Software Defect Prediction Using Supervised Machine Learning and Ensemble Techniques: A Comparative Study

Article *in* Journal of Software Engineering and Applications · January 2019

DOI: 10.4236/jsea.2019.125007

Abdullah Alsaeedi
Taibah University
35 PUBLICATIONS 384 CITATIONS
SEE PROFILE

Mohammad Zubair Khan
Sarhad University of Science & IT
21 PUBLICATIONS 264 CITATIONS
SEE PROFILE

Scientific Research Publishing

Journal of Software Engineering and Applications, 2019, 12, 85-100
http://www.scirp.org/journal/jsea
ISSN Online: 1945-3124
ISSN Print: 1945-3116

Software Defect Prediction Using Supervised Machine Learning and Ensemble Techniques: A Comparative Study

Abdullah Alsaeedi, Mohammad Zubair Khan

Department of Computer Science, College of Computer Science and Engineering, Taibah University, Madinah, KSA
Email: aasaeedi@taibahu.edu.sa, zubair.762001@gmail.com

How to cite this paper: Alsaeedi, A. and Khan, M.Z. (2019) Software Defect Prediction Using Supervised Machine Learning and Ensemble Techniques: A Comparative Study. *Journal of Software Engineering and Applications*, **12**, 85-100.
https://doi.org/10.4236/jsea.2019.125007

Received: April 10, 2019
Accepted: May 18, 2019
Published: May 21, 2019

Abstract

An essential objective of software development is to locate and fix defects ahead of schedule that could be expected under diverse circumstances. Many software development activities are performed by individuals, which may lead to different software bugs over the development to occur, causing disappointments in the not-so-distant future. Thus, the prediction of software defects in the first stages has become a primary interest in the field of software engineering. Various software defect prediction (SDP) approaches that rely on software metrics have been proposed in the last two decades. Bagging, support vector machines (SVM), decision tree (DS), and random forest (RF) classifiers are known to perform well to predict defects. This paper studies and compares these supervised machine learning and ensemble classifiers on 10 NASA datasets. The experimental results showed that, in the majority of cases, RF was the best performing classifier compared to the others.

Keywords

Machine Learning, Ensembles, Prediction, Software Metrics, Software Defect

1. Introduction

A software defect is a bug, fault, or error in a program that causes improper outcomes. Software defects are programming errors that may occur because of errors in the source code, requirements, or design. Defects negatively affect software quality and software reliability [1]. Hence, they increase maintenance costs and efforts to resolve them. Software development teams can detect bugs by analyzing software testing results, but it is costly and time-consuming by testing entire software modules. As such, identifying defective modules in early stages is necessary to aid software testers in detecting modules that required intensive testing [2] [3].

In the field of software engineering, software defect prediction (SDP) in early stages is vital for software reliability and quality [1] [4]. The intention of SDP is to predict defects before software products are released, as detecting bugs after release is an exhausting and time-consuming process. In addition, SDP approaches have been demonstrated to improve software quality, as they help developers predict the most likely defective modules [5] [6]. SDP is considered a significant challenge, so various machine learning algorithms have been used to predict and determine defective modules [7]. With the end goal of expanding the viability of software testing, SDP is utilized to distinguish defective modules in current and subsequent versions of a software product. Therefore, SDP approaches are very helpful in allocating more efforts and resources for testing and examining likely-defective modules [8].

Commonly-used SDP strategies are regression and classification strategies. The objective of regression techniques is to predict the number of software defects [5]. In the literature, there are a number of regression models used for SDP [9] [10] [11] [12]. In contrast, classification approaches aim to decide whether a software module is faulty or not. Classification models can be trained from the defect data of the previous version of the same software. The trained models can then be used to predict further potential software defects. Mining software repository becomes a vital topic in research for predicting defects [13] [14].

Supervised machine learning classifiers are commonly employed to predict software defects such as support vector machines [15] [16] [17], k-nearest neighbors (KNN) [18] [19], Naive Bayes [19] [20] [21], and so on. In addition, Bowes [22] suggested the use of classifier ensembles to effectively predict defects. A number of works have been accomplished in the field of SDP utilizing ensemble methods such as bagging [23] [24] [25], voting [22] [26], boosting [23] [24] [25], random tree [22], RF [27] [28], and stacking [22]. Neural networks (NN) can be used to predict defect prone software modules [29] [30] [31] [32].

Clustering algorithms such as k-means, x-means, and expectation maximization (EM) have also been applied to predict defects [33] [34] [35]. In addition, the experiment outcomes in [34] [35] showed that x-means clustering performed better than fuzzy clustering, EM, and k-means clustering at identifying software defects. Aside from those, transfer learning is a machine learning approach that expects to exchange the information learned in one dataset and utilize that learning to help tackle issues in an alternate dataset [36]. Transfer learning has also been introduced to the field of SDP [37] [38].

Software engineering data, such as defect prediction datasets, are very imbalanced, where the number of samples of a specific class is vastly higher than another class. To deal with such data, imbalanced learning approaches have been proposed in SDP to mitigate the data imbalance problem [7]. Imbalanced learning approaches include re-sampling, cost-sensitive learning, ensemble learning, and imbalanced ensemble learning (hybrid approaches) [7] [39]. Re-sampling approaches can be either oversampling and under-sampling methods, and these can add or remove instances from the training data only. Several previous studies [40] [41] [42] in SDP utilized oversampling approaches, especially, Synthetic Minority Over-sampling Technique (SMOTE). Pelayo and Dick [40] combined SMOTE with DS to study the effect of oversampling on the accuracy of predictive models at detecting software defects. The results showed that SMOTE led to improvements in the classification accuracy, especially when the percentage of resampling was 300%.

Cost-sensitive learning is another approach to dealing with data imbalance. It works by adding weight to samples or resampling them by allocating cost to each class in a predefined matrix. However, the issue with cost-sensitive classifiers is that there is no intelligent and systematic way to set the cost matrices [39]. Ensemble approaches combine multiple models to obtain better predictions. Three ensemble methods are widely used in SDP includes: bagging, boosting, and stacking. The common boosting algorithm for SDP is adaptive boosting (AdaBoost). On the other hand, hybrid approaches [39], such as SMOTEBoost and RUSBoost are known approaches for dealing with the problem of class imbalance. In this paper, we will use the SMOTE oversampling approach to deal with class imbalance, with the intent to compare the performance of supervised machine learning and Ensemble techniques in predicting software defects.

