JOURNAL OF SOFTWARE ENGINEERING & INTELLIGENT SYSTEMS
ISSN 2518-8739
30 April 2020, Volume 5, Issue 1, JSEIS, CAOMEI Copyright © 2016-2020
www.jseis.org

SOFTWARE FAULT PREDICTION BASED ON RANDOM FOREST ALGORITHM

TAHMINEH SADAT MOOSAVI [1], KEYVAN MOHEBBI [2]
[1,2]Islamic Azad University, Mobarakeh, Isfahan, Iran
Email: k.mohebbi@mau.ac.ir [2]

Academic Editor: Sajid Khan, APCOMS, Rawalpindi.
Published: 30 April, 2020
Funding: Authors have no funds.
Article Type: Research Article

ABSTRACT

Software fault prediction is the process of identifying the faulty modules that helps to reduce the complexity of the test phase. The selection of proper features from the source code is the most important step in this process. These features, called software metrics, are a measure of the characteristic of a piece of software that gives us an overall view of the software information. According to the literature, there is still no single set of features that are related to software faults. In addition, improvement of the fault prediction is still a challenge. This research proposes a new approach to software fault prediction. First, the existing data are divided into testing and training categories using the k-fold technique and each time these categories are randomly changed. Then, the optimal features related to the software faults are selected using the F-score and a new dataset is constructed with these features. Finally, the model is classified and trained using the random forest algorithm. The proposed approach is implemented and evaluated via the NASA dataset. The results indicate that this approach has precision of 90.7%, accuracy of 91.5%, sensitivity of 92%, and F-score of 91.8%.

Keywords: software fault prediction; random forest; feature selection; classification; software metrics; software quality;

1. INTRODUCTION

Software quality is very important in the field of software engineering. On the other hand, making high-quality software is very expensive. In order to increase the quality of the software and the test phase, we are interested in eliminating the faults in the initial stage [1]. A fault or defect is an incorrect definition of information in a program that causes the software to operate in an unwanted or unplanned state. Software fault generally refers to an issue that is related to the software products, their external behavior, or even their intrinsic properties. Many causes can lead to faults. Coding faults, inaccurate identification of requirements, programmer faults, program execution environment such as operating system or hardware, and etc. [2]. The cost of the software fault is high. It includes the cost of finding and correcting faults and testing the modified software. According to a Cambridge University report, software developers spend about 50 percent of their time to find and correct defects, which accounts for approximately 312 million dollars each year [3].

Fault prediction means detecting the potential faults that can occur in the future [4]. The purpose of the software fault prediction is to recognize the fault-prone modules by using some of the basic software features before the testing process begins. This will help improve the quality of the software [5]. The presence of faults not only decreases the quality of software but also increases the cost of software development and maintenance. Therefore, recognizing which software module is vulnerable to fault during the initial phase of software development, helps improve the quality of the software [6]. The role of fault prediction is the detection of faulty modules before the test phase, using some of the structural features of the software system. The software fault prediction model is mainly constructed based on the information extracted from the previous projects and then used to predict the current faults while developing the software.

So far, many approaches have been proposed to predict software faults. A number of such approaches rely on decision trees. The decision trees are one of the efficient means for both classification and estimation problems. However, they are prune to overfitting and have limited generalization. Generalization refers to the ability of the model to conform well with the new data while having the least prediction error. Another disadvantage of decision trees is the instability of their results in the presence of data noise. To overcome these shortcomings, the random forest algorithm is introduced which is an ensemble learning technique based on a group of decision trees with different training models. This paper proposes a new approach to software fault prediction. It first processes the dataset to extract and select the best features related to the faults. Then, it uses random forest algorithm to learn and predict the software faults.

Corresponding Author: Keyvan Mohebbi
Email Address: k.mohebbi@mau.ac.ir

The rest of this paper is organized as follows. Section 2 reviews the related works. Section 3 describes the proposed approach for software fault prediction. Section 4 evaluates this approach. Section 5 concludes the paper.

2. RELATED WORKS

In recent years, different approaches have been proposed to predict software faults. Tosun et al. (2008) improved the accuracy of software fault prediction significantly by combining the Naïve Bayes, neural networks, and voting classifiers [7]. Zheng et al. (2010) focused their study on three cost-sensitive boosting algorithms to increase the accuracy of neural networks [8]. Wang et al. (2010) noted there have several versions of defect predictor based on Naïve Bayes theory and analyzed their difference estimation method and algorithm complexity via the MDP dataset. In this comparison, Multi-variants Gauss Naïve Bayes (MvGNB) showed better results than other methods, such as decision tree learner J48 [9].

