_Opnd	total operands
branchCount	of the flow graph
defects	{false,true} module has/has not one or more reported defects

McCabe metrics include four software metrics, consisting of essential complexity, cyclomatic complexity, design complexity, and Lines of Code (LOC) . Cyclomatic Complexity or "v (G)" measures the number of "linearly independent paths". If no path in the cluster is a linear combination of other paths in the cluster via a program's "flowgraph", a path set is said to be linear independent. The flowchart is a directed graph in which each node corresponds to a program expression, and each arc shows the control flow from one expression to another.

"v (G)" is calculated by $v(G) = e - n + 2$ where "G" is the flow chart of a program, "e" is the number of springs in the flow chart, and "n" is the nodes in the flow chart. Standard McCabes rules ("v (G)"> 10) are used to define the error-prone module.

Essential Complexity or "ev(G)" is a measure of a flow chart that can be "reduced" by parsing all G subflowgraphs that are "D structured primers". Such "D-structured primers" are sometimes referred to as "single inlet single outlet downstream diagrams". "ev (G)" is calculated using $ev(G) = v(G) - m$; where m is the number of sub flowgraphs of "G", which are D-structured primers.

Design Complexity or "iv (G)" is the cyclomatic complexity of a module's reduced flow chart. The "G" flowgraph of a module is reduced to eliminate the complexity that does not affect the relationship between the design modules. According to McCabe, this complexity measurement reflects the modules that immediately call models to their dependent modules.
Lines of code is measured according to McCabe's line count rules. Halstead metrics are divided into three groups: basic measures, derived measures and lines of code measures [21].

Table 4. Metric Details

Basic Measures		Derived Measures	
mu1	number of unique operators	$P = volume = V = N * log2(mu)$	the number of mental comparisons needed to write a program of length N
mu2	number of unique operands	$V^* = (2 + mu2') * log2(2 + mu2')$	volume on minimal implementation
N1	total occurrences of operators	$L = V^*/N$	program length
N2	total occurrences of operands	$D = 1/L$	difficulty
length	$N = N1 + N2$	$L' = 1/D$	
vocabulary	$mu = mu1 + mu2$	$I = L' * V'$	intelligence
mu1' = 2	potential operator count (just the function name and the "return" operator)	$E = V/L$	effort to write program
mu2'	potential operand count. (the number of arguments to the module)	$T = \frac{E}{18} second$	time to write program

3.2. Confusion Matrix

A confusion matrix is a powerful analysis tool that allows to visualize estimated values against actual values. In the machine learning context, a confusion matrix is a metric used to quantify the performance of a machine learning classifier. In addition to this confusion matrices are useful because they give direct comparisons of values like True Positives, False Positives, True Negatives and False Negatives. The Confusion matrix is the state of a data set and the number of true and false estimates of our classification model, transformed into a table. The general form of the confusion matrix is given in Table 5.

Table 5. General Confusion Matrix

Actual Class	**Predicted Class**	
	Positives	Negatives
Positives	TP (True Positives)	FN (False Negatives)
Negatives	FP (False Positives)	TN (True Negatives)

TP (True Positives): It shows examples that are actually positive and classified as positive.
FP (False Positives): It shows examples that are actually negative and classified as positive.
TN (True Negatives): It shows examples that are actually negative and classified as negative.
FN (False Negatives): It shows examples that are actually positive and classified as negative.
The following performance measures can be calculated from the confusion matrix.

Accuracy:

It is the ratio of the number of correctly classified samples to the number of samples.

$$Accuracy = \frac{TP+TN}{TP + FP + FN + TN} \quad (1)$$

Precision:

The ratio of the number of positive samples correctly classified to the number of positive classified samples.

$$Presicion = \frac{TP}{TP+FP} \quad (2)$$

Mean Absolute Error (MAE):

It calculates absolute average error differences between predicted values and actual values.

$$MAE = \frac{\sum_{i=1}^{N} |x_i - x|}{N} \quad (3)$$

Root Mean Squared Error:

It calculates by the square root of the sum of squares of the difference between the estimated and actual values.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{N} |x_i - x|^2}{N}} \quad (4)$$

where $x_i \quad (i = 1,2, \dots, N)$ is the real output, N is the number of outputs.

