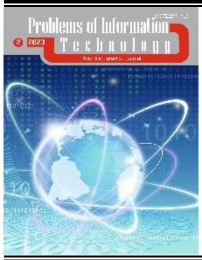

Available online at www.jpit.az

14 (2)
2023

# Software defect prediction using the machine learning methods

**Tamilla A. Bayramova**

Institute of Information Technology, B. Vahabzade str., 9A, AZ1141 Baku, Azerbaijan

tamilla@iit.science.az

0000-0002-8377-3572

ARTICLE INFO

http://doi.org/10.25045/jpit.v14.i2.03

*Article history:*
Received 12 January 2023
Received in revised form 28 February 2023
Accepted 21 April 2023

**Keywords:**
Random Forest
Naive Bayes
Bagging
Boosting
Ensemble
Software defect prediction

ABSTRACT

Reliability of software systems is one of the main indicators of quality. Defects occurring when developing software systems have a direct effect on reliability. Precise prediction of defects in software systems helps software engineers to ensure the reliability of software systems and to properly allocate resources for the trial process. The development of an ensemble method by combining several classification methods occupies one of the main places in research conducted in the field of error prediction in software modules. This paper proposes a method based on the application of ensemble training for defect detection. Here, a database obtained from PROMISE and GITHUB software engineering registry is used to detect defects. Experiments are conducted using Weka software. The prediction efficiency is evaluated based on F-measure and ROC-area. As a result of experiments, the defect detection accuracy of the proposed method is proven to be higher than that of individual machine learning methods.

## 1. Introduction

The rapid development of information technologies has posed high demands on hardware and software. Developing quality software systems and meeting customer requirements is one of the most important issues for the software industry. Anyone using a software system should be guaranteed that the system will operate at a certain level of reliability. A precise assessment of reliability gives both developers and users some confidence about the successful operation of software system. Currently, the development of reliable software systems is one of the most urgent problems, however time and budget constraints create great problems in achieving this goal. Furthermore, the complexity of software systems continues to increase, which leads to an increase in the number of defects in them. These defects are mainly due to incorrect operation of the software system or incorrect specification. One of the most important and expensive processes in the development of software systems is the detection and correction of defects that may occur under different conditions.

## 2. Prediction of software defects

Software trial is a key tool to ensure its reliability, however it is impossible to detect all defects in the testing process. Moreover, the testing process requires huge effort, expense and proficiency in the relevant field. The costs, time and effort involved in security-critical software systems increase even further. Therefore, the development of a good testing strategy is one of the difficult issues facing software engineers (Kazimov,. Bayramova & Malikova, 2021).

Along with supporting the testing process, software defect prediction methods play an important role in developing more reliable software products and speeding time for launching them on the market. Software defect prediction is a model developing process that will be used to detect defective elements (modules or classes) in the early stages of the software life cycle. As a result of the classification of the modules, it is determined whether it is defective or not. Modules classified as defective are checked first and more rigorously in the testing process, while those recognized as non-defective are checked if time and resources remain (Thota, Shajin & Rajesh, 2020). Software defect prediction is referred to as a step in improving the software system reliability and enables software developers and testers to discover which modules of the system are most disposed to defects. Careful inspection of these modules during testing directly affects the timely detection and elimination of defects (Menzies et al, 2010). Forecasting leads to reduced software testing costs, timely software product development, and improved overall quality (Kaur & Kaur, 2014).

Software defect prediction is the detection of defective software modules with the application of machine learning methods and is performed by two strategies: regression and classification. Regression methods aim to predict the number of defects in a software system. Numerous studies have been conducted in this area using a number of regression models (Alsaeedi & Zubair Khan, 2019; Yan, Chen & Guo, 2010; Rathore & Kumar, 2016; Rathore & Kumar, 2017).

Classification methods determine whether the software module is defective or not. Classification models learn from known software defects of the previous edition. The learned models are applied to predict the remaining potential defects in the software system (Wang, 2014).

Different types of machine learning classifiers are applied to predict software defects. They can be grouped into three main categories: supervised learning, unsupervised learning, and semi-supervised learning (Chug & Dhall, 2013; Malhotra, 2015).

Various machine learning methods are used to improve the software defect prediction. Research shows that the most commonly used methods in this field are:

- Decision Trees (Zhang, Jing & Wang, 2017);
- Bayesian learners (BL) (Jayanthi & Florence, 2019);
- Neural networks (NN) (Jin C., Jin Sh.. , 2015; Kanmani et al, 2007);
- Support vector machines (SVM) (Qiao et al, 2020);.
- Rule based learning (RBL) (Singh et al, 2017);
- Ensemble learners (EL).

