Journal of Advanced Research in Applied Sciences and Engineering Technology

Journal homepage: https://jaraset.com/
ISSN: 2462-1943

SIMULATING AND EVALUATING ACCURACY OF SOFTWARE DEFECT PREDICTION MODEL USING DIVERSE MACHINE LEARNING TECHNIQUES

Farukh Khan [1*], Dr. Chandikaditya Kumawat [2]

[1] Research Scholar, Department of Computer Science and Engineering, Mewar University, Rajasthan, India
[2] *Associate Professor, Department of Computer Science and Engineering, Mewar University, Rajasthan, India*

ARTICLE INFO

Article history:
Received: 22-08-2024
Received in revised form: 20-09-2024
Accepted: 28-09-2024
Available online: 11-11-2024

Keywords:

Support vector machine (SVM), SDP models, artificial neural network (ANN), KELM, ELM.

ABSTRACT

A software bug that is moved to the next stage of the software development lifecycle (SDLC) costs ten times more to remove. This lowers the quality of the final software product and makes the job of the project managers more challenging. As a result, the software industry has mandated that high-quality software projects must be completed on schedule and within budget Support vector machine (SVM) and artificial neural network (ANN), two classification techniques with the prediction power to manage the intricate non-linear correlations between the software characteristics and the software fault, have been empirically compared. Artificial neural networks (ANNs) are suitable to construct defect prediction models because of their capacity to manage the intricate nonlinear interactions between the software metrics and the defect data. The feature selection techniques solve this issue. Two classifiers, ELM and KELM, which are based on wrapper and filter-based feature selection techniques, are used to build SDP models. The study aims to ascertain two things: (1) the efficacy of feature selection-based classification models in software defect prediction; and (2) whether or not the elimination of superfluous features significantly alters the performance of the SDP models.

Introduction

Software testing is a highly important and crucial part of software development. Up to 50% of the total cost of software development is covered by software testing [1]. There are efforts underway to lower the price of software testing. One area of study that attempts to identify the early software lifecycle modules most likely to result in errors is software defect prediction. The modules in question may be addressed in a prioritized manner, hence reducing the work and resources needed to ensure their flawless functioning.

Because software technology is becoming more and more necessary in every aspect of our lives, software quality has become a crucial topic. Predicting software defects is seen as a quality assurance task that reduces the amount of errors in the product that is being built beforehand. It works by predicting a module's likelihood of having a defect before making the necessary preparations to prevent a fault from occurring. This guarantees that sufficient time and resources are allocated to the defect-prone components in order to adequately cover them and that neither time nor resources are spent on a module that is defect-free. In addition to providing the customer with a quantitative output, defect prediction aids in the development of a qualitative product.

One of the most practical and economical software operations is defect prediction. It is regarded by software professionals as a crucial stage that determines the quality of the product being built. It has significantly contributed to dispelling the claims made against the software industry that it cannot meet deadlines and budgets. In addition, there has been a noticeable change in the clientele's reaction from unsatisfactory to better about the product quality.

The older statistical methodologies for defect prediction have been mostly supplanted by data miners today. The classification model that forms the foundation of data mining assigns the component to either the fault-prone or non-fault-prone class. First, we provide the classifier with known examples, whose class we already know. After training, the model is evaluated on instances that are unknown, and the accuracy of the prediction made by the method is assessed. The rest of this chapter is dedicated to providing a quick overview of the disciplines and subject that are relevant to this dissertation.

Predicting Software Defects

The hardest task for a software engineer is handling bugs in the product once they are found. Software flaws reduce the quality of the programmed and raise the cost of the resources used to create it. Since fixing a fault becomes progressively more expensive as software development progresses, it is essential to find these flaws as soon as possible. Because of this procedure, the testing phase accounts for over half of the software project's overall cost, necessitating cautious handling [2]. This has led the software community to create a variety of techniques to find software flaws early in the software development lifecycle. In the research on software defect

prediction, creating models for software defects is determined to be one of the noteworthy approaches [2].

Static source code and design metrics have been useful in developing classification models since they capture the majority of the software's coding and design elements [3]. Early in the software development lifecycle, these indicators assist software managers in creating models for predicting software defects. The quality analysts are guided by the expected fault proneness of a module, which allows the resources to be effectively and efficiently focused on the problem likely regions. This has made it easier for software firms to provide their clients with affordable, dependable, high-quality products.

