Engineering Science and Technology, an International Journal xxx (xxxx) xxx

HOSTED BY

ELSEVIER

Contents lists available at ScienceDirect

Engineering Science and Technology, an International Journal

journal homepage: www.elsevier.com/locate/jestch

Full Length Article

Multiple-classifiers in software quality engineering: Combining predictors to improve software fault prediction ability

Fatih Yucalar [a,*], Akin Ozcift [a], Emin Borandag [a], Deniz Kilinc [a,b]

[a] Manisa Celal Bayar University, Department of Software Engineering, Manisa, Turkey
[b] Izmir Bakircay University, Department of Computer Engineering, Izmir, Turkey

ARTICLE INFO

Article history:
Received 16 April 2019
Revised 26 September 2019
Accepted 18 October 2019
Available online xxxx

Keywords:
Classification
Ensemble learning
Software fault prediction
Software quality engineering

ABSTRACT

Software development projects require a critical and costly testing phase to investigate efficiency of the resultant product. As the size and complexity of project increases, manual prediction of software defects becomes a time consuming and costly task. An alternative to manual defect prediction is the use of automated predictors to focus on faulty modules and let the software engineer to examine the defective part with more detail. In this aspect, improved fault predictors will always find a software quality application project to be applied on. There are many base predictors tested-designed for this purpose. However, base predictors might be combined with an ensemble strategy to further improve to increase their performance, particularly fault-detection abilities. The aim of this study is to demonstrate fault-prediction performance of ten ensemble predictors compared to baseline predictors empirically. In our experiments, we used 15 software projects from PROMISE repository and we evaluated the fault-detection performance of algorithms in terms of F-measure (FM) and Area under the Receiver Operating Characteristics (ROC) Curve (AUC). The results of experiments demonstrated that ensemble predictors might improve fault detection performance to some extent.

1. Introduction

Software development projects require a software testing phase, a critical and costly step, to investigate efficiency of the resultant product. In this aspect, the required time to test the software, the staff to be allocated and hence the cost are the major points to consider. As the size and complexity of project increase, manual prediction of software defects becomes a time consuming and costly task [1]. As an alternative to manual code reviews, defect predictors have been widely used in organizations to predict software bugs while saving time and reducing cost [2]. In case of a limited budget, instead of a complete testing of an entire system, software managers might use defect predictors to focus on parts of the system that seem defect-prone. These potential defect-prone modules can then be examined with more detail. Tree defect predictors are trained with the use of historical defect databases and are expected to investigate software faults of future projects [1].

From machine learning perspective, defect prediction is a classification task that aims to discriminate fault prone (fp) and non-fault prone (nfp) code modules defined with static code attributes [3,4]. Thus, there are many candidate classification algorithms in the literature that might help software analysts to investigate whether a software component will be predicted fp or nfp [4]. As a part of this study, we have reviewed a selection of relevant publications from software fault prediction model literature. We will discuss some of the algorithms in related work section and the reader is referred to current review of Catal et al. for a fine discussion of prediction algorithms [5].

The usage of defect predictors is strictly dependent to easiness of obtaining static code attributes [6]. Automated code metric extraction tools answer this intuitional question of whether the use of defect predictors is feasible or not. "Once a local repository is built with such extraction tools, it would be easy to train an algorithm with the historical data and test for current releases simultaneously" [7]. In this concept, it is reasonable for software developers to build local repositories with automated tools and hence to make use of defect predictors to guide code reviewers focus on problematic modules.

The motivation of this study comes from the survey of defect predictor literature: there are many studies aiming to locate software defects with the use of single learners. However, in the related literature, we did not find any comprehensive study evaluating ensemble predictor algorithms for software quality

* Corresponding author.
E-mail address: fatih.yucalar@cbu.edu.tr (F. Yucalar).
Peer review under responsibility of Karabuk University.

https://doi.org/10.1016/j.jestch.2019.10.005
2215-0986/

estimation. There are only a few studies that make use of ensemble predictors applied to software fault detection [7,8]. From the software defect prediction point of view, ensemble algorithms combine signals from base defect predictors in the committee to produce an enhanced fault detection algorithm. Furthermore, the committee of predictors (ensemble) provides capturing the strengths of multiple predictors while helping to overcome noisy nature of code metrics [7–9] that define software modules to be tested.

In this study, we perform an extensive research on the performance of ten ensemble predictors from WEKA machine learning workbench [10] applied to fifteen datasets from PROMISE repository [11]: (i) In order to make a baseline (determining the best predictor) for evaluation of ensemble algorithms, we first selected sixteen widely used base predictors from software fault detection literature. We calculated performance of predictors for sample datasets in terms of F-measure (FM) and Area under Receiver Operating Characteristics Curve (AUC) metrics. (ii) As a second step, we tested ten ensemble predictors with the same datasets to determine their performance using FM and AUC metrics. (iii) We compared best ensemble predictors with baseline predictor of step one to interpret the performance of ensemble algorithms in software fault detection.

2. Related work

A typical software fault detection strategy makes use of a predictor (so called a classifier from machine learning perspective) trained with software metrics and fault data (collected from previous releases or similar projects) to estimate defects of future projects. In this aspect, the software fault prediction problem is often a supervised classification task that aims to discriminate fp and nfp modules (classes) from each other with maximum accuracy [12,13].

In the literature, there are many predictors applied to software fault prediction. In the recent review [5], Catal et al. discusses 90 software fault prediction articles and their corresponding prediction algorithms. The context of this work does not let discuss all the studies conducted. Instead, we use Catal et al.'s review as reference for survey of base predictors frequently used in software defect prediction. We briefly mention those predictors as follows:

According to Catal et al., Logistic Regression (LR) was used in seventeen studies for various datasets as software fault predictor. Another widely used predictor in software fault detection is Naïve Bayes (NB). This algorithm was used in eleven studies from various software domains. Another most frequently used defect predictor was C4.5 and its java implementation J48. The usage of these similar algorithms was thirteen in total. Furthermore, Random forests (RF) is another predictor noticed in Catal et al.'s study frequently. RF is an ensemble predictor and it was used in five studies. Some of other miscellaneous algorithms mentioned in5 are Support Vector Machine (SVM), Radial Basis Function (RBF), K-Star (KS), Voting Feature Intervals (VFI), Voted Perceptron (VP), IBk, CART, Multi-Layer Perceptron (MLP), Dempster-Shafer Belief Networks (DSBN) and 1-R [5].

In particular, ensemble fault predictors were not frequently used in software defect prediction studies. Other than RF, we came across four recent studies using ensemble predictors in fault prediction domain. In a recent study, Arisolma et al. used Decorate and Adaboost ensemble of C4.5 predictor to evaluate fault-proneness of a large Java legacy system project [14]. They stated that Adaboost could be applied to further improve the C4.5 prediction model. They evaluated their study with the use of cost effectiveness and AUC metrics and they obtained Adaboost to be the best predictor among other evaluated algorithms [1]. Another ensemble prediction strategy was a simple majority voting scheme applied to public datasets. The authors combined Neural Networks, Naïve Bayes and Voting Feature Intervals to improve prediction performance of base predictors [15]. In their similar study, the authors used the same three predictors with a "all or nothing" voting scheme. They applied a voting strategy to label a sample as defective under the condition that all algorithms also predict that sample as defective. The authors applied their voting strategy to public datasets from PROMISE and they reported that their ensemble strategy decreased false alarm down to 15% while increasing precision by 43%. Another study that we mention was evaluated by Yiang et al. with the use of bagging ensemble strategy [16]. In the empirical study, the authors used six major classifiers to assess public datasets from PROMISE and bagging was one of their predictors.

In their study, Laradji et al. [17] examined the effects of combining feature selection and ensemble learning algorithms on the performance of defect classification. In this study, the authors proposed a new two-variant ensemble learning algorithm in order to increase the effectiveness of feature selection and to prevent the negative effects of data imbalance. In the study of Abdou and Darwish [18], they have introduced the resampling technique with three types of different ensemble learners using eight of base learners tested on seven types of benchmark datasets provided by the PROMISE repository. In most of the algorithms used in the experimental results showed that accuracy has been enhanced using ensemble learners more than single leaners. In another study, Ardimento et al. [19] proposed a multi-source machine learning approach based on both product and process metrics for defect prediction. The results obtained in this study showed that the proposed approach is effective for defect prediction. In the study performed by Kaur and Kaur [20], software metrics such as Chidamber&Kemerer (C&K), Henderson & Sellers, McCabe were used for software defect estimation. According to these metrics, the use of Random Forest and Bagging methods showed good results for software fault prediction. The last study by Boucher and Badri [21], they used the threshold calculation method to software fault prediction. The authors have compared empirically the selected threshold calculation methods as part of fault prediction techniques. Also, they have used a machine learning technique namely Bayes Network as a baseline for comparison. Thresholds have been calculated for different object-oriented metrics using four different datasets obtained from the PROMISE repository and another one based on the Eclipse project.

Software fault prediction has been studied in the literature by many scientists. However, there is still a need for benchmarking studies for governing future works. In this context, the main contribution of this paper is the evaluation of many diverse software projects with the comparison of extensive ensemble algorithms with those of corresponding base learners.

3. Methodology

Software fault detection literature borrows lots of algorithms from machine learning techniques to develop automated software fault predictors. However, there is no universal predictor that might be used in every software project. Consequently, there is an ongoing effort to build enhanced fault predictors.

This study in an experimental contribution to this research area with the application of various base predictors and ensemble predictors to fifteen real-world projects. The methodology of our study is described with the following steps:

(i) Obtain fifteen diverse datasets from PROMISE repository to use in evaluation.
(ii) If necessary, pre-process selected datasets.