Section 2 summarizes software metrics that can be used as attributes to identify software defects. Section 3 presents evaluation metrics that can be used to measure the performance of SDP models. Sections 4 and 5 detail the experimental methodology and results, respectively. Section 6 presents the threats to validity. Related works are described in Section 7.

2. Software Metrics

A software metric is a proportion of quantifiable or countable characteristics that can be used to measure and predict the quality of software. A metric is an indicator describing a specific feature of a software [6]. Identifying and measuring software metrics is vital for various reasons, including estimating programming execution, measuring the effectiveness of software processes, estimating required efforts for processes, deduction of defects during software development, and monitoring and controlling software project executions [5].

Various software metrics have been commonly used for defect prediction. The first group of software metrics is called lines of code (LOC) metrics and is considered basic software metrics. LOC metrics are typical proportions of software development. Many studies in SDP have proven a clear correlation between LOC metrics and defect prediction [43] [44]. One of the most common software metrics widely used for SDP are the cyclomatic complexity metrics, which were proposed by McCabe [45] and are used to represent the complexity of software products. McCabe's metrics (cyclomatic metrics) are computed based on the control flow graphs of a source code by counting the number of nodes, arcs, and connected components. Ohlsson and Alberg [46] used McCabe's cyclomatic metrics to predict defect-prone modules before starting coding. Many previous studies have used McCabe's cyclomatic metrics to build SDP models [47] [48], as well. Another set of software metrics are the software size metrics proposed by Halstead [49]. Halstead software size metrics are based on the number of operands and operators from source codes [49]. In addition, these metrics are related to program size of program vocabulary, length, volume, difficulty, effort, and time [49] and have been used in SDP [48] [50].

According to [51], the majority of software fault prediction approaches rely on object-oriented software metrics. Chidamber and Kemerer [52] proposed several software metrics called CK object-oriented metrics, which include the depth of inheritance tree (DIT), weighted method per class (WMC), number of children (NOC), and so on. Many studies using object-oriented metrics have been used in SDP [53] [54] [55]. Radjenovic *et al.* [51] identified effective software metrics used in SDP and aimed to enhance software quality by finding defects. The outcome of their study [51] outlined that object-oriented and process metrics were more effective in finding defects compared to other size and complexity metrics.

3. Evaluation Measures for Software Bugs Prediction

In this section, we will discuss different measurements for software defect prediction such as true positive (TP), true negative (TN), false positive (FP) and false negative (FN). TP denotes the number of defective software instances that are correctly classified as defective, while TN is the number of clean software instances that are correctly classified as clean. FP denotes the number of clean software instances that are wrongly classified as defective, and FN denotes the number of defective software instances that are mistakenly classified as clean

One of the primary simple metrics to evaluate the performance of predictive models is classification accuracy, also called the correct classification rate. It is utilized to quantify the extent of the effectively classified instances to the aggregate instances. Another measure is called precision, and it is calculated by dividing the number of instances correctly classified as defective (TP) by the total number of instances classified as defective (TP + FP) [16]. In addition, recall measures the percentage of the number of instances correctly classified as defective (TP) to the total number of faulty instances (TP + FN) [16]. F-score is a harmonic mean of precision and recall, and many studies in the literature used F-score metrics [56] [57]. ROC-AUC calculates the area under the receiver operating characteristic (ROC) curve by computing trade-offs between TPR and FPR.

$$\text{Accuracy} = (\text{TP}+\text{TN})/(\text{TP}+\text{TN}+\text{FP}+\text{FN}) \tag{1}$$

$$\text{Precision} = \text{TP}/(\text{TP}+\text{FP}) \tag{2}$$

$$\text{Recall} = \text{TP}/(\text{TP}+\text{FN}) \tag{3}$$

$$\text{F-Score} = \left(2 * \text{Precision} * \text{Recall}\right) / \left(\text{Precision} + \text{Recall}\right) \quad (4)$$

$$\text{TPR} = \text{TP} / \left(\text{TP} + \text{FN}\right) \quad (5)$$

$$\text{FPR} = \text{FP} / \left(\text{TN} + \text{FP}\right) \quad (6)$$

G-measure is another measure used in software defect prediction. It is defined as a harmonic mean of recall and specificity. Probability of false alarm (PF) is the ratio of clean instances wrongly classified as defective (FP) among the total clean instances (FP + TN).

$$\text{PF} = \text{FP} / \left(\text{FP} + \text{TN}\right) \quad (7)$$

$$\text{Specificity} = 1 - \text{PF} = \text{TN} / \text{FP} + \text{TN} \quad (8)$$

$$\text{G-measure} = 2 * \text{Recall} * \text{Specificity} / \text{Recall} + \text{Specificity} \quad (9)$$

4. Experimental Methodology

For the experiments, 10 well-known software defect datasets [62] were selected. The majority of related works used these datasets to evaluate the performance of their SDP techniques and this is the reason behind selecting the above-mentioned dataset for further comparisons. Table 1 reports the datasets used in the experiments along with the statistics. RF, DS, Linear SVC SVM, and LR were chosen to be the base classifiers. Boosting and bagging classifiers for all the base classifiers were also considered. The experiments were conducted on a Python environment. The classifiers' performances in this study were measured using classification accuracy, precision, recall, F-score, and ROC-AUC score. It is important to highlight that these metrics were computed using the weighted average. The intuition behind selecting the weighted average was to calculate metrics for each class label and take the label imbalance into the account.

Table 1. Dataset summaries.

Dataset	**Number of Modules**	Number of Defective Modules	Number of Attributes (Software Metrics)
JM1	7782	1672	21
KC3	194	36	39
MC1	1988	46	38
MC2	125	44	39
MW1	253	27	37
PC1	705	61	37
PC2	745	16	36
PC3	1077	134	37
PC4	1287	177	37
PC5	1711	471	38

The performance of classifiers was evaluated based on 10-fold cross-validation to split the datasets into 10 consecutive folds. One of them for testing and the remaining folds for training. Afterwards, features were standardized and scaled using the standard Scaler function in Python, which works by removing the mean and scaling the features into unit variance. Since the datasets were very imbalanced, the oversampling approach using SMOTE was performed for the training data only, as it has been widely used in the literature to mitigate imbalance issues in training data for SDP.