Wang et al. (2011) in another large-scale empirical study of supervised learning algorithms concluded that ensemble decision trees overcome binary classifiers. Forest rotation has achieved remarkably significant results in classification [10]. Fault distribution is one of the important factors that can affect prediction performance, such as an unbalanced dataset in which most modules are fault-free, while the few faulty modules are more important. To overcome this problem, Shatnawi et al. (2012) proposed a sampling approach. This approach is applied to an object-oriented system from the Eclipse dataset using three classifiers: Naïve Bayes, Bayesian network, and k-nearest neighborhood. The results showed that the predictability is better than the traditional sampling approaches [11].

Most of the works in literature use past project data to build prediction models. Mizuno et al. (2013) proposed an approach that uses the textual properties of the module code instead of other metrics. Here, a large set of metrics is constructed that represents the frequency of words in the module code. The main idea is based on the filtering of spam emails, as they usually contain a specific pattern of words and sentences. From the source code perspective, similar conditions are generally found in faulty modules. This study also compared the Bernoulli multivariate and the polynomial models [12].

Gayathri et al. (2014) investigated the improved multi-layer perceptron neural network on NASA collection. In this paper, the fuzzy system is used in the middle layer of the neural network and according to the experiments has promising results. However, today statistical learning algorithms combine multiple learners' predictions, which is called ensemble learning and outperforms single learners and many studies prove this [13].

Suresh et al. (2014) examined the application of statistical methods such as linear regression, logistic regression, and machine learning methods such as neural network (and its different forms) for software fault prediction using Chidamber and Kemerer (CK) metrics. Here, a fault is considered as a dependent variable and CK metric suite as independent variables. The comparison approach was applied for Apache integration framework (AIF). The analysis shows that the hybrid approach of radial basis function network obtained better fault prediction rate when compared with the other three neural network models [14].

Moeyersoms et al. (2015) compared 32 different types of Bayesian networks to prediction via AUC and H criteria. The experimental results showed that the improved Naïve Bayes has the greater or equal performance to the Naïve Bayes classifier [15]. Mahajan et al. (2015) examined the Bayesian Regularization (BR) to detect the faults and reduce the cost of the test phase. The BR technique produces the best combination to generate an efficient network based on neural networks. This technique is compared with the Levenberg-Marquardt and backpropagation algorithms and outperforms both of them. Precise measurement and accurate prediction is a very important issue in any software. Because measurement varies based on different criteria, a proper subset of criteria that are more relevant to fault prediction should be selected [16]. Several articles deal with the selection of metrics. In their research, Arora et al. (2015) used the F-score to select effective metrics in fault prediction, based on the Twin Support Vector Machine (TSVM) method. The proposed approach was evaluated using the NASA dataset. The results showed that the prediction efficiency increased with decreasing the set of metrics through feature reduction [17]. Kumari et al. (2015) used the association rule mining to get the best object-oriented metrics, via the Apriori algorithm. The experiments were conducted on the Eclipse dataset and showed that the approach discovers the best rules and performs well on large scale databases. One of the disadvantages of this approach is that it might discover lots of rules which are confusing [18]. Singh et al. (2015) showed that supervised learning techniques outperform random forest and logistic regression in fault prediction. They used the Eclipse dataset and the results were promising [19].

de Assis Boldt et al. (2015) in their work relied on the cascade feature selection. Three cascading combinations have been tested with the extreme machine learning technique as the base architecture for classification. The combination of mixed ranking, sequence feature selection, and genetic algorithms performed better. Experimental results showed that this approach generates smaller subsets, consumes less time, and performs better classification than using the genetic algorithm alone for feature selection [20].

Liu et al. (2015) proposed a two-stage data preprocessing approach which incorporates both feature selection and instance reduction. In the feature selection stage, they first performed relevance analysis, and then proposed a threshold-based clustering method. In the instance reduction stage, they applied random under-sampling to keep the balance between the faulty and non-faulty instances. For evaluation, they chose datasets from real-world software projects, such as Eclipse and NASA. The comparison between this approach and some classical baseline methods demonstrated the effectiveness of the proposed work [21].