Please cite this article as: F. Yucalar, A. Ozcift, E. Borandag et al., Multiple-classifiers in software quality engineering: Combining predictors to improve software fault prediction ability, Engineering Science and Technology, an International Journal, https://doi.org/10.1016/j.jestch.2019.10.005

(iii) Select suitable performance metrics to interpret results of experiments.
(iv) Select the most widely used base predictors from software fault detection literature.
(v) Determine the most successful base predictor testing algorithms with selected datasets in step (i).
(vi) Select possible ensemble predictors from machine learning literature and test them with datasets of step (i).
(vii) Determine the most promising ensemble predictor(s), comparing the results with the most successful base predictor from step (v).

3.1. Selection of datasets and preprocessing

In order to realize our experiments, we have utilized 15 public datasets obtained from PROMISE repository. These datasets reflect industrial software engineering practices and they belong to three real world applications: (i) Data from a Turkish white-goods manufacturer [7] (ii) NASA projects [22,23] and (iii) Various academic software projects and Apache related open-source applications [24]. The concise information about characteristics of datasets and group they belong to is given in Table 1.

From prediction point of view, each project is defined with software code metrics (attributes) such as Halstead's metrics, functional metrics, McCabe's Cyclomatic complexity, Lines of Code (LOC), Chidamber&Kemerer measures etc. that are extracted from modules of the corresponding project. The NASA and white-good projects are basically defined with the use of Halstead-McCabe and LOC metrics [25,26]. However, the third group of projects is defined in terms of Chidamber&Kemerer metric suite [27]. As the three groups of projects in average require fifty code metrics to be explained, we disregard attribute information of datasets and we refer interested reader to the mentioned references.

As additional information, we have processed third group software projects to label the modules as fp and nfp. The third group projects included a column that shows the number of the defects in the corresponding module. Since ML algorithms require that attribute as binary, i.e. fp and nfp, we converted attribute into binary as follows: the modules with bug counts greater than zero were labeled as fp otherwise nfp.

3.2. Selection of performance metrics

Defect prediction, as a two-group classification problem, has four possible prediction outcomes: (i) true positive (TP) for correctly classified positive instances, (ii) false positive (FP) for negative instance classified as positive, (iii) true negative (TN) for correctly classified negative instances and (iv) false negative (FN) for positive instance classified as negative. The numbers of cases from the four sets (outcomes) are used in the derivation of several well-known performance measures such as Accuracy (ACC), Probability of Detection (PD), Probability of False Alarms (PF), F-measure (FM), AUC and ROC curve. The confusion matrix that connects four prediction outcomes to each other is given in Table 2.

In software fault prediction literature, PD, PF, ACC, FM, AUC and ROC are frequently used performance metrics. First, we define these concepts briefly and then we make explanation why to prefer FM and AUC as basic performance metrics for our evaluation.

- Probability of Detection (PD) is also called recall and it is defined as the probability of correct prediction of a module that contains defects. For a defect prediction task, it is desired to obtain PD as one.

$$PD = \frac{TP}{TP + FN} \tag{1}$$

- Probability of False Alarms (PF) is defined as the ratio of false positives to all non-defective modules. As a performance metric, PF is desired to be as small as possible. In ideal prediction task, it is equal to zero.

$$PF = \frac{FP}{FP + TN} \tag{2}$$

- Accuracy (ACC) is a widely used evaluation metric to compare performance of machine learning algorithms quantitatively. It is defined as the proportion of true outcomes.

$$ACC = \frac{TP + TN}{TP + FP + TN + FN} \tag{3}$$

- Precision (PR), similar to ACC, is defined as the ratio of correctly detected modules.

$$PR = \frac{TP}{TP + FP} \tag{4}$$

- F-score or F-measure is another widely used performance metrics that is defined as the harmonic mean of Recall and Precision. FM can take values of (0, 1) and it is accepted to be better as it approaches to one.

$$FM = \frac{2 \times Recall \times Precision}{Recall + Precision} \tag{5}$$

Table 1
Major characteristics of datasets used in this study.

No	Project	#Attributes	#Modules	Defect rate	Group
1	ar5	29	36	0.20	1
2	ar6	29	101	0.15	1
3	cm1	21	498	0.09	2
4	jm1	21	10,885	0.19	2
5	kc1	95	145	0.05	2
6	mc2	40	161	0.32	2
7	pc1	21	1109	0.06	2
8	pc3	21	1563	0.10	2
9	intercafe	23	27	0.14	3
10	ivy2_0	23	352	0.11	3
11	jedit_4_3	23	538	0.06	3
12	lucene_2_4	23	341	0.59	3
13	serapion	23	47	0.15	3
14	tomcat	23	855	0.09	3
15	workflow	21	427	0.12	3

Please cite this article as: F. Yucalar, A. Ozcift, E. Borandag et al., Multiple-classifiers in software quality engineering: Combining predictors to improve software fault prediction ability, Engineering Science and Technology, an International Journal, https://doi.org/10.1016/j.jestch.2019.10.005

Table 2
Major characteristics of datasets used in this study.

Actual Data		
Predicted	Defective	Non-defective
Defective	TP	FP
Non-defective	FN	TN

- ROC curve, from software prediction point of view, is defined as a tradeoff between the predictor's ability to correctly detect defective modules (PD) and non-defective modules (PF) [16]. ROC is plot of PD (on the Y-axis) versus PF (on the X-axis) while adjusting predictor's threshold. As the ROC curve of a predictor comes close to upper left corner (PD = 1 and PF = 0), the performance is better. A sample ROC curve comparing Decision Stump, Jrip and Bayesian Logistic Regression classifiers on AR5 dataset is given in Fig. 1. The best predictor in Fig. 1 is obviously Decision Stump algorithm.
- AUC is defined as the area of under ROC curve and it is a common numeric metric to compare performance of predictors practically.

"Accuracy is a good measure of a learner's performance when the possible outcomes occur with similar frequencies. The data sets used in this study, however, have imbalanced class distributions" [3]. Therefore, we eliminated the usage of ACC as a performance metric to evaluate our predictors.

One of the performance metrics widely used in software fault prediction studies is PD-PF pair. Basically, while PD is maximized, PF is reversely minimized for a successful prediction algorithm. "PD defines exact identification of faults whereas PF gives the cost to validate that faults" [28]. Whenever it is needed for a deeper analysis, we make use of PD-PF pair. However, for the sake of convenience, while discriminating the predictors, we needed a single metric and we preferred FM. It is empirically explained in [29] that a good prediction model should achieve high Recall and high Precision [29]. FM that combines Recall (PD)-Precision pair and it is therefore a suitable single metric for our basic performance evaluation.

In this study, we selected FM metric in place of ACC to obtain a more confident evaluation framework. However, in practice, predictors might produce equal FM outputs and hence another measure is necessary to make preference among predictors. Similar to ACC, two predictors might also produce similar FM measures while having different fault detection abilities. In the literature, ROC curve or AUC is used to serve a confidence reference for empirical evaluations. AUC as a single scalar metric might separates predictive performance in terms of class and cost distributions [30]. In this concept, AUC, as discussed by Menzies et al. [3], might provide researchers to decide on the predictive performances of two hypothetical classifiers A and B. Furthermore, the statistical interpretation of AUC is easy: "any classifier achieving AUC well above 0.5 is demonstrably effective for identifying fp modules and gives valuable advice as to which modules should receive particular attention in software testing" [30]. Through this study, we therefore selected FM and AUC measures, while evaluating the results of our experiments.

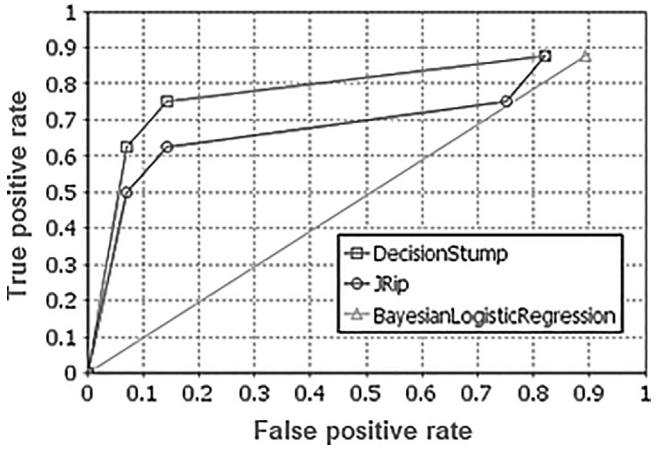

Fig. 1. Sample ROC curves for three classifiers applied to dataset AR5.

3.3. Selection of base predictors

The major aim of this study is to demonstrate the prediction performances of ensemble algorithms compared to widely used base predictors. For this aim, we should first evaluate various base predictors from literature and then select the best algorithm to be a reference line for ensemble predictor experiments. We used Catal et al.'s review [5] to determine frequency of base predictors in the software fault prediction literature. And hence, we determined sixteen base predictors. The names of the algorithms and their corresponding abbreviations with brief prediction strategies are given in Table 3.

3.4. Selection of base predictors

The aim of ensemble learning strategies is to build a committee of learners combining diverse base predictors to obtain an enhanced learning algorithm that is superior to any of the members of committee. Categorization of ensemble learning approaches primarily depends on (i) the generation strategy of base predictors in the ensemble and (ii) the integration method of base predictors [40]:

Ensemble predictor building strategies: in general, ensemble strategies use either the same type of base predictors (homogeneous) or dissimilar learners (heterogeneous) while generating committee. In ensemble prediction strategies, one of the key concepts that shape success of ensemble prediction strategies is diversity. Diversity of predictors means that the decision boundaries of classifiers are to be different than one another [41]. The most popular diversity techniques use different training datasets for each base predictor of committee. Bootstrap aggregating (bagging) [42] and boosting [43] are examples of data resampling techniques providing distinct training data for each base predictor. In bagging each member of committee is built using a bootstrap sample from the original dataset. The fault prediction of ensemble is obtained either with a uniform average or with a voting strategy [44]. Similar to bagging, boosting creates ensemble using resampling data strategy and voting mechanism.