4. Results and Discussion

In this study, on NASA's 4 different data sets (JM, PC1, KC1, CM1), WEKA data mining software tool with various algorithms were implemented with a 10-fold validation rule (Figure 1). Table 6 shows the accuracy rates of the algorithms applied for 4 data sets. The data sets used were analyzed by classification method and the algorithms were implemented in the open source data mining software tool WEKA, a 10-fold cross validation rule.

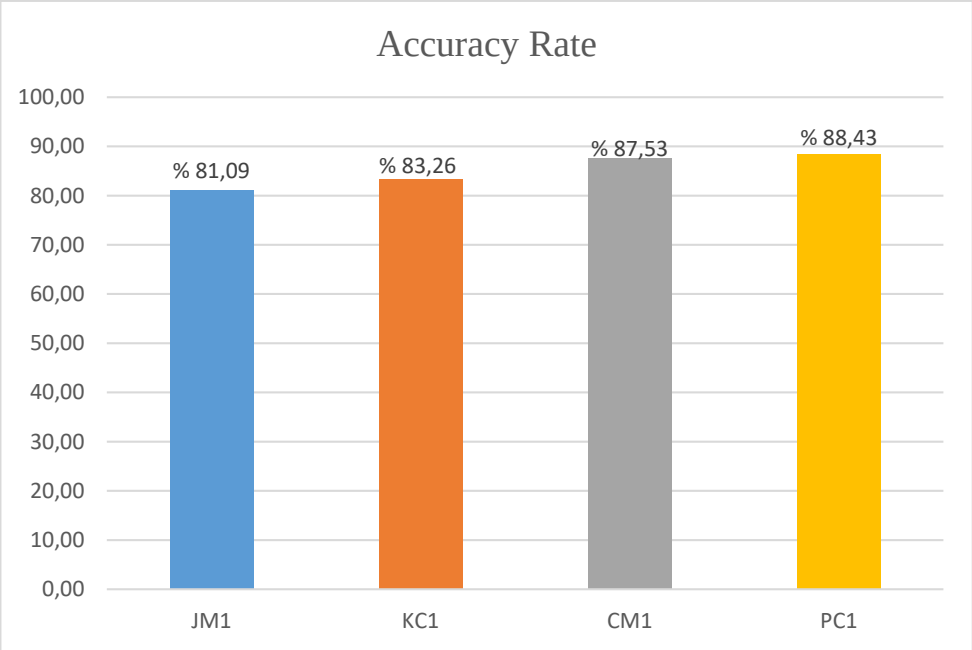

Figure 1. Average Accuracy Rate Graph showing comparison between accuracy values of Table 6

Table 6. Accuracy Results of All Algorithms

ALGORITHMS	*JM1 (%)*	*KC1 (%)*	*CM1 (%)*	*PC1 (%)*
BAYES				
Bayes Net	68.05	76.42	64.15	74.39
Naive Bayes	80.42	80.75	85.34	89.17
Naive Bayes Multinomial		83.68	70.68	90.53
Naive Bayes Multinominal Text	80.65	83.12	90.16	93.05
Naive Bayes Multinomial Updateable		83.68	70.68	90.53
Naive Bayes Updatable	80.42	80.75	85.34	89.17
FUNCTIONS				
Logistic	81.35	84.65	88.35	92.42
Multilayer Perceptron	80.95	84.10	87.55	93.59
SGD	70.77	84.10	89.55	93.05
SGDText	80.65	83.12	90.16	93.05
Simple Logistic	81.12	84.65	89.15	92.60
SMO	80.75	83.26	89.15	92.96
Voted Perceptron	52.21	83.12	90.16	92.60
LAZY				
lBk	76.97	84.10	84.73	92.06
K*	78.56	82.14	87.14	91.79
LWL	80.65	83.12	89.75	93.23
META				
Iterative Classifier Optimizer	80.89	84.37	89.15	93.05
Adaboost1	80.79	83.54	90.16	93.05
Attribute Selected Classifier	80.86	83.12	89.35	93.41
Bagging	81.19	84.10	89.75	94.13
Classification via Regression	81.24	84.93	89.35	93.14
CVParameter Selection	80.65	83.12	90.16	93.05
Filtered Classsifier	81.12	83.40	90.16	93.50
Logi Boost	80.89	84.51	88.95	93.14
Multi Class Classifier	81.35	84.65	88.35	92.42
Multi Class Classifier Updatable	80.77	84.10	89.55	93.05
Multi Scheme	80.65	83.12	90.16	93.05
Random Comittee	81.03	84.79	87.75	93.59
Randomizable Filtered Classifer	75.26	82.56	85.54	89.45
Random Sub Space	81.79	84.10	90.16	93.86
Stacking	80.65	83.12	90.16	93.05
Vote	80.65	83.12	90.16	93.05
Weighted Instances Handler Wrapper	80.65	83.12	90.16	93.05
MISC				
Input Mapped Classifier	80.65	83.12	90.16	93.05
RULES				
Desicion Table	80.90	82.98	89.15	92.87
JRip	81.04	83.26	89.35	93.32
OneR	79.39	82.14	88.35	92.87
Part	80.74	83.68	88.75	93.68
ZeroR	80.65	83.12	90.16	93.05
TREE				
Desicion Stump	80.65	83.12	90.16	93.05
Hoeffding Tree	80.71	83.12	90.16	93.05
J48	79.50	82.98	87.95	93.32
LMT	81.24	84.65	89.15	92.42
Random Forest	81.75	85.07	88.75	93.68
Random Tree	75.47	80.61	84.33	91.07
REP Tree	80.67	83.96	89.15	93.59
AVERAGE(%)	*81.09*	*83.26*	*87.53*	*88.43*

