International Journal for Multidisciplinary Research (IJFMR)
E-ISSN: 2582-2160 ● Website: www.ijfmr.com ● Email: editor@ijfmr.com

A Comprehensive Study on Ensemble Methods for Software Error Detection

Nagib Mahfuz[1], Md. Mahedi Hasan[2]

[1,2]Senior Lecturer, Dept. of Computer Science and Engineering, North Western University, Bangladesh

Abstract

Software Developers often find it painful and tedious to locate or pinpoint the errors that reside in the source code which causes serious hindrance in the progression of developing any software. In the field of software engineering, it is very crucial to understand the software metrics that are directly involved with the progression of the software. Besides, various classification algorithms have been used to foresee the errors in building the software. In this paper, we focus especially on ensemble algorithms as they tend to provide more precise and statistically efficient outcomes than the other traditional algorithms. This paper contains twenty software metrics that are pivotal in identifying errors in software applications. Eight Java projects have been gathered to showcase the significance of the software metrics in predicting errors. In this study, three ensemble methods are considered, MultiBoostAB, Dagging, and Decorate. For a detailed inspection of the performance, accuracy, recall, precision, F-measure, and ROC Curve were appraised. The comparisons exhibit Decorate as the highest-performing method and Dagging as the lowest.

Keywords: Ensemble, Software Metrics, Software Error Detection.

1. INTRODUCTION

The impact of software metrics adoption in machine learning models for software error detection is important. In recent times, there have been some results that contradict the main goal of importing software metrics as features [1]. Because of this, the selection of the proper set of metric suits is a crucial part of software error detection. The dataset must be chosen with the precise attribute set that correlates with the metric suits as well [2]. There are lots of matters involved in hampering the progression level and performance of these predictive systems and usages of the models as well. Besides, different software application systems require different types of approaches in terms of solutions. Nowadays, the features that are considered as the software metrics have changed significantly because of the changing nature of building the software. Two of these issues are imbalanced data and the high dimensionality of the error-prone datasets [3] that are used to build the prediction models. So, the software metrics as features are very cautiously maintained for better performance of these models. Before software testing, various models based on defects prediction analysis methods are used to diagnose any existing errors in the modules [4].

Software error defection is one of the most impactful criteria of the testing phase of the software development life cycle. These models point out the errors in the software that require further testing and processing without breaching any parameters. There are lots of matters involved in hampering the progression level and performance of these predictive systems and usages of the models as well. Besides, different software application systems require different types of approaches in terms of solutions.

Depending on the application types, various software metric suits have been proposed. These metrics are called process metrics. Some popular metric suits are Halstead complexity measures [5], Chidamber and Kemerer (CK) metrics [6], measures of complexity by Brian Henderson-sellers [7], Abreu's Metrics [8], Bansiya and Davis' Quality Metrics [9] and so on.

Most of these errors but certainly not all of them can be identified through special arrangements. Here comes the predictive analysis of the defects containing systems that can provide further assistance and less error-prone environments. When it comes to software engineering and detecting various defects in the application, it usually varies which type of application it is. The ensemble method is comprised of various base algorithms and the main goal for it to enhance the base algorithms that already exist. In this study, we scrutinize some of the ensemble algorithms and their performances to detect any error in a software system to aid the developers beforehand.

This study is arranged as follows: Section 2 sums up the related works. Section 3 gives an overview of the research approach and the defective and imbalanced datasets as well as the dataset preprocessing methods. Section 4 contains a brief analysis of the ensemble algorithms used in this paper. Section 5 gives a comprehensive analysis and discussion of the final results of the performances of the ensemble methods. Section 6 presents the threats to validity and the conclusion is contained in section 7.

2. Literature Review

A detailed overview of the related works is given below. At first, some of the generic classification methods are reviewed followed by methods with feature selection techniques are covered. Afterward, some ensemble methods are assessed for software error detection.

2.1 Generic Techniques

In their paper, Surendra Naidu, et al., (2013) proposed decision tree-based classification algorithms such as C4.5 and ID3 [10]. The number of errors can be found and then reduced to minimize the cost and time to build the project as the defect configurations are sometimes ignored.

In another study, Logan Perreault, et al., (2017) worked on various classification methods, namely Neural Networks, K-NN, Logistic Regression, Naïve Bayes, and Support Vector Machine [11]. They used five datasets from NASA and relied on pattern learning between the features in the datasets. The final result displays that all of them have decent performances when using the static software attributes with a high degree of confidence.