SDP is often approached as a binary classification issue, in which a module is classified as either non-defect prone or prone to defects. A module is an individual, indivisible unit of source code. It may be a class or a procedure, and it has a set of properties that are specific to objects, such Chidamber and Kemerer metrics [4] for classes, or procedural metrics, like Halstead metrics [5], for procedures. Using labeled historical defect data, a software defect prediction model is constructed during a training phase. The trained model then functions as a classifier for newly discovered unknown data. The SDP model's classification performance is assessed based on a number of factors, including specificity, sensitivity, and accuracy.

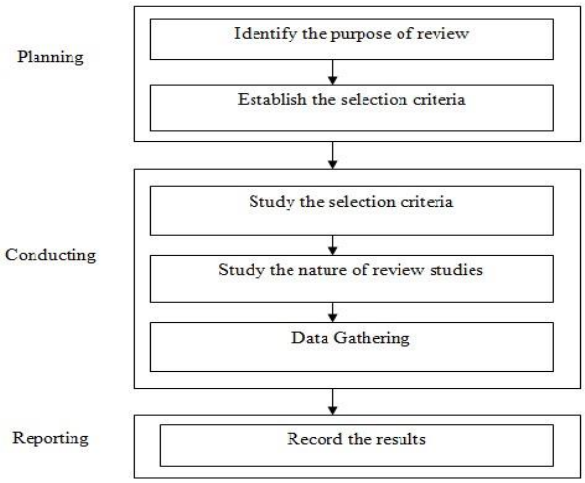


Figure 1 Review Process

REVIEW OF RELATED LITERATURE

Software practitioners tend to use phrases like defect, fault, mistake, malfunction, and failure interchangeably. They are specifically defined by Parhami [52] as distinct states where the system may go above owing to a repair or below due to a fluctuation. Anticipating the possibility of a flaw occurring in a certain software module is one of the potential remedies that saved the software engineers. The probability of a defect occurring aided them in organizing these modules based on the expected severity of the fault and allocating enough time and personnel to them in order to minimize errors in the final product and avoid going over budget. This approach, which has been studied and used by many academics, is known as "Software Defect Prediction [6]." Researchers have shown a link between several software process and product indicators and the incidence of software defects [3]. Early software lifecycle defect forecasting is facilitated by prediction models that are constructed using a mix of software metrics and previous defect data.

The inability of machine learning approaches to provide a benchmark result increases with the complexity and imprecision of the software project data collection process. In order to prevent the performance of software defect prediction systems from declining, a set of algorithms that could include this imprecision and uncertainty into their operation became necessary. These techniques eventually led to the development of the term "Soft Computing" (SC). In the early 1990s, SC developed as an approach to address the software defect prediction issue. It served as the foundation for a significant number of machine learning approaches at first, and it later developed to include fuzzy-based methods and evolutionary algorithms.

It is critical to understand the importance of using these approaches and, therefore, to summaries the available literature research in order to analyze and further improve the employability of SC in software defect prediction. To the best of the authors' knowledge, none of the review articles that have been published in the SDP literature [54, 55, 56] have addressed the use of soft computing approaches to help determine if a module belongs to a faulty or non-defective label at the method or class level.

This made it easier for us to research and look at how approaches are represented as models for tackling SDP problems, how new approaches are represented, what difficulties have been addressed, and what traditional challenges in this field still need to be solved. Motivation

In 2009, Catal and Diri [4] conducted a comprehensive analysis of 74 research studies on software fault prediction. The utilisation trends of procedural and object-oriented metrics, public and private data sets, machine learning, and statistical approaches utilized in SDP techniques were the main topics of this study. In a different study, Catal [5] reviewed 90 papers published between 1990 and 2009, covering the machine learning and statistical methods used by the authors[7].

Abaei and Selamat [7] conducted an empirical analysis using four NASA [8] datasets that included method level metrics, principal component analysis, correlation-based feature selection techniques, and naive Bayes, decision trees, decision tables, random forests, neural networks, artificial immune systems (AIS), CLONALG, and Immunos. They also presented a survey of various machine learning approaches of software defect detection.

This literature review is unique in the ways listed below:

- The prior research has only looked at machine learning methods used in software fault prediction. Machine learning approaches are just one subset of the soft computing strategies used for SDP issues, which are covered in full in this review article. As such, it provides a clearer picture of the relationship between machine learning techniques and other components of soft computing, such fuzzy and evolutionary techniques[8].

- This work has 120 research publications overall, spanning the years 2005 through 2023. As a result, it will ultimately provide a deeper and more thorough picture than the previous ones.