The algorithm creates three weak-predictors: "the first classifier C1 is trained with a random subset of the available training data. The training data subset for the second classifier C2 is chosen as the most informative subset, given C1. That is, C2 is trained on a training data only half of which is correctly classified by C1, and the other half is misclassified. The third classifier C3 is trained with instances on which C1 and C2 disagree. The three classifiers are combined through a three-way majority vote [41]". The bagging and boosting algorithms are from [45] and they are given in Figs. 2 and 3.

Ensemble integration strategies: particularly, for heterogeneous predictor committee, the combination rule determines the strategy of ensemble algorithm. In general, two kinds of ensemble combination rules exist: (i) voting and algebraic combinations of predictors and (ii) a second level predictor to obtain the output for the committee.

Please cite this article as: F. Yucalar, A. Ozcift, E. Borandag et al., Multiple-classifiers in software quality engineering: Combining predictors to improve software fault prediction ability, Engineering Science and Technology, an International Journal, https://doi.org/10.1016/j.jestch.2019.10.005

Table 3
Base predictors selected from literature.

No	Algorithm	Prediction Model	Prediction Strategy
1	Naïve Bayes (NB)	Statistical	Statistical classifiers construct an optimal Bayes classifier in two approaches: (i) estimating posterior probabilities directly (LR) or (ii) using class-conditional probabilities then obtaining posterior probabilities with Bayes' theorem (NB).
2	Logistic Regression (LR)		
3	Multi-Layer Perceptron (MLP)	Neural Network	Neural network classifiers resemble a network structure comprising weighting, aggregation and thresholding functions. These functions are applied to software data attributes to determine if a module is fp by calculating its posterior probability [30]. We used two kinds of neural networks in our study as MLP and RBF.
4	RBF Network (RBF)		
5	Sequential Minimal Optimization (SMO)	Support Vector	SVM classifiers construct a hyperplane to discriminate fp and nfp modules with the use of linear functions as PG [31]. However, in some cases linear separation is impossible and the problem is mapped to a higher space by means of kernel functions. SMO [32], and VP [33] uses polynomial kernels to achieve this mapping.
6	Pegasos (PG)	Machine (SVM)	
7	Voted Perceptron (VP)		
8	Instance Based Learner (IB)	K Nearest Neighbor	These learners use an instance-based approach to classify modules depending on their similarity. We used IB [34] and KS [35] to detect faults and the two algorithms uses a different similarity concept while making discrimination of two classes (being fp or nfp). For similarity measure, while IB uses an Euclidian approach, KS makes use of an entropy-based distance measure.
9	KStar (KS)	(KNN) Knearest	
10	Jrip (JR)	Rule Based (RB)	RB classifiers are built with the use of IF-THEN rules and these rules are used to classify software modules. Rules of a rule-based classifier might be extracted from a decision tree. In this scheme, one rule is created for each path from the root to a leaf node. PART [13] uses C4.5 tree and OR [36] makes use of a one-level decision tree. Our last rule-based predictor, JR, implements a propositional rule learner that is based on association rules with reduced error pruning [37].
11	OneR (OR)		
12	PART		
13	J48	Decision Tree (DT)	These set of classifiers partition data with the use of attribute splits. There are many DT predictors and the main difference between them is the attribute split criteria such as Gini index and information gain. In this aspect, while J48 [13] uses information gain to find the best splits, CART [38] makes use of Gini index to determine decision tree splits.
14	CART		
15	Hyperpipes (HYP)	Miscellaneous	HYP is a predictor with the similar strategy of OR algorithm. It makes use of a simple rule for fp and nfp classes to make a prediction [13] "VFI [7–39] places minimum and maximum values of each attribute for each module. The difference between these values forms an interval for each attribute. When a new input is taken, the number of samples of each class within this interval are counted and summed to form votes. The prediction is based on selecting the class with the highest vote."
16	Voting Feature Intervals (VFI)		

Input: *training set S, base predictor B, integer T (# of bootstrap samples)*

A. *for* $i = 1$ *to* T {

B. $S' =$ *bootstrap sample from* S

C. $E_i = B(S')$

D. }

E. $E^*(x) = \underset{y \in Y}{arg\ max} \sum_{i:E_i(x)=y} 1$ *(predicted label y)*

Output: *ensemble predictor* E^*

Fig. 2. Pseudocode of Bagging Algorithm.

Input: *training set S of size m, base predictor B, integer T (# of trials*

A. $S' = S$ *instance weights assigned to be* 1

B. *For* $i = 1$ *to* T {

C. $E_i = B(S')$

D. $\epsilon_i = \frac{1}{m} \sum_{x_j \epsilon S': E_i(x_j) = y_j} weight(x)$ *(weighted error on training set)*

E. *If* $\epsilon_i > 1/2$, *set* S' *to bootstrap sample from* S *with weight* 1 *for every instance and GO TO step C*

F. $\beta_i = \epsilon_i / (1 - \epsilon_i)$

G. $For - each\ x_j \epsilon S', if\ E_i(x_j) = y_j\ then\ weight(x_j) = weight(x_j).\beta_i$

H. *Normalize the weight of instances so that* S' *to be* m

İ. }

J. $E^*(x) = \underset{y \in Y}{arg\ max} \sum_{i:E_i(x)=y} \log \frac{1}{\beta_i}$

Output: *ensemble predictor* E^*

Fig. 3. Pseudocode of Boosting Algorithm.

Voting based strategies makes prediction in two ways: if all predictors agree on the prediction (unanimous voting) and if at least one more than half of the classifiers agree on the prediction (simple majority). Algebraic combinations use some rules, i.e. mean, median, weighted average, to decide on the output of the predictors [41].

In a different approach, stacking (stacked generalization) [46], the output of the predictors is combined through a second level predictor. Stacking replaces algebraic combination rule with a predictor to learn the reliability of the members in the ensemble. Therefore, the algorithm discovers a better way to combine output of predictors compared to voting mechanism or algebraic combination rules [47].

In our study, we selected ten ensemble predictors from WEKA environment and we provide brief information about those algorithms in Table 4.

3.5. Research questions

In this study, we aim to answer the following major research questions to analyze performance of selected 10 ensemble software fault predictors:

(i) What is the most powerful base predictor from literature (creating a baseline algorithm for evaluation)?
(ii) What is the performance of selected ensemble predictors compared to reference predictor?
(iii) How might the number of base predictors in the committee affect fault detection performance of ensemble predictors?
(iv) What are the possible ensemble combinations that might change performance of ensemble predictors? In this query, we need to answer miscellaneous questions; (a) performance of combination of best base predictors, (b) combina-

Please cite this article as: F. Yucalar, A. Ozcift, E. Borandag et al., Multiple-classifiers in software quality engineering: Combining predictors to improve software fault prediction ability, Engineering Science and Technology, an International Journal, https://doi.org/10.1016/j.jestch.2019.10.005

Table 4
Ensemble predictors selected from literature.

No	Algorithm	Prediction Model	Prediction Strategy
1	AdaboostM1 (AB)	Boosting	The most well-known boosting technique trains predictors sequentially. Algorithm at the end of each training round determines mis-classified examples and in the next round it focuses on mis-classified examples while training a new predictor [43,44].
2	Logic Boost (LB)		LB uses regressors in the ensemble and it makes use of additive logistic regression (a variant of boosting applied to regression) strategy to decide on the output of base regressors [48].
3	Multiboost AB (MAB)		MAB is a technique that combines AdaBoost and wagging. It makes use of variance reduction techniques of AdaBoost's wagging. MAB produces decision committees using J48 having lower error than either AdaBoost or wagging [10].
4	Bagging (BG)	Bagging	As it is explained, each predictor of bagging ensemble is constructed with the use of bootstrap sample from the original data.
5	Random Forest (RF)		RF [49] is a combination of bagging algorithm and random subspaces [50] method. The algorithm uses decision trees as predictors while constructing them with bootstrap samples of original data. The major novelty of algorithm compared to generic bagging is that it makes use of random attributes in the training of predictors hence providing further diversity.
6	Dagging (DG)		DG [51] is a kind of bagging algorithm. However, it is different in the usage of data to construct base predictors. While bagging provides bootstrap samples to predictors, DG makes use of disjoint samples to build predictors.
7	Rotation Forest (ROF)		ROF [52] is similar to random forests using random attributes while building ensemble of decision trees. However, ROF uses a Principal Components Analysis transformation to provide predictors with diverse attribute sets, while training the members of committee.
8	Stacking (ST)	Combination Rule	As it is explained, ST uses a predictor to combine output of base learners in the ensemble. In general, the algorithm uses heterogeneous predictors to increase performance of its ensemble strategy.
9	Multi scheme (MS)		MS might use multiple predictors in its ensemble. While it makes decision on the predictions of base learners, it makes use of a cross-validation strategy to determine the best predictor through the ensemble [10].
10	Voting (VT)		VT makes use of 'votes' or 'algebraic combinations' of heterogeneous predictors taking place in the ensemble, while making decision on a prediction.

tion of best ensemble predictors (c) performance of ensemble predictor while its default base predictor is replaced with another base predictor.

4. Experimental case studies

We have to state some essential points that were followed while carrying out experiments of this study:

(i) We used default parameters of WEKA suit for the all predictors used in the study.
(ii) We used 10-fold-cross validation through all experiments.
(iii) We limited ourselves to use default predictors of ensemble algorithms in the experiments. However, it must be noted that each ensemble algorithm might also be used with different base predictors.
(iv) We selected three predictors namely J48, Naïve Bayes and Logistics Regression for voting ensemble scheme (with average voting strategy) among many variations.
(v) We selected three predictors i.e. SMO, KStar and J48 for Multi Scheme ensemble algorithm from various possible combinations.