The following Algorithm 1 was used for the experiments. It began by providing a list of datasets and a list of classifiers and then proceeded to iterate over all datasets, as shown in Line 8. The datasets were split into training and testing data based on 10-fold cross-validation with shuffling of the data before splitting, as shown in Line 9. One the dataset was split, the perform Standard Scaler function was utilized to standardize and scale the features.

Once the features were standardized, the training data for each fold were re-sampled using the SMOTE technique, as shown in Line 11. As mentioned above, SMOTE oversampling has been widely used in SDP. The loop in Lines 12 - 25 aimed to train the classifiers, obtain predictions, and compute evaluation metrics. The average metrics were computed in Lines (27 - 31) as the datasets were split using 10-folds. The process from Lines 9 - 31 was iterated throughout all provided datasets.

```
   Input : Datasets, Classifiers
   Result: AvgAccurecy, AvgRecall, AvgPrecision, AvgF-score, and AvgAUC
 1 Datasets ← {PC1, PC3, PC4, PC5, JM1, KC2, KC3, MC1, MC2, CM1};
 2 Classifiers ← {RF, DS, SVM, LR, AdaBoost-RF, AdaBoost-DS, AdaBoost-SVM,
     AdaBoost-LR, Bagging-RF, Bagging-DS, Bagging-SVM, Bagging-LR};
 3 AllAccuracyScores ← {};
 4 AllRecallScores ← {};
 5 AllPrecisionScores ← {};
 6 AllFScores ← {};
 7 AllAUCScores ← {};
 8 for DS ∈ Datasets do
 9     for Xtrain, Xtest ∈ KFold (nsplits = 10, shuffle = True).split(DS) do
10         (Xtrain, Xtest) ← PerformStandardScaler(Xtrain, Xtest);
11         ResampledXtrain, ResampledYtrain ← SMOTE(Xtrain, Ytrain);
12         for clf ∈ Classifiers do
13             clf ← TrainClassifier(clf, ResampledXtrain, XtrainLabels);
14             predictions ← predict(cls, Xtest);
15             Accurecy ← ComputeAccurecy(predictions, XtestLabels);
16             Recall ← ComputeRecall(predictions, XtestLabels);
17             Precision ← ComputePrecision(predictions, XtestLabels);
18             F-score ← ComputeFmeasure(predictions, XtestLabels);
19             AUC ← ComputeAUC(predictions, XtestLabels);
20             AllAccuracyScores ← AllAccuracyScores ∪ Accurecy;
21             AllRecallScores ← AllRecallScores ∪ Recall;
22             AllPrecisionScores ← AllPrecisionScores ∪ Precision;
23             AllFScores ← AllFscoreScores ∪ F-score;
24             AllAUCScores ← AllAUCScores ∪ AUC;
25         end
26     end
27     AvgAccurecy ← ComputeAvgAccurecy(AllAccuracyScores);
28     AvgRecall ← ComputeAvgRecall(AllRecallScores);
29     AvgPrecision ← ComputeAvgPrecision(AllPrecisionScores);
30     AvgF-score ← ComputeAvgFmeasure(AllFScores);
31     AvgAUC ← ComputeAvgAUC(AllAUCScores);
32 end
33 return AvgAccurecy,AvgRecall,AvgPrecision,AvgF-score,AvgAUC
```

Algorithm 1. The experimental procedure for software defect perdition.

5. Experimental Results and Discussion

Table 2 summarizes the performance of the different classifiers based on the classification accuracy. The RF classifier achieved accuracies of 0.91, 0.84, 0.90, 0.82, 0.97, and 0.83 for the PC1, PC3, PC4, KC2, MC1, and CM1 datasets, respectively. In addition, it is obvious that the RF classifier attained the highest accuracy scores for the PC1, PC3, PC4, KC2, MC1, and CM1 datasets compared to other classifiers, indicating better predictions of defective instances performed by the RF classifiers in these datasets. Moreover, the reported scores in Table 2 show that the bagging classifier with DS as a base learner performed well on the PC5, KC3, and MC2 datasets as compared to the other classifiers.

In Figure 1, it is clear that the RF classifier obtained the highest accuracy scores for all datasets, except PC5, JM1, KC3 and MC2. Furthermore, the maximum accuracy attained for PC1 was 0.91 whereas the minimum value was 0.78

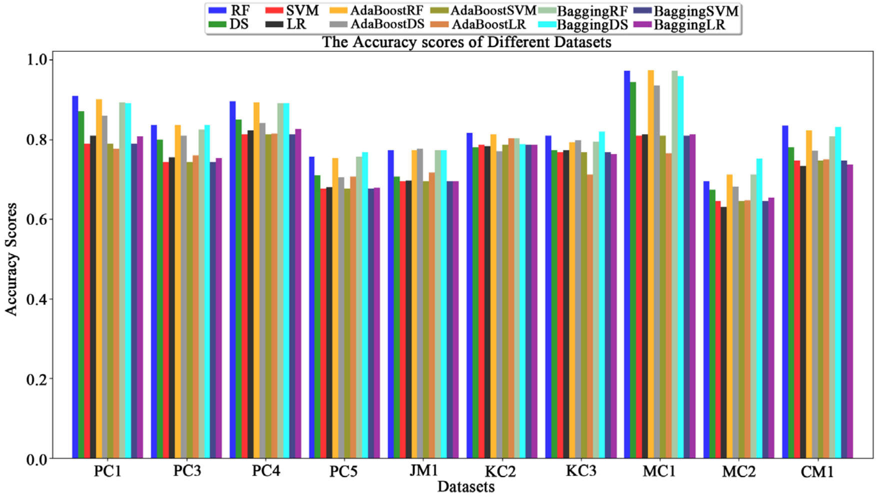

Figure 1. Classification accuracy scores of different classifiers.

Table 2. The accuracy scores obtained using different classifiers.