- This review aims to do two things: first, it will analyse the research based on metrics, datasets, and techniques; second, it will examine the rise of published studies on SC in SDP year over year.

Material and Methodology

The procedural steps that were taken throughout the review process. Three steps comprised the process: planning, carrying out, and reporting the outcomes. The goal of the review was determined during the planning stage, and the requirements for research publications to be included for consideration under review were set. Examining the selection criteria, examining the nature of the review studies, and collecting data are all part of the second phase. In the last step, the analysis and review findings were presented. Selection Criteria

In order to investigate the viability of SC approaches as a solution to software defect prediction problems, this review work looks at how techniques are represented as a model for addressing SDP problems, as well as the concerns that have been overcome and the classic challenges that need more investigation[9]. The selection criteria developed for the research papers to be considered under review are described in

State of the Art

A survey of the applications of soft computing techniques to software defect prediction issues was provided by the literature analysis. Depending on the kind of SC method used by the authors, the included research were further categorised into three groups.

Evolutionary Methods

In order to decrease the amount of faulty software modules, Vandecruys et al. [9] introduced a data mining approach called AntMiner+ that is based on ACO and is used to create efficient defect prediction models. On three NASA public datasets, the authors evaluated their technique's accuracy performance against that of C4.5, logistic regression, and support vector machines. The results demonstrate that the suggested technique is superior and can be used to identify the crucial stages of the software development lifecycle.

Mausa et al.'s analysis [6] of the ensembles' defect prediction ability for the imbalanced datasets made use of genetic programming. The multi objective evolutionary algorithm used three distinct mechanisms for ensemble selection. Three public datasets the Java Development Tool, the Eclipse Plugin, and Apache Hadoop were utilised in various versions. Four datasets from the UCI library were also used to duplicate the experiments [1]. In contrast to the previous

evolutionary algorithms, the findings demonstrated that the multilevel selection approach that was given produced dependable results and had a quick rate of convergence.

Research Process

The research process outlines the procedures to be followed in order to complete the assignment efficiently. The research methodology used in this thesis project is shown in Figure 1. The steps shown in Figure 2 are explained in the next section. In the next chapters of this dissertation, these procedures are followed.

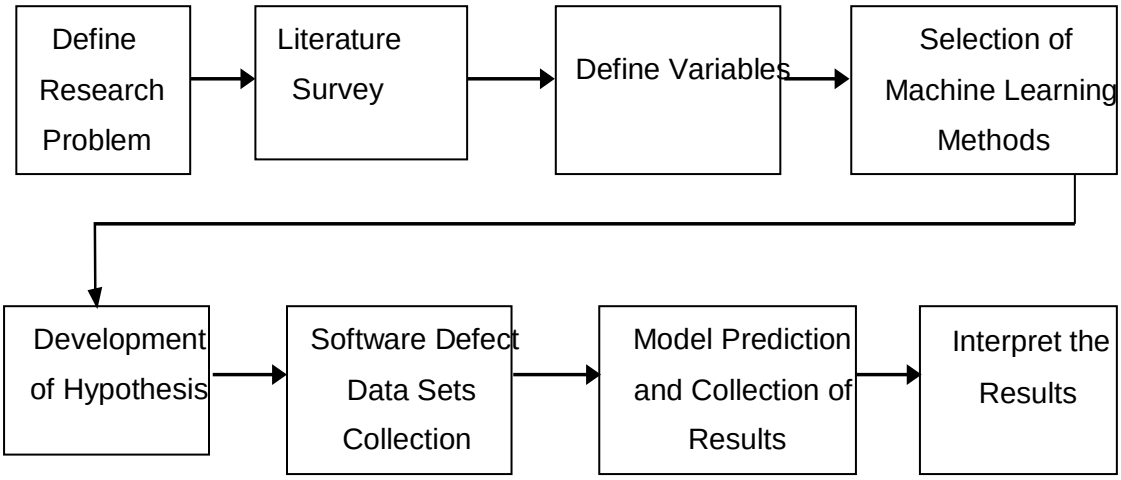


Figure 2 The Methodology of Research

Selecting Data Analysis Techniques

The choice of data analysis techniques is aided by the nature of the dependent variable being used [10]. Four machine learning techniques were evaluated in this work: Kernel-based Extreme Learning Machine (KELM), Support Vector Machine (SVM), Artificial Neural Network (ANN), and Extreme Learning Machine (ELM). The fault proneness, a binary dependent variable, is predicted using these machine learning techniques.

