Research Journal of Applied Sciences, Engineering and Technology 11(1): 63-69, 2015
DOI: 10.19026/rjaset.11.1676
ISSN: 2040-7459; e-ISSN: 2040-7467

Submitted: February 6, 2015 Accepted: March 1, 2015 Published: September 05, 2015

Research Article

Investigation of Software Defect Prediction Using Data Mining Framework

[1]M. Anbu and [2]G.S. Anandha Mala
[1]Department of Information Technology, St. Joseph's Engineering College,
[2]Department of Computer Science and Engineering, Easwari Engineering College, Chennai, India

Abstract: A software defect is a error, failure, fault in a computer program or system producing an incorrect or unexpected result, or causing it to behave in an unintended way. Software Defect Prediction (SDP) locates defective modules in software. The final product should have null or minimal defects to produce high quality software. Software defect detection at the earliest stage reduces development cost, reworks effort and improves the quality of software. In this study, the efficiency of different classifiers such as Naïve Bayes, Support Vector Machine (SVM) and K-Nearest Neighbor (KNN) are evaluated for SDP.

Keywords: Naïve bayes, K-Nearest Neighbor (KNN), Partial decision Tree Algorithm (PART), Software Defect Prediction (SDP), software quality, Support Vector Machine (SVM)

INTRODUCTION

Software defects are errors, flaws, bugs, mistakes, failures or faults in computer programs or systems that generate inaccurate/unexpected outcome, or preclude software from its intended behavior. A project team aspires to create quality software products with null or minimal defects. High risk components of a software project need to be detected at the earliest stage of software life cycle to enhance the software quality. Software defects, affect quality and time. Also, identifying and rectifying defects are time consuming and expensive. It is impractical to eliminate all defects, but reducing defects magnitude and adverse effects on projects is possible (Rawat and Dubey, 2012). A software defect is a flaw or imperfection in a software work product or software process. A software process is an activity, method, practice and transformation that people use to develop and maintain software work products (Clark and Zubrow, 2001).

Software Defect Prediction (SDP) locates defective modules in software. The final product should have as few defects as possible to ensure high quality software. Software defect detection at the earliest stage leads to reduce development cost, rework efforts and reliable software. Thus defect prediction study is important to ensure software quality. The most discussed problems in software quality and software reliability is SDP. As Boehm says, finding and fixing a problem after delivery is 100 times more expensive than fixing it in the requirement and design phase. Also, software projects spend 40 to 50% of effort in avoidable rework (Boehm and Basili, 2007).

Software is a complex entity of various modules with varied defect occurrence possibility range. Efficient and timely defect occurrence prediction allows software project managers to use people, cost and time for improved quality assurance. Defects in software, results in poor quality software and leads to software project failure. Occasionally, it is impossible to identify defects and fix them during development and so they are handled when noticed by team members. Hence, defect-prone software modules should be predicted before deployment to plan better maintenance strategies. Early detection of defect prone software module knowledge ensures an efficient process improvement plan within stipulated cost and time. This results in better software release and high customer satisfaction. Accurate defect measurement and prediction is crucial in software as it is an indirect measurement based on many metrics (Nam, 2010).

A common SDP process based on machine learning models is seen in Fig. 1. The first step in building a prediction model is to generate instances from software archives like version control systems, e-mail archives and issue tracking systems. Each instance represents a system, class, function (method), a software component (or package), source code file and/or code change based on prediction granularity. An instance, has many metrics (features) culled from software archives and labeled with buggy/clean or number of bugs. For example, instances generated from software archives are labeled with 'B' (buggy), 'C' (clean), or the number of bugs (Verner and Tate, 1992) in Fig. 1.

Corresponding Author: M. Anbu, Department of Information Technology, St. Joseph's Engineering College, Chennai, India