4.1. Selection of performance metrics

The average performance of base predictors in terms of FM and AUC is calculated from Tables 5–8 as 0.749 and 0.631. Use of FM as a reference metric, we determine the best three predictors as J48, PART and JR with values of 0.802, 0.798 and 0.794. However, this does not illustrate the real case. The AUC values of three predictors are 0.634, 0.580 and 0.684. These three values are only around average AUC value that means a poor PD-PF performance. The ROC curves that make deeper analysis of MLP and J48 while they applied to CM1 dataset is given in Fig. 4. Though MLP and J48 have similar FM values of 0.845 and 0.852 respectively, it is inspected from Fig. 4 that PD-PF rate of MLP is positioned (more inclined to upper left corner) healthier compared to the ROC curve of J48. Therefore, with the use of AUC values, the best three predictors are concluded to be MLP, LR and NB with values of 0.743, 0.724 and 0.693 respectively. Furthermore, the FM measures of these predictors are 0.790, 0.786 and 0.757. The performance of the most successful base predictor is hence determined to be as MLP algorithm. The algorithm has the highest AUC value of 0.743 with acceptable FM measure of 0.790 compared to overall average FM value.

As a last point for base predictor performance, we can inspect Tables 5–8 to determine performance of algorithms in terms of domain i.e. the source of software projects. As it was given in Table 1, there are three types of software sources and we calculated average AUC qualities of algorithms with regard to software groups.

We observed that MLP is still the best algorithm with values 0.775, 0.755 and 0.723. However, in white-good software projects NB is slightly better with AUC value of 0.779. In overall performance, MLP is still decided to be the best predictor in fault detection performance.

In the following section, we continue with the analysis of ensemble predictor performances in fault detection.

4.2. Experiments to evaluate ensemble predictor performance

We determined the best base predictor to be MLP in section 5.1 and we will make use of FM and AUC values of this predictor to be a reference for evaluating the results of ensemble predictions given in Table 9 and 10.

Inspection of Table 9 in terms of FM value reveals that ROF is better than the rest of the ensemble predictors with value of 0.808. At the same time, VOT, LB and DG with FM values of 0.805, 0.804 and 0.801 are observed to be competitive at first glance. However, an evaluation of Table 10 in terms of AUC reveals that DG might not be a successful fault predictor compared to ROF. Both ROF and DG has the same FM value of 0.86 for ivy2_0 dataset and sample ROC curves of the algorithms applied to ivy2_0 is given in Fig. 5.

ROC curve comparison from Fig. 5 demonstrates that fault prediction efficiency (PD-PF balance) of ROF is better than that of DG algorithm for sample ivy2_0 project. Similarly, in Tables 9 and 10, BG and MS have relatively high FM with irrelevant AUC. Most significantly, ST has the lowest AUC value among all predictors of

Please cite this article as: F. Yucalar, A. Ozcift, E. Borandag et al., Multiple-classifiers in software quality engineering: Combining predictors to improve software fault prediction ability, Engineering Science and Technology, an International Journal, https://doi.org/10.1016/j.jestch.2019.10.005

Table 5
Software Fault Detection Performance of Base Predictors for FM value (first eight algorithms).

	Base Predictors							
Dataset	NB	J48	LR	RBF	SMO	KS	OR	PG
ar5	0.840	0.833	0.710	0.868	0.833	0.701	0.904	0.901
ar6	0.816	0.788	0.774	0.813	0.827	0.810	0.023	0.855
cm1	0.858	0.852	0.861	0.852	0.852	0.863	0.840	0.755
jm1	0.770	0.769	0.759	0.741	0.722	0.770	0.866	0.849
kc1	0.918	0.893	0.908	0.922	0.924	0.904	0.348	0.513
mc2	0.710	0.682	0.724	0.623	0.664	0.631	0.680	0.693
pc1	0.895	0.921	0.901	0.897	0.897	0.911	0.867	0.889
pc3	0.573	0.868	0.873	0.849	0.849	0.869	0.711	0.851
intercafe	0.745	0.915	0.778	0.681	0.859	0.765	0.820	0.839
ivy2_0	0.848	0.855	0.865	0.847	0.853	0.845	0.744	0.723
jedit_4_3	0.682	0.712	0.711	0.693	0.664	0.725	0.446	0.684
lucene_2_4	0.572	0.678	0.696	0.678	0.681	0.655	0.784	0.804
serapion	0.750	0.725	0.732	0.778	0.832	0.736	0.855	0.851
tomcat	0.867	0.877	0.889	0.867	0.888	0.876	0.541	0.697
workflow	0.521	0.664	0.615	0.586	0.511	0.615	0.922	0.931
AVERAGE	0.757	**0.802**	0.786	0.779	0.790	0.778	0.690	0.789

Table 6
Software Fault Detection Performance of Base Predictors for AUC value (first eight algorithms).

	Base Predictors							
Dataset	NB	J48	LR	RBF	SMO	KS	OR	PG
ar5	0.907	0.717	0.725	0.830	0.759	0.707	0.529	0.513
ar6	0.652	0.478	0.667	0.557	0.567	0.507	0.500	0.554
cm1	0.658	0.558	0.730	0.663	0.497	0.644	0.804	0.661
jm1	0.679	0.653	0.713	0.669	0.502	0.638	0.545	0.500
kc1	0.780	0.599	0.670	0.609	0.555	0.753	0.500	0.513
mc2	0.700	0.630	0.723	0.585	0.587	0.613	0.681	0.693
pc1	0.650	0.668	0.809	0.646	0.500	0.655	0.500	0.566
pc3	0.756	0.599	0.823	0.675	0.500	0.689	0.500	0.708
intercafe	0.231	0.598	0.592	0.296	0.625	0.457	0.561	0.638
ivy2_0	0.765	0.681	0.764	0.648	0.545	0.681	0.533	0.503
jedit_4_3	0.725	0.718	0.810	0.733	0.671	0.819	0.500	0.677
lucene_2_4	0.728	0.687	0.766	0.714	0.666	0.734	0.500	0.582
serapion	0.768	0.639	0.722	0.762	0.694	0.710	0.517	0.496
tomcat	0.799	0.619	0.794	0.745	0.556	0.751	0.463	0.628
workflow	0.605	0.666	0.560	0.615	0.514	0.660	0.607	0.614
AVERAGE	0.693	0.634	0.724	0.649	0.582	0.667	0.549	0.589

Table 7
Software Fault Detection Performance of Base Predictors for FM value (last eight algorithms).

	Base Predictors							
Dataset	VP	IB	HYP	VFI	JR	PART	CART	MLP
ar5	0.895	0.921	0.897	0.221	0.918	0.926	0.907	0.701
ar6	0.813	0.854	0.682	0.506	0.850	0.855	0.833	0.816
cm1	0.386	0.786	0.766	0.819	0.786	0.833	0.778	0.845
jm1	0.838	0.870	0.849	0.187	0.854	0.855	0.849	0.741
kc1	0.348	0.588	0.485	0.564	0.631	0.590	0.348	0.931
mc2	0.511	0.678	0.346	0.394	0.755	0.732	0.689	0.68
pc1	0.865	0.880	0.874	0.292	0.872	0.884	0.867	0.917
pc3	0.529	0.736	0.802	0.777	0.714	0.725	0.711	0.865
intercafe	0.827	0.824	0.831	0.781	0.813	0.781	0.778	0.859
ivy2_0	0.571	0.766	0.723	0.488	0.768	0.763	0.758	0.848
jedit_4_3	0.495	0.679	0.473	0.531	0.655	0.653	0.446	0.707
lucene_2_4	0.454	0.704	0.784	0.710	0.859	0.915	0.784	0.679
serapion	0.855	0.843	0.850	0.420	0.861	0.851	0.855	0.841
tomcat	0.284	0.659	0.659	0.494	0.668	0.692	0.664	0.889
workflow	0.903	0.946	0.916	0.832	0.908	0.917	0.918	0.535
AVERAGE	0.638	0.782	0.729	0.534	0.794	0.798	0.745	0.790

Table 10 and the algorithm produces unacceptable fault prediction performance at least for these software projects.

As it was mentioned, we utilize MLP prediction outcomes to evaluate the performance of ensemble predictors. Therefore, from Tables 9 and 10, we determined ensemble predictors similar or better than MLP in terms of FM and AUC values and the results of corresponding comparison is given in Table 11.

Table 11 demonstrates the promising ensemble algorithms (with default parameters and with default number of base predictors) compared to baseline predictor MLP. On the other hand, we

Please cite this article as: F. Yucalar, A. Ozcift, E. Borandag et al., Multiple-classifiers in software quality engineering: Combining predictors to improve software fault prediction ability, Engineering Science and Technology, an International Journal, https://doi.org/10.1016/j.jestch.2019.10.005

Table 8
Software Fault Detection Performance of Base Predictors for AUC value (last eight algorithms).