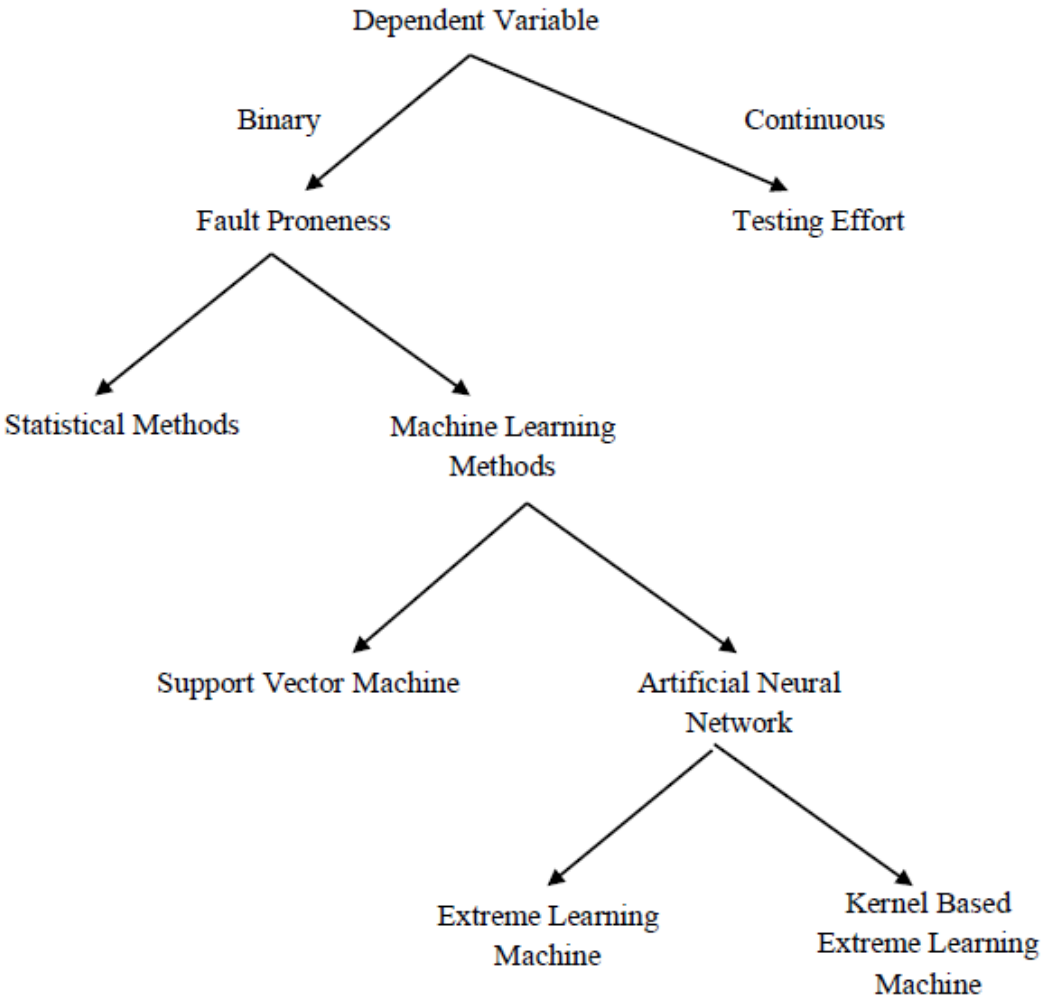


Figure 3 Choosing Data Analysis Techniques

Support Vector Machine

In a Support Vector Machine (SVM) model, the examples are represented as a collection of points in space that are spaced so that the gaps between the various categories are as large as feasible. Depending on which side the unlabeled data falls on, predictions are made for them. Kernel functions are those that carry out this mapping into the space. The most prevalent kinds of kernel functions are sigmoid, polynomial, linear, and Gaussian functions [4, 5, 6]. A single data point is placed in a p-dimensional space and a linear SVM classifier, determines whether or not these points may be separated by a (p-1) dimensional hyper plane. The hyper plane that divides the data points with the biggest margin between the two classes is the best option. Stated

otherwise, the optimal separation hyper plane is selected to minimize the generalization error by having the largest buffer from the closest training data point.