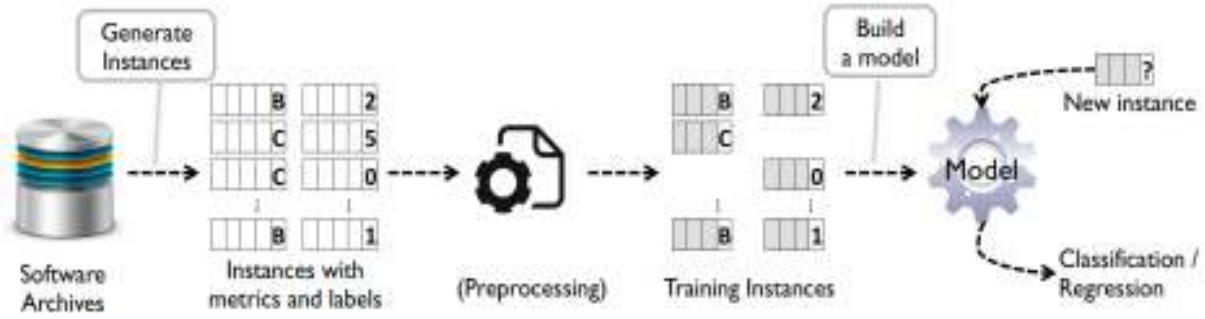

Fig. 1: Common process of software defect prediction

Software metrics are defined by measuring some property of a software portion or its specifications. Software metrics ensure quantitative method to assess software quality. Software metrics are defined as "continuous application of measurement based techniques to software development and its products to ensure supply of meaningful and timely Management Information with use of techniques to improve products and process" (Rawat *et al*., 2012). Software metrics is used for measuring the process and product of software. Various software metrics are classified as Software Product metrics, Software Process metrics and Software Project metrics. Process metrics highlights the software development process. It aims to measure the process duration, cost and type of method used. Process metrics augment software development and maintenance. Examples include efficacy of defect removal during development, patterning of testing defect arrival and fix process response time. Project metrics monitor the project situation and status. Project metrics preclude problems or risks by calibrating projects and optimize the software development plan. Project metrics describe project characteristics and execution. Measuring the number of software developers, staffing pattern over a software life cycle, cost, schedule and productivity are the examples of project metrics. Product metrics describe the software product attributes during any development phase. Product metrics measure program size, software design complexity, performance, portability, maintainability and product scale. Product metrics presume and invent product quality. Product metrics measure the medium or final product (Rawat *et al*., 2012). These metrics play an important role in classifying the modules as defective or non-defective.

In this study, the efficiency of the classifiers in classifying defective modules are evaluated. A KC1 dataset from the PROMISE software dataset repository is used for evaluation.

LITERATURE REVIEW

An association rule mining based method to predict defect associations and defect correction effort was presented by Song *et al*. (2006) to help developers detect software defects and assists project managers to allocate testing resources effectively. This study applied the new method to the SEL defect data of more than 200 projects over more than 15 years. Results reveal that for defect association prediction, accuracy is very high and false-negative rate very low. Likewise, for defect correction effort prediction, accuracy for defect isolation effort prediction and defect correction effort prediction are high. This study compared defect correction effort prediction method with other methods-PART, C4.5 and Naive Bayes and showed that accuracy improved by 23%. Support and confidence levels impact on prediction accuracy was evaluated, along with the false-negative rate, false-positive rate and rules.

Various classifications and clustering methods aimed at predicting software defects was proposed by Chug and Dhall (2013). It analyzed classification and clustering techniques to predict software defects. Performance of 3 data mining classifier algorithms called J48, Random Forest and Naive Bayesian Classifier based on criteria like ROC, Precision, MAE and RAE were evaluated. Clustering technique using k-means are then applied on Hierarchical Clustering and Make Density Based Clustering algorithm on the data set. Evaluation of clustering is based on criteria like Cluster Instance, Time Taken, Incorrectly Clustered Instance, Number of Iterations and Log Likelihood. An exploration of 10 real time NASA software projects defect datasets followed by applications finally leads to defect prediction.

A new SDP model using Particle Swarm Optimization (PSO) and Support Vector Machine (SVM) named P-SVM model was proposed by Can *et al*. (2013). The authors use PSO algorithm to calculate best SVM parameters and adopts optimized SVM model to predict software defect. P-SVM model and three other prediction models predict software defects in JM1 data set on an experimental basis, the results showing that P-SVM has a higher prediction accuracy compared to BP Neural Network model, SVM and GA-SVM models.

A technique to select best attributes set to improve SDP accuracy was proposed by Khan *et al*. (2014). Software attribute characteristics influence the defect prediction model's performance and effectiveness. The

new method is evaluated using NASA metric data repository data sets and demonstrates acceptable accuracy using a simple algorithm.

