Software Defect Prediction using Tree-Based Ensembles

Hamoud Aljamaan
King Fahd University of Petroleum and Minerals
Dhahran, Saudi Arabia
hjamaan@kfupm.edu.sa

Amal Alazba
King Fahd University of Petroleum and Minerals
Dhahran, Saudi Arabia
King Saud University
Riyadh, Saudi Arabia
aalazba@ksu.edu.sa

ABSTRACT

Software defect prediction is an active research area in software engineering. Accurate prediction of software defects assists software engineers in guiding software quality assurance activities. In machine learning, ensemble learning has been proven to improve the prediction performance over individual machine learning models. Recently, many Tree-based ensembles have been proposed in the literature, and their prediction capabilities were not investigated in defect prediction. In this paper, we will empirically investigate the prediction performance of seven Tree-based ensembles in defect prediction. Two ensembles are classified as bagging ensembles: Random Forest and Extra Trees, while the other five ensembles are boosting ensembles: Ada boost, Gradient Boosting, Hist Gradient Boosting, XGBoost and CatBoost. The study utilized 11 publicly available MDP NASA software defect datasets. Empirical results indicate the superiority of Tree-based bagging ensembles: Random Forest and Extra Trees ensembles over other Tree-based boosting ensembles. However, none of the investigated Tree-based ensembles was significantly lower than individual decision trees in prediction performance. Finally, Adaboost ensemble was the worst performing ensemble among all Tree-based ensembles.

CCS CONCEPTS

• **General and reference** → **Empirical studies**; • **Computing methodologies** → **Machine learning**; • **Software and its engineering** → *Software defect analysis.*

KEYWORDS

Machine Learning, Ensemble Learning, Software Defect, Classification, Prediction, Bagging, Boosting

ACM Reference Format:
Hamoud Aljamaan and Amal Alazba. 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 '20), November 8–9, 2020, Virtual, USA.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3416508.3417114

PROMISE '20, November 8–9, 2020, Virtual, USA

ACM ISBN 978-1-4503-8127-7/20/11…$15.00
https://doi.org/10.1145/3416508.3417114

1 INTRODUCTION

Developing high quality software is essential, however, due to time and budget constraints, achieving such quality is challenging. Applying the most effective quality assurance methods to find and correct software defects is a tedious task and very expensive [28]. Therefore, instead of inspecting all parts of the system, software defect prediction can be used to identify which parts of the system are most defective, hence apply quality assurance methods on those selected parts of the system. Software defect prediction is the process of identifying software modules that are defect prone using machine learning classifiers. Software defect prediction has many advantages like reducing the cost of software testing and delivering the software product on time, consequently, increasing the overall software products quality.

Recently, different machine learning classifiers have been investigated to improve the prediction of software defects such as decision trees and ensemble learning [26]. Prediction models are usually built using different software metrics extracted from the source code. Static code metrics, Halstead and McCabe metrics, are the mostly used software metrics in the area of software defect prediction [26]. In this paper, we are utilizing 11 MDP NASA defect datasets, however, before building the prediction models, data preprocessing is crucial since most of these defect datasets suffer from two main issues: (1) high dimensionality, and (2) class imbalance. To overcome the first issue, a feature selection technique is used to reduce the number of software metrics (i.e. independent variables) in defect datasets. Given the fact that more than 50% of the defects are in 20% of the components [29], non-defective instances represent the majority class in any software defect dataset. Data sampling techniques can be employed to overcome this imbalance issue.

Ensemble learning (e.g. boosting, bagging, stacking and voting) have proven to be effective in software defect prediction, and it showed promising prediction results compared to individual classifiers. Although boosting ensemble was the mostly investigated ensemble learning in defect prediction, only two boosting algorithms (AdaBoost and XGBoost) have been studied previously. In this paper, we aim to predict software defects based on obtainable software metrics using Tree-based bagging and boosting ensembles including: Extra Trees, XGBoost, CatBoost, Gradient Boosting and Hist Gradient Boosting, which have not been widely used in this research area along with the two mainly used ensembles: Random Forests and Adaboost. A comprehensive empirical study will be performed to investigate the effectiveness of different Tree-based ensembles in software defect prediction. Furthermore, gain ratio feature selection method was used to reduce the dimensionality

of the defect datasets. Finally, imbalanced distribution of defective and non-defective modules was handled by the SMOTE technique.

Paper Organization. The remaining of this paper is organized as follows. Section 2 sheds the light over background information of Tree-based ensembles employed for predicting software defects. Section 3 reviews related work in software defect prediction using ensemble learning. A detailed description of the conducted empirical study is presented in section 4. The results analysis with the identified threats to validity are discussed in section 5. Finally, conclusion and future work are summarized in section 6.

2 TREE-BASED ENSEMBLES

Ensemble learning [35] is a computational model built using several machine learning classifiers to produce a final predictive classifier. Ensembles can learn complex data patterns for decision making and class prediction. There are two main problem categories of software defect prediction: regression where the number of defects is being predicted by the regression model and classification where the classifier predicts the class label of a module (e.g. binary classification: defective or non-defective). In a classification problem, the ensemble learning process has two stages [35]: (1) training a set of individual classifiers, called base classifiers, and (2) combining the output of the base classifiers using averaging or voting for a final prediction [47]. Ensembles are considered homogeneous if the base classifiers of a single-type algorithm whereas it is called heterogeneous if the base classifiers have different types of algorithms.

In this paper, we focus on Tree-based bagging and boosting homogeneous ensembles with decision trees [36] as a base classifier. Bagging [4] is a parallel ensemble technique that trains multiple base classifiers independently by creating several bootstrap samples for each base classifier. Then it uses averaging or voting to combine the output of all classifiers. Boosting [10] is a sequential ensemble technique that combines several classifiers to produce a model with low bias, and therefore improve the overall prediction. The main essence of boosting is to build the classifiers iteratively and sequentially where each model depends on the classifier fitted previously considering the instances that were misclassified by the previous classifiers. Many boosting algorithms have been proposed where the differences are mainly on how they learn and combine the base classifies. In this paper, we utilize a set of Tree-based ensembles for predicting software defects. The selected Tree-based ensembles are briefly described as follows:

- **Random Forests (RF)** [16] is a bagging ensemble built using several small decision trees where each decision tree is built using random subsets of the dataset. To increase the diversity among the tree, a random subset of features is selected at each node to produce the best split. The randomness of the dataset and features reduces the risk of overfitting. In a classification problem, the majority vote is used to generate a final prediction (i.e. class label).
- **Extra Trees (ET)** [12], or extremely randomized trees, is similar to RF but with further randomness. ET differs from RF in two ways (1) each decision tree is built using the whole dataset, and (2) at each node, it selects the splits randomly (i.e. not the best splits).
- **Adaboost (Ada)** [10], or Adaptive boosting, is a popular boosting algorithm proposed by Freund and Schapire. The algorithm fits a sequence of base classifiers and updates the weights of the instances by giving the misclassified instances a higher weight and then fit a new base learner with the updated weights. The final prediction is obtained from all base classifiers by combining the outputs using a weighted majority vote technique where each base classifier is contributed, given a higher weight, based on its performance.
- **Gradient Boosting (GB)** [11] is a generalization of Adaboost ensemble that can be built using a diversity of loss functions. While Ada uses the weight of misclassified instances to fit a new base classifier, GB uses gradients. Using GB can enhance the quality of fitting the base classifiers, however, it is inefficient in memory consumption and processing time.
- **Hist Gradient Boosting (HGB)** [14], or histogram-based gradient boosting, is a boosting ensemble that use features histograms for fast and accurate selection of best splits. It is more efficient than GB in terms of processing speed and memory consumption.
- **XGBoost (XGB)** [5], or extreme gradient boosting, is a GB ensemble that uses the second-order derivatives of the loss function to find the best base classifier more accurately and efficiently. Whereas GB uses gradients to fit a new base classifier, XGB utilizes the second-order gradients.
- **CatBoost (CAT)** [8], or categorical boosting, is a boosting ensemble that differs from other ensembles in two ways: (1) it can handle categorical features using one-hot encoding, and (2) it builds oblivious decision trees as base classifiers where the same features are used as a splitting criterion across all nodes within the same level of the tree. Oblivious trees are symmetric; hence, CAT reduces overfitting and decreases the training time.

3 LITERATURE REVIEW

Software defect prediction using machine learning is an active research area in software quality assurance [26]. Different types of machine learning classifiers have been explored in software defect prediction that can be categorized into three main categories: supervised learning, unsupervised learning and semi-supervised learning [25, 45]. Supervised learning is the most widely used approach in software defect prediction such as: decision trees [27], neural networks [19, 20] , Naive Bayes [15], support vector machines [43], Bayesian networks [30] and ensembles [31, 40] . Unsupervised learning [23] has been also investigated like K-means [44], expectation maximization clustering [6] and clustering ensemble [3]. Recently, ensemble learning gained more attention in the area of software defect prediction. In this section, we will review research papers that have utilized ensemble learning, which is a supervised learning approach, in the context of software defect prediction.

Ensemble learning builds prediction classifiers by combining the output of multiple machine learning classifiers to enhance the prediction accuracy [35, 47]. Ensemble learning was proven to be an effective approach in improving the performance of individual

classifiers for predicting software defects [2] and software maintainability [9]. Wide range of ensemble learning classifiers have been investigated in the literature to enhance the prediction of software defects. Ensemble learning can be classified into two categories: homogeneous [2, 42] and heterogeneous [31, 33] ensembles. Aljamaan and Elish [2] performed an empirical study to utilize the prediction capabilities of bagging and boosting ensembles in software defect prediction within the context of KC1 NASA dataset. The study compared the performance of both ensembles over six individual classifiers such as decision trees (DT), support vector machines (SVM), etc. Experimental results indicated that both ensembles performed better than individual classifiers. Yohannese et al. [42] investigated the capability of bagging and boosting ensembles in software defect prediction using eight NASA datasets. They utilized information gain (IG) as a feature selection technique to reduce the dimensionality of the dataset along with synthetic minority oversampling technique (SMOTE) as a data balancing technique. They concluded that combining ensembles with IG and SMOTE may enhance the prediction performance. Goel et al. [13] utilized Random Forests and XGBoost ensembles on four software defect datasets. They used PCA as a dimensionality reduction technique and SMOTE to balance the defect and non-defect instances. Their results showed the superiority of the ensembles (i.e RF and XGB) over individual classifiers in detecting software defects. Deep forest was explored by Zhou et al. [46] where GcForest ensemble was used with two base classifiers: Completely-Random Forests and XGBoost. They conducted the experiment on 25 software defect datasets, and concluded that GcForest outperformed individual classifiers.

Heterogeneous ensembles such as voting and stacking have also been explored in software defect prediction [31, 33] . Pandey et al. [31] evaluated a different type of ensemble learning technique which is voting to improve software defect prediction. They combined multiple base classifiers including DT, SVM, random forest, etc., and then evaluated the proposed ensemble on 12 NASA

Table 1: Comparison between prior machine learning-based approaches

Ref	Dataset	Base Classifiers	Ensembles	Feature selection	Imbalanced data	Statistical analysis
[2]	KC1 (class)	MLP, RBF, BBN, NB, SVM, DT	Bagging, AdaBoost	no	no	no
[42]	ar1, ar4, JM1', KC2, MC1", MW1', PC3', PC4"	J48	Bagging, AdaBoost.M2	IG	SMOTE	no
[13]	ivy-2.0, ant-1.7, camel 1.4, jedit-4.1, camel 1.6	DT	Random Forest, XGB	PCA	SMOTE	no
[46]	JM1, MC1, MC2, MW1, PC1, PC2, PC3, PC4, PC5, Xalan v2.6, Ant v1.7, Camel v1.6, Jedit v4.0, Log4j v1.0, Lucene v2.4, Poi v3.0, Tomcat v6.0, LC, JDT, PDE, EQ, ML, Apache, Safe, Zxing	completely-RF, XGB	GcForest	no	no	Mann–Whitney U test
[31]	CM1, KC1, KC2, KC3, MC1, MC2, PC1, JM1, MW1, PC2, PC3, PC4	IBK, MLP, SVM, RF, NB, Logistic Boost, PART, JRip, J48, Decision Stump	Voting	no	SMOTE	Wilcoxon
[33]	ant-1.5, ant-1.6, ant-1.7, jedit-4.1, jedit-4.2, tomcat. xalan-2.5, xalan-2.6	NB, DT, KNN, SMO	Stacking	no	no	Wilcoxon
[18]	Ant-1.7, Camel-1.6, e-learning, Forrest-0.8, Jedit-4.3, Tomcat, Xalan-2.7, Xerces-1.4, Zuzel, Berek, Pbean2, Velocity-1.6	NB, LR, J48, VP, SMO	AdaBoostM1, Voting, Stacking	no	no	no
[24]	CM1, JM1, KC1, KC2, PC1	J48	Adaboost, Bagging, RSM, RF, Voting	no	SMOTE	no
[40]	CM1, KC1, KC2, KC3, MC1, MC2, MW1, PC1, PC2, PC3, PC4, JM1	Bagging, RF, AdaBoost	Weighted average probabilities	no	no	Wilcoxon
[41]	KC1, KC2, KC3, PC1, PC2, PC3, PC4, MC1, MC2, CM1, JM1, MW1	Bagging, RF, AdaBoost	Weighted average probabilities	GFS	no	no
[22]	Ant-1.7, Camel-1.6, KC3, MC1, PC2, PC4	RF, GB, SGD, W-SVMs, LR, M-NB, B-NB	Average probability	GFS	no	no

datasets. They used SMOTE technique to tackle the data imbalance problem. In another experimental study conducted by Petrić et al. [33], the stacking heterogeneous ensemble was explored in software defect prediction. They combined different classifiers (naïve Bayes (NB), DT, sequential minimal optimisation (SMO) and k-nearest neighbors (KNN)) to build the stacking ensemble. They evaluated the ensemble using 8 publicly available datasets. Their experimental results revealed that stacking ensemble with diversity selection outperformed the bagging ensemble.