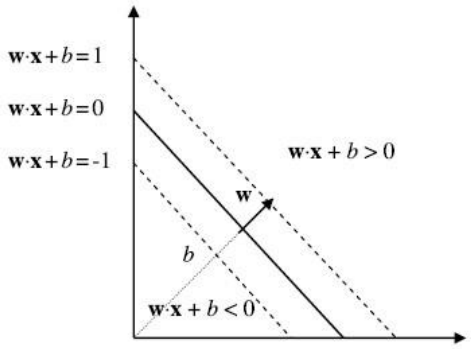


Figure 4 A basic classifier using linear hyperplane

RESULT

The tests were carried out using the MATLAB 2022 software. The MATLAB code for the LM, BR, and RP training methods can be found in Table 1.The study used multilayer feed forward neural network topologies with a single hidden layer consisting of 10 default neurons. The input layer has a number of neurons that is equal to the number of characteristics in the data set. The final layer consists of two neurons, one representing the class of non-defective modules and the other representing the class of defective modules [12].

Table 1 Functions for implementing MATLAB

S. No.	Algorithm	MATLAB Implementation
1	Resilient back propagation	Trainlm
2	Resilient back propagation	Trainrp
3	Bayesian Regularization	Trainbr

Table 2 Accuracy

Dataset	LM	RP	BR
PC1	94.13	94.72	97.92

PC2	98.79	99.19	98.43
PC3	88.76	88.76	99.97
PC4	92.16	88.35	92.62
PC5	98.25	99.74	97.59
KC2	83.35	89.93	93.16
KC3	85.88	83.48	97.77

Table 3 Value of the Test Set's R Square

Dataset	**LM**	**RP**	**BR**
PC1	0.86	0.86	0.41
PC2	0.97	0.99	0.96
PC3	0.76	0.78	0.5
PC4	0.88	0.80	0.31
PC5	0.94	0.96	0.99
KC2	0.9	0.95	0.9
KC3	0.88	0.5	0.9

Table 4 Sensitivity

Dataset	**LM**	**RP**	**BR**
PC1	38.23	3.3	82.39
PC2	0	0	0
PC3	8.23	3.23	72.89
PC4	39.23	12.23	75.16
PC5	21.76	18.86	66.37
KC2	27.96	22.56	70.16
KC3	17.76	0	95.55

Table 5 Specificity

Dataset	**LM**	**RP**	**BR**
PC1	99.91	93.04	97.83
PC2	100	100	99.72
PC3	100	99.15	96.71
PC4	99.95	99.91	95.23
PC5	100	95.82	99.81
KC2	98.19	95.23	96.87
KC3	100	100	93.04

Table 6 MSE and RMSE

Dataset	**LM**		**RP**		**BR**	
	MSE	**RMSE**	**MSE**	**RMSE**	**MSE**	**RMSE**
PC1 PC2	0.07 0.03	0.26 0.17	0.05 0.02	0.22 0.14	1.11E- 0.0310	0 0.17
PC3	0.23	0.33	0.09	0.3	0.02	0.14
PC4	0.06	0.24	0.08	0.28	0.01	0.1
PC5	0.02	0.14	0.02	0.14	0.01	0.1
KC2	0.1	0.31	0.15	0.38	0.03	0.17
KC3	0.22	0.47	0.18	0.42	4.91E-	2.21E-

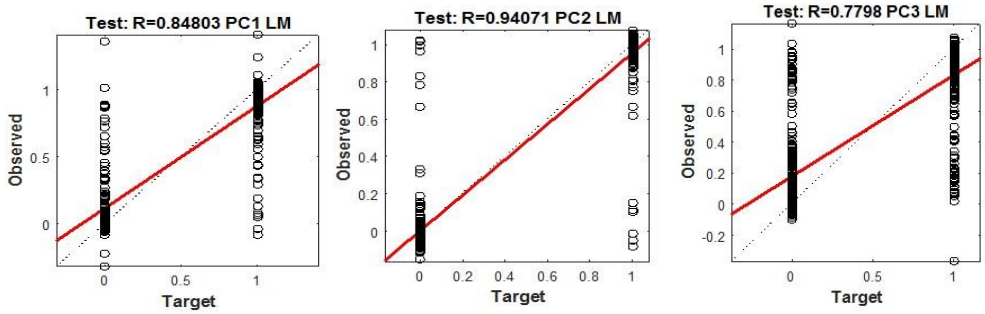
Table 7 False Negative Rate (FNR/Type II Error)