	Base Predictors							
Dataset	VP	IB	HYP	VFI	JR	PART	CART	MLP
ar5	0.499	0.740	0.519	0.653	0.602	0.814	0.600	0.799
ar6	0.544	0.619	0.663	0.689	0.564	0.742	0.497	0.751
cm1	0.384	0.750	0.866	0.772	0.656	0.717	0.676	0.734
jm1	0.485	0.642	0.528	0.713	0.530	0.724	0.499	0.690
kc1	0.536	0.570	0.576	0.562	0.613	0.663	0.476	0.916
mc2	0.574	0.682	0.493	0.566	0.748	0.748	0.695	0.690
pc1	0.518	0.639	0.623	0.747	0.530	0.697	0.486	0.723
pc3	0.644	0.529	0.739	0.923	0.610	0.639	0.414	0.780
intercafe	0.598	0.666	0.619	0.652	0.537	0.526	0.397	0.772
ivy2_0	0.559	0.640	0.503	0.663	0.570	0.712	0.665	0.726
jedit_4_3	0.556	0.677	0.609	0.683	0.645	0.668	0.487	0.768
lucene_2_4	0.272	0.543	0.625	0.543	0.435	0.598	0.174	0.762
serapion	0.500	0.589	0.552	0.651	0.529	0.721	0.490	0.750
tomcat	0.527	0.582	0.669	0.671	0.578	0.682	0.560	0.677
workflow	0.704	0.724	0.809	0.852	0.562	0.619	0.391	0.611
AVERAGE	0.526	0.639	0.626	0.689	0.580	0.684	0.500	**0.743**

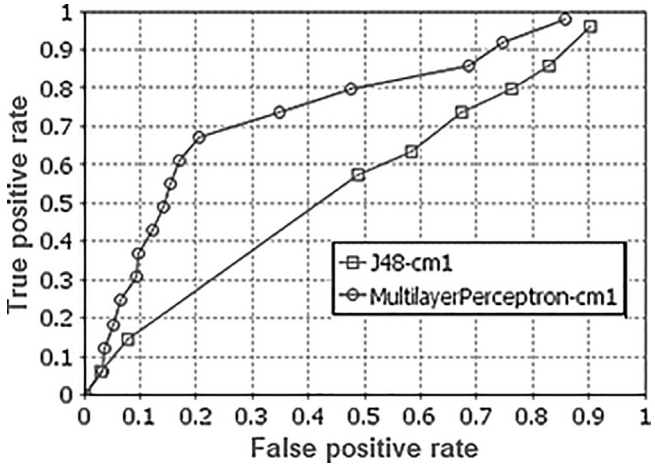

Fig. 4. ROC curves of MLP and J48 applied to CM1 dataset.

will further analyze performance of unselected ensemble predictors in Section 5.3 to evaluate their performances in terms of parameters of corresponding algorithms.

From Table 11 it is easily seen that ROF has the best average fault detection performance among all ensemble predictors with best FM and AUC values. On the other hand, for the analysis of software projects in group base, it is observed that ROF has slightly variable performance. In the first and second group projects ROF has satisfactory values whereas LB is slightly better in third group projects.

We here provide a visual comparison of the best three ensemble predictors, ROF, VOT and RF from Table 11 with the use of FM and AUC values in Figs. 6 and 7 respectively. For the sake of convenience, we make use of numbers from Table 1 instead of software project names.

Inspection of Fig. 6 shows that there is no dramatic performance difference between three best ensemble predictors except a few projects. However, it is easily seen that ROF has better performance in terms of AUC, i.e. major evaluation metric in fault prediction efficiency, compared to VOT and RF.

It might be concluded from evaluation of overall experimental results that ROF is the most efficient fault predictor compared to the algorithms in Tables 9 and 10. Furthermore, VOT and RF (partially LB) are also promising ensemble predictors for fault detection purposes. However, other than algorithm specific parameters, there are mainly two key points that affect the performance of ensemble predictors: (i) the type of the base predictor and (ii) the number of the predictors in the ensemble. These two points are hence important research questions to be evaluated. On the other hand, we will not take algorithms specific parameters of ensemble algorithms into the scope of this study. In the next section, we design a few experiments to evaluate effect of the above-mentioned points on the fault prediction performance of algorithms.

4.3. Experiments to evaluate ensemble predictor performance

In this section, we design experiments to answer following questions empirically:

(i) How does the number of predictors in the ensemble affect fault detection performance?
(ii) How does the type of the base predictor in the ensemble affect fault detection performance of algorithms?

Because, it is not feasible to design an experiment for each ensemble algorithm, we make selection among ten ensemble predictors randomly.

4.4. Number of base predictors in the ensemble:

As Tables 9 and 10 are examined for ROF, AB and RF algorithms, it is observed that the three algorithms have relatively low fault prediction performance in workflow, ar6 and mc2 respectively. We design experiments to evaluate performance of three sample algorithms applied to mentioned datasets while increasing base predictor number starting from 10 to 100. In the experiments, we use 10-fold cross validation and we make measures in terms of PD-PF and AUC. The results of the experiments are given in Table 12.

Table 12 demonstrates that performance of ensemble predictors might be affected with the number of base predictors used in the committee. High PD-AUC and low PF rates might be provided with larger number of base predictors. The corresponding performance charts for each of the algorithms are given in Figs. 8–10.

It may be observed from inspection of Figs. 8–10 that ROF, AB and RF has best performances with the base predictor numbers of 20, 80 and 90 in respective order. The three ensemble algorithms with increased number of base predictors have better PD-PF and AUC values to 10-base predictors as default.

Please cite this article as: F. Yucalar, A. Ozcift, E. Borandag et al., Multiple-classifiers in software quality engineering: Combining predictors to improve software fault prediction ability, Engineering Science and Technology, an International Journal, https://doi.org/10.1016/j.jestch.2019.10.005

Table 9
Software Fault Detection Performance of Ensemble Predictors for FM value.

	Ensemble Predictor Algorithms									
Dataset	AB	BG	DG	MAB	LB	ROF	ST	RF	MS	VOT
ar5	0.801	0.728	0.883	0.778	0.824	0.791	0.681	0.810	0.701	0.864
ar6	0.765	0.783	0.806	0.794	0.806	0.831	0.783	0.838	0.810	0.824
cm1	0.855	0.852	0.853	0.853	0.858	0.854	0.855	0.852	0.863	0.850
jm1	0.727	0.771	0.723	0.72	0.756	0.761	0.732	0.782	0.77	0.767
kc1	0.920	0.918	0.936	0.936	0.936	0.941	0.918	0.941	0.904	0.917
mc2	0.655	0.710	0.688	0.699	0.667	0.690	0.547	0.669	0.631	0.716
pc1	0.897	0.918	0.897	0.898	0.914	0.929	0.897	0.924	0.911	0.913
pc3	0.849	0.855	0.849	0.849	0.864	0.866	0.849	0.876	0.869	0.878
intercafe	0.859	0.784	0.784	0.830	0.859	0.830	0.784	0.765	0.859	0.859
ivy2_0	0.842	0.833	0.860	0.835	0.845	0.860	0.833	0.849	0.854	0.865
jedit_4_3	0.686	0.741	0.691	0.682	0.726	0.752	0.346	0.762	0.725	0.706
lucene_2_4	0.626	0.446	0.696	0.639	0.658	0.681	0.446	0.664	0.696	0.688
serapion	0.802	0.711	0.838	0.804	0.802	0.804	0.711	0.700	0.832	0.725
tomcat	0.869	0.867	0.881	0.867	0.886	0.879	0.867	0.867	0.876	0.889
workflow	0.641	0.494	0.631	0.615	0.666	0.664	0.348	0.641	0.535	0.615
AVERAGE	0.786	0.761	0.801	0.786	0.804	**0.808**	0.706	0.796	0.789	0.805

Table 10
Software Fault Detection Performance of Ensemble Predictors for AUC value.

	Ensemble Predictor Algorithms									
Dataset	AB	BG	DG	MAB	LB	ROF	ST	RF	MS	VOT
ar5	0.808	0.795	0.844	0.790	0.804	0.853	0.366	0.777	0.707	0.786
ar6	0.585	0.687	0.693	0.499	0.593	0.674	0.402	0.639	0.507	0.656
cm1	0.700	0.734	0.551	0.712	0.724	0.771	0.490	0.723	0.644	0.741
jm1	0.710	0.743	0.517	0.689	0.713	0.729	0.491	0.718	0.638	0.721
kc1	0.755	0.717	0.798	0.688	0.674	0.87	0.391	0.812	0.753	0.757
mc2	0.591	0.699	0.681	0.634	0.653	0.740	0.48	0.699	0.613	0.758
pc1	0.803	0.827	0.569	0.748	0.843	0.859	0.486	0.813	0.655	0.842
pc3	0.782	0.820	0.548	0.793	0.819	0.827	0.499	0.795	0.689	0.792
intercafe	0.761	0.174	0.554	0.707	0.761	0.652	0.174	0.658	0.625	0.565
ivy2_0	0.781	0.499	0.676	0.807	0.767	0.742	0.497	0.769	0.566	0.786
jedit_4_3	0.801	0.819	0.767	0.793	0.805	0.820	0.491	0.832	0.819	0.806
lucene_2_4	0.664	0.502	0.745	0.685	0.707	0.767	0.487	0.742	0.688	0.766
serapion	0.818	0.423	0.835	0.887	0.818	0.849	0.414	0.787	0.694	0.737
tomcat	0.777	0.501	0.626	0.759	0.787	0.811	0.486	0.791	0.751	0.810
workflow	0.643	0.508	0.642	0.645	0.632	0.687	0.476	0.624	0.541	0.664
AVERAGE	0.731	0.629	0.669	0.722	0.740	**0.776**	0.442	0.745	0.659	0.745

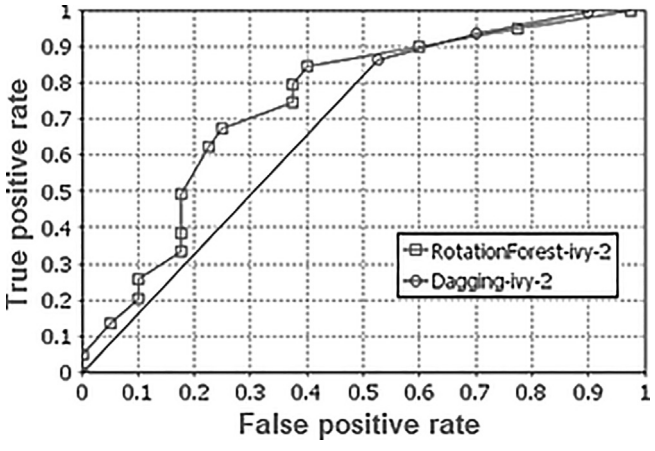

Fig. 5. ROC curves of ROF and DG applied to ivy2_0 dataset.