Dataset	Base Learner				AdaBoost				Bagging			
	RF	DS	SVM	LR	RF	DS	SVM	LR	RF	DS	SVM	LR
PC1	**0.91**	0.87	0.79	0.81	0.90	0.86	0.79	0.78	0.89	0.89	0.79	0.81
PC3	**0.84**	0.80	0.74	0.76	**0.84**	0.81	0.74	0.76	0.82	**0.84**	0.74	0.75
PC4	**0.90**	0.85	0.81	0.82	0.89	0.84	0.81	0.82	0.89	0.89	0.81	0.83
PC5	0.76	0.71	0.68	0.68	0.75	0.70	0.68	0.71	0.76	**0.77**	0.68	0.68
JM1	0.77	0.71	0.69	0.70	0.77	**0.78**	0.69	0.72	0.77	0.77	0.69	0.69
KC2	**0.82**	0.78	0.79	0.78	0.81	0.77	0.79	0.80	0.80	0.79	0.79	0.79
KC3	0.81	0.77	0.77	0.77	0.79	0.80	0.77	0.71	0.79	**0.82**	0.77	0.76
MC1	**0.97**	0.94	0.81	0.81	**0.97**	0.94	0.81	0.77	**0.97**	0.96	0.81	0.81
MC2	0.69	0.67	0.65	0.63	0.71	0.68	0.65	0.65	0.71	**0.75**	0.65	0.65
CM1	**0.83**	0.78	0.75	0.73	0.82	0.77	0.75	0.75	0.81	**0.83**	0.75	0.74

obtained by LR for the same dataset. Among the base learners, RF was the best performing classifier for all datasets, while SVM was the worst classifier for all datasets, except KC2, MC2, and CM1. Besides, Bagging with DS achieves higher accuracy scores for PC3, PC5, KC3, MC2, CM1 compared to the other bagging and boosting methods.

Table 3 reports the F-scores attained using different classifiers. In general, it is apparent that the RF classifier was the best performing for six different datasets, as illustrated in Table 2 and Table 3. For PC1, PC3, PC4, KC2, MC1, and CM1, the RF classifier attained the highest F-scores compared to the other classifiers, indicating better predictions obtained by RF. In addition, the reported F-scores presented that AdaBoost classifier with RF as a base learner attained similar scores to RF for the PC3, PC4, KC2, and MC1 datasets. Furthermore, bagging with DS achieved higher F-scores compared to other classifiers for PC5, KC, and MC2.

Figure 2 illustrates bar plots of the F-scores attained using classifiers for all datasets. For the PC3, PC4, PC5, and JM1 datasets, it is obvious that the SVM,

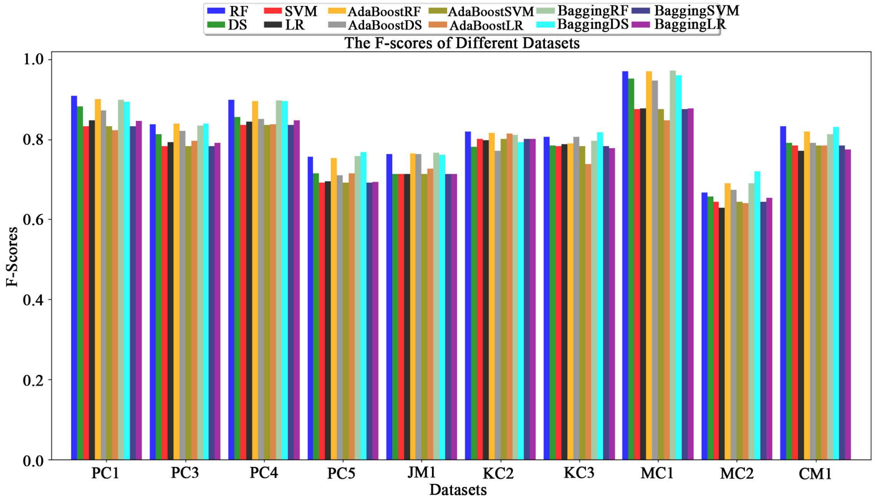

Figure 2. F-scores attained by different classifiers.

Table 3. The F-scores obtained using different classifier.

Dataset	Base Learner				AdaBoost				Bagging			
	RF	DS	SVM	LR	RF	DS	SVM	LR	RF	DS	SVM	LR
PC1	**0.91**	0.88	0.83	0.85	0.90	0.87	0.83	0.82	0.90	0.89	0.83	0.85
PC3	**0.84**	0.81	0.78	0.79	**0.84**	0.82	0.78	0.80	0.83	**0.84**	0.78	0.79
PC4	**0.90**	0.86	0.84	0.84	**0.90**	0.85	0.84	0.84	**0.90**	**0.90**	0.84	0.85
PC5	0.76	0.72	0.69	0.70	0.75	0.71	0.69	0.71	0.76	**0.77**	0.69	0.69
JM1	0.76	0.71	0.71	0.71	0.76	0.76	0.71	0.73	**0.77**	0.76	0.71	0.71
KC2	**0.82**	0.78	0.80	0.80	**0.82**	0.77	0.80	0.81	0.81	0.79	0.80	0.80
KC3	0.81	0.78	0.78	0.79	0.79	0.81	0.78	0.74	0.80	**0.82**	0.78	0.78
MC1	**0.97**	0.95	0.88	0.88	**0.97**	0.95	0.88	0.85	**0.97**	0.96	0.88	0.88
MC2	0.67	0.66	0.64	0.63	0.69	0.67	0.64	0.64	0.69	**0.72**	0.64	0.65
CM1	**0.83**	0.79	0.78	0.77	0.82	0.79	0.78	0.79	0.81	**0.83**	0.78	0.77

AdaBoost (SVM), and bagging (SVM) classifiers performed badly, as reported in Table 3. For JM1, the highest F-score was 0.77, attained by bagging (RF). Additionally, the lowest score was 0.71, which was attained using six different classifiers. Furthermore, the F-scores achieved by bagging (LR) were the minimum for KC3 and MC1. LR was the worst classifier for the MC2 and CM1 datasets.

The ROC-AUC scores achieved by all participating classifiers are shown in Table 4. For PC1 and PC3, the LR and bagging with LR classifiers attained the highest ROC-AUC scores, achieving 0.77 for PC1 and 0.74 for PC3. The bagging with the RF algorithm as base estimator performed well in terms of ROC-AUC scores, reaching 0.84, 0.71, and 0.64 for the PC4, PC5, and JM1 datasets, respectively. The ROC-AUC score of AdaBoost with the LR classifier on data set KC2 was the best among all the classifiers, achieving a score of 0.78, while the lowest value was 0.66 and was attained by the AdaBoost with DS. The SVM, AdaBoost (SVM), and bagging (SVM) classifiers achieved the highest ROC-AUC scores for the CM1 and MC1 datasets.

Figure 3 shows the bar plots of ROC-AUC scores attained by all classifiers. It is clear there is no dominant classifier and this may due to the nature of datasets. For instance, LR and bagging (LR) classifiers performed well on PC1 and PC3 datasets, while these classifiers did not achieve the highest ROC-AUC scores for other datasets.