Predictive models can be built based on various metrics taken from the software's source code. A software system metric is an indicator describing a specific software feature. Various approaches have been proposed to predict software system defects based on software metrics in recent years (Ge, Liu & Liu, 2018). These metrics are used to evaluate the reliability of software systems, detect defects, monitor and manage software projects (Zhang, 2009; Kazimov & Bayramova, 2022). The most commonly used metrics for defect prediction are listed below (table 1):

**Table 1.** Software metrics most commonly used in predicting software defects

| | Metrics | Description | |
|---|---|---|---|
| 1. | LOC | Number of code lines | MakKeyb |
| 2. | v(g) | Cyclomatic complexity | |
| 3. | ev(g) | Basic complexity | |
| 4. | iv(g) | Design complexity | |
| 5. | n | Total number of operators and operands | Halstead |
| 6. | v | Volume | |
| 7. | l | Software length | |
| 8. | d | Difficulty | |
| 9. | i | Intelligence | |
| 10. | e | Effort | |
| 11. | b | Error | |
| 12. | t | Time | |
| 13. | lOCode | Number of code lines | |
| 14. | lOComment | Number of comment lines | |
| 15. | lOBlank | Blank lines | |
| 16. | lOCodeAnd Comment | Number of comment and code lines | |
| 17. | uniq_Op | Number of unique operators | |
| 18. | uniq_Opnd | Number of unique operands | |
| 19. | total_Op | Number of total operators | |
| 20. | total_Opnd | Number of total operands | |
| 21. | branchCount | Number of total graph branches | |

## 3. Releated work

Currently, the analysis and prediction of defects in software systems has become one of the main trends in the field of software engineering. Software defect prediction using machine learning is one of the dynamic research areas in software systems reliability assurance. Research in this field has started since the 1990s (Malhotra, 2015).

Ensemble training has performed effectiveness in predicting software system defect and promises more positive results than individual classifiers.

The authors in their paper (Aljamaan & Alazba, 2020) conducted research to test the efficiency of tree structures ensemble methods, which have not been widely applied in software systems forecasting. Here, experiments are conducted on 11 datasets from the NASA MDP database of software system defects. Random Forest and Extra Trees, XGBoost, Cat-Boost, Gradient Boosting and Hist Gradient Boosting and AdaBoost ensemble algorithms are used for forecasting. Random forest and Extra trees show better performance. The worst performance is shown by AdaBoost algorithm.

(Catal & Diri, 2009) conducts experiments on a dataset taken from NASA PROMISE repository. These experiments show that the Random Forest algorithm performs better when the data set is large (when the number of modules is large), and the Naive Bayes algorithm performs better on small data sets. This experiment shows that the most important factor in predicting software system defects is not software performance, but the correct selection of the algorithm.

In (Alazzam, Alsmadi & Akour, 2017), researchers compare Bagging, Boosting and Stacking ensembles. Naive Bayes, Bayes Network, SMO, PART, J48, Random Forest, Random Tree, IB1, Decision table and NB tree are taken as basis algorithms. Their experimental results show that Boosting perform better than Bagging, and Stacking and Random Forest perform well. In order to increase the efficiency of defect detection, they recommend combining the Random Forest algorithm in the ensemble, which performs better than other base algorithms.

(Li et al, 2019) focuses on combining different machine learning algorithms to predict software defects. MDP data set is used as experimental data. Calculations are performed using 5 different ensemble algorithms, and as a result of the comparative analysis, the Random Forest algorithm is concluded to show better performance.

(Matloob et al, 2021) provides a review of the literature on the application of the ensemble algorithm in software defect prediction. The review is based on scientific articles published in four online libraries (ACM, IEEE, Springer Link, and Science Direct) since 2012. This review addresses five questions covering various aspects of the application of ensemble training algorithms to predict software defects. It provides a brief summary of the latest trends and achievements in ensemble training for software system defect prediction and lays some foundation for future innovation and analysis. As a result of the research, it is concluded that the most commonly used ensemble algorithms are Random Forest, Bagging, Boosting, and the least used ones are Stacking, Voting and Extra Trees. AUC, accuracy, F-measure, Recall, Precision, and MCC are mainly used to test the prediction efficiency of the models. The most used machine learning platform for conducting experiments is WEKA. Engineer programmers use data collected in the PROMISE repository, created in 2005, when predicting defects in software systems. Here, data collected based on NASA software systems and defects occurring in them are stored in the ARFF format. This allows them to be analyzed using machine learning tools.

(Hussain et al, 2015) explores AdaboostM1, Vote and StackingC ensemble algorithms by selecting Naive Bayes, Logistic, J48, Voted Perceptron and SMO at base level in Weka machine learning tool. 12 data sets taken from the PROMISE database are used to test the efficiency of the ensemble algorithms. This experiment uses 10-fold cross-validation and ROC analysis to evaluate the efficiency of the algorithm. Besides, recall, precision, accuracy, F-measure are also used to check the efficiency of base classifiers and ensemble algorithms. As a result, the Stacking algorithm is found to perform better than other methods and base classifiers.

(Bowes & Hall, 2018) studies the specific defects detected by four classifiers. The performances of Random Forest, Naïve Bayes, RPart and SVM classifiers are explored on different datasets taken from NASA, open and commercial databases. Even though the prediction performance is approximately the same, each classifier detects a different set of defects. Therefore, applying voting-based ensemble methods to find defects in the software system can provide better performance.

## 4. Ensemble training methods

In machine learning problems, the main goal is to find a single model that is capable to predict the expected result more accurately. Machine learning algorithms can provide different predictions even when the model is trained on the same data. This is called the dispersion of the predictions or the stability of the model (Zhang & Ma, 2012).