Blaser et al. (2016) introduced an extension of random forest classifier, namely random forest rotation, which has not been used to predict software fault. This approach has better results on numerical than nominal data. In this study, a multidimensional and scalable pre-processing strategy is presented to reduce the complexity of the dimensions of the software features. The results show that the semi-supervised learning algorithm with feature reduction is significantly better than the random forest that is one of the best supervised algorithms. This approach is suggested when there is not enough data to train the model [22]. Hamill et al. (2017) analyzed the necessary tasks to resolve a software fault and how they are affected by different factors. In addition, they predicted the effort required to resolve a fault based on the information obtained from the software changes. This analysis is based on data from 1,200 failures in a tracking system of a major mission in NASA. The predictions are performed using three supervised machine learning algorithms and three sampling techniques [23].

Begum and Dohi (2017) proposed a data-agnostic approach to predict software fault. They used the multilayer perceptron neural network to classify software faults and thus increased the ability to create various classes. But it may have computational complexity [24]. Muhamad et al. (2017) divided their work into two main parts: feature selection and software classification. They tested their approach via the NASA dataset. The results showed that feature selection could increase the predictability of the classifier [25]. Ranjan et al. (2017) reviewed the works on software fault prediction. The comparative evaluation showed that among various features, v(G), ev(G), iv(G), and Loc which are all dependent on the programming language, have more important roles in fault prediction [26]. Choudhary et al. (2018) tried to find the most important features to detect and predict faults. They modified the features of each software and transformed them into effective features to examine the software code. Then, they examined the application of decision tree, random forest, and nearest neighbor algorithms on the modified features [27]. Singh and Sharma (2018) evaluated various works and intended to find the best approach to software fault prediction. They used 10 data sets for this evaluation. The results showed that the superiority of the backpropagation approaches [28]. Maddipati et al. (2018) believed that the NASA dataset is unbalanced and it is important to preprocess it. They used an enhanced fuzzy approach to normalize the data and thus improving the evaluation criteria [29].

3. PROPOSED APPROACH

In order to software fault prediction, this research relies on the random forest that is an ensemble learning algorithm. This algorithm is used for both classification and regression purposes. It trains various decision trees. Then predict or classify a new data item via voting the trained trees. The proposed approach is described as follows.

After reading the dataset of software modules, it is necessary to prepare the text data. The purpose of this phase is to extract and initialize the proper features. First, the data is divided into training and testing categories using k-fold cross-validation. Then, the most proper data is selected based on F-score to learn the model. To do this, a prediction is made for each row of the testing data and the number of both correct and incorrect predictions are identified from the predicted and actual results. A matrix is constructed in which each row and column represent the actual and predicted values for a classification, respectively. This matrix is indeed the data set of feature vectors. The values of such vectors are used to calculate the precision and f-score of the prediction. Those features that have the highest frequency in the top-ranked precision and f-score are then selected to construct the training model.

After pre-processing, the software will be classified. In this research, the random forest algorithm that is an ensemble learning technique is used for classification. Here, two parameters are determined by the user. The first parameter is the number of decision trees to be made in the forest. This parameter varies based on the number of training data and available features. The second parameter is the set of features that should be selected randomly when the failure feature in each tree node is determined. The optimal feature for data classification will be selected from such a set. Also, this parameter is constant in the whole process of training the random forest and for all trees. The usual values considered for this parameter are *Log(nVariable) + 1* and *Sqrt(nVariable)*, where *nVariable* denotes the number of features in the data set.

The developing process of the training model is described below:

a) For each decision tree, a subset of the initial training data which contains the optimal features determined by the f-score is selected.

b) To estimate the error rate, in each training subset, a part of the data is set aside. This data is called out-of-bag (OOB). The calculated error using this data is a bias-free estimation for the generalization error.
c) In each training subset, a decision tree is developed. In each node, instead of the best failure feature, the top *m* features with the highest f-score are selected. Among these *m* features, the best one is selected as the failure feature.
d) The test data is applied to all trees. The majority of trees' voting is considered as the final data tag.
e) Steps 1 through 4 are applied to all of the training subsets.