Our findings demonstrate that there was uncertainty in the classifiers' performances, as some classifiers performed well in specific datasets but worse in others. Similar to other studies [6] [22] [58], our results recommend using ensembles as predictive models to detect software defects. Additionally, their findings [6] [22] [58] agreed with our outcome that RF performed well. However, the experiments conducted by Hammouri *et al.* [61] purported that the best performing algorithm was DS, while our study's findings confirmed that DS performed badly, unless it was used as a base learner with bagging classifiers for some datasets, as reported in Table 2 and Table 3.

Table 4. The ROC-AUC scores obtained using classifiers.

Dataset	Base Learner				AdaBoost				Bagging			
	RF	DS	SVM	LR	RF	DS	SVM	LR	RF	DS	SVM	LR
PC1	0.72	0.70	0.76	**0.77**	0.68	0.66	0.76	0.74	0.73	0.66	0.76	**0.77**
PC3	0.64	0.61	0.73	**0.74**	0.64	0.64	0.73	0.73	0.67	0.65	0.73	**0.74**
PC4	0.81	0.73	0.82	0.83	0.79	0.73	0.82	0.80	**0.84**	0.81	0.82	0.83
PC5	0.70	0.66	0.68	0.68	0.69	0.65	0.68	0.67	**0.71**	**0.71**	0.68	0.68
JM1	0.62	0.59	0.63	0.63	0.63	0.62	0.63	0.63	**0.64**	0.62	0.63	0.63
KC2	0.74	0.67	0.77	0.77	0.74	0.66	0.77	**0.78**	0.75	0.71	0.77	0.77
KC3	0.69	0.69	0.66	0.67	0.65	**0.71**	0.66	0.63	0.67	**0.71**	0.66	0.66
MC1	0.66	0.58	**0.75**	0.71	0.65	0.57	**0.75**	0.70	0.67	0.61	**0.75**	0.71
MC2	0.61	0.59	0.59	0.58	0.63	0.61	0.59	0.61	0.63	**0.66**	0.59	0.61
CM1	0.58	0.56	**0.68**	0.63	0.56	0.57	**0.68**	**0.68**	0.57	0.59	**0.68**	0.65

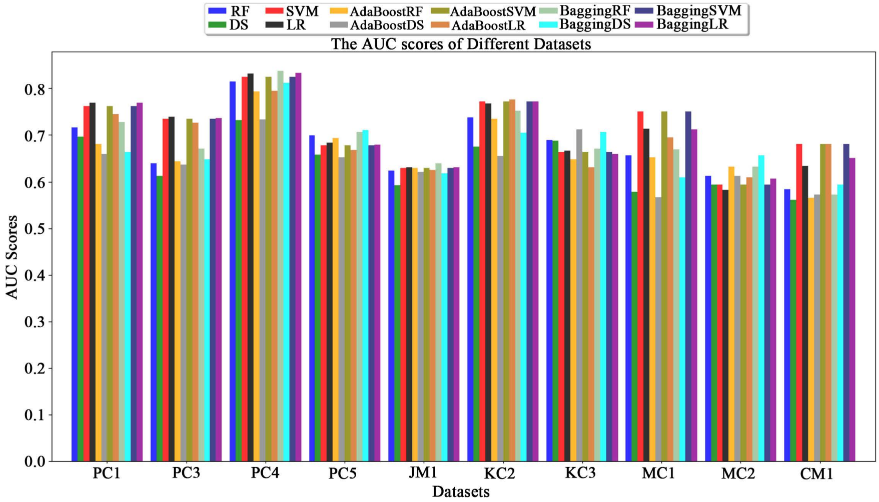

Figure 3. ROC-AUC scores of different classifiers.

6. Threats to Validity

In this section, we list some potential threats in our study and responses to construct validity.

1) The selection of datasets may not be representative. One potential threat to validity is the selection of datasets where they might not be representative. In our study, this threat is mitigated by evaluating the performance of the classifiers on ten well-known datasets that are commonly used in the literature review.

2) The generalization of our results. We have attempted to mitigate this threat by measuring the performance of the base learners, boosting, and bagging classifier on diverse datasets that have different sizes.

3) The trained classifiers may over-fitting and bias the results. Instead of splitting the datasets randomly using the simple train-test split (70% - 80% for training and 30% - 20% for testing), we split the dataset into training and testing sets using the 10-fold cross validation to avoid the over-fitting issue that might be caused using the random splitting.

7. Related Works

Kalai Magal *et al.* [28] combined feature selection with RF to improve the accuracy of software defect predication. Feature selection was based on correlation computation and aimed to choose the ideal subset of features. The selected features using correlation-based feature selection were then used with RF to predict software defects. Various experiments were conducted on open NASA datasets from the PROMISE repository. The outcome showed clear improvements obtained using the improved RF compared to the traditional RF.

Venkata *et al.* [9] explored various machine learning algorithms for real-time system defect identification. They investigated the impact of attribute reduction on the performance of SDP models and attempted to combine PCA with different classification models which did not show any improvements. However, the outcomes of the experimental results demonstrated that combining the correlation-based feature selection technique with 1-rule classifier led to improvements in classification accuracy.

Anuradha and Shafali [58] investigated three supervised classifiers: J48, NB, and RF. Various datasets were selected to assess the classifiers efficiency at detecting defective modules. The conducted experiments demonstrated that the RF classifier outperformed the others. Moreover, Ge *et al.* [6] showed that RF performed well compared to LWL, C4.5, SVM, NB, and multilayer feed forward neural networks. On the other hand, Singh and Chug [59] analyzed five classifiers—ANN, particle swarm optimization (PSO), DS, NB, Linear classifier (LC)—and compared their performance in terms of detecting software defects. The experiment results showed that LC outperformed the other classifiers.

Aleem *et al.* [27] compared the performance of 11 machine learning methods and used 15 NASA datasets from the PROMISE repository. NB, MLP, SVM, AdaBoost, bagging, DS, RF, J48, KNN, RBF, and k-means were applied in their study. The results showed that bagging and SVM performed well in the majority of datasets. Meanwhile, Wang *et al.* [22] carried out a comparative analysis of ensemble classifiers for SDP and demonstrated that voting ensemble and RF attained the highest classification accuracy results compared to AdaBoost, NB, stacking, and bagging. Perreault *et al.* [19] compared NB, SVM, ANN, LR, and KNN on five NASA datasets. The outcomes of the conducted experiments did not show a superior classifier at identifying software defects. Hussain *et al.* [60] used the AdaboostM1, Vote and StackingC ensemble classifier with five base classifiers: NB, LR, J48, Voted-Perceptron and SMO in Weka tool for SDP. The experimental results showed that StackingC performed well compared to the other classifiers.