Ensemble training model enables to create a reliable model by combining several machine learning classifiers to increase the prediction efficiency and get a more accurate result. Ensemble models are ideal for regression and classification, increasing model accuracy and minimizing dispersion.

Certain errors of individual classifiers may lead to deficiencies in their predictions under certain circumstances (Cortes et al, 2008; Stapor, 2017). Therefore, they combine several classifiers to take advantage of their strengths. In recent years, researchers have proven experimentally that the ensemble method performs better than individual classifiers (Rokach, 2009; Rodriguez, Kuncheva & Alonso, 2006).

Ensemble training models can be homogeneous and heterogeneous. The homogeneous ensemble uses the same algorithm in the base training. The homogeneous approach trains a base method with different subsets of the training data and makes a decision. Whereas, the heterogeneous ensemble uses different base algorithms. The heterogeneous approach uses different base models, but the same training data (Rokach, 2010). The most commonly used ensemble methods for predicting defects in software systems are Bagging, Boosting, Random Forest and Voting methods.

In the case of classification, ensemble training is performed in two stages:

- Training of base classifiers;
- Calculating the average value based on the output data of the base classifiers or obtaining a general result by voting.

There are three main classes of ensemble training methods: boosting, bagging, and stacking.

1. The main elements of ***BAGG**ing* (***B**ootstrap**AGG**regating*) ensemble model can be explained as follows:

- Initial subsets are loaded from the training dataset;
- A decision tree is formed for each subset of loaded data;
- Based on the decisions, the final decision is made by voting or calculating the average value (figure 1.).

2. ***Stacking*** typically has a two-layer (maybe more) hierarchy. On the 1st layer, the models selected for the ensemble are placed, on the 2nd layer, the final forecasting model based on the predictions of these models is placed (Figure 2).

3. The models included in the ***Boosting*** ensemble are sequentially added. Each model corrects the forecast of the previous model and calculates the weighted average value of the forecast. Based on these values, the forecast is calculated (Figure 3).

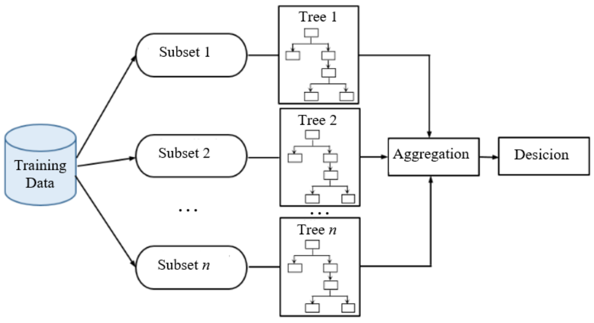

Figure 1. ***BAGG**ing* method structural scheme

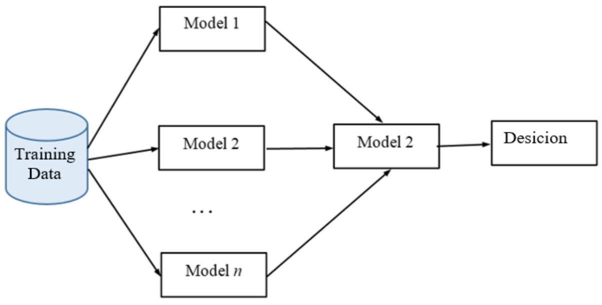

Figure 2. Stacking method structural scheme

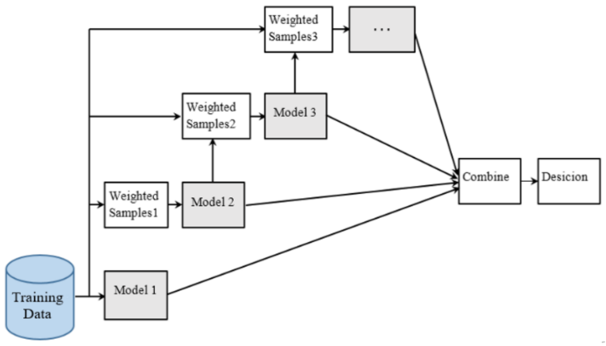

Figure 3. Boosting method structural scheme

## 5. Ensemble method for defect detection

This section makes efforts to develop an efficient method for predicting software systems defects using machine learning methods.

Figure 4 illustrates the decision-making system architecture of the proposed approach for software systems defects prediction. As figure shows, the decision-making system includes three classifiers.

Bagging (base learner PART), Random Forest and Logistic algorithms are combined through the Vote ensemble algorithm to predict defective modules in software systems. The aim of combining these classifiers is to improve the model's quality of predicting software system defects. To perform this process, each classifier is trained using the classified data.

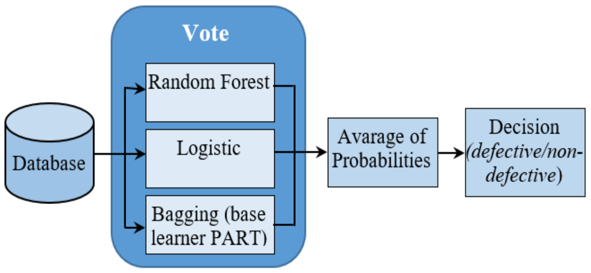
Vote
Random Forest
Logistic
Bagging (base learner PART)
Database
Avarage of Probabilities
Decision (defective/non-defective)

Figure 4. Software defect prediction model

Here, the individual decision of each of these classifiers is combined to make a collaborative decision.