A comparative analysis between homogeneous and heterogeneous ensembles has been conducted [18, 24]. Hussain et al. [18] compared the performance of boosting, voting and stacking ensembles in detecting software defect using five base classifiers: NB, logistic regression (LR), J48, Voted-Perceptron (VP) and SMO. Using 12 publicly available datasets, they concluded that stacking ensemble outperformed boosting and voting ensembles. Li et al. [24] used four homogeneous ensembles: Adaboost, bagging, random subspace method (RSM) and random forest (RF) with J48 as a base classifier and a voting heterogeneous ensemble with the four homogeneous ensembles as base classifiers. They utilized five NASA datasets and used SMOTE technique to overcome the imbalanced distribution of defect and non-defect instances in selected datasets. The outcomes of the experiment revealed a superior performance of voting and RF over the other ensembles in software defect prediction.

Some studies proposed ensemble learning technique that combine multiple base classifiers using average probabilities [22, 40, 41]. Tong et al. [40] and Tran et al. [41] proposed a two-stage ensemble for software defect prediction built using three well-known ensemble learning techniques: bagging, RF, and AdaBoost. The three ensembles were combined using the weighted average probabilities. They evaluated their method using 12 NASA MDP datasets. The results of [40] showed that the proposed method significantly outperformed bagging, RF, and AdaBoost. While in [41], they utilized the greedy forward selection (GFS) as a feature selection technique and their experimental results yielded better performance in predicting software defects. GFS was also used by Laradji et al. [22] in combination with the average probability ensemble. They used seven base classifiers to build the ensemble classifier: RF, LR, weighted SVMs, gradient boosting (GB), stochastic gradient descent (SGD), multinomial NB, and Bernoulli NB. Six publicly available datasets were used to evaluate their proposed method against W-SVMs and RF. The results showed that the built ensemble outperformed the other two classifiers.

Ensembles have shown to be an effective approach in software defect prediction and it show promising prediction results over individual classifiers. Table 1 presents a summary of the explored related studies that have utilized ensembles in software defect prediction. In terms of ensembles investigation, the most widely explored ensembles are boosting, bagging and voting. Although boosting ensemble was the mostly used, only two boosting algorithms (i.e. AdaBoost and XGBoost) have been investigated. The most commonly used datasets in the area of software defect prediction are the NASA datasets. It was reported that these datasets are imbalanced and they need to be preprocessed. However, it was observed that only four studies used a data balancing technique to overcome the imbalanced class distribution of software defect datasets. Moreover, most of the existing studies did not employ a feature selection technique to reduce the number of features in defect datasets, although it was reported to be effective in enhancing the prediction performance. Lastly, low number of empirical studies used statistical analysis methods to test the significance of prediction performance differences between investigated classifiers.

In this paper, we will investigate the application of Tree-based ensembles that have not been extensively explored in the area of software defect prediction including Extra Trees, XGBoost, CatBoost, Gradient Boosting and Hist Gradient Boosting along with AdaBoost and RF. A comprehensive empirical study among different Tree-based ensembles will be performed to investigate their effectiveness in software defect prediction.

4 EMPIRICAL STUDY

In this section, we will present the conducted empirical study to investigate the prediction performance of Tree-based ensembles in predicting software defects. The empirical study was fully conducted and implemented using Python where we: trained and tested all boosting ensembles, applied feature selection using gain ratio, balanced the defect datasets with the SMOTE technique and performed statistical analysis tests. We used scikit-learn [32], CatBoost [1] and XGBoost [2] libraries to build the Tree-based ensembles.

4.1 Goal

Goal of this empirical study is defined using the GQM template as follows: ***evaluate*** Tree-based ensembles, namely Random Forest, Extra Trees, Adaboost, Gradient Boosting, Hist Gradient Boosting, XGBoost and CatBoost, for the ***purpose*** of software defect prediction with ***respect*** to their prediction performance measures (accuracy and AUC) from the ***perspective*** of both researchers and software practitioners in the ***context*** of 11 software defect datasets. To achieve our goal, we formulated the following two research questions:

- **RQ 1.** Will Tree-based ensembles improve software defect prediction performance significantly over individual decision trees classifiers?
- **RQ 2.** Which Tree-based ensemble is superior among ensembles in predicting software defects? Is there a significant performance difference between bagging and boosting ensembles?

4.2 Datasets

In this empirical study, we used 11 publicly available software defect datasets of NASA software projects [37]. The original datasets suffer from missing values, conflicting features and duplicate instances which might influence the prediction performance [38]. Therefore, we used a cleaned version of the datasets provided by Shepperd et al. [38] that are publicly accessible [3]. The description of used datasets in this paper is presented in Table 2. Each dataset contains a different number of: instances (i.e. size), independent variables and defect ratio. Each instance in the datasets represents a function, subroutine, or method. An instance is represented by: (1) one binary

[1] https://catboost.ai/docs/
[2] https://xgboost.readthedocs.io/en/latest/index.html
[3] https://figshare.com/collections/NASA_MDP_Software_Defects_Data_Sets/4054940

dependent variable indicating whether the instance is defective or non-defective, and (2) a set of Halstead and McCabe static code metrics as independent variables.

Table 2: Software defect datasets description

		No. of instances		
Dataset	**No. of Ind. variables**	Defective	Non-defective	**Total**
CM1	37	42	285	327
JM1	21	1672	6110	7782
KC1	21	314	869	1183
KC3	39	36	158	194
MC1	38	46	1942	1988
MC2	39	44	81	125
MW1	37	27	226	253
PC1	37	61	644	705
PC2	36	16	729	745
PC3	37	134	943	1077
PC4	37	177	1110	1287

We further optimized the datasets using data balancing and feature selection techniques. We used SMOTE technique to resolve the problem of imbalanced distribution of defective and non-defective instances. SMOTE is the most commonly used over-sampling technique in software defect prediction and it has proven to enhance the performance of ensemble learning [24, 42]. The main idea of SMOTE is to create new defective instances (i.e. the minority class) by interpolating several defective instances. A proper selection of relevant features (i.e. independent variables) can improve the reliability and effectiveness of the machine learning classifiers [1] and it has shown to be effective in improving the performance of software defect prediction [41, 42]. Yohannese et al. [42] performed feature selection using IG to increase machine learning classifiers prediction performance. In this paper, we used the Gain Ratio as a feature selection technique which is a normalization of IG that takes the number and size of branches into consideration [34]. The value of gain ratio ranges between 0 and 1 and we selected independent variables that have a value greater than or equal to the average gain [34]. Table 3 presents the number of independent variables before and after applying gain ratio feature selection. Nearly more than 30% of the independent variables were considered irrelevant, thus excluded from the datasets. The preprocessed datasets are available for download [4].