Tables 7 – 10 show the confusion matrices of the most accurate algorithm for each data set which are Random Sub Space Algorithm for JM1 DataSet, Random Forest Algorithm for KC1 DataSet, Naive Bayes Multinominal Text, SGDText, Voted Perceptron, Adaboost1, CVParameter Selection, Filtered Classsifier, Multi Scheme, Random Sub Space, Stacking, Vote, Weighted Instances Handler Wrapper, Input Mapped Classifier, ZeroR, Desicion Stump and Hoeffding Tree for CM1 DataSet, Bagging Algorithm for PC1 DataSet.

Table 7. JM1 DataSet- Random Sub Space Algorithm Confusion Matrix

Random SubSpace	TP	FP
False	0,978	0,864
True	0,136	0,022

Table 8. KC1 DataSet- Random Forest Algorithm Confusion Matrix

Random Forest	TP	FP
False	0,965	0,672
True	0,328	0,035

Table 9. CM1 DataSet- Algorithm Confusion Matrix

	TP	FP
False	1,000	1,000
True	0,000	0,000

Table 10. PC1 DataSet- Bagging Algorithm Confusion Matrix

Bagging	TP	FP
False	0,993	0,753
True	0,247	0,007

5. Conclusion

The performances of the all algorithms are evaluated by using 4 datasets from NASA projects in terms of accuracy rate. In our study, average accuracy was calculated for each data set. In terms of general accuracy rates, the most successful accuracy rates were taken in PC1 data set compared to other data sets. The highest rate of 81.79% in the JM1 dataset was obtained with the Random Sub Space algorithm. Random Forest with 85.07% in the KC1 dataset, CM 1 data set Naive Bayes Multinominal Text, SGDText, Voted Perceptron, Adaboost1, CVParameter Selection, Filtered Classsifier, Multi Scheme, Random Sub Space, Stacking, Vote, Weighted Instances Handler Wrapper, Input Mapped Classifier, ZeroR, Desicion Stump and Hoeffding Tree Algorithms of 90.16% accuracy rate was obtained and Bagging algorithm with 94.13% accuracy rate in PC1 data set. When the results are analyzed in terms of accuracy of machine learning algorithms, it is seen that the majority of the algorithms have very appropriate values. The accuracy rates of the algorithms are quite high in 4 datasets. In some algorithms, the accuracy rates differ greatly compared to other algorithms, and these algorithms appear to be successful in accuracy rate calculations, as in the literature.

In future work, better results can be obtained from previous studies by replicating more object-oriented metric sets and different data sets. In future, we want to replicate our study with more object oriented metrics and different data sets so that generalized observations and conclusions can be made. We also intend to apply different machine learning algorithms such as slow learner genetic algorithm to further assess the accuracy of machine learning techniques.

Acknowledgement

This research work was supported by The Scientific and Technological Research Council of Turkey (TÜBİTAK), Project Number: 118E682. Also, we are thankful to the PROMISE software engineering repository for providing free and easy access to the NASA defect data sets for use in our research.

References

[1] Gayatri, M. and Sudha, A. (2014). Software Defect Prediction System using Multilayer Perceptron Neural Network with Data Mining. International Journal of Recent Technology and Engineering (IJRTE), 3(2): 54-59.