## 6. Experimental study of the proposed method

Ensemble training has been proven to be an effective approach for predicting software defects and ensuring software stability along with improving the performance of individual classifiers. Defective software modules have a major impact on the reliability of software systems, leading to bigger expenses, longer release times, and significantly increased maintenance costs for deployed systems. This article analyzes the most popular and widely used machine learning algorithms. The data used in this study is obtained from the open-source NASA Promise and Github databases.

CM1, DATATRIEVE, JM1, KC1, KC2, KC3, MC1, PC1, PC2, PC3 databases are used for experiments (PROMISE & Github). Table 2 presents the parameters of these databases.

**Table 2.** Used Databases

| N | Data set | Number of Modules | Number of defective modules | Features (Software metrics) |
|---|---|---|---|---|
| 1. | CM1 | 498 | 49 | 21 |
| 2. | JM1 | 10885 | 2106 | 21 |
| 3. | KC1 | 2109 | 326 | 21 |
| 4. | KC2 | 522 | 107 | 21 |
| 5. | PC1 | 1109 | 77 | 21 |
| 6. | Datatrieve | 130 | 11 | 9 |
| 7. | KC3 | 194 | 36 | 39 |
| 8. | MC1 | 1952 | 46 | 38 |
| 9. | PC2 | 722 | 16 | 36 |
| 10. | PC3 | 1053 | 153 | 37 |

Classification is performed on this data set using 10-fold cross-validation. Naive Bayes, Support Vector Machine, J48, PART, IBk, Multilayer Perceptron (MLP) and Random forest algorithms are taken for comparison (Table 3.).

**Table 3.** Algorithms taken for comparison

| Learner | Description |
|---|---|
| Naive Bayes(NB) | A Naive Bayes classifier takes all attributes in the training data to be equally important and independent and enables them to contribute to the classifier's decision. The algorithm is based on Bayes' conditional probability theorem. |
| SMO | It implements John Platt's minimal sequential optimization algorithm to train a support vector classifier. A support vector machine is a maximum margin learning method that works by finding the optimal hyperplane separating positive and negative samples from a database. |
| J48 | J48 constructs decision trees from a set of labeled training samples using the concept of data entropy. |
| PART | PART is a recursive algorithm based on the divide-and-conquer strategy, and is a rule induction method derived from the combination of C4.5 and RIPPER. Rules are created, covered training samples are removed, and rules are recursively genera-ted for the remaining samples till the end. |
| IBk | IBk is a lazy nearest neighbor method. It is a sample-based simple training object that uses the class of the nearest k training samples to the class of test samples. A training sample with minimum Euclidean distance from a given test sample is predicted. When there is more than one sample within the minimum distance, the first found sample is used. |
| Multi-Layer Perceptron (MLP) | MLP is a network structure of input, output and hidden layers comprising perceptron or neurons. The neural layers are trained by a back-propagation algorithm based on the error correction rule. |
| Logistic Regression (LR) | Logistic regression is a type of statistical model (also known as a *logit model*) that is often used for classification and predictive analytics. Logistic regression estimates the probability of an event occurring based on a data set of given independent variables. |
| Random Forest | Random Forest or Random Decision Forests is an ensemble training method for classifica-tion, regression, and other problems that works by building multiple decision trees during training. In classification problems, the output of a random forest is the class chosen by the majority of trees. |

Experiments are performed in Weka software. Weka is one of the most popular data mining tools, developed in Java at the University of Waikato, New Zealand. It is widely used due to its mobility, availability and ease of use (Witten, 2009).

To evaluate the effectiveness of the proposed method, the true positive ate (TP), false positive rate (FP), true negative rate (TN), false negative rate (FN), F-measure, accuracy, precision, and recall parameters are used. This article uses ROC (receiver operating characteristic, in other words "errors curve") and F-measure to check the precision of the Forecast. The value of the ROC curve varies within the range of (0,1). A value of 1 is an ideal model. Table 4 describes confusion matrix used in the classification process.

**Table 4.** Confusion matrix

| | | **Predicted values** | |
|---|---|---|---|
| | | ***Positive*** | ***Negative*** |
| **Actual values** | ***Positive*** | TP | FP |
| | ***Neqative*** | FN | TN |

The efficiency parameters of the proposed model are calculated as follows (Ezekiel et al, 2020):

One of the key metrics for evaluating the performance of predictive models is classification precision.