4.3 Model Validation

The performance of ensembles was validated using a stratified 10-fold cross-validation technique [21], where each dataset is randomly divided into ten folds of equal size, nine folds are used as a training dataset and one fold is used to evaluate the model. This procedure is repeated ten times where each fold is used exactly once as a testing dataset. Then, the final prediction estimation is calculated by averaging the prediction performance of the ten iterations. To ensure that the proportion of the defective and non-defective instances is approximately equal to the original dataset, a stratified cross-validation was used. The main advantage of this technique is that all instances are used for both model training and model testing.

[4] https://github.com/hjamaan/PROMISE20-DefectPredictionTreeEnsembles

4.4 Evaluation Measures

We selected a set of evaluation measures to evaluate the prediction performance of the boosting ensembles. These measures are derived from the four possible prediction outcomes of a binary classification problem: (1) true positives (TP) which is the number of defective instances correctly classified as defective, (2) false positives (FP) which is the number of defective instances incorrectly classified as non-defective, (3) true negatives (TN) which is the number of non-defective instances correctly classified as non-defective, and (4) false negatives (FN) which is the number of non-defective instances incorrectly classified as defective. We selected the measures that have been used in the literature to evaluate the performance of software defect prediction, and they are:

- **Accuracy**: is the percentage of correctly predicted instances to the total number of instances, and is calculated as follows:

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \times 100. \quad (1)$$

- **Area Under Curve (AUC)**: is the percentage of the area that is underneath the Receiver Operator Characteristic (ROC) curve. For illustration purposes, consider Figure 1, where the blue curve represents a classifier performance. The dotted diagonal line represents a classifier performing random guessing. The AUC metric calculates the area under the curve, where a large area indicates high performance. AUC can have a value between 0 and 1, where 1 indicates the best performance and zero is considered the worst performance.

4.5 Statistical Analysis

We investigated the significance of prediction performance differences between classifiers using the non-parametric Wilcoxon signed rank statistical test at a significance level of (0.05) [7, 17]. Data was checked for normality using the Shapiro-Wilk test, and found to be not normally distributed. Moreover, by examining the plotted boxplots shapes, it shows different skewness directions (i.e. right or left). Thus, we choose the non-parametric Wilcoxon signed rank statistical test to find the best performing classifier in each defect dataset.

Table 3: Number of independent variables before and after feature selection

	Dataset	**CM1**	**JM1**	**KC1**	**KC3**	**MC1**	**MC2**	**MW1**	**PC1**	**PC2**	**PC3**	**PC4**
No. of ind. variables	Before	37	21	21	39	38	39	37	37	36	37	37
	After	24	8	6	23	26	19	27	20	23	20	20
	Retention Ratio %	64.86	38.1	28.57	58.97	68.42	48.72	72.97	54.05	63.89	54.05	54.05

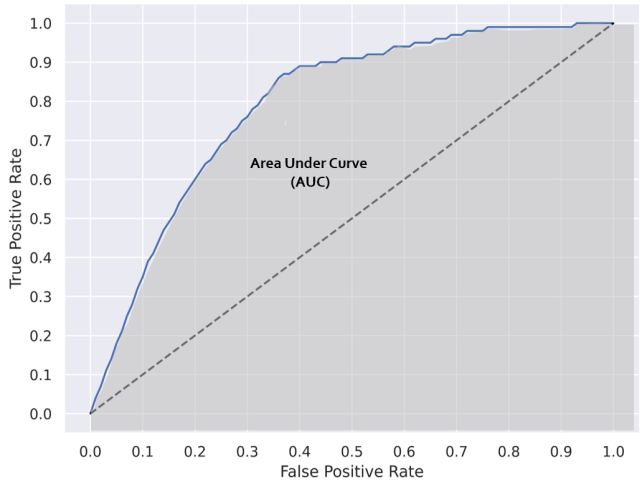

Figure 1: ROC

5 RESULTS AND DISCUSSION

In this section, we will review the prediction performance of individual decision trees classifiers in comparison to Tree-based ensembles. Then, we will compare the prediction performance between Tree-based ensembles. Table 4 presents the investigated prediction classifiers performance in terms of accuracy and AUC scores over all defect datasets. DT classifier was consistently one of worst performing classifiers across all defect datasets by scoring the lowest accuracy and AUC scores. The most noticeable performance difference was observed in the small sized dataset MC2. Ada boost ensemble showed similar prediction performance to individual DTs, with marginal differences in accuracy and AUC scores. Excluding Ada ensemble, other Tree-based ensembles showed a consistent higher prediction performance over individual DTs. Considering the prediction performance of all ensembles, RF and ET bagging ensembles competed for the best performing ensemble in each defect dataset. In the context of the largest sized defect dataset (JM1), both ensembles (RF and ET) were distinguishably superior over the other boosting ensembles with a difference of (4%) in accuracy score over other ensembles.

Next, we plotted the ensembles accuracy boxplots for each defect dataset, as shown in Figure 2, to further examine the accuracies dispersion and differences between ensembles. Boxplots shows that