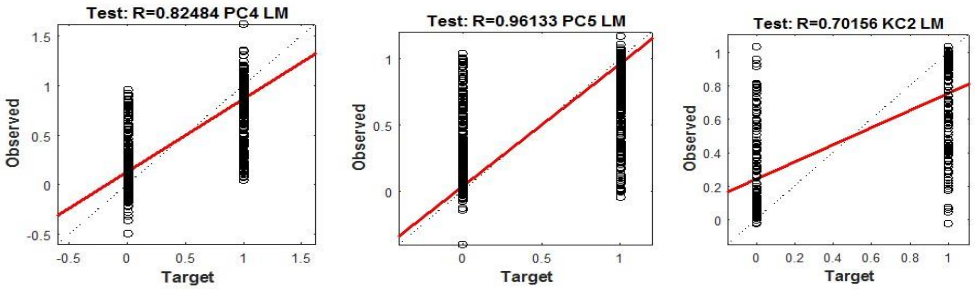
Dataset	**LM**	**RP**	**BR**
PC1	65.11	97.67	18.6
PC2	100	100	100
PC3	92.54	97.76	29.1
PC4	63.48	88.76	25.84

PC5	80.23	83.14	36.62
KC2	70.09	79.44	30.84
KC3	80.55	100	19.44

Table 8 False Positive Rate (Type I Error)

Dataset	**LM**	**RP**	**BR**
PC1	1.53	0.19	2.49
PC2	0	0	0.27
PC3	0.64	0.84	3.28
PC4	1.33	1.71	4.76
PC5	0.28	0.27	0.37
KC2	2.89	2.16	3.13
KC3	0.63	0	6.96





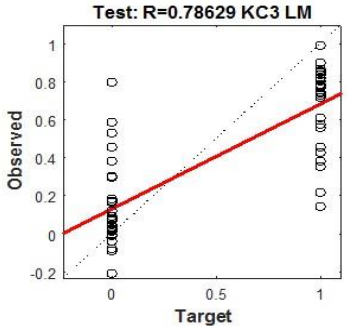
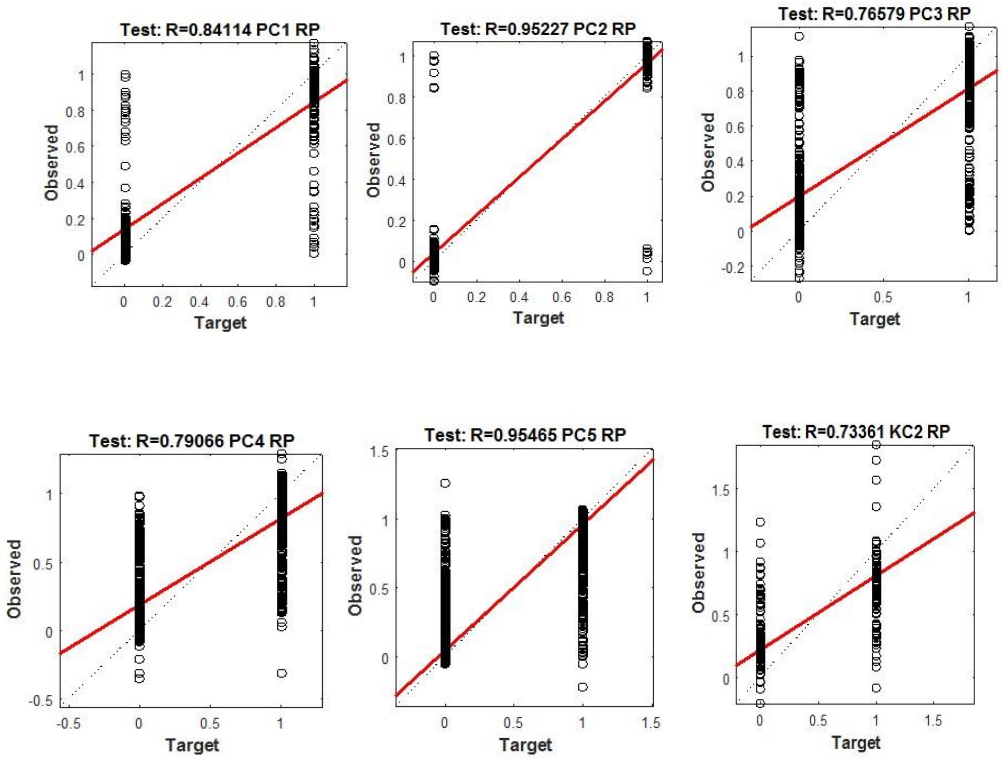


Figure 5 Regression plots of observed vs. target plots of LM algorithm for the seven datasets.