***Precision*** is defined as the ratio of true positive (TP) modules to the total number of modules classified as positive:

$$\text{Precision} = \frac{TP}{TP + FP}$$

***Recall*** is defined as the ratio of true positive modules to the number of truly classified modules:

$$\text{Recall} = \frac{TP}{TP + FN}$$

***Accuracy*** is calculated as the ratio of the number of truly classified modules to the total number of modules:

$$Accuracy = \frac{TP + FN}{TP + TN + FP + FN}$$

***False Positive rate*** (FPR):

$$FPR = \frac{FP}{TP + TN}$$

***True Positive rate*** (TPR):

$$TPR = \frac{TP}{TP + TN}$$

***ROC –area*** shows the dependence of FPR on TPR (Brownlee, 2020).

Area under the ROC curve - AUC is a measure of how well a parameter can distinguish between two classes (impaired/impaired).

$$AUC = \frac{1 + TP_R - FP_R}{2}$$

***F-measure*** is the harmonic mean of precision and recall:

$$f - measure = \frac{2 * \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

In this article, the prediction accuracy of the model is evaluated with ROC-area and F-measure, which are most commonly used in experiments.

Table 5 presents the results of the experiments conducted on the database according to the F-measure value, and table 6 presents the results according to the ROC-area evaluation. Here, the test results of different algorithms and the proposed ensemble model (EM) are compared. These results are graphically illustrated in figure 5 and figure 6, respectively.

**Table 5.** Comparison of the proposed method with existing methods by F-measure value

| | NB | SMO | J48 | RF | PART | IBk | MLP | Ensemble |
|---|---|---|---|---|---|---|---|---|
| CM1 | **0,858** | 0,852 | 0,852 | 0,854 | 0,851 | 0,843 | 0,845 | 0,846 |
| Datatrieve | 0,840 | ? | 0,867 | 0,883 | 0,863 | 0,883 | 0,873 | **0,883** |
| JM1 | 0,770 | 0,722 | 0,769 | **0,783** | 0,763 | 0,766 | 0,741 | 0,767 |
| KC1 | 0,820 | 0,786 | 0,832 | **0,843** | 0,817 | 0,837 | 0,828 | 0,837 |
| KC2 | 0,821 | 0,784 | 0,810 | 0,825 | 0,810 | 0,802 | **0,835** | 0,814 |
| KC3 | 0,785 | 0,743 | 0,783 | 0,769 | 0,762 | 0,708 | 0,762 | **0,811** |
| MC1 | 0,937 | ? | 0,973 | 0,976 | 0,970 | 0,977 | 0,974 | **0,978** |
| PC1 | 0,895 | 0,897 | 0,921 | **0,927** | 0,926 | 0,921 | 0,917 | 0,924 |
| PC2 | 0,930 | ? | 0,965 | ? | 0,963 | 0,958 | 0,965 | **0,966** |
| PC3 | 0,458 | ? | 0,839 | 0,840 | 0,845 | 0,843 | 0,842 | **0,848** |

**Table 6.** Comparison of the proposed method with existing methods by ROC-area value

| | NB | SMO | J48 | RF | PART | IBk | MLP | Ensembl |
|---|---|---|---|---|---|---|---|---|
| CM1 | 0,658 | 0,497 | 0,558 | 0,750 | 0,721 | 0,589 | 0,734 | **0,789** |
| DATATRIEVE | 0,736 | 0,500 | 0,482 | 0,685 | 0,550 | 0,574 | 0,751 | **0,791** |
| JM1 | 0,679 | 0,502 | 0,653 | 0,755 | 0,712 | 0,640 | 0,690 | **0,763** |
| KC1 | 0,790 | 0,516 | 0,689 | 0,823 | 0,747 | 0,735 | 0,771 | **0,831** |
| KC2 | 0,830 | 0,597 | 0,704 | 0,825 | 0,704 | 0,643 | 0,828 | **0,831** |
| KC3 | 0,662 | 0,514 | 0,653 | 0,736 | 0,601 | 0,539 | 0,639 | **0,759** |
| MC1 | 0,747 | 0,500 | 0,566 | 0,850 | 0,580 | 0,665 | 0,728 | **0,886** |
| PC1 | 0,650 | 0,500 | 0,668 | 0,875 | 0,814 | 0,740 | 0,723 | **0,884** |
| PC2 | 0,717 | 0,500 | 0,463 | 0,780 | 0,605 | 0,551 | 0,770 | **0,804** |
| PC3 | 0,749 | 0,500 | 0,591 | 0,832 | 0,767 | 0,603 | 0,783 | **0,843** |