Table 4: Tree-based ensembles performance across defect datasets

	CM1		KC1		KC3		JM1	
Classifier	**Accuracy**	**AUC**	**Accuracy**	**AUC**	**Accuracy**	**AUC**	**Accuracy**	**AUC**
DT	82.28	0.82	69.1	0.69	87.64	0.87	78.08	0.78
Ada	82.28	0.82	71.23	0.77	87.04	0.86	79.21	0.85
RF	**91.23**	0.97	72.55	0.79	89.87	**0.97**	**83.92**	**0.91**
ET	90.53	**0.98**	**73.07**	**0.8**	**91.76**	**0.97**	83.19	**0.91**
GB	90	0.95	69.04	0.76	89.87	0.95	78.63	0.86
HGB	89.82	0.97	70.83	0.77	89.57	0.96	79.89	0.88
XGB	88.25	0.95	67.55	0.74	89.24	0.95	78.33	0.86
CAT	88.95	0.96	67.66	0.74	89.87	0.96	79.86	0.88
	MC1		**MC2**		**MW1**		**PC1**	
Classifier	**Accuracy**	**AUC**	**Accuracy**	**AUC**	**Accuracy**	**AUC**	**Accuracy**	**AUC**
DT	97.68	0.98	54.17	0.52	86.52	0.88	88.59	0.89
Ada	98.3	0.99	57.44	0.5	86.52	0.87	88.9	0.89
RF	98.97	**1**	70.38	0.71	92.94	0.98	93.79	**0.99**
ET	**99.28**	**1**	68.91	0.7	94.02	**0.99**	**95.19**	**0.99**
GB	97.94	**1**	68.97	0.7	92.94	**0.99**	92.24	0.98
HGB	98.95	**1**	68.65	0.69	**94.48**	**0.99**	94.88	**0.99**
XGB	97.53	0.99	68.59	0.71	92.05	0.98	91.15	0.97
CAT	98.94	**1**	**72.05**	**0.72**	93.37	0.98	93.48	0.98
	PC2		**PC3**		**PC4**			
Classifier	**Accuracy**	**AUC**	**Accuracy**	**AUC**	**Accuracy**	**AUC**		
DT	96.78	0.97	85.79	0.86	91.44	0.91		
Ada	96.71	0.97	86.06	0.85	90.81	0.91		
RF	98.49	**1**	90.51	0.96	95.5	0.99		
ET	**98.7**	**1**	90.62	**0.97**	**95.77**	**1**		
GB	97.12	**1**	88.5	0.95	93.56	0.99		
HGB	97.94	**1**	**91.47**	**0.97**	95.32	0.99		
XGB	97.33	**1**	87.28	0.95	93.56	0.98		
CAT	97.87	**1**	90.67	0.96	94.46	0.99		

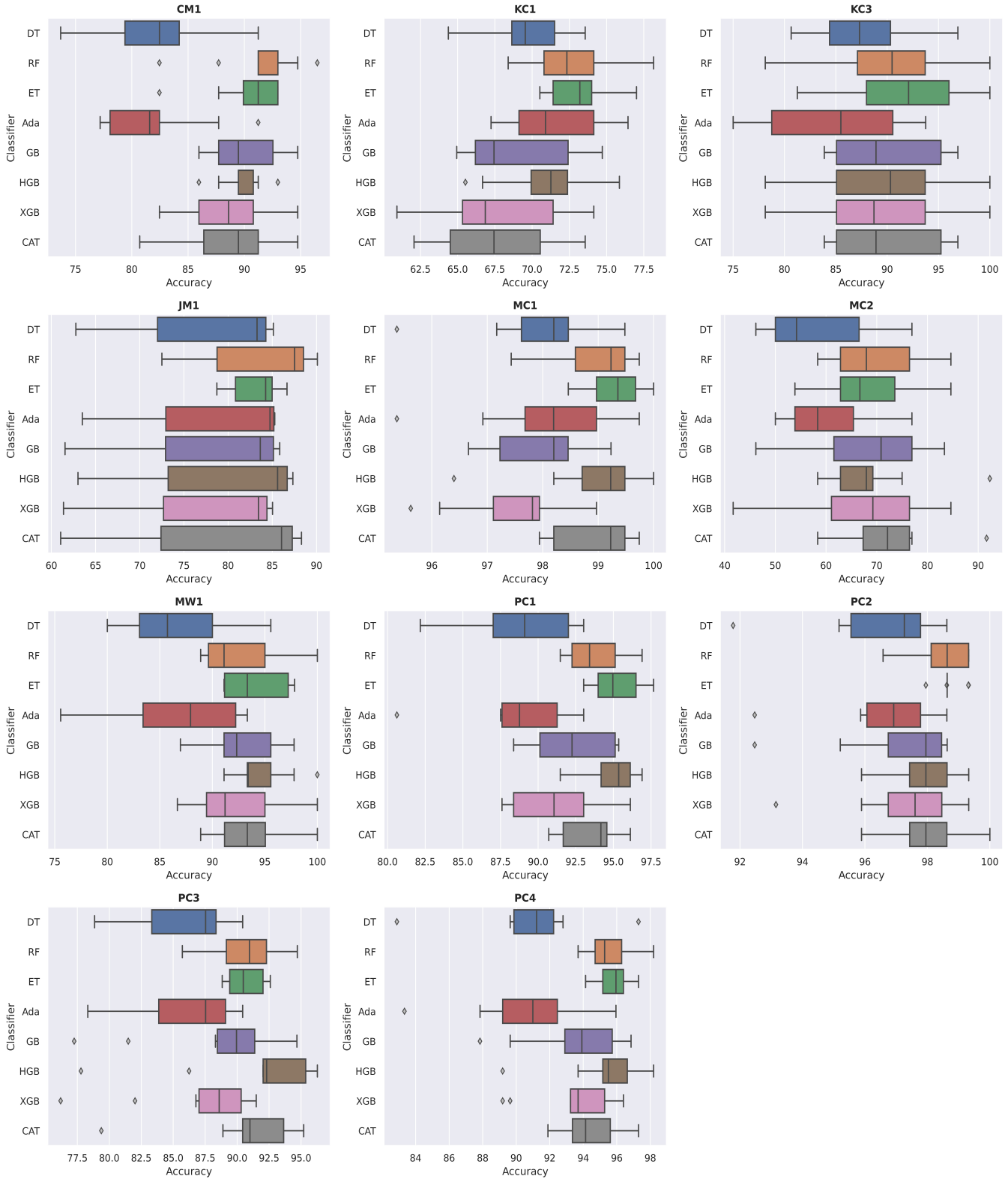

Figure 2: Tree-based ensembles accuracy boxplot

Table 5: Classifiers statistical performance outcome for each dataset

	CM1		KC1		KC3		JM1		CM1		CM2		MW1		PC1		PC2		PC3		PC4		Total	
Classifier	**W**	**L**	**W**	**L**	**W**	**L**	**W**	**L**	**W**	**L**	**W**	**L**	**W**	**L**	**W**	**L**	**W**	**L**	**W**	**L**	**W**	**L**	**W**	**L**
DT	0	6	0	3	0	3	0	4	0	5	0	4	0	6	0	5	0	5	0	5	0	6	0	52
Ada	0	6	3	2	0	3	1	2	2	4	0	5	0	6	0	5	0	5	0	5	0	6	6	49
RF	3	0	5	0	2	0	6	0	4	0	2	0	2	0	4	1	5	0	3	0	5	0	41	1
ET	2	0	6	0	2	0	3	0	5	0	2	0	2	0	6	0	5	0	3	0	5	0	41	0
GB	3	0	0	2	0	0	0	3	1	4	1	0	2	0	3	3	0	2	2	2	2	3	14	19
HGB	2	0	2	1	1	0	4	1	4	0	2	0	3	0	4	0	2	2	4	0	4	0	32	4
XGB	2	2	0	4	0	0	0	3	0	6	0	0	2	1	0	5	2	2	0	4	2	3	8	30
CAT	2	0	0	4	1	0	0	1	4	1	2	0	2	0	3	1	2	0	4	0	2	2	22	9

there is a significant difference in scored accuracies between DT classifier and Tree-based ensembles in three defect datasets (CM1, PC3 and PC4), where the DT box is lower than the other ensembles boxes. Similarly, in other defect datasets, DT boxes overlaps with the other ensembles boxes indicating a possibility of accuracy differences between classifiers. However, this observation is not applicable in the case of Ada boost ensemble, since it shows similar accuracy scores dispersion (i.e. boxes and whiskers) to DT boxplots. Thus, we observe that the Ada boost ensemble was the least performing ensemble on most defect datasets compared to other ensembles.