Hammouri *et al.* [61] assessed NB, ANN, and DS for SDP. Three real debugging datasets were used in their study. Measurements such as accuracy, precision, recall, F-measure, and RMSE were utilized to analyze the results. The results of their study showed that DS performed well.

The above-mentioned approaches differ from the proposed approach in this paper in two ways. Firstly, we compared the performance of different supervised and Ensemble methods on the oversampled training data, while other works such as Kalai Magal *et al.* [28] and Venkata *et al.* [9] focused on the impact of feature selection and attribute reduction on the performance of classifiers. Secondly, a very similar study to our approach presented in this paper was conducted by Alsawalqah *et al.* [63], where they studied the impact of SMOTE on the Adaboost ensemble method with J48 as a base classifier. Their findings demonstrated that SMOTE can help to boost the performance of the ensemble method on four NASA datasets. This differs from our study presented in this paper is that we compared varieties of ensemble methods on the oversampled training dataset, while Alsawalqah *et al.* [63] used only Adaboost with J48 as a base classifier.

The general finding in these related works is that classifiers such as RF, bagging, DS, Adaboost performed well in the SDP problem. Therefore, we have focused on studying the performance of these classifiers on the condition that SMOTE oversampling techniques were applied to training data only.

8. Conclusions and Future Works

This paper focused on comparing the most well-known machine learning algorithms that are widely used to predict software defects. The performances of different algorithms were evaluated using classification accuracy, F-measure, and ROC-AUC metrics. The SMOTE resampling strategy was used to mitigate the data imbalance issues. The outcomes of the conducted experiment showed that RF, AdaBoost with RF, and bagging with DS generally performed well.

Interesting future extensions could include studying the impact of various metaheuristic feature selection approaches to select the optimal set of features for SDP. One future direction is to explore and compare the performance of deep learning approaches and ensemble classifiers with other resampling techniques, as data imbalance is still an issue that badly affects the performance of the existing SDP approaches.

Conflicts of Interest

The authors declare no conflicts of interest regarding the publication of this paper.

References

[1] Rawat, M.S. and Dubey, S.K. (2012) Software Defect Prediction Models for Quality Improvement: A Literature Study. *International Journal of Computer Science Issues*, **9**, 288-296.

[2] Li, J., He, P., Zhu, J. and Lyu, M.R. (2017) Software Defect Prediction via Convolutional Neural Network. 2017 *IEEE International Conference on Software Quality, Reliability and Security*, 25-29 July 2017, Prague, 318-328. https://doi.org/10.1109/QRS.2017.42

[3] Hassan, F., Farhan, S., Fahiem, M.A. and Tauseef, H. (2018) A Review on Machine Learning Techniques for Software Defect Prediction. *Technical Journal*, **23**, 63-71.

[4] Punitha, K. and Chitra, S. (2013) Software Defect Prediction Using Software Metrics: A Survey. 2013 *International Conference on Information Communication and Embedded Systems*, 21-22 February 2013, Chennai, 555-558. https://doi.org/10.1109/ICICES.2013.6508369

[5] Kalaivani, N. and Beena, R. (2018) Overview of Software Defect Prediction Using Machine Learning Algorithms. *International Journal of Pure and Applied Mathematics*, **118**, 3863-3873.

[6] Ge, J., Liu, J. and Liu, W. (2018) Comparative Study on Defect Prediction Algorithms of Supervised Learning Software Based on Imbalanced Classification Data Sets. 2018 19*th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing*, 27-29 June 2018, Busan, 399-406. https://doi.org/10.1109/SNPD.2018.8441143

[7] Song, Q., Guo, Y. and Shepperd, M. (2018) A Comprehensive Investigation of the Role of Imbalanced Learning for Software Defect Prediction. *IEEE Transactions on Software Engineering*, 1. https://doi.org/10.1109/TSE.2018.2836442

[8] Chang, R.H., Mu, X.D. and Zhang, L. (2011) Software Defect Prediction Using Non-Negative Matrix Factorization. *Journal of Software*, **6**, 2114-2120. https://doi.org/10.4304/jsw.6.11.2114-2120

[9] Challagulla, V.U.B., Bastani, F.B., Yen, I.L. and Paul, R.A. (2005) Empirical Assessment of Machine Learning Based Software Defect Prediction Techniques. *Proceedings of the* 10*th IEEE International Workshop on Object-Oriented Real-Time Dependable Systems*, 2-4 February 2005, Sedona, 263-270. https://doi.org/10.1109/WORDS.2005.32

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

[11] Rathore, S.S. and Kumar, S. (2016) A Decision Tree Regression Based Approach for the Number of Software Faults Prediction. *ACM SIGSOFT Softw Are Engineering Notes*, **41**, 1-6. https://doi.org/10.1145/2853073.2853083

[12] Rathore, S.S. and Kumar, S. (2017) An Empirical Study of Some Software Fault Prediction Techniques for the Number of Faults Prediction. *Soft Computing*, **21**, 7417-7434. https://doi.org/10.1007/s00500-016-2284-x

[13] Wang, H. (2014) Software Defects Classification Prediction Based on Mining Software Repository. Master's Thesis, Uppsala University, Department of Information Technology.

[14] Vandecruys, O., Martens, D., Baesens, B., Mues, C., Backer, M.D. and Haesen, R. (2008) Mining Software Repositories for Comprehensible Software Fault Prediction Models. *Journal of Systems and Software*, **81**, 823-839. https://doi.org/10.1016/j.jss.2007.07.034 http://www.sciencedirect.com/science/article/pii/S0164121207001902

[15] Vapnik, V. (2013) The Nature of Statistical Learning Theory. Springer, Berlin.