The best size of feature subset to build a prediction model to prove that feature selection establishes SDP model was discussed by Wang *et al*. (2012a). Mutual information is an outstanding relevance indicator between variables and used as a measurement in the new feature selection algorithm. A nonlinear factor for evaluation function was introduced for feature selection to improve performance. The results of the feature selection algorithm were validated by varying machine learning methods. Experimental results reveal that all classifiers achieved high accuracy.

Three new defect prediction models based on C4.5 model were proposed by Wang *et al*. (2012b). Spearman's rank correlation coefficient was introduced to choose the root node of the decision tree which improves models on defect prediction. An experimental scheme was designed to verify the improved model's effectiveness and this paper compared prediction accuracies of existing and improved models. Results showed that improved models reduced decision tree size by 49.91% on average and increased prediction accuracy by 4.58 and 4.87% on two modules in the experiment.

A defect predictor based on Naive Bayes theory was presented by Tao and Wei-Hua (2010), analyzed their difference estimation methods and algorithm complexity. This study concluded that the best defect predictor to be Multi-variants Gauss Naive Bayes (MvGNB) through evaluation and compared it with a decision tree learner J48. Results on benchmarking MDP data sets proved that MvGNB was useful to predict defects.

Different feature selection and dimensionality reduction methods were used to determine most important software metrics by Xia *et al*. (2013). Three different classifiers, namely Naïve Bayes, SVM and decision tree were used. On the NASA data, comparative experiment results show that instead of 22 or more metrics, less than 10 metrics ensure better performance.

The problem of defect prediction was focused on by Czibula *et al*. (2014) which was important during software maintenance and evolution. As conditions which result in software modules having defects are hard to identify, machine learning based classification models was used to offset this issue. This study proposed a new relational association rules mining based classification model which in turn is based on the discovery of relational association rules to predict whether a software module is defective.

The positive effects of combining feature selection and ensemble learning on defect classification performance were demonstrated by Laradji *et al*. (2014). Added to the efficient feature selection, a new two-variant (with/without feature selection) ensemble learning algorithm was proposed to ensure robustness of data imbalance and feature redundancy. This study shows software dataset features which are carefully chosen for defective components accurate classification. Further, tackling software data issues mentioned above, with the new combined learning model lead to remarkable classification performance and paved the way for successful quality control.

A new algorithm called Transfer Naive Bayes (TNB) proposed by Ma *et al*. (2012), used information of all training data features. This solution estimates test data distribution and transfers cross-company data information to training data weights. The defect prediction model is built on the weighted data. This article presents a theoretical analysis of comparative methods, showing data sets' experiment results from various organizations. It indicates that TNB is more accurate regarding AUC (Area Under receiver operating characteristic Curve), with reduced runtime than state of the art methods.

Three cost-sensitive boosting algorithms to boost neural networks for SDP were presented by Zheng (2010). Based on threshold-moving, the first algorithm tries to move the classification threshold to not-fault-prone modules as more fault-prone modules are classified correctly. The other two weight-updating based algorithms incorporate misclassification costs into the boosting procedure's weight-update rule so that algorithms boost more weights on samples associated with misclassified defect-prone modules. Performances of the three algorithms were evaluated using 4 NASA project datasets regarding a singular measure, Normalized Expected Cost of Misclassification (NECM). Experiments suggested that threshold-moving is the best choice to build cost-sensitive SDP models with boosted NNs, among the three algorithms.

Combining meta-heuristic optimization methods and bagging technique to improves SDP performance was proposed by Wahono *et al*. (2014). Meta-heuristic optimization methods (Genetic Algorithm (GA) and PSO) were applied to handle feature selection and bagging technique is used to deal with class imbalance issues. Results indicated that new methods improved classifier performance greatly. Based on comparisons, it is concluded that there is no great difference between PSO and GA when used as feature selection for classifiers in SDP.

A new Two-Stage Cost-Sensitive learning (TSCS) method for SDP, using cost information in the classification and feature selection stages was proposed by Liu *et al*. (2014). For feature selection, it specifically developed 3 new cost-sensitive feature selection algorithms, Cost Sensitive Variance Score (CSVS), Cost-Sensitive Laplacian Score (CSLS) and Cost-Sensitive Constraint Score (CSCS), by including cost information into traditional feature selection algorithms.

The new methods were evaluated in NASA projects 7 real data sets. Results suggest that TSCS method achieved better performance in SDP compared to current single-stage cost-sensitive classifiers. This experiment also reveals that new cost-sensitive feature selection methods outperform conventional cost-blind feature selection methods, validating the use of cost information at the feature selection stage.