Tree-based ensembles prediction performance varied from a defect dataset to another. For instance, the XGB ensemble performed well in most of the defect datasets, however, in the case of KC1 and MC1 datasets, the prediction performance deteriorated to a lower level than individual DTs. In a similar situation, the CAT ensemble was placed as one of worst performing ensembles in KC1 dataset. On the contrary, RF and ET ensembles maintained a superior, or at least slightly better, performance over other ensembles in all datasets. Both ensembles were competing for the best performing ensemble and maintained smaller boxes and shorter whiskers compared relatively to the other ensembles, indicating lower dispersion in their accuracy scores. Finally, observations in the context of the largest defect dataset JM1 indicates the stable performance of the ET ensemble by having the smallest box and shortest whiskers compared all other ensembles.

We used the Wilcoxon test to test whether two classifiers accuracy scores are equal, or if the first classifier is significantly higher in accuracy scores than the second classifier, which makes the first classifier a winner and the second classifier a loser. If the test outcome shows insignificant difference, then both classifiers were not reported as a winner, nor a loser. Table 5 presents the outcome of performing the Wilcoxon statistical test to check the significance of performance differences between classifiers in each dataset. For illustration, let us consider the prediction performance of XGB ensemble in the context of the MW1 defect dataset. XGB ensemble was reported to be significantly better than two other classifiers, and significantly lower than a single classifier. Analyzing the overall statistical comparison results, we observe that it aligns well with our previous analysis indicating the superiority of RF and ET ensembles over other ensembles. Both were ranked as the first ensembles with 41 wins, however, RF ensemble lost against ET ensemble in the context of PC1 dataset. Following them, the HGB ensemble came third with 32 wins. On the contrary, Ada boost ensemble was the lowest ranked ensemble among all ensembles with 49 losses. Lastly, individual DTs wasn't found significantly better than any other ensemble in predicting software defects.

We further expanded the previous statistical test results to present classifiers paired comparisons as shown in Table 6. Each row represents the number of wins for row classifier against the column classifier, and vice versa, the same number represents the column classifier losses against the row classifier. For example, ET ensemble won against XGB ensemble 7 times and similarly XGB ensemble lost against ET ensemble 7 times. Last column calculates the percentage of each classifier total wins out of all possible paired comparisons, where each classifier will have a total of 77 paired comparisons

Table 6: Classifiers paired comparisons outcomes

Classifier	DT	Ada	RF	ET	GB	HGB	XGB	CAT	**Win**	**Wins %**
DT	-								**0**	**0%**
Ada	3	-					2	1	**6**	**8%**
RF	11	11	-		6	2	8	3	**41**	**53%**
ET	11	10	1	-	6	2	7	4	**41**	**53%**
GB	5	6			-		3		**14**	**18%**
HGB	9	10			5	-	7	1	**32**	**42%**
XGB	4	4					-		**8**	**10%**
CAT	9	8			2		3	-	**22**	**29%**
Loss	**52**	**49**	**1**	**0**	**19**	**4**	**30**	**9**		
Loss %	**68%**	**64%**	**1%**	**0%**	**25%**	**5%**	**39%**	**12%**		

(11 datasets multiplied by 7 other competing classifiers). Similarly, the last row calculates the losses percentage. Note that a paired comparison may not result as a win nor a loss, if the statistical test outcome stated insignificant difference in competing classifiers performance.

Paired comparisons outcome emphasizes the superiority of ET and RF ensembles over other ensembles, however, the ET ensemble performed significantly better than the competing RF ensemble in one dataset. Moreover, not a single ensemble performed better than the ET ensemble in all utilized defect datasets. Hence, the losses percentage for the ET ensemble was zero. HGB ensemble also performed well in defect prediction with a low percentage of losses (5%), and all HGB ensemble 4 losses were against the best ensembles (i.e. ET and RF). Ada boost ensemble was the worst performing ensemble with the highest percentage of losses, however, it performed significantly better than individual DTs in the context of three defect datasets.

RQ1 Answer: Tree-based ensembles prediction performance was significantly better, or at least competitive, compared to individual DTs in defect prediction. Specifically, bagging ensembles (RF and ET) showed significantly superior prediction performance over individual DTs in all 11 defect datasets.

RQ2 Answer: Bagging ensembles (RF and ET) performed significantly better, or at least competitive, compared to boosting ensembles in defect prediction. In addition, Ada boost ensemble was the least performing ensemble among all Tree-based ensembles.

5.1 Threats to Validity

Similar to other empirical studies utilizing public datasets, an external threat to validity is associated with the defect datasets selection. Our results are limited to utilized NASA defect datasets and can't be generalized to all software systems. Therefore, replication of this empirical study using other publicly available or new defect datasets is needed. A construct threat to validity is related to the metrics used as independent variables for defect prediction. NASA defect datasets uses a set of software metrics as predictors for software defects. We further refined the selection of software metrics using gain ratio feature selection. Resulting in different metrics selection for each defect dataset. In our study, we used the default ensembles hyperparameters values set by the utilized libraries. Previous research [39] showed the impact of machine learning parameters optimization on the overall performance, however, we leave this as a possible future work.

6 CONCLUSION

This paper empirically investigated the prediction capabilities of Tree-based ensembles in software defect prediction. A comprehensive study was conducted to investigate the prediction performance of individual decision trees and 7 Tree-based ensembles within the context of 11 publicly available defect datasets. Gain ratio technique was applied for feature selection, and SMOTE technique was used to preprocess the imbalanced defect data. The main contributions of this paper are as follows: First, we applied and compared the performance of Tree-based ensembles against individual decision trees and examined to which extent ensembles offer an increase in prediction performance over individual DTs. Second, we compared the prediction performance between ensembles in defect prediction.