4.5. Number of base predictors in the ensemble

We design two kinds of experiments in order to analyse the effect of base predictor on the fault prediction performance of algorithms: (i) ROF algorithm with miscellaneous base predictors and (ii) Vote strategy with best three base predictors (MLP, LR and NB) and with best three ensemble predictors.

(i) Each ensemble algorithm in Tables 9 and 10 have a default base predictor. However, it is also possible to use the algorithms with different base classifiers. As a sample, we use ROF with two successful base predictors i.e. MLP and LB to analyse the performance of the corresponding algorithm. The results of experiments in terms of PD-PF and AUC are given in Table 13.

It is seen from Table 13 that ROF with default base predictor J48 seems better in terms of PD and AUC. However, ROF-MLP combination has a better PF value compared to ROF-J48. However, there is no significant difference between the two algorithms. On the other hand, compared to the performance of ROF-LB combination, either of the two algorithms might be preferred as successful fault predictors.

(ii) Vote algorithm has almost infinite ensemble strategies to make enhanced fault predictor combinations. In this study, first we selected VOT with J48, LR and NB. As a sample, we replace J48 with MLP from VOT strategy and observe the performance of new ensemble. The result of this experiment once again in terms of PD-PF and AUC is given in Table 14. For a second experiment, we research the answer of performance of three best ensembles (ROF, RF and LB) in voting strategy and the result of this experiment is given in Table 15.

Please cite this article as: F. Yucalar, A. Ozcift, E. Borandag et al., Multiple-classifiers in software quality engineering: Combining predictors to improve software fault prediction ability, Engineering Science and Technology, an International Journal, https://doi.org/10.1016/j.jestch.2019.10.005

Table 11
Software Fault Detection Performance of Ensemble Predictors.

	Ensemble Predictors versus MLP									
	MLP		LB		ROF		RF		VOT	
Dataset	FM	AUC	FM	AUC	FM	AUC	FM	AUC	FM	AUC
ar5	0.701	0.799	0.824	0.804	0.791	0.853	0.810	0.777	0.864	0.786
ar6	0.816	0.751	0.806	0.593	0.831	0.674	0.838	0.639	0.824	0.656
Avg1	**0.758**	**0.775**	0.815	0.698	**0.811**	**0.763**	0.824	0.708	0.844	0.721
cm1	0.845	0.734	0.858	0.724	0.854	0.771	0.852	0.723	0.850	0.741
jm1	0.741	0.690	0.756	0.713	0.761	0.729	0.782	0.718	0.767	0.721
kc1	0.931	0.916	0.936	0.674	0.941	0.870	0.941	0.812	0.917	0.757
mc2	0.680	0.690	0.667	0.653	0.690	0.740	0.669	0.699	0.716	0.758
pc1	0.917	0.723	0.914	0.843	0.929	0.859	0.924	0.813	0.913	0.842
pc3	0.865	0.780	0.864	0.819	0.866	0.827	0.876	0.795	0.878	0.792
Avg2	0.855	0.757	0.861	0.771	**0.860**	**0.799**	0.864	0.759	0.864	0.766
intercafe	0.859	0.772	0.859	0.761	0.830	0.652	0.765	0.658	0.859	0.565
ivy2_0	0.848	0.726	0.845	0.767	0.860	0.742	0.849	0.769	0.865	0.786
jedit_4_3	0.707	0.768	0.726	0.805	0.752	0.82	0.762	0.832	0.706	0.806
lucene_2_4	0.679	0.762	0.658	0.707	0.681	0.767	0.664	0.742	0.688	0.766
serapion	0.841	0.750	0.802	0.818	0.804	0.849	0.700	0.787	0.725	0.737
tomcat	0.889	0.677	0.886	0.787	0.879	0.811	0.867	0.791	0.889	0.810
workflow	0.535	0.611	0.666	0.632	0.664	0.687	0.641	0.624	0.615	0.664
Avg3	0.697	0.691	**0.762**	**0.696**	0.747	0.669	0.703	0.641	0.737	0.614
AVERAGE	0.790	0.743	0.804	0.740	**0.808**	**0.776**	0.796	0.745	0.805	0.745

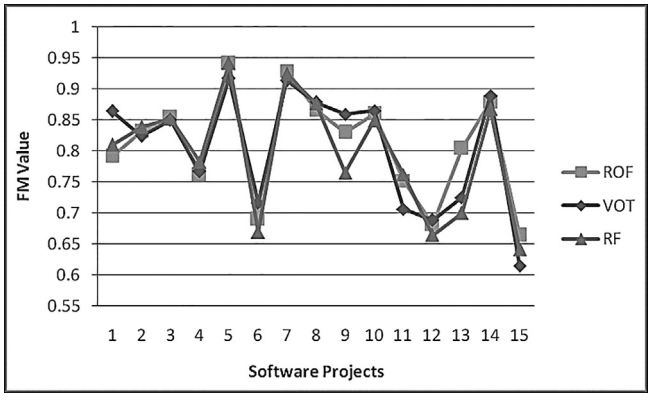

Fig. 6. Comparison of ROF, VOT and RF in terms of FM value.

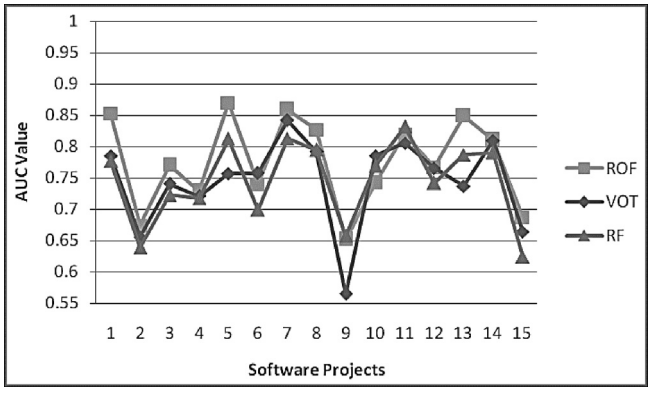

Fig. 7. Comparison of ROF, VOT and RF in terms of AUC value.

Inspection of Table 14 demonstrates that VOT with (MLP, NB, LR) has decrease in false alarm rate (PF) and in prediction rate (PD) as much as % 1.2. Though decrease in PF is better for a predictor, the same amount of decrease in PD is not a desired case. On the other hand, the new combination has increase in AUC as % 2.7 that might be a preference for the performance of the new VOT strategy.

Results of Table 15 demonstrate that the best three ensemble predictors do not provide best results that might be expected. PD value of 0.817 is somewhat around the previous findings of Section 5.3.2. However, PF is significantly the worst of the results of this section. Original average AUC values of ROF, RF and LB were 0.776, 0.745 and 0.740. However, the VOT combination of these predictors produces average AUC of 0.729 that is lower than each of the three algorithms alone.

5. Threads to validity

This empirical study uses 15 bug prediction datasets from a Turkish white-goods manufacturer [7], NASA projects [22,23], various academic software projects and Apache related open-source applications [24].

In the experiments above, there is not substantial threats for internal validity analysis. However, the main issues of such studies is that any proposed method is not tested sufficiently to prove its generalized efficiency. To handle the mentioned issue in this study, we made experiments with 15 datasets and 26 classifiers. With the goal of benchmarking various ensemble predictors in fault prediction ability, it was observed that ensemble predictors are efficient to improve base classifier performances with the use of static code metrics. In particular, Rotation Forest perform well.

Another important aspect that should be taken into account is that the nature of the software fault prediction datasets is unbalanced. In order to evaluate the experimental results of unbalanced nature, we preferred FM and AUC metrics which are more reliable measures in evaluation of classification algorithms.

One other thread is that the performance of machine learning algorithms are affected with feature filtering. Though, we did not benefit feature filtering approaches to strictly evaluate ensemble learners, it could further improve the prediction performance of the evaluated algorithms.

6. Conclusion and contributions

Development of new defect predictors is a continuous research area among the software quality engineers. Hence, it is anticipated that better software predictors will decrease the required effort to detect faults in software modules. In the literature many base predictors have been developed to satisfy this need. It is also seen

Please cite this article as: F. Yucalar, A. Ozcift, E. Borandag et al., Multiple-classifiers in software quality engineering: Combining predictors to improve software fault prediction ability, Engineering Science and Technology, an International Journal, https://doi.org/10.1016/j.jestch.2019.10.005

Table 12
Three sample ensemble predictors evaluating base predictor number effect on performance.

	ROF in workflow			AB in ar6			RF in mc2		
# of predictors	PD	PF	AUC	PD	PF	AUC	PD	PF	AUC
10	0.667	0.338	0.687	0.792	0.807	0.585	0.689	0.490	0.699
20	**0.718**	**0.286**	**0.745**	0.822	0.747	0.607	0.677	0.506	0.702
30	0.615	0.384	0.742	0.842	0.688	0.608	0.696	0.497	0.693
40	0.667	0.335	0.721	0.822	0.692	0.628	0.702	0.484	0.694
50	0.615	0.386	0.695	0.822	0.747	0.625	0.708	0.471	0.699
60	0.590	0.413	0.695	0.832	0.690	0.657	0.702	0.474	0.701
70	0.692	0.308	0.718	0.832	0.690	0.643	0.708	0.471	0.710
80	0.564	0.435	0.682	**0.842**	**0.633**	**0.645**	0.702	0.474	0.703
90	0.641	0.359	0.703	0.832	0.635	0.639	**0.714**	**0.458**	**0.704**
100	0.615	0.386	0.708	0.832	0.635	0.650	0.708	0.461	0.703

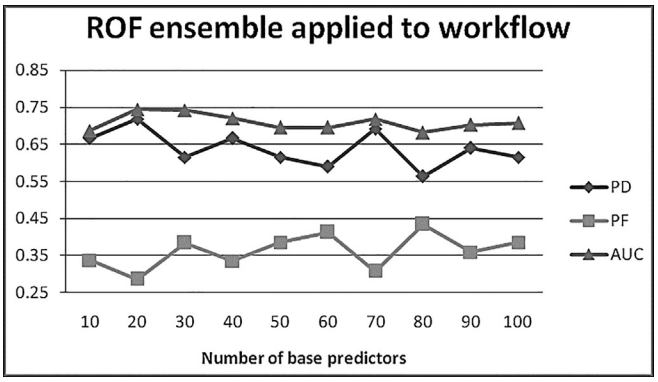

Fig. 8. ROF performance with varying base predictor numbers.