Four semi-supervised classification methods for semi-supervised defect prediction was evaluated by Catal (2014). Low-Density Separation (LDS), SVM, Expectation-Maximization (EM-SEMI) and Class Mass Normalization (CMN) methods were investigated on NASA data sets including CM1, KC1, KC2 and PC1. Results proved that SVM and LDS algorithms outperformed CMN and EM-SEMI algorithms. LDS algorithm performs better than SVM in large data sets. When there is limited fault data, LDS-based prediction approach is suggested for SDP.

MATERIALS AND METHODS

Dataset: CM1, PC1, KC1 and KC2 dataset are from the PROMISE software dataset repository and are widely used for evaluating SDP. This study used these datasets so that the predictive model's performance could be compared. Table 1 shows the dataset of software defects.

This dataset has many software metrics like Line of Code, number of operands and operators, Program length, Design complexity, effort and time estimator and other metrics which identify software with defects (Agarwal and Tomar, 2014).

The KC1 data is obtained from a science data processing project coded in C++, containing 2108 modules (Challagulla *et al*., 2008). NASA's Metric Data Program (MDP) data repository's KC1 data set comprises logical groups of Computer Software Components (CSCs) in a large ground system. KC1 has 43,000 lines of code, coded in C++ the data set having 2,107 instances (modules). In these instances, 325 have one or more faults and 1,782 have nil faults. A module's maximum number of faults is seven. Defect prediction models have independent variables as product and process metrics and a dependent variable indicates whether a module has a fault. In this data set, class_label is the dependent variable and the rest independent variables (Gayathri and Sudha, 2014).

KC1 dataset is a NASA Metrics Data Program (Shirabad and Menzies, 2005) verifying/improving predictive software engineering models. KC1 is a C++ system implementing storage management for ground data receipt/processing containing of McCabe and Halstead features code extractors and module based measures.

Table 1: Details of software defect dataset

Dataset	Language	No. of modules	Defective (%)
CM1	C	496	9.7
PC1	C	1,107	6.9
KC1	C++	2,109	15.5
KC2	C++	522	20.5

Defect detectors are calculated as:

a = Classifier predicts no defects and module has no error
b = Classifier predicts no defects and module has error
c = Classifier predicts some defects and module has no error
d = Classifier predicts some defects and module has error

Accuracy, probability detection (pd) or recall, precision (prec), probability of false alarm (pf) and effort are calculated as:

$$Accuracy = \frac{a+d}{a+b+c+d} \quad (1)$$

$$recall = \frac{d}{b+d} \quad (2)$$

$$pf = \frac{c}{a+c} \quad (3)$$

$$prec = \frac{d}{c+d} \quad (4)$$

$$effort = \frac{c.LOC + d.LOC}{TotalLOC} \quad (5)$$

KC1 dataset has 2109 instances and 22 varied attributes including 5 different LOC, 3 McCabe metrics, 12 Halstead metrics, a branch count and 1 goal-field. Dataset's attribute information is: total operands, design complexity, McCabe's Line count of Code (LOC), cyclomatic complexity, program length, effort, Halstead, class and others. Some samples from the dataset are given:

Example 1 : 1.1, 1.4, 1.4, 1.4, 1.3, 1.3, 1.3, 1.3, 1.3, 1.3, 1.3, 1.3, 2, 2, 2, 2, 1.2, 1.2, 1.2, 1.2, 1.4, false

Example 2 : 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, true

Example 3 : 83, 11, 1, 11, 171, 927.89, 0.04, 23.04, 40.27, 21378.61, 0.31, 1187.7, 65, 10, 6, 0, 18, 25, 107, 64, 21, true

Classifiers: Classification divides data samples into target classes. For example, a software module is

categorized as "defective" and "not-defective" using classification. In Classification, class categories are already known and so is a supervised learning approach (Han and Kamber, 2006). There are two classification methods: Binary and Multilevel. Binary classification divides a class only into two categories "defective" and "not-defective". Multi-level classification is used when there are more than two classes and they are split into "highly complex", "complex" or "simple." Learning and Testing are two phases of a classification approach: hence, it divides a dataset into two parts: training and testing. Approaches like cross fold and Leave-one-out partition the dataset. During the learning phase, a classifier is learned using training dataset and evaluated with a testing dataset. Varied classification techniques are available and classifiers used for this are discussed below (Ma *et al*., 2012).