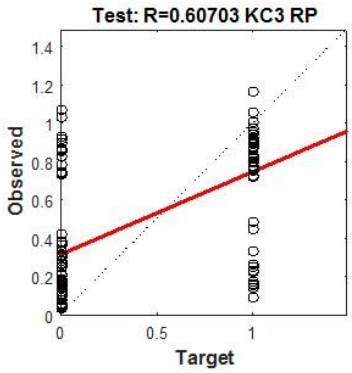
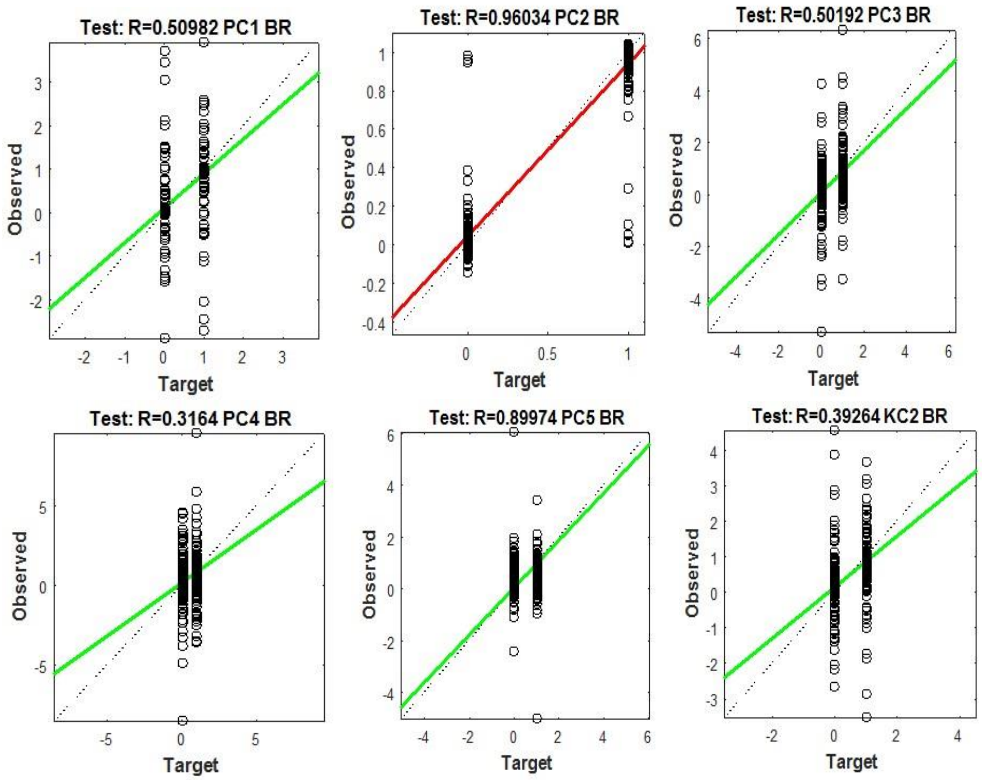


Figure 6 Regression plots were created to compare the observed values with the target values of the RP method for the seven datasets.



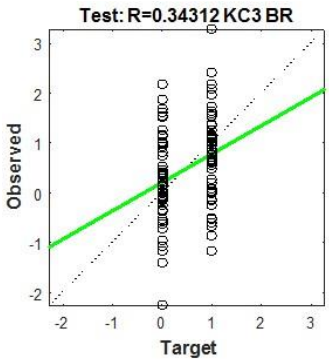


Figure 7 Regression graphs comparing the observed values to the desired values using the BR method for the seven datasets.

Discussion

Nevertheless, the R2 values exhibit complete inconsistency. A higher R2 result indicates a stronger match. LM and RP exhibited comparable performance, achieving an 84 percent data fit specifically for PC1. Overall, LM outperformed the other two equivalents in four out of seven data sets. Typically, a value of R2 greater than 0.9 is considered to indicate a strong fit [6]. LM and BR were found to be robust modelling strategies for the PC5 and PC2 defect datasets, respectively, based on this criteria.

The Bayesian based training function outperformed the LM and RP approaches in terms of accuracy and sensitivity parameters, achieving an accuracy of above 90 percent on all datasets. Furthermore, BR had the highest level of accuracy in six out of seven instances. Nevertheless, accuracy may often provide a misleading perception of performance as a result of the presence of imbalanced datasets with defects. While LM and RP demonstrated superior specificity compared to BR, specificity is not a crucial performance metric. Sensitivity is more significant since accurately classifying faulty modules is vital, rather than properly identifying non-defective modules [14].