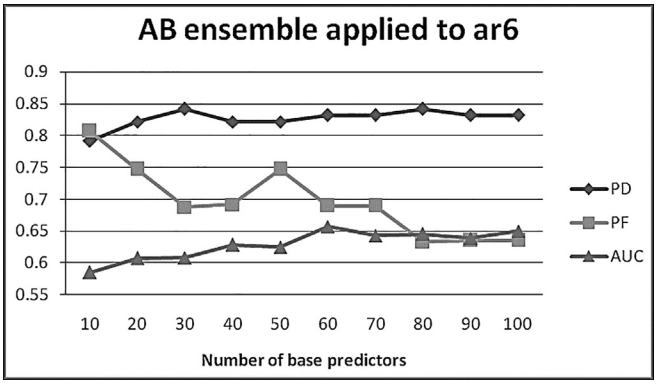

Fig. 9. AB performance with varying base predictor numbers.

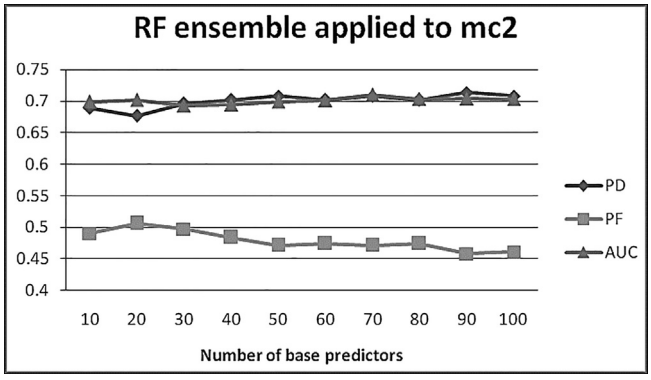

Fig. 10. RF performance with varying base predictor numbers.

from machine learning literature that ensemble of predictors might be another solution to increase performance of predictors. Most of the scientific disciplines make use of ensemble strategies to enhance performance of a prediction system. However, up to now only a few studies have used ensemble prediction methods in software fault detection. This main contribution of this study is to evaluate performance of a wide range of ensemble algorithms as software fault predictors. As the experimental results demonstrate in terms of FM and AUC measures, ensemble algorithms, particularly Rotation Forest, might be used as highly developed fault predictors. Other than ROF, Random Forest, Logic Boost, Adaboost and Voting are alternative successful fault predictors that might be used in software quality studies.

Another contribution of our study is to demonstrate the effect of number of base predictors in the ensemble on the prediction performance of the algorithms. It is empirically shown for ROF, AB and RF ensemble predictors that the fault prediction performance of the algorithms might be enhanced with the use of more predictors in the committee. One of the main drawbacks of such increase is the extra computational load in the fault detection process. However, such a cost is still tolerable compared to effort necessary with manual inspection of faults. Furthermore, though we have not included in the study, we checked some error metrics while making experiments. And we have observed that the error metrics have become better as the number of base predictors is increased in the community.

As a third contribution, we empirically showed that there might be many ensemble choices to be used as enhanced fault predictors. For instance, ROF-MLP combination, though it is not outperforming original ROF-J48 strategy, has a decreased false alarm rate (PD) that might be an asset. Particularly, VOT with (MLP, NB, LR) compared to VOT with (J48, NB, LR) has % 1.2 decrease in false alarm rate with a % 2.7 increase in AUC. On the other hand, the combination has % 1.2 decrease in prediction rate. In mission-critical projects, increase in prediction rate is preferred to decrease in false alarm rate. However, in a commercial software project, it is important to optimize PD-PF at the same time. Therefore, the predictor selection is up to the project type and it must be decided by the software engineer.

One other contribution of our study is the evaluation of two relatively new software projects i.e. first and third group projects. In the literature, NASA projects are studied frequently as a reference to test performance of fault predictors. In our study, we evaluated 16 base predictors and particularly 10 ensemble predictors applied to these data sets. This would also be a one benefit of our study to the researchers that will use mentioned datasets as references to test their predictors.

Though we have not included in this empirical study, we have also tested Decorate ensemble algorithm with three kinds of software projects. The restrictions of algorithm have only let 9 projects to be tested and it has produced average FM and AUC values as

Please cite this article as: F. Yucalar, A. Ozcift, E. Borandag et al., Multiple-classifiers in software quality engineering: Combining predictors to improve software fault prediction ability, Engineering Science and Technology, an International Journal, https://doi.org/10.1016/j.jestch.2019.10.005

Table 13
Comparison of ROF-J48, ROF-LB and ROF-MLP ensemble fault predictors.

	ROF (J48)			ROF (LR)			ROF (MLP)		
Dataset	PD	PF	AUC	PD	PF	AUC	PD	PF	AUC
ar5	0.778	0.242	0.853	0.722	0.526	0.737	0.750	0.429	0.772
ar6	0.861	0.685	0.674	0.762	0.592	0.689	0.901	0.513	0.742
cm1	0.900	0.902	0.771	0.884	0.813	0.733	0.884	0.831	0.751
jm1	0.817	0.717	0.729	0.814	0.717	0.713	0.812	0.745	0.711
kc1	0.945	0.592	0.870	0.897	0.712	0.747	0.938	0.592	0.933
mc2	0.702	0.444	0.740	0.727	0.372	0.721	0.696	0.447	0.739
pc1	0.942	0.689	0.859	0.924	0.871	0.807	0.935	0.762	0.809
pc3	0.900	0.814	0.827	0.896	0.748	0.824	0.888	0.777	0.810
Intercafe	0.852	0.645	0.652	0.778	0.658	0.606	0.852	0.439	0.533
ivy2_0	0.889	0.755	0.742	0.886	0.690	0.763	0.869	0.692	0.777
jedit_4_3	0.752	0.247	0.820	0.712	0.290	0.810	0.734	0.267	0.804
lucene_2_4	0.685	0.362	0.767	0.697	0.316	0.766	0.694	0.323	0.775
Serapion	0.800	0.383	0.849	0.733	0.317	0.728	0.778	0.556	0.787
Tomcat	0.909	0.84	0.811	0.910	0.758	0.794	0.902	0.759	0.799
Workflow	0.667	0.338	0.687	0.667	0.335	0.666	0.590	0.411	0.611
AVG	**0.826**	0.577	**0.776**	0.800	0.581	0.740	0.814	**0.569**	0.756

Table 14
Comparison of VOT strategies with (J48, LR, NB) and with (MLP, LR, NB) combinations.

	VOT (J48, LR, NB)			VOT (MLP, LR, NB)		
Dataset	PD	PF	AUC	PD	PF	AUC
ar5	0.861	0.218	0.786	0.750	0.339	0.830
ar6	0.842	0.633	0.656	0.832	0.580	0.711
cm1	0.876	0.850	0.741	0.88	0.777	0.782
jm1	0.807	0.674	0.721	0.806	0.665	0.707
kc1	0.917	0.711	0.757	0.931	0.593	0.890
mc2	0.739	0.446	0.758	0.733	0.459	0.782
pc1	0.929	0.774	0.842	0.931	0.738	0.778
pc3	0.882	0.584	0.792	0.861	0.498	0.799
intercafe	0.889	0.639	0.565	0.852	0.645	0.707
ivy2_0	0.872	0.605	0.786	0.878	0.691	0.778
jedit_4_3	0.708	0.295	0.806	0.697	0.308	0.808
lucene_2_4	0.685	0.286	0.766	0.685	0.312	0.748
serapion	0.711	0.489	0.737	0.822	0.378	0.809
tomcat	0.899	0.677	0.810	0.843	0.621	0.820
workflow	0.615	0.384	0.664	0.513	0.481	0.626
AVG	**0.815**	0.551	0.745	0.801	**0.539**	**0.772**

Table 15
Fault prediction performance of best three ensemble predictors (ROF, RF, LB).

	VOT (ROF, RF, LB)		
Dataset	PD	PF	AUC
ar5	0.861	0.218	0.817
ar6	0.851	0.741	0.671
cm1	0.888	0.903	0.784
jm1	0.816	0.694	0.753
kc1	0.945	0.592	0.845
mc2	0.720	0.475	0.718
pc1	0.939	0.702	0.876
pc3	0.902	0.792	0.845
Intercafe	0.889	0.639	0.723
ivy2_0	0.886	0.756	0.787
jedit_4_3	0.774	0.226	0.835
lucene_2_4	0.688	0.360	0.756
serapion	0.844	0.456	0.827
tomcat	0.915	0.828	0.808
workflow	0.692	0.311	0.663
AVG	0.817	0.717	0.729

0.820 and 0.746. The algorithm in this aspect is promising and it might be an alternative fault predictor.