Study outcomes present promising empirical results for both software engineers and machine learning researchers of the applicability of Tree-based ensembles in software defect prediction. Ensemble prediction capabilities can be utilized to accurately predict defects and assist in the overall software quality assurance activities. Empirical results show that all ensembles offer an increase, or at least competitive, prediction performance compared to individual DTs. Between ensembles, RT and ET ensembles outperformed other competing ensembles in most defect datasets. However, ET ensemble performed significantly better than the RF ensemble in a single defect dataset. Moreover, in the context of the largest defect dataset, the prediction performance of the ET ensemble was uniquely stable compared to the other ensembles. In the lower side, Ada boost ensemble was the least performing ensemble in defect prediction, nevertheless, the Ada boost ensemble performed better than individual DTs in several defect datasets.

For future directions, the work in this paper can be extended in these directions: the empirical study can be replicated with new defect datasets of different sizes, and compare the new results with the reported results in this paper. In addition, different feature selection techniques may be applied to examine their effect on boosting ensembles prediction performance. Finally, prediction performance of heterogeneous ensembles (e.g. voting ensemble) can be compared against the Tree-based ensemble.

ACKNOWLEDGMENTS

The authors would like to acknowledge the support of King Fahd University of Petroleum and Minerals (KFUPM) in the development of this work.

REFERENCES

[1] Ghadah Aldehim and Wenjia Wang. 2017. Determining appropriate approaches for using data in feature selection. *International Journal of Machine Learning and Cybernetics* 8, 3 (June 2017), 915–928. https://doi.org/10.1007/s13042-015-0469-8

[2] Hamoud I. Aljamaan and Mahmoud O. Elish. 2009. An empirical study of bagging and boosting ensembles for identifying faulty classes in object-oriented software. In *2009 IEEE Symposium on Computational Intelligence and Data Mining*. 187–194. https://doi.org/10.1109/CIDM.2009.4938648 ISSN: null.

[3] Hamad Alsawalqah, Neveen Hijazi, Mohammed Eshtay, Hossam Faris, Ahmed Al Radaideh, Ibrahim Aljarah, and Yazan Alshamaileh. 2020. Software Defect Prediction Using Heterogeneous Ensemble Classification Based on Segmented Patterns. *Applied Sciences* 10, 5 (Jan. 2020), 1745. https://doi.org/10.3390/app10051745 Number: 5 Publisher: Multidisciplinary Digital Publishing Institute.

[4] Leo Breiman. 1996. Bagging predictors. *Machine Learning* 24, 2 (Aug. 1996), 123–140. https://doi.org/10.1007/BF00058655

[5] Tianqi Chen and Carlos Guestrin. 2016. XGBoost: A Scalable Tree Boosting System. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16)*. Association for Computing Machinery, San Francisco, California, USA, 785–794. https://doi.org/10.1145/2939672.2939785

[6] Rodrigo A. Coelho, Fabrício dos R.N. Guimarães, and Ahmed A.A. Esmin. 2014. Applying Swarm Ensemble Clustering Technique for Fault Prediction Using Software Metrics. In *2014 13th International Conference on Machine Learning and Applications*. 356–361. https://doi.org/10.1109/ICMLA.2014.63

[7] Janez Demšar. 2006. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine learning research* 7, Jan (2006), 1–30.

[8] Anna Veronika Dorogush, Vasily Ershov, and Andrey Gulin. 2018. CatBoost: gradient boosting with categorical features support. *ArXiv* (2018).

[9] Mahmoud O. Elish, Hamoud Aljamaan, and Irfan Ahmad. 2015. Three empirical studies on predicting software maintainability using ensemble methods. *Soft Computing* 19, 9 (Sept. 2015), 2511–2524. https://doi.org/10.1007/s00500-014-1576-2

[10] Y. Freund. 1995. Boosting a Weak Learning Algorithm by Majority. *Information and Computation* 121, 2 (Sept. 1995), 256–285. https://doi.org/10.1006/inco.1995.1136

[11] Jerome H. Friedman. 2001. Greedy Function Approximation: A Gradient Boosting Machine. *The Annals of Statistics* 29, 5 (2001), 1189–1232. https://www.jstor.org/stable/2699986 Publisher: Institute of Mathematical Statistics.

[12] Pierre Geurts, Damien Ernst, and Louis Wehenkel. 2006. Extremely randomized trees. *Machine Learning* 63, 1 (April 2006), 3–42. https://doi.org/10.1007/s10994-006-6226-1

[13] Lipika Goel, Mayank Sharma, Sunil Kumar Khatri, and D. Damodaran. 2020. Defect Prediction of Cross Projects Using PCA and Ensemble Learning Approach. In *Micro-Electronics and Telecommunication Engineering*, Devendra Kumar Sharma, Valentina Emilia Balas, Le Hoang Son, Rohit Sharma, and Korhan Cengiz (Eds.). Springer Singapore, Singapore, 307–315.

[14] Aleksei Guryanov. 2019. Histogram-Based Algorithm for Building Gradient Boosting Ensembles of Piecewise Linear Decision Trees. In *Analysis of Images, Social Networks and Texts (Lecture Notes in Computer Science)*, Wil M. P. van der Aalst, Vladimir Batagelj, Dmitry I. Ignatov, Michael Khachay, Valentina Kuskova, Andrey Kutuzov, Sergei O. Kuznetsov, Irina A. Lomazova, Natalia Loukachevitch, Amedeo Napoli, Panos M. Pardalos, Marcello Pelillo, Andrey V. Savchenko, and Elena Tutubalina (Eds.). Springer International Publishing, Cham, 39–50. https://doi.org/10.1007/978-3-030-37334-4_4

[15] Peng He, Bing Li, Xiao Liu, Jun Chen, and Yutao Ma. 2015. An empirical study on software defect prediction with a simplified metric set. *Information and Software Technology* 59 (March 2015), 170–190. https://doi.org/10.1016/j.infsof.2014.11.006

[16] Tin Kam Ho. 1995. Random decision forests. In *Proceedings of 3rd International Conference on Document Analysis and Recognition*, Vol. 1. 278–282 vol.1. https://doi.org/10.1109/ICDAR.1995.598994

[17] Myles Hollander, Douglas A. Wolfe, and Eric Chicken. 2000. *Nonparametric statistical methods* (3 rd ed.). Wiley, New York.

[18] Shahid Hussain, Jacky Keung, Arif Ali Khan, and Kwabena Ebo Bennin. 2015. Performance Evaluation of Ensemble Methods For Software Fault Prediction: An Experiment. In *Proceedings of the ASWEC 2015 24th Australasian Software Engineering Conference (ASWEC ' 15 Vol. II)*. Association for Computing Machinery, Adelaide, SA, Australia, 91–95. https://doi.org/10.1145/2811681.2811699

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

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

[21] Ron Kohavi. 1995. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Proceedings of the 14th international joint conference on Artificial intelligence - Volume 2 (IJCAI'95)*. Morgan Kaufmann Publishers Inc., Montreal, Quebec, Canada, 1137–1143.