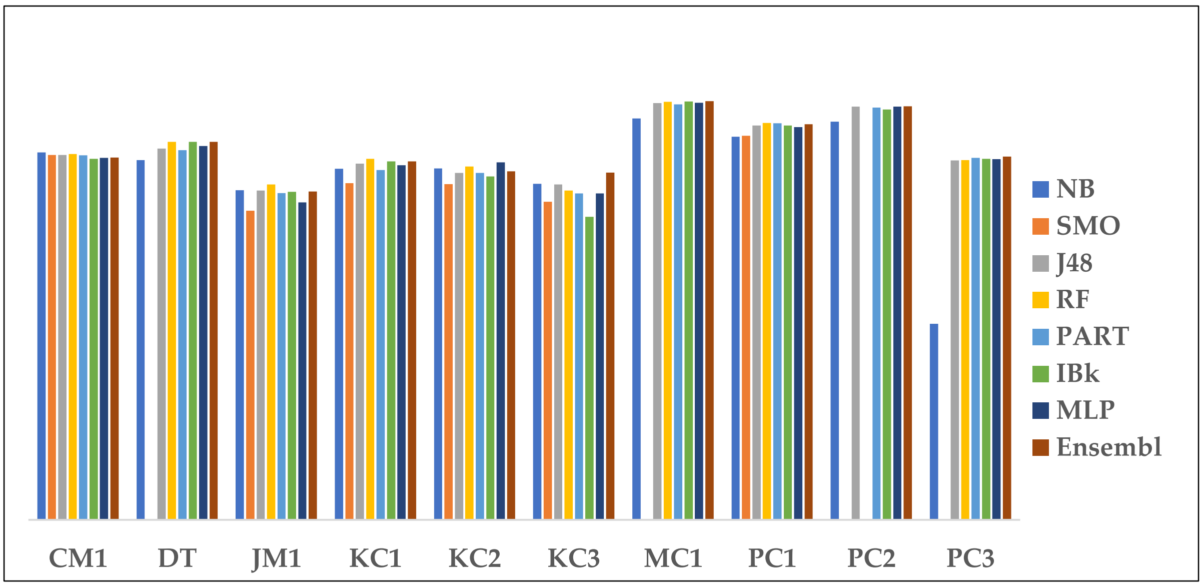

**Figure 5.** Comparison of the effectiveness of algorithms by F-measure value

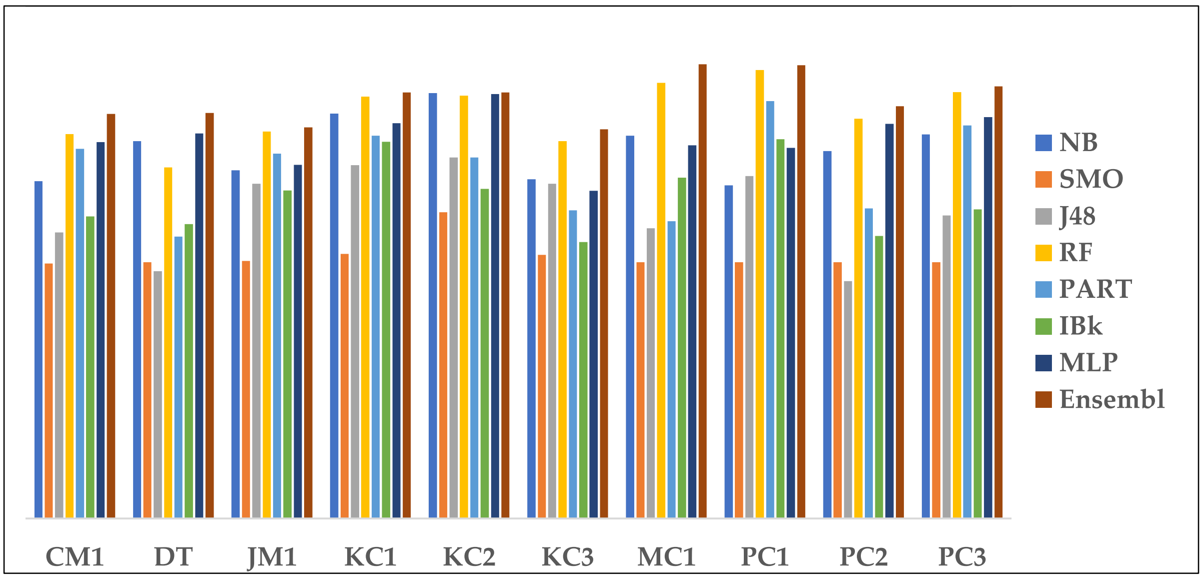

**Figure 6.** Comparison of the effectiveness of algorithms by ROC-area value

As table 6 shows, according to ROC-area indicator, the defect module detection precision of the ensemble model is superior to the detection precision of individual algorithms. As table 5 shows, according to F-measure value, in most cases EM prevails, in some cases the RF model shows a high result. However, unlike RF, EM performs stable results in both small and large software systems (Figures 5 and 6).

Figures 7 and 8 illustrate the comparison of the efficiency of the ensemble model with the RF algorithm.

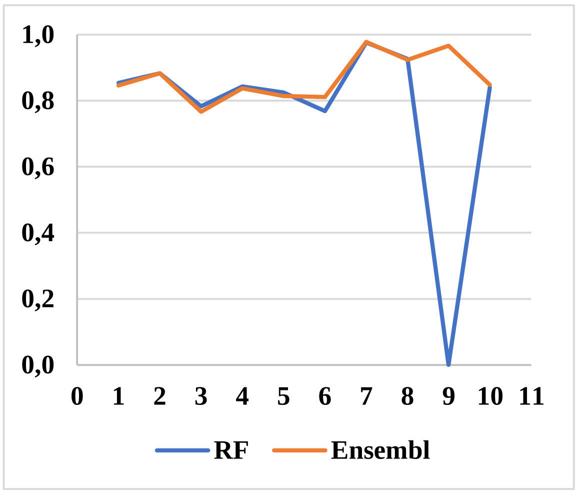

**Figure 7.** Comparison of the efficiency of RF and the proposed model by F-measure value

The ensemble model is obviously superior to the RF model for both indicators and has more stable forecasting performance.