[16] Elish, K.O. and Elish, M.O. (2008) Predicting Defect-Prone Software Modules Using Support Vector Machines. *Journal of Systems and Software*, **81**, 649-660. https://doi.org/10.1016/j.jss.2007.07.040 http://www.sciencedirect.com/science/article/pii/S016412120700235X

[17] Gray, D., Bowes, D., Davey, N., Sun, Y. and Christianson, B. (2009) Using the Support Vector Machine as a Classification Method for Software Defect Prediction with Static Code Metrics. In: Palmer-Brown, D., Draganova, C., Pimenidis, E. and Mouratidis, H., Eds., *Engineering Applications of Neural Networks*, Springer, Berlin, 223-234. https://doi.org/10.1007/978-3-642-03969-0_21

[18] Wang, H., Khoshgoftaar, T.M. and Seliya, N. (2011) How Many Software Metrics Should Be Selected for Defect Prediction? 24*th International FLAIRS Conference*, 18-20 May 2011, Palm Beach, 69-74.

[19] Perreault, L., Berardinelli, S., Izurieta, C. and Sheppard, J. (2017) Using Classifiers for Software Defect Detection. 26*th International Conference on Software Engineering and Data Engineering*, 2-4 October 2017, Sydney, 2-4.

[20] Wang, T. and Li, W. (2010) Naive Bayes Software Defect Prediction Model. 2010 *International Conference on Computational Intelligence and Software Engineering*, 10-12 December 2010, Wuhan, 1-4. https://doi.org/10.1109/CISE.2010.5677057

[21] Jiang, Y., Cukic, B. and Menzies, T. (2007) Fault Prediction Using Early Lifecycle Data. 18*th IEEE International Symposium on Software Reliability*, 5-9 November 2007, Trollhättan, 237-246. https://doi.org/10.1109/ISSRE.2007.24

[22] Wang, Tao, Li, W., Shi, H. and Liu, Z. (2011) Software Defect Prediction Based on

Classifiers Ensemble. *Journal of Information & Computational Science*, **8**, 4241-4254.

[23] Jiang, Y., Cukic, B. and Menzies, T. (2008) Cost Curve Evaluation of Fault Prediction Models. 2008 19*th International Symposium on Software Reliability Engineering*, 10-14 November 2008, Seattle, 197-206. https://doi.org/10.1109/ISSRE.2008.54

[24] Jiang, Y., Lin, J., Cukic, B. and Menzies, T. (2009) Variance Analysis in Software Fault Prediction Models. 2009 20*th International Symposium on Software Reliability Engineering*, 16-19 November 2009, San Jose, 99-108. https://doi.org/10.1109/ISSRE.2009.13

[25] Abdou, A. and Darwish, N. (2018) Early Prediction of Software Defect Using Ensemble Learning: A Comparative Study. *International Journal of Computer Applications*, **179**, 29-40. https://doi.org/10.5120/ijca2018917185

[26] Moustafa, S., El Nainay, M., El Makky, N. and Abougabal, M.S. (2018) Software Bug Prediction Using Weighted Majority Voting Techniques. *Alexandria Engineering Journal*, **57**, 2763-2774. https://doi.org/10.1016/j.aej.2018.01.003 http://www.sciencedirect.com/science/article/pii/S1110016818300747

[27] Aleem, S., Capretz, L. and Ahmed, F. (2015) Benchmarking Machine Learning Technologies for Software Defect Detection. *International Journal of Software Engineering & Applications*, **6**, 11-23. https://doi.org/10.5121/ijsea.2015.6302

[28] Jacob, S.G., *et al*. (2015) Improved Random Forest Algorithm for Software Defect Prediction through Data Mining Techniques. *International Journal of Computer Applications*, **117**, 18-22. https://doi.org/10.5120/20693-3582

[29] Kumar, R. and Gupta, D.L. (2016) Software Bug Prediction System Using Neural Network. *European Journal of Advances in Engineering and Technology*, 3, 78-84.

[30] Jindal, R., Malhotra, R. and Jain, A. (2014) Software Defect Prediction Using Neural Networks. *Proceedings of* 3*rd International Conference on Reliability*, *Infocom Technologies and Optimization*, 8-10 October 2014, Noida, 1-6. https://doi.org/10.1109/ICRITO.2014.7014673

[31] Sethi, T. (2016) Improved Approach for Software Defect Prediction Using Artificial Neural Networks. 2016 5*th International Conference on Reliability*, *Infocom Technologies and Optimization*, 7-9 September 2016, Noida, 480-485. https://doi.org/10.1109/ICRITO.2016.7785003

[32] Jayanthi, R. and Florence, L. (2018) Software Defect Prediction Techniques Using Metrics Based on Neural Network Classifier. *Cluster Computing*, 1-12. https://doi.org/10.1007/s10586-018-1730-1

[33] Bishnu, P.S. and Bhattacherjee, V. (2012) Software Fault Prediction Using Quad Tree-Based K-Means Clustering Algorithm. *IEEE Transactions on Knowledge and Data Engineering*, **24**, 1146-1150. https://doi.org/10.1109/TKDE.2011.163

[34] Park, M. and Hong, E. (2014) Software Fault Prediction Model Using Clustering Algorithms Determining the Number of Clusters Automatically. *International Journal of Software Engineering and Its Applications*, **8**, 199-204.

[35] Catal, C., Sevim, U. and Diri, B. (2009) Software Fault Prediction of Unlabeled Program Modules. *Proceedings of the World Congress on Engineering*, **1**, 1-3. https://doi.org/10.1109/ITNG.2009.12

[36] Han, J., Pei, J. and Kamber, M. (2011) *Data Mining*: *Concepts and Techniques*. The Morgan Kaufmann Series in Data Management Systems. https://books.google.com.sa/books?id=pQws07tdpjoC

[37] Ma, Y., Luo, G., Zeng, X. and Chen, A. (2012) Transfer Learning for Cross-Company

Software Defect Prediction. *Information and Software Technology*, **54**, 248-256. http://www.sciencedirect.com/science/article/pii/S0950584911001996 https://doi.org/10.1016/j.infsof.2011.09.007

[38] Cao, Q., Sun, Q., Cao, Q. and Tan, H. (2015) Software Defect Prediction via Transfer Learning Based Neural Network. 2015 1*st International Conference on Reliability Systems Engineering*, 21-23 October 2015, Beijing, 1-10. https://doi.org/10.1109/ICRSE.2015.7366475

[39] Rodriguez, D., Herraiz, I., Harrison, R., Dolado, J. and Riquelme, J.C. (2014) Preliminary Comparison of Techniques for Dealing with Imbalance in Software Defect Prediction. *Proceedings of the* 18*th International Conference on Evaluation and Assessment in Software Engineering*, 13-14 May 2014, London, 1-10. https://doi.org/10.1145/2601248.2601294