Support Vector Machine (SVM): SVM divides data samples of two classes by determining a hyper-plane in original input space maximizing their separation. SVM works well for classification of data samples inseparable linearly by using the kernel function theory. Many kernel functions for example Gaussian, Polynomial and Sigmoid are available and are used to map data samples to higher dimensional feature space. SVM then determines a hyper-plane in the feature space to divide data samples of various classes. This is a better choice for linearly and nonlinearly separable data classification. SVM has many advantages like ensuring a global solution for data classification. It generates a unique global hyper-plane to separate different class data samples rather than local boundaries as compared to existing data classification approaches. As SVM follows Structural Risk Minimization (SRM) principle, it reduces risk during training and enhances its generalization capability (Xing *et al*., 2005).

SVM uses machine learning technique and Method level metrics. SVM is a supervised learning models with associated learning algorithms analyzing data and recognizing patterns for classification and regression analysis (Challagulla *et al*., 2008). SVM was successfully solved classification and regression problems in applications. SVM's capability in predicting defect prone software modules and comparing its performance against 8 statistical and machine learning models in the context of four NASA datasets is studied. Results indicate that SVM prediction performance is better than the models compared. SVM's advantage is ensuring better performance and its disadvantage is that it does not work well on public datasets (Singh *et al*., 2010). SVM was used in bioinformatics, image classification and handwriting recognition. SVM is used for data classification and is robust in nature than other software quality prediction techniques. It adapts to modeling non-linear functional relationships, achieves improved performance and is an efficient software quality prediction technique. To minimize cost and improve software testing process effectiveness, researchers used SVM on a Data set from NASA's Metrics Data Program (MDP) data repository (Voulgaris and Magoulas, 2008). The advantage of SVM is that they provide better performance. The main disadvantage of SVM is that it does not work well on public datasets.

K-Nearest Neighbor (KNN): This classifier's working is based on a voting system. KNN locates new or unidentified data sample aided by earlier identified data samples, called nearest neighbor and samples being assigned using voting strategy. More than one nearest neighbor participates in data samples classification. KNN learning is slow and hence is called Lazy Learner. KNN technique is used in clustering and classification.

An algorithm in k Nearest Neighbor classification (Sridhar and Babu, 2012), finds a set of k objects in training set close to the input and classifies it based on the majority of that group class. The elements needed for this approach are: distance metrics, labeled objects set and a number of k.

PART: PART is a partial decision tree algorithm is the extension of RIPPER and C4.5 algorithms. PART algorithm need not perform global optimization to produce the appropriate rules. Class for generating a PART decision list use separate-and-conquer method.

DTNB: Decision Tree and Naïve Bayes algorithm uses a decision table/naive bayes hybrid classifier. At each point during the search, the algorithm evaluates the advantage of dividing the attributes into two disjoint subsets. Initially all attributes are modeled by decision table initially. A forward selection search is used where each attribute is modeled by Naïve Bayes and remaining by decision table. At each step, the algorithm also dropping an attribute entirely from the model.

NNge: Nearest-neighbor-like algorithm using non-nested generalized exemplars which can be viewed as if-then rules.

Naïve Bayes: Naive Bayesian classifier is based on Bayes theorem with independence assumptions between the predictors. Naive Bayesian model is easy to build, having no complicated iterative parameter estimation making it specifically useful for large datasets. Despite simplicity, Naive Bayesian classifier often does well and is used as it outperforms sophisticated classification methods.

Bayes net: Bayes Network classifier uses a variety of search algorithms and quality measures. It supports different data structures like network structure, conditional probability distributions.

NB updateable t: Naive Bayes classifier uses estimator classes. It is the updateable version of Naïve Bayes.