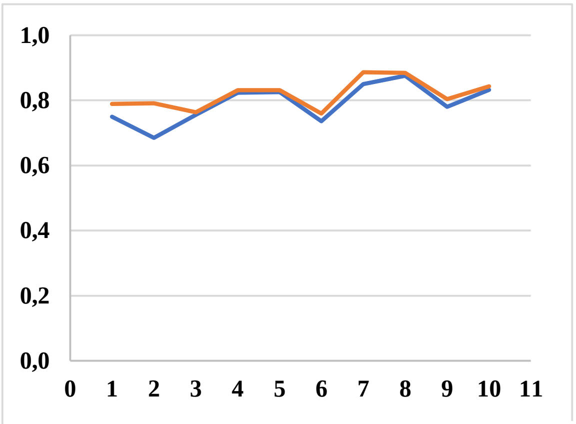

**Figure 8.** Comparison of the efficiency of RF and the proposed model by ROC-area value

## Conclusion

Software defect is a serious problem in software systems development. Predetermining the module that contains potential defects increase the reliability of the software system. Software defect prediction methods urges quality professionals to carefully inspect the modules classified as defective during software code testing.

This article proposed an ensemble training method for software defect prediction. PROMISE and GITHUB open databases were used for experiments. ROC-area and F-measure parameters were used to evaluate the effectiveness of the proposed method. The results of the experiments showed that the proposed method is superior to the most commonly used NB, SMO, J48, PART, İBk, MLP and RF methods in the prediction of software defects due to its detection precision.

This research concluded that the application of ensemble methods in software system defect prediction provided more precise results than individual machine learning methods.

## References

Alazzam I., Alsmadi I., Akour M., (2017). Software fault proneness prediction: A comparative study between bagging, boosting, and stacking ensemble and base learner methods, Int. J. Data Anal. Techn. Strategies, vol. 9, no. 1, p. 1, doi: 10.1504/ijdats.2017.10003991.)

Aljamaan H., Alazba A. (2020). Software defect prediction using tree-based ensembles. In Proceedings of the 16th ACM International Conference on Predictive Models and Data Analytics in Software Engineering (PROMISE 2020). Association for Computing Machinery, New York, NY, USA, pp.1–10. https://doi.org/10.1145/3416508. 3417114.

Alsaeedi A., Zubair Khan M. (2019). Software Defect Prediction Using SupervisedMachine Learning and Ensemble Techniques:A Comparative Study, Journal of Software Engineering and Applications, 12, pp.85-100.

Bowes D., Hall T., (2018). Software defect prediction: do different classifiers find the same defects? Petrić J. Software defect prediction: do different classifiers find the same defects? Software Qual J., 26, pp.525–552 https://doi.org/10.1007/ s11219-016-9353-3.

Catal C., Diri B. (2009). Investigating the effect of dataset size, metrics sets, and feature selection techniques on software fault prediction problem. Information Sciences, 179, pp. 1040–1058.

Chug A., Dhall S., (2013). Software defect prediction using supervised learning algorithm and unsupervised learning algorithm, Confluence 2013: The Next Generation Information Technology Summit (4th International Conference), Noida, pp. 173-179, doi: 10.1049/cp.2013.2313.

Cortes C., LeCun Y., Vapnik V., Drucker H., Jackel L. D.

(2008). Boosting and other ensemble methods, Neural Comput., vol. 6, no. 6, pp. 1289–1301,

Ezekiel O.O., Irhebhude M. E., Evwiekpaefe A.E. and Nonyelum O., F. (2020). Evaluation of Machine Learning Classification Techniques in Predicting Software Defects, Transactionson Machine Learning and Artificial Intelligence, Volume 8 No 5 August, pp: 1-15

Ge J., Liu J., Liu, W. (2018). Comparative Study on Defect Prediction Algorithms of Supervised Learning Software Based on Imbalanced Classification DataSets. 2018 19th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, 27-29 June, Busan, 399-406.

Hussain S., Keung J., Khan A. A., and Bennin K. E., (2015). Performance evaluation of ensemble methods for software fault prediction, in Proc. ASWEC 24th Australas. Softw. Eng. Conf. (ASWEC), vol. 2, Sep. pp. 91–95, doi: 10.1145/2811681.2811699.

Jayanthi R., Florence L. (2019). Software defect prediction techniques using metrics based on neural network classifier. Cluster Comput 22 (Suppl 1), pp.77–88. https://doi.org/10.1007/s10586-018-1730-1.

Jin C., Jin Sh. (2015). Prediction approach of software fault-proneness based on hybrid artificial neural network and quantum particle swarm optimization. Applied Soft Computing 35 (Oct. 2015), pp.717–725. https://doi.org/10.1016/j. asoc.2015.07.006

Kanmani S., Uthariaraj V. R., Sankaranarayanan V., Thambidurai P. (2007). Object-oriented software fault prediction using neural networks. Information and Software Technology 49, 5, pp.483–492. https://doi.org/10.1016/ j.infsof.2006.07.005),

Kaur A., Kaur K. (2014). Performance analysis of ensemble learning for predicting defects in open source software, 2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI), Delhi, India, pp. 219-225, doi: 10.1109/ICACCI. 2014.6968438

Kazimov T.H., Bayramova T. A., Malıkova N.C. (2021). Research of intelligent methods of software testing. System Research & Information Technologies, № 4, pp. 42- 52.