[40] Pelayo, L. and Dick, S. (2007) Applying Novel Resampling Strategies to Software Defect Prediction. *NAFIPS* 2007 *Annual Meeting of the North American Fuzzy Information Processing Society*, 24-27 June 2007, San Diego, 69-72. https://doi.org/10.1109/NAFIPS.2007.383813

[41] Pak, C., Wang, T. and Su, X.H. (2018) An Empirical Study on Software Defect Prediction Using Over-Sampling by Smote. *International Journal of Software Engineering and Knowledge Engineering*, **28**, 811-830. https://doi.org/10.1142/S0218194018500237

[42] Shatnawi, R. (2012) Improving Software Fault-Prediction for Imbalanced Data. 2012 *International Conference on Innovations in Information Technology*, 18-20 March 2012, London, 54-59. https://doi.org/10.1109/INNOVATIONS.2012.6207774

[43] Zhang, H. (2009) An Investigation of the Relationships between Lines of Code and Defects. 2009 *IEEE International Conference on Software Maintenance*, 20-26 September 2009, Edmonton, 274-283. https://doi.org/10.1109/ICSM.2009.5306304

[44] Mende, T. and Koschke, R. (2009) Revisiting the Evaluation of Defect Prediction Models. *Proceedings of the* 5*th International Conference on Predictor Models in Software Engineering*, 18-19 May 2009, Canada, 1-10. https://doi.org/10.1145/1540438.1540448

[45] McCabe, T.J. (1976) A Complexity Measure. *IEEE Transactions on Software Engineering*, **2**, 308-320. https://doi.org/10.1109/TSE.1976.233837

[46] Ohlsson, N. and Alberg, H. (1996) Predicting Fault-Prone Software Modules in Telephone Switches. *IEEE Transactions on Software Engineering*, **22**, 886-894. https://doi.org/10.1109/32.553637

[47] Lessmann, S., Baesens, B., Mues, C. and Pietsch, S. (2008) Benchmarking Classification Models for Software Defect Prediction: A Proposed Framework and Novel Findings. *IEEE Transactions on Software Engineering*, **34**, 485-496. https://doi.org/10.1109/TSE.2008.35

[48] Song, Q., Jia, Z., Shepperd, M., Ying, S. and Liu, J. (2011) A General Software Defect-Proneness Prediction Framework. *IEEE Transactions on Software Engineering*, **37**, 356-370. https://doi.org/10.1109/TSE.2010.90

[49] Halstead, M.H. (1977) Elements of Software Science (Operating and Programming Systems Series).

[50] Menzies, T., Greenwald, J. and Frank, A. (2007) Data Mining Static Code Attributes to Learn Defect Predictors. *IEEE Transactions on Software Engineering*, **33**, 2-13. https://doi.org/10.1109/TSE.2007.256941

[51] Radjenovi, D., Heriko, M., Torkar, R. and Radjenovi, A. (2013) Software Fault Prediction Metrics: A Systematic Literature Review. *Information and Software Tech-*

nology, **55**, 1397-1418. https://doi.org/10.1016/j.infsof.2013.02.009 http://www.sciencedirect.com/science/article/pii/S0950584913000426

[52] Chidamber, S.R. and Kemerer, C.F. (1994) A Metrics Suite for Object Oriented Design. *IEEE Transactions on Software Engineering*, **20**, 476-493. https://doi.org/10.1109/32.295895

[53] Jureczko, M. and Spinellis, D.D. (2010) Using Object-Oriented Design Metrics to Predict Software Defects.

[54] Gupta, D.L. and Saxena, K. (2017) Software Bug Prediction Using Object-Oriented Metrics. *Sadhana*, **42**, 655-669.

[55] Singh, A., Bhatia, R. and Singhrova, A. (2018) Taxonomy of Machine Learning Algorithms in Software Fault Prediction Using Object Oriented Metrics. *Procedia Computer Science*, **132**, 993-1001. http://www.sciencedirect.com/science/article/pii/S1877050918308470

[56] Kim, S., Zhang, H., Wu, R. and Gong, L. (2011) Dealing with Noise in Defect Prediction. 2011 33*rd International Conference on Software Engineering*, 21-28 May 2011, Waikiki, 481-490. https://doi.org/10.1145/1985793.1985859

[57] Lee, T., Nam, J., Han, D., Kim, S. and In, H. (2011) Micro Interaction Metrics for Defect Prediction. *Proceedings of the* 19*th ACM SIGSOFT Symposium and the* 13*th European Conference on Foundations of Software Engineering*, 5-9 September 2011, Szeged, 311-321. https://doi.org/10.1145/2025113.2025156

[58] Chug, A. and Dhall, S. (2013) Software Defect Prediction Using Supervised Learning Algorithm and Unsupervised Learning Algorithm. *Confluence* 2013: *The Next Generation Information Technology Summit*, 26-27 September 2013, Uttar Pradesh, 173-179. https://doi.org/10.1049/cp.2013.2313

[59] Deep Singh, P. and Chug, A. (2017) Software Defect Prediction Analysis Using Machine Learning Algorithms. 2017 7*th International Conference on Cloud Computing, Data Science Engineering-Confluence*, 12-13 January 2017, Noida, 775-781. https://doi.org/10.1109/CONFLUENCE.2017.7943255

[60] Hussain, S., Keung, J., Khan, A. and Bennin, K. (2015) Performance Evaluation of Ensemble Methods for Software Fault Prediction: An Experiment. *Proceedings of the ASWEC* 2015 24*th Australasian Software Engineering Conference*, **2**, 91-95. https://doi.org/10.1145/2811681.2811699

[61] Hammouri, A., Hammad, M., Alnabhan, M. and Alsarayra, F. (2018) Software Bug Prediction Using Machine Learning Approach. *International Journal of Advanced Computer Science and Applications*, **9**, 78-83. https://doi.org/10.14569/IJACSA.2018.090212

[62] Tantithamthavorn. An R package of Defect Prediction Datasets for Software Engineering Research.

[63] Alsawalqah, H., Faris, H., Aljarah, I., Alnemer, L. and Alhindawi, N. (2017) Hybrid Smote-Ensemble Approach for Software Defect Prediction. In: Silhavy, R., Silhavy, P., Prokopova, Z., Senkerik, R. and Oplatkova, Z., Eds., *Software Engineering Trends and Techniques in Intelligent Systems*, Springer, Berlin, 355-366. https://doi.org/10.1007/978-3-319-57141-6_39