CONCLUSION

This conducts an empirical research to compare the performance of three conventional back propagation based training algorithms, including Laverberg Marquardt, Resilient back propagation, and Bayesian Regularization, in the context of software fault prediction. A

multilayer feed forward artificial neural network was constructed using the MATLAB command line interface. Seven faulty data sets from the PROMISE repository were used in the experiments. The classification models were evaluated based on parameters derived from the confusion matrix and statistical metrics including Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R2 value. BR outperformed LM and RP in terms of MSE, R2 value, accuracy, recall, and false negative rate, according to a comprehensive comparison. The findings indicate that the context and criticality of the software project play a crucial role in helping project manager's priorities performance measurements and choose the appropriate training algorithm based on the available objectives and resources [15].

The back propagation training algorithms are a kind of optimisation algorithms that aim to optimize the weights in order to attain optimal performance. Studies have also shown the use of nature-inspired search-based optimisation algorithms in the domain of software engineering. Future work might include doing an empirical investigation of the back propagation learning functions and search-based approaches within the domain of software fault prediction.

To precisely measure the software defect prediction model's performance, one must carefully choose and comprehend the relevant metrics, and then gauge the model's effectiveness in light of the particular requirements of the software project. Different metrics may help understand the categorization performance, but they can also complicate the conclusion-making process. As previously said, it is not unexpected for disparate performance metrics to provide contradictory comparison outcomes. Previous investigations have also noted a similar tendency. In theory, the confusion matrix is used to construct the performance indices, and this is a straightforward process. In actuality, however, the prediction models' comparisons are only meaningful if the performance metrics are intimately connected to the project's particular needs.

We came to the conclusion that there are several dimensions to the software defect prediction issue and that it is unusual for a single model to perform optimally across all software quality situations. It becomes clear from this that the objective is to increase both the effectiveness of software verification processes and the classifier's performance. This prompts us to think about and look into more cost-sensitive variables in subsequent work, including the F-measure and its numerous derivatives.

REFERENCES

[1] Y. Singh, "Software testing," Cambridge University Press, 2012.

[2] A. Singh and R. Malhotra, "Object oriented software engineering", PHI Learning, India, 2012.

[3] S. Henry and D. Kafura, "Software structure metrics based on information flow", IEEE Transactions on Software Engineering, vol. 7, no. 5, pp. 510–518, 1981.

[4] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design", IEEE Transactions on Software Engineering, vol. 20, pp. 476-493, 1994.

[5] M. H. Maurice, "Elements of software science", Elsevier, 1977.

[6] T. M. Khoshgoftaar, K. Gao, R. M. Szabo, "An application of zeroinflated Poisson regression for software fault prediction", Proceedings of 12th International Symposium on Software Reliability Engineering, Hong Kong, China, pp. 66-73, 2001.

[7] V. U. B. Challagulla, F. B. Bastani, I-L. Yen and R. A. Paul, "Empirical assessment of machine learning based defect prediction techniques", International Journal on Artificial Intelligence Tools, vol. 17, no. 2, pp. 389-400, 2008.

[8] K. O. Elish, M. O. Elish, "Predicting defect-prone software modules using support vector machines", Journal of Systems and Software, vol. 81, no. 5, pp. 649-660, 2008.

[9] C. Elken, "The foundations of cost-sensitive learning", in 17th International Joint Conference on Artificial Intelligence (IJCAI'01), Seattle, WA, USA, pp. 973-978, 2001.

[10] V. Chandwani, V. Agrawal and R. Nagar, "Modeling slump of ready mix concrete using genetic algorithms assisted training of artificial neural networks", Expert Systems with Applications, vol. 42, no. 2, pp. 885-893, 2015.

[11] R. S. Rathore and S. Kumar, "Predicting number of faults in software system using genetic programming", International Conference on Soft Computing and Software Engineering, Berkeley, pp. 303-311, 2015.

[12] J. Kennedy and R. Eberhart, "Particle swarm optimisation", IEEE International Conference on Neural Networks, Australia, pp. 19421948, 1995.

[13] L. J. Fogel, A. J. Owens, and M. J. Walsh, "Artificial intelligence through simulated evolution, Wiley, New York, 1966.

[14] L. C. Briand, P. Devanbu, and W. Melo, "An investigation into coupling measures for C++," In the Proceedings of the ICSE 97, Boston, USA, 1997.

[15] J. Bieman, and B. Kang, "Cohesion and reuse in an object-oriented system," In Proceedings of the CM Symposium on Software Reusability, pp. 259-262, 1995.