Kazimov T.H., Bayramova T. A. (2022). Development of a hybrid method for calculation of software complexity // System Research & Information Technologies, № 2, pp. 32- 44.

Qiao L, Li X., Umer Q., Guo P. (2020). Deep learning based software defect prediction, Neurocomputing, Vol. 385, pp. 100-110, https://doi.org/10.1016/j.neucom. 2019.11.067

Li R., Zhou L., Zhang Sh., Liu H., Huang X., Sun Z. (2019). Software Defect Prediction Based on Ensemble Learning. In Proceedings of the 2019 2nd International Conference on Data Science and Information Technology (DSIT 2019). Association for Computing Machinery, New York, NY, USA, 1–6. https://doi.org/10.1145/3352411. 3352412.

Malhotra R. (2015). A systematic review of machine learning techniques for software fault prediction. Applied Soft Computing, 27, pp.504–518. https://doi.org/10.1016/j.asoc.2014.11.023,

Matloob F. et al., (2021). Software Defect Prediction Using Ensemble Learning: A Systematic Literature Review, in IEEE Access, vol. 9, pp. 98754-98771, doi: 10.1109/ACCESS.2021.3095559.

Menzies T., Milton Z., Turhan B., Cukic B., Jiang Y., and Bener A. (2010). Defect prediction from static code features: current results, limitations, new approaches. Automated Software Engineering 17, 4, pp.375–407. https://doi.org/10.1007/s10515-010-0069-5.

NASA Defect Dataset, https://github.com/klainfo/NASADefectDataset/tree/master/OriginalData/MDPI.

NASA metrics data program, PROMISE software engineering repository, 2004, http://promise.site.uottawa.ca/SERepository/datasets-page.html

Rathore S.S., Kumar S. A. (, 2016). Decision Tree Regression Based Approach forthe Number of Software Faults Prediction. ACM SIGSOFT Softw Are Engineering Notes41, pp.1-6. https://doi.org/10.1145/2853073.2853083

Rathore S.S., Kumar, S. (2017). An Empirical Study of Some Software Fault Prediction Techniques for the Number of Faults Prediction. Soft Computing, 21,7417-7434.

Rodriguez J. J., Kuncheva L. I., Alonso C. J., (2006). Rotation forest: A new classifier ensemble method,'' IEEE Trans. Pattern Anal. Mach. Intell., vol. 28, no. 10, pp. 1619.

Rokach L., (2009). Taxonomy for characterizing ensemble methods in classification tasks: A review and annotated bibliography, Comput. Statist. Data Anal., vol. 53, no. 12, pp. 4046–4072.

Rokach L. (2010). Ensemble-based classifiers. Artificial Intelligence Review 33, 1, pp.1-39. https://doi.org/10.1007/s10462-009-9124-7.

Singh P., Pal N. R., Verma S.. Vyas O. P. (2017). Fuzzy Rule-Based Approach for Software Fault Prediction, in IEEE Transactions on Systems, Man, and Cybernetics: Systems, vol. 47, no. 5, pp. 826-837, doi: 10.1109/TSMC.2016.2521840.

Stapor K. (2017). Evaluating and comparing classifiers: Review, some recommendations and limitations, in Proc. Int. Conf. Comput. Recognit. Syst., pp. 12–21.

Thota M. K., Shajin F. H, Rajesh P., (2020). Survey on software defect prediction techniques, Vol. 17, No. 4, pp.100-110.

Wang H. (2014). Software Defects Classification Prediction Based on Mining Software Repository. Master's Thesis, Uppsala University, Department of InformationTechnology. p.93.

Witten H., Frank E., Hall M. A., Pal C. J., (2009). Data Mining: Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. 2009. The WEKA data mining software: an update. SIGKDD Explor. Newsl. 11, 1, 10–18. https://doi.org/10.1145/1656274.1656278

Yan Z., Chen X. and Guo P. (2010). Software Defect Prediction Using Fuzzy Sup-port Vector Regression. In: Zhang, L., Lu, B. and Kwok, J., Eds., Advances in NeuralNetworks, Springer, Berlin, pp.17-24. https://doi.org/ 10.1007/978-3-642-13318-3_3

Zhang Ch., Ma Y. (2012). Ensemble Machine Learning: Methods and Applications, Springer New York, NY, p.332.

Zhang H. (2009). An Investigation of the Relationships between Lines of Code and Defects. 2009 IEEE International Conference on Software Maintenance, 20-26 September Edmonton, 274-283.

Zhang Z., Jing X., Wang T. (2017). Label propagation based semi-supervised learning for software defect prediction. Automated Software Engineering, 24, pp.47–69. https://doi.org/10.1007/s10515-016-0194-x.

Brownlee J., (2020). ROC Curves and Precision-Recall Curves for Imbalanced Classification, https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-imbalanced-classification/ (accessed Jun. 1, 2022)