The characteristics of the proposed approach are summarized as follows:

- It is based on ensemble learning which has a promising learning and extending capability.
- It resists the over-fitting problem. This is because of the random policy in selecting both the training subsets and the features.
- It can provide an internal bias-free estimation of the generalized fault during the training phase.
- It calculates the locality (or similarity) between any pair of data. This can be effective in clustering, estimating missing values, and providing a general view of data.

4. IMPLEMENTATION AND EVALUATION

Four databases: KC1, KC2, CM1, and PC1 of NASA dataset[1] are used to evaluate the proposed approach. The results of feature extraction and selection for these databases are shown in Table 1.

Table. 1 Selected features of databases

Database	KC1	KC2	CM1	PC1
Selected Features	Locoed	ev(g)	Locoed	D
	ev(g)	Loc	I	Locodeandcomment
	B	I	E	L
	Loc	E	T	uniq_opnd
	I	T	uniq_opnd	iv(g)
	T	uniq_op	Loc	M
	Locomment	total_opnd	v(g)	V
	Locodeandcomment	v(g)	V	T
	uniq_opnd	iv(g)	D	Locoed
	v(g)	M	Locomment	Locomment
	M	Loblank	Loblank	total_opnd
	L	uniq_opnd	uniq_op	Branchcount
	Loblank	L	total_opnd	Loc
	uniq_op	D	ev(g)	v(g)

The database columns that are related to each selected feature are used as the dataset for the next step, i.e. classification. In order to classify the software, the training model is first created and after training, is used to classify a new sample into correct or defective categories.

A comparative evaluation is performed between the proposed approach and the similar works in [14], [15], [16], and [21]. Table 2 shows the results with respect to four well-known criteria, namely precision, sensitivity, accuracy, and f-score.

Table. 2 Evaluation results

Approach	Evaluation Criteria (percent)			
	Precision	Sensitivity	Accuracy	F-score
[14]	81.7	61.1	84	61
[15]	87.4	86.5	84.4	86.6
[16]	88	83.4	90.4	90
[21]	89.1	89.9	89.7	91
Proposed	**91.5**	**92**	**90.7**	**91.8**

As can be seen, the proposed approach outperforms similar works with respect to evaluation criteria. The reason is that for selecting the most effective features, the datasets of the training and testing phases, are changed

[1] https://figshare.com/collections/NASA_MDP_Software_Defects_Data_Sets/4054940

randomly, using the k-fold technique. Then, the f-score criteria is used on the random datasets to select the best features.

5. CONCLUSION AND FUTURE WORKS

This paper proposed an approach to software fault prediction. It is based on the random forest algorithm that is an ensemble learning technique. The main idea of such techniques is that a group of weak learners can make a strong one. Although it is not the first time to rely on the random forest for software fault prediction, the difference between the proposed approach and the similar works is that to select the most proper features, it randomly changes the training and testing sets each time. These sets are then used to identify the features related to software faults. This process starts over and over again, while the optimal features are determined as those with high frequency in the iterations. The empirical evaluation indicated promising results.

For the future, we consider the following paths to extend this work. To improve the common evaluation criteria, the number of decision trees may be increased. Besides, the combination of other classifiers such as support vector machines, boosted trees, or neural networks can be examined. Moreover, to reduce the time of the training phase, the trees can be trained in parallel.