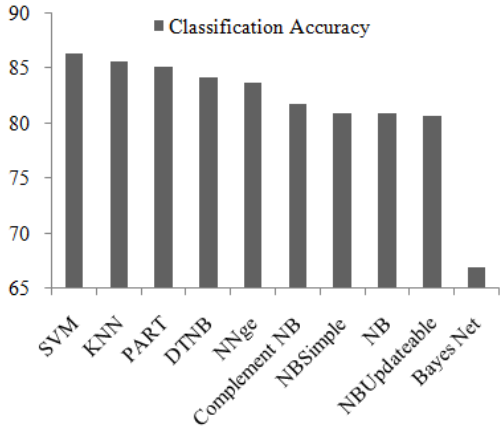

Fig. 2: Classification accuracy for different classifiers

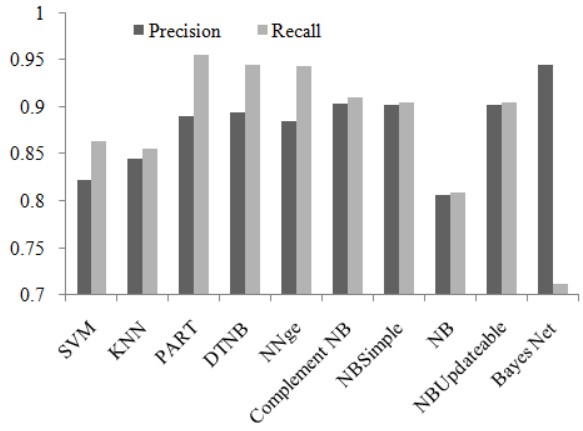

Fig. 3: Precision and recall for different classifiers

Table 2: Classification accuracy, precision, recall for different classifiers using K folds cross validation

Algorithm	Classification accuracy	Precision	Recall
SVM	86.27	0.822	0.863
KNN	85.51	0.845	0.855
PART	85.13	0.890	0.955
DTNB	84.13	0.894	0.945
NNge	83.61	0.885	0.943
Complement NB	81.71	0.903	0.910
NB simple	80.90	0.902	0.905
NB	80.85	0.806	0.809
NB updateable	80.57	0.902	0.905
Bayes net	66.88	0.944	0.711

This classifier uses a default precision of 0.1 for numeric attributes with zero training instances.

NB simple: It is a Class for building and using a simple Naive Bayes classifier and numeric attributes are modelled by a normal distribution.

RESULTS AND DISCUSSION

In this study experiment are conducted for obtaining classification accuracy, precision and recall for classifiers such as Naïve Bayes, SVM and KNN. Table 2 and Fig. 2 shows the results of accuracy, precision and recall for different classifiers.

For Classification accuracy, SVM classifier performs better than all the other classifiers. For Classification accuracy, SVM classifier performs better by 25.32% than Bayes Net and by 6.83% than NB Updateable (Fig. 3).

For Precision, Bayes Net classifier performs better than all the other classifiers. For Precision, Bayes Net classifier performs better by 15.77% than NB and by 13.82% than SVM. For Recall, PART classifier performs better than all the other classifiers. For Recall, PART classifier performs better by 29.29% than Bayes Net and by 16.55% than NB.

CONCLUSION

In this study, experiments are carried out for analyzing the defect prediction using different types of classifiers such as NB, SVM, KNN etc. The classifiers are evaluated for KC1 dataset. The classification accuracy of the SVM classifier performs better when compared to other classifiers. The study can be extended to improve the classifier for SDP which outperforms well for classification accuracy, precision and recall.

REFERENCES

Agarwal, S. and D. Tomar, 2014. A feature selection based model for software defect prediction. Int. J. Adv. Sci. Technol., 65: 39-58.

Boehm, B. and V.R. Basili, 2007. Software defect reduction top 10 list. Software Eng. Barry W. Boehm's Lifetime Contribut. Software Develop. Manage. Res., 34(1): 75.

Can, H., X. Jianchun, Z. Ruide, L. Juelong, Y. Qiliang and X. Liqiang, 2013. A new model for software defect prediction using particle swarm optimization and support vector machine. Proceeding of 25th IEEE Chinese Control and Decision Conference (CCDC, 2013), pp: 4106-4110.

Catal, C., 2014. A comparison of semi-supervised classification approaches for software defect prediction. J. Intell. Syst., 23(1): 75-82.

Challagulla, V.U.B., F.B. Bastani, I.L. Yen and R.A. Paul, 2008. Empirical assessment of machine learning based software defect prediction techniques. Int. J. Artif. Intell. T., 17(02): 389-400.

Chug, A. and S. Dhall, 2013. Software defect prediction using supervised learning algorithm and unsupervised learning algorithm. Proceeding of 4th International Conference Confluence 2013: The Next Generation Information Technology Summit, pp: 173-179.