As a future direction, we plan combining ensemble predictors with feature selection techniques to further evaluate fault prediction performance of ensemble strategies.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

References

[1] B. Turhan, G. Kocak, A. Bener, Data mining source code for locating software bugs: a case study in telecommunication industry, Expert Syst. Appl. 36 (6) (2009) 9986–9990.
[2] A. Tosun, A. Bener, R. Kale, AI-based software defect predictors: applications and benefits in a case study, AI Magazine 32 (2) (2011) 57–68.
[3] T. Menzies, J. Greenwald, A. Frank, Data mining static code attributes to learn defect predictors, IEEE Trans. Software Eng. 33 (1) (2007) 2–13.
[4] Y. Peng, G. Kou, G. Wang, W. Wu, Y. Shi, Ensemble of software defect predictors: an Ahp-based evaluation method, Int. J. Inform. Technol. Decision Making 10 (1) (2011) 187–206.
[5] C. Catal, Software fault prediction: a literature review and current trends, Expert Syst. Appl. 38 (4) (2011) 4626–4636.
[6] A. Tosun, A. Bener, B. Turhan, T. Menzies, Practical considerations in deploying statistical methods for defect prediction: a case study within the Turkish telecommunications industry, Inf. Softw. Technol. 52 (2010) 1242–1257.
[7] A. Tosun Misirli, A. Bener, B. Turhan, An, industrial case study of classifier ensembles for locating software defects, Software Quality Control 19 (3) (2011) 515–536.
[8] B. Twala, Software faults prediction using multiple classifiers, in: Computer Research and Development (ICCRD), in: 3rd International Conference on Computer Research and Development (ICCRD), 2011, pp. 504–510.
[9] S. Zhong, T.M. Khoshgoftaar, N. Seliya, Analyzing software measurement data with clustering techniques, IEEE Intell. Syst. 19 (2) (2004) 20–27.

Please cite this article as: F. Yucalar, A. Ozcift, E. Borandag et al., Multiple-classifiers in software quality engineering: Combining predictors to improve software fault prediction ability, Engineering Science and Technology, an International Journal, https://doi.org/10.1016/j.jestch.2019.10.005

[10] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, I.H. Witten, The WEKA data mining software: an update, ACM SIGKDD Explorations 11 (1) (2009) 10–18.
[11] G. Boetticher, T. Menzies and T. J. Ostrand, The PROMISE repository of empirical software engineering data West Virginia University, Lane Department of Computer Science and Electrical Engineering (2007).
[12] H. Wang, T.M. Khoshgoftaar, N. Seliya, How many software metrics should be selected for defect prediction, Twenty-Fourth International Florida Artificial Intelligence Research Society Conference, 2011.
[13] I. Gondra, Applying machine learning to software fault-proneness prediction, J. Syst. Softw. 81 (2) (2008) 186–195.
[14] E. Arisholm, L.C. Briand, E.B. Johannessen, A systematic and comprehensive investigation of methods to build and evaluate fault prediction models, J. Syst. Softw. 83 (1) (2010) 2–17.
[15] A. Tosun, B. Turhan, 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, pp. 318–320.
[16] Y. Jiang, B. Cukic, Y. Ma, Techniques for evaluating fault prediction models, Journal Empirical Software Engineering 13 (5) (2008) 561–595.
[17] I.H. Laradji, M. Alshayeb, L. Ghouti, Software defect prediction using ensemble learning on selected features, Inf. Softw. Technol. 58 (2015) 388–402.
[18] A.S. Abdou, N.R. Darwish, Early prediction of software defect using ensemble learning: a comparative study, Int. J. Computer Appl. 179 (46) (2018) 29–40.
[19] P. Ardimento, M. Luca Bernardi, M. Cimitile, A multi-source machine learning approach to predict defect prone components, in: Proceedings of the 13th International Conference on Software Technologies, 2018, pp. 272–279.
[20] A. Kaur, I. Kaur, An empirical evaluation of classification algorithms for fault prediction in open source projects, J. King Saud University – Computer Inform. Sci 30 (1) (2018) 2–17.
[21] A. Boucher, M. Badri, Using software metrics thresholds to predict fault-prone classes in object oriented software, in: 2016 4th Intl Conf on Applied Computing and Information Technology/3rd Intl Conf on Computational Science/Intelligence and Applied Informatics/1st Intl Conf on Big Data, Cloud Computing, Data Science Engineering (ACIT-CSII-BCD), 2016, pp. 169–176.
[22] T. Menzies, J.S.D. Stefano, How good is your blind spot sampling policy, in: Proceedings of the Eighth IEEE international conference on High assurance systems engineering, IEEE Computer Society, 2004, pp. 129–138.
[23] T. Menzies, J.D. Stefano, A. Orrego, R. Chapman, Assessing predictors of software defects, Proceedings, Workshop on Predictive Software Models, 2004.
[24] M. Jureczko, L. Madeyski, Towards identifying software project clusters with regard to defect prediction, in: Proceedings of the 6th International Conference on Predictive Models in Software Engineering, 2010, pp. 1–10.
[25] A.A. Khan, A. Mahmood, S.M. Amralla, T.H. Mirza, Comparison of software complexity metrics, Int. J. Comput. Network Technol. 4 (1) (2016) 19–26.
[26] G.K. Armah, G. Luo, K. Qin, A.S. Mbandu, Applying variant variable regularized logistic regression for modeling software defect predictor, Lecture Notes Softw. Eng. 4 (2) (2016) 107–115.
[27] S.R. Chidamber, C.F. Kemerer, A Metrics Suite for Object Oriented Design, IEEE Trans. Software Eng. 20 (6) (1994) 476–493.
[28] A. Kaur, A.S. Brar, P.S. Sandhu, An empirical approach for software fault prediction, in: International Conference on Industrial and Information Systems (ICIIS), 2010, pp. 261–265.
[29] H. Zhang, X. Zhang, Comments on "Data Mining Static Code Attributes to Learn Defect Predictors", IEEE Trans. Software Eng. 33 (9) (2007) 635–637.
[30] S. Lessmann, B. Baesens, C. Mues, S. Pietsch, Benchmarking classification models for software defect prediction: a proposed framework and novel findings, IEEE Trans. Software Eng. 34 (4) (2008) 485–496.
[31] S. Shwartz, Y. Singer, N. Srebro, A. Cotter, Pegasos: primal estimated sub-GrAdient solver for SVM, Math. Program. 127 (1) (2011) 3–30.
[32] Y. Peng, G. Kou, G. Wang, H. Wang, F.I.S. Ko, Empirical evaluation of classifiers for software risk management, Int. J. Inform. Technol. Decision Making 8 (4) (2009) 749–767.
[33] Y. Freund, R.E. Schapire, Large margin classification using the perceptron algorithm, Mach. Learn. 37 (3) (1999) 277–296.
[34] M. Bal, M.F. Amasyali, H. Sever, G. Kose, A. Demirhan, Performance Evaluation of the Machine Learning Algorithms Used in Inference Mechanism of a Medical Decision Support System, Hindawi Publishing Corporation, The Scientific World Journal, 2014, pp. 1–15.
[35] D.Y. Mahmood, M.A. Hussein, Intrusion detection system based on k-star classifier and feature set reduction, IOSR J. Comput. Eng. 15 (5) (2013) 107–112.
[36] R. Raju, V. Vallikumar, An empirical comparison of classifier algorithms for anomaly detection using uci spambase dataset, Int. J. Computer Sci. Knowl. Eng. 5 (1) (2011) 7–10.
[37] W. Cohen, Fast effective rule induction, in: Proceedings of the Twelfth International Conference on Machine Learning, 1995, pp. 115–123.
[38] J. Kaur, J.S. Gurm, Optimizing the accuracy of CART algorithm by using genetic algorithm, in: International Journal of Computer Science Trends and Technology (IJCST), 2015, pp. 142–147.
[39] G. Demiroz, A. Guvenir, Classification by Voting Feature Intervals, in: Proceedings of the 9th European Conference on Machine Learning, Springer-Verlag, 1997, pp. 85–92.
[40] N. Rooney, D. Patterson, A. Tsymbal, S. Anand, Random subspacing for regression ensembles, in: Proceedings of the 7th International Florida Artificial Intelligence Research Society Conference, 2004, pp. 532–537.
[41] R. Polikar, Ensemble based systems in decision making, IEEE Circuits Syst. Mag. 6 (3) (2006) 21–45.
[42] K. Machová, F. Barčák, P. Bednár, A bagging method using decision trees in the role of base classifiers, Acta Polytechnica Hungarica 3 (2) (2006) 121–132.
[43] Y. Freund, R.E. Schapire, A decision-theoretic generalization of on-line learning and an application to boosting, J. Comput. Syst. Sci. 55 (1) (1997) 119–139.
[44] G. Brown, Ensemble Learning, Encycl. Mach. Learn. (2011) 312–330.
[45] E. Bauer, R. Kohavi, An empirical comparison of voting classification algorithms: bagging boosting, and variants, Machine Learning 36 (1–2) (1999) 105–139.
[46] D.H. Wolpert, Original contribution: stacked generalization, Neural Networks 5 (2) (1992) 241–259.
[47] K.M. Ting, I.H. Witten, Stacked generalization: when does it work?, in: Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence, 1997, pp 866–871.
[48] V. Jayaraj, N. Saravana Raman, Software defect prediction using boosting techniques, Int. J. Computer Appl. 65 (13) (2013) 1–4.
[49] G. Abaei, A. Selamat, A survey on software fault detection based on different prediction approaches, Vietnam J. Computer Sci. 1 (2) (2014) 79–95.
[50] T.K. Ho, The random subspace method for constructing decision forests, IEEE Trans. Pattern Anal. Mach. Intell. 20 (8) (1998) 832–844.
[51] K.M. Ting, I.H. Witten, Stacking bagged and dagged models, in: Proceedings of the Fourteenth International Conference on Machine Learning, 1997, pp. 367–375.
[52] J.J. Rodriguez, L.I. Kuncheva, C.J. Alonso, Rotation forest: a new classifier ensemble method, IEEE Trans. Pattern Anal. Mach. Intell. 28 (10) (2006) 1619–1630.

Please cite this article as: F. Yucalar, A. Ozcift, E. Borandag et al., Multiple-classifiers in software quality engineering: Combining predictors to improve software fault prediction ability, Engineering Science and Technology, an International Journal, https://doi.org/10.1016/j.jestch.2019.10.005