REFERENCES

1. Czibula, G., Z. Marian, and I.G. Czibula, *Software defect prediction using relational association rule mining.* Information Sciences, 2014. **264**: p. 260-278.
2. Munson, J.C., A.P. Nikora, and J.S. Sherif, *Software faults: A quantifiable definition.* Advances in Engineering Software, 2006. **37**(5): p. 327-333.
3. Gotthans, T., J.C. Sprott, and J. Petrzela, *Simple chaotic flow with circle and square equilibrium.* International Journal of Bifurcation and Chaos, 2016. **26**(08): p. 1650137.
4. Zhou, Z.-Q., Q.-X. Zhu, and Y. Xu, *Time Series Extended Finite‐State Machine‐Based Relevance Vector Machine Multi‐Fault Prediction.* Chemical Engineering & Technology, 2017. **40**(4): p. 639-647.
5. Rathore, S.S. and S. Kumar, *A study on software fault prediction techniques.* Artificial Intelligence Review, 2019. **51**(2): p. 255-327.
6. Rathore, S.S. and S. Kumar, *An empirical study of some software fault prediction techniques for the number of faults prediction.* Soft Computing, 2017. **21**(24): p. 7417-7434.
7. Tosun, A., B. Turhan, and A. Bener. *Ensemble of software defect predictors: a case study*. in *Proceedings of the Second ACM-IEEE international symposium on Empirical software engineering and measurement*. 2008.
8. Zheng, J., *Cost-sensitive boosting neural networks for software defect prediction.* Expert Systems with Applications, 2010. **37**(6): p. 4537-4543.
9. Wang, T. and W.-h. Li. *Naive bayes software defect prediction model*. in *2010 International Conference on Computational Intelligence and Software Engineering*. 2010. Ieee.
10. Wang, T., et al., *Software defect prediction based on classifiers ensemble.* Journal of Information & Computational Science, 2011. **8**(16): p. 4241-4254.
11. Shatnawi, R. *Improving software fault-prediction for imbalanced data*. in *2012 international conference on innovations in information technology (IIT)*. 2012. IEEE.
12. Mizuno, O. *On effects of tokens in source code to accuracy of fault-prone module prediction*. in *2013 International Computer Science and Engineering Conference (ICSEC)*. 2013. IEEE.
13. Gayathri, M. and A. Sudha, *Software defect prediction system using multilayer perceptron neural network with data mining.* International Journal of Recent Technology and Engineering, 2014. **3**(2): p. 54-59.
14. Suresh, Y., L. Kumar, and S.K. Rath, *Statistical and machine learning methods for software fault prediction using CK metric suite: a comparative analysis.* ISRN Software Engineering, 2014. **2014**.
15. Moeyersoms, J., et al., *Comprehensible software fault and effort prediction: A data mining approach.* Journal of Systems and Software, 2015. **100**: p. 80-90.
16. Mahajan, R., S.K. Gupta, and R.K. Bedi, *Design of software fault prediction model using BR technique.* Procedia Computer Science, 2015. **46**: p. 849-858.
17. Arora, I., V. Tetarwal, and A. Saha, *Open issues in software defect prediction.* Procedia Computer Science, 2015. **46**: p. 906-912.
18. Kumari, D. and K. Rajnish, *A new approach to find predictor of software fault using association rule mining.* Int. J. Eng. Technol, 2015. **7**(5): p. 1671-1684.
19. Singh, P.K., R. Panda, and O.P. Sangwan, *A critical analysis on software fault prediction techniques.* World applied sciences journal, 2015. **33**(3): p. 371-379.

20. de Assis Boldt, F., T.W. Rauber, and F.M. Varejão. *Single sequence fast feature selection for high-dimensional data*. in *2015 IEEE 27th International Conference on Tools with Artificial Intelligence (ICTAI)*. 2015. IEEE.
21. Liu, W., et al., *Empirical studies of a two-stage data preprocessing approach for software fault prediction.* IEEE Transactions on Reliability, 2015. **65**(1): p. 38-53.
22. Blaser, R. and P. Fryzlewicz, *Random rotation ensembles.* The Journal of Machine Learning Research, 2016. **17**(1): p. 126-151.
23. Hamill, M. and K. Goseva-Popstojanova, *Analyzing and predicting effort associated with finding and fixing software faults.* Information and Software Technology, 2017. **87**: p. 1-18.
24. Begum, M. and T. Dohi, *A neuro-based software fault prediction with Box-Cox power transformation.* Journal of Software Engineering and Applications, 2017. **10**(03): p. 288.
25. Muhamad, F.P.B., D.O. Siahaan, and C. Fatichah, *Software fault prediction using filtering feature selection in cluster-based classification.* IPTEK Journal of Proceedings Series, 2018(1): p. 59-64.
26. Ranjan, P., S. Kumar, and U. Kumar, *Software fault prediction using computational intelligence techniques: A survey.* Indian Journal of Science and Technology, 2017. **10**(18).
27. Choudhary, G.R., et al., *Empirical analysis of change metrics for software fault prediction.* Computers & Electrical Engineering, 2018. **67**: p. 15-24.
28. Singh, J. and S. Sharma, *Fault detection technique for test cases in software engineering.* Int. J. Eng. Technol, 2018. **7**(1): p. 53.
29. Maddipati, S.S., G. Pradeepini, and A. Yesubabu, *Software Defect Prediction using Adaptive Neuro Fuzzy Inference System.* International Journal of Applied Engineering Research, 2018. **13**(1): p. 394-397.

AUTHORS PROFILE