[22] Issam H. Laradji, Mohammad Alshayeb, and Lahouari Ghouti. 2015. Software defect prediction using ensemble learning on selected features. *Information and Software Technology* 58 (Feb. 2015), 388–402. https://doi.org/10.1016/j.infsof.2014.07.005

[23] Ning Li, Martin Shepperd, and Yuchen Guo. 2020. A systematic review of unsupervised learning techniques for software defect prediction. *Information and Software Technology* 122 (June 2020), 106287. https://doi.org/10.1016/j.infsof.2020.106287

[24] Ran Li, Lijuan Zhou, Shudong Zhang, Hui Liu, Xiangyang Huang, and Zhong Sun. 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, Seoul, Republic of Korea, 1–6. https://doi.org/10.1145/3352411.3352412

[25] Huihua Lu, Bojan Cukic, and Mark Culp. 2012. Software defect prediction using semi-supervised learning with dimension reduction. In *2012 Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering*. 314–317. https://doi.org/10.1145/2351676.2351734

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

[27] Tim Menzies, Jeremy Greenwald, and Art Frank. 2007. Data Mining Static Code Attributes to Learn Defect Predictors. *IEEE Transactions on Software Engineering* 33, 1 (Jan. 2007), 2–13. https://doi.org/10.1109/TSE.2007.256941

[28] Tim Menzies, Zach Milton, Burak Turhan, Bojan Cukic, Yue Jiang, and Ayşe Bener. 2010. Defect prediction from static code features: current results, limitations, new approaches. *Automated Software Engineering* 17, 4 (Dec. 2010), 375–407. https://doi.org/10.1007/s10515-010-0069-5

[29] Niclas Ohlsson, Ann Christin Eriksson, and Mary Helander. 1997. Early Risk-Management by Identification of Fault-prone Modules. *Empirical Software Engineering* 2, 2 (Feb. 1997), 166–173. https://doi.org/10.1023/A:1009757419320

[30] Ahmet Okutan and Olcay Taner Yıldız. 2014. Software defect prediction using Bayesian networks. *Empirical Software Engineering* 19, 1 (Feb. 2014), 154–181. https://doi.org/10.1007/s10664-012-9218-8

[31] Sushant Kumar Pandey, Ravi Bhushan Mishra, and Anil Kumar Tripathi. 2020. BPDET: An effective software bug prediction model using deep representation and ensemble learning techniques. *Expert Systems with Applications* 144 (April 2020), 113085. https://doi.org/10.1016/j.eswa.2019.113085

[32] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake Vanderplas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Édouard Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12, 85 (2011), 2825–2830. http://jmlr.org/papers/v12/pedregosa11a.html

[33] Jean Petrić, David Bowes, Tracy Hall, Bruce Christianson, and Nathan Baddoo. 2016. Building an Ensemble for Software Defect Prediction Based on Diversity Selection. In *Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '16)*. Association for Computing Machinery, Ciudad Real, Spain, 1–10. https://doi.org/10.1145/2961111.2962610

[34] J. R. Quinlan. 1986. Induction of decision trees. *Machine Learning* 1, 1 (March 1986), 81–106. https://doi.org/10.1007/BF00116251

[35] Lior Rokach. 2010. Ensemble-based classifiers. *Artificial Intelligence Review* 33, 1 (Feb. 2010), 1–39. https://doi.org/10.1007/s10462-009-9124-7

[36] S.R. Safavian and D. Landgrebe. 1991. A survey of decision tree classifier methodology. *IEEE Transactions on Systems, Man, and Cybernetics* 21, 3 (June 1991), 660–674. https://doi.org/10.1109/21.97458

[37] Sayyad Shirabad, J. and Menzies, T.J. 2005. The PROMISE Repository of Software Engineering Databases. http://promise.site.uottawa.ca/SERepository/

[38] Martin Shepperd, Qinbao Song, Zhongbin Sun, and Carolyn Mair. 2013. Data Quality: Some Comments on the NASA Software Defect Datasets. *IEEE Transactions on Software Engineering* 39, 9 (Sept. 2013), 1208–1215. https://doi.org/10.1109/TSE.2013.11

[39] Chakkrit Tantithamthavorn, Shane McIntosh, Ahmed E Hassan, and Kenichi Matsumoto. 2016. Automated parameter optimization of classification techniques for defect prediction models. In *Proceedings of the 38th International Conference on Software Engineering*. 321–332.

[40] Haonan Tong, Bin Liu, and Shihai Wang. 2018. Software defect prediction using stacked denoising autoencoders and two-stage ensemble learning. *Information and Software Technology* 96 (April 2018), 94–111. https://doi.org/10.1016/j.infsof.2017.11.008

[41] Hung Duy Tran, LE Thi My Hanh, and Nguyen Thanh Binh. 2019. Combining feature selection, feature learning and ensemble learning for software fault prediction. In *2019 11th International Conference on Knowledge and Systems Engineering (KSE)*. 1–8. https://doi.org/10.1109/KSE.2019.8919292 ISSN: 2164-2508.

[42] Chubato Wondaferaw Yohannese, Tianrui Li, Macmillan Simfukwe, and Faisal Khurshid. 2017. Ensembles based combined learning for improved software fault prediction: A comparative study. In *2017 12th International Conference on Intelligent Systems and Knowledge Engineering (ISKE)*. 1–6. https://doi.org/10.1109/ISKE.2017.8258836

[43] Xiao Yu, Jin Liu, Jacky Wai Keung, Qing Li, Kwabena Ebo Bennin, Zhou Xu, Junping Wang, and Xiaohui Cui. 2020. Improving Ranking-Oriented Defect Prediction Using a Cost-Sensitive Ranking SVM. *IEEE Transactions on Reliability* 69, 1 (March 2020), 139–153. https://doi.org/10.1109/TR.2019.2931559

[44] Feng Zhang, Quan Zheng, Ying Zou, and Ahmed E. Hassan. 2016. Cross-Project Defect Prediction Using a Connectivity-Based Unsupervised Classifier. In *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*. 309–320. https://doi.org/10.1145/2884781.2884839 ISSN: 1558-1225.

[45] Zhi-Wu Zhang, Xiao-Yuan Jing, and Tie-Jian Wang. 2017. Label propagation based semi-supervised learning for software defect prediction. *Automated Software Engineering* 24, 1 (March 2017), 47–69. https://doi.org/10.1007/s10515-016-0194-x

[46] Tianchi Zhou, Xiaobing Sun, Xin Xia, Bin Li, and Xiang Chen. 2019. Improving defect prediction with deep forest. *Information and Software Technology* 114 (2019), 204–216.

[47] Zhi-Hua Zhou. 2012. *Ensemble Methods: Foundations and Algorithms* (1st ed.). Chapman & Hall/CRC.