[2] Menzies, T., Greenwald, J., and Frank, A. (2006). Data Mining Static Code Attributes to Learn Defect Predictors. IEEE Transactions on Software Engineering, 33(1): 2-13.

[3] Elish, K.O. and Elish, M.O. (2008). Predicting Defect-Prone Software Modules Using Support Vector Machines, Journal of Systems and Software, 81: 649-660.

[4] Lessmann, S., Baesens, B., Mues, C., and Pietsch, S. (2008). Benchmarking Classification Models for Soft ware Defect Prediction: A Proposed Framework and Novel Findings. IEEE Transactions on Software Engineering, 34(4): 485-496.

[5] Moeyersoms, J., de Fortuny, E. J., Dejaeger, K., Baesens, B., and Martens, D. (2015). Comprehensible Software Fault and Effort Prediction: A Data Mining Approach. Journal of Systems and Software, 100: 80-90.

[6] Gyimothy, T., Ferenc, R., and Siket, I. (2005). Empirical validation of object-oriented metrics on open source software for fault prediction. IEEE Transactions on Software engineering, 31(10): 897-910.

[7] Dhankhar, S., Rastogi, H., and Kakkar, M. (2015) Software fault prediction performance in software engineering, 2nd International Conference on Computing for Sustainable Global Development (INDIACom), New Delhi, 11-13 March 2015, pp. 228-232.

[8] Koru, A. G. and Liu, H. (2005). Building effective defect-prediction models in practice. IEEE software, 22(6): 23-29.

[9] Ma, Y., Guo, L., and Cukic, B. (2007). A Statistical Framework for the Prediction of Fault-Proneness. In Advances in Machine Learning Applications in Software Engineering IGI Global, 237-263.

[10] Wang, T. and Li, W. (2010). Naive Bayes Software Defect Prediction Model, 2010 International Conference on Computational Intelligence and Software Engineering, Wuhan, pp. 1-4.

[11] Wang, H., Khoshgoftaar, T. M., and Napolitano, A. (2011). An Empirical Study of Software Metrics Selection Using Support Vector Machine. In SEKE July, pp. 83-88.

[12] Choudhary, G. R., Kumar, S., Kumar, K., Mishra, A., and Catal, C. (2018). Empirical Analysis of Change Metrics for Software Fault Prediction. Computers & Electrical Engineering, 67: 15-24.

[13] Pandey, A. K. and Goyal, N. K. (2010). Predicting Fault-Prone Software Module Using Data Mining Technique and Fuzzy Logic. International Journal of Computer and Communication Technology, 2(2):56-63.

[14] Khoshgoftaar, T.M. and Seliya, N. (2002). Software Quality Classification Modeling Using the SPRINT Decision Tree Algorithm, In the proceedings of the 4th IEEE International Conference on Tools with Artificial Intelligence, Washington, DC, pp. 365-374.

[15] Thwin, M.M. and Quah, T. (2003). Application of Neural Networks for Software Quality Prediction Using Object-Oriented Metrics, In the proceedings of the 19th International Conference on Software Maintenance, Amsterdam, The Netherlands, pp. 113-122.

[16] Elish K.O. and Elish M.O. (2008). Predicting defect-prone software modules using support vector machines, Journal of Systems and Software, 81:649-660.

[17] Pai, G.J. and Dugan, J.B. (2007). Empirical Analysis of Software Fault Content and Fault Proneness Using Bayesian Methods, IEEE Transactions on Software Engineering, 33: 675-686.

[18] Yu, Menzies, T., Greenwald, J., and Frank, A. (2007). Data Mining Static Code Attributes to Learn Defect Predictors, IEEE Transactions on Software Engineering, 33: 2-13.

[19] https://machinelearningmastery.com/what-is-the-weka-machine-learning-workbench/ Accessed: 29 January 2020

[20] Chaudhary, N., Mehta, G., and Bajaj, K. (2015). Comparison Of Classification Algorithms And Design Of A Percentage-Split Based Method For Data Classification, IJCSIT, 2(5):1-6.

[21] Aydilek, İ . (2018). Yazılım Hata Tahmininde Kullanılan Metriklerin Karar Ağaçlarındaki Bilgi Kazançlarının İncelenmesi ve İyileştirilmesi. Pamukkale Üniversitesi Mühendislik Bilimleri Dergisi, 24(5):906-914.

[22] http://promise.site.uottawa.ca/SERepository/datasets-page.html/Accessed:09.04.2020