Clark, B. and D. Zubrow, 2001. How good is the software: A review of defect prediction techniques? Sponsored by the US Department of Defense. Carnegie Mellon University, Pittsburgh, PA.

Czibula, G., Z. Marian and I.G. Czibula, 2014. Software defect prediction using relational association rule mining. Inform. Sci. Int. J., 264: 260-278.

Gayathri, M. and A. Sudha, 2014. Software defect prediction system using multilayer perceptron neural network with data mining. Int. J. Recent Technol. Eng. (IJRTE), 3(2).

Han, J. and M. Kamber, 2006. Data Mining, Southeast Asia Edition: Concepts and Techniques. Morgan Kaufmann, pp: 770.

Khan, J.I., A.U. Gias, M. Siddik, M. Rahman, S.M. Khaled and M. Shoyaib, 2014. An attribute selection process for software defect prediction. Proceeding of the International Conference on Informatics, Electronics and Vision (ICIEV, 2014), pp: 1-4.

Laradji, I.H., M. Alshayeb and L. Ghouti, 2014. Software defect prediction using ensemble learning on selected features. Inform. Software Tech., 58: 388-402.

Liu, M., L. Miao and D. Zhang, 2014. Two-stage cost-sensitive learning for software defect prediction. IEEE T. Reliab., 63(2): 676-686.

Ma, Y., G. Luo, X. Zeng and A. Chen, 2012. Transfer learning for cross-company software defect prediction. Inform. Software Tech., 54(3): 248-256.

Nam, J., 2010. Survey on Software Defect Prediction.

Rawat, M.S. and S.K. Dubey, 2012. Software defect prediction models for quality improvement: A literature study. Int. J. Comput. Sci., 9(5).

Rawat, M.S., C.O.E.T. MGM's, A. Mittal and S.K. Dubey, 2012. Survey on impact of software metrics on software quality. Int. J. Adv. Comput. Sci. Appl., 3(1).

Shirabad, J.S. and T.J. Menzies, 2005. The PROMISE repository of software engineering databases. School of Information Technology and Engineering, University of Ottawa, Canada.

Singh, Y., A. Kaur and R. Malhotra, 2010. Prediction of fault-prone software modules using statistical and machine learning methods. Int. J. Comput. Appl., 1(22): 8-15.

Song, Q., M. Shepperd, M. Cartwright and C. Mair, 2006. Software defect association mining and defect correction effort prediction. IEEE T. Software Eng., 32(2): 69-82.

Sridhar, S.M. and B.R. Babu, 2012. Evaluating the classification accuracy of data mining algorithms for anonymized data. Int. J. Comput. Sci. Telecommun., 3(8).

Tao, W. and L. Wei-Hua, 2010. Naïve bayes software defect prediction model. Proceeding of International Conference on Computational Intelligence and Software Engineering (CiSE), pp: 1-4.

Verner, J. and G. Tate, 1992. A software size model. IEEE T. Software Eng., 18(4): 265-278.

Voulgaris, Z. and G.D. Magoulas, 2008. Extensions of the k nearest neighbour methods for classification problems. Proceeding of 26th IASTED International Conference on Artificial Intelligence and Applications, CD Proceedings ISBN: 978-0-88986-710-9, pp: 23-28.

Wahono, R.S., N. Suryana and S. Ahmad, 2014. Metaheuristic optimization based feature selection for software defect prediction. J. Software, 9(5): 1324-1333.

Wang, J., B. Shen and Y. Chen, 2012b. Compressed C4. 5 models for software defect prediction. Proceeding of 12th International Conference on Quality Software (QSIC), pp: 13-16.

Wang, P., C. Jin and S.W. Jin, 2012a. Software defect prediction scheme based on feature selection. Proceeding of International Symposium on Information Science and Engineering (ISISE, 2012), pp: 477-480.

Xia, Y., G. Yan and Q. Si, 2013. A study on the significance of software metrics in defect prediction. Proceeding of 6th International Symposium on Computational Intelligence and Design (ISCID, 2013), 2: 343-346.

Xing, F., P. Guo and M.R. Lyu, 2005. A novel method for early software quality prediction based on support vector machine. Proceeding of 16th IEEE International Symposium on Software Reliability Engineering (ISSRE, 2005), pp: 10.

Zheng, J., 2010. Cost-sensitive boosting neural networks for software defect prediction. Expert Syst. Appl., 37(6): 4537-4543.