In Naïve Bayes, attributes are presumed as not dependent on each other and have equal weight but they are correlated in practice. Omer Faruk Arer, et al., (2017) integrated the attributes as pairs in the Naïve Bayes algorithm and proposed Feature dependent Naïve Bayes method [12]. The experiment was carried out on eight datasets from the NASA datasets repository and the result was satisfactory compared to the other altered Naïve Bayes method.

During the development phase, although some errors get accepted sometimes, they can create bigger problems in the later stages. Pointing to this issue, Hiba Alsghaier, et al., (2020) used Support Vector Machine (SVM) with the help of Genetic Algorithm (GA) and Particle Swarm Optimization (PSO) for better error detection in software [13]. A total of 24 datasets from NASA MDP and Java open-source projects were considered to carry out the investigation and improved results were found when SVM is merged with GA and PSO.

While investigating error detection in software using Support Vector Machine, Mohammad Azzeh, et al., (2023) observed that the kernels have significant effects on the performance of SVM [14]. The Radial

Basis Function (RBF) and Sigmoid Function were better at accuracy while the Linear and Polynomial Kernel Functions underperformed. Also, reducing dataset dimensionality did not improve the performance of SVM notably, and the kernel function's effects vary with different attribute subsets.

In another study, Lov Kumar et al., tried to find out the faulty components of a software program. They utilized Least Squared Support Vector Machine (LSSVM) to construct the defect prediction model. Datasets from thirty different projects were collected from NASA, PROMISE and GitHub repository. As for the performance metrics, accuracy and F-measure were deployed for the T-test to yield statistical significance.

2.2 Feature Selection Techniques

Feature selection is a major preprocessing stage in machine learning and has been broadly utilized in a large section ranging from text recognition [15] to biomedical engineering [16]. Datasets with high dimensions often generate unwanted outcomes and thus it is suggested that less effective features be removed to enhance the classification performance as different metric sets could correlate with the errors in the system.

Using search-based software engineering problem, Kehan Gao, et al., (2011) proposed an automatic hybrid search algorithm that enhanced the searching performance of feature metrics that other traditional feature selection techniques. In this study, seven feature ranking techniques and three feature subset selection methods were investigated on real-world software projects [17]. Two of the best-known algorithms, Random Forrest and Support Vector Machine were considered for classification purposes. Although the software metrics involved in the software defect detection field were not considered.

Kun Zhu, et al., (2021) proposed a Search-based Enhanced Metaheuristic Attribute Selection method using Whale Optimization Algorithm (WOA) and Simulated Annealing (SA) which can impressively select lesser but highly correlated attributes [18]. Twenty datasets were collected from PROMISE, ReLink, and AEEEM data repositories and the performance was seemingly better than other traditional feature selection techniques.

Abdullateef Balogun et al., (2020) considered Naïve Bayes and Decision Tree classifiers to investigate forty-six feature selection methods. In this study, twenty-five datasets from NASA, PROMIS, ReLink and AEEEM were chosen. The final result showed that the performance of the selected feature selection technique depends on the corresponding classification method, datasets, and performance evaluation metrics.

In another study on feature reduction, Prava, et al.,(2020) proposed a mixed method where they used principal component analysis to lessen the dataset dimension and particle swarm optimization for subset extraction [19]. In their paper, Random Forest, Support Vector Machine and Naïve Bayes were utilized as classifiers. Five datasets from PROMISE and Kaggle dataset were considered to evaluate the performance of the proposed model. It is found that the proposed model performed up to 98.70% in terms of ROC-AUC curve, much higher than the traditional classification algorithms. It is also mentioned that whenever there is a lack of feature subsets, accuracy gap increases.

Tumar Lyad, et al., (2020) proposed Enhanced Binary Moth Flame Optimization (EBMFO) algorithm which is a derived form of Moth Flame Optimization (MFO) algorithm [20]. They converted the continuous form of MFO to binary form of EBMFO. In this paper, fifteen real-time projects from the PROMISE data repository were considered. For classification purposes, they employed Linear discriminant analysis (LDA), K-nearest neighbor (K-NN) and Decision Tree (DT) algorithms. Adaptive synthetic sampling (ADASYN) technique was utilized to overcome the imbalanced datasets for further

improved results. 20 of all basic object-oriented software metrics were used and this study's final result shows that the Decision tree performs the worst and LDA has the best performance in terms of ROC-AUC. On the other hand, K-NN conducts the result in less time compared to the other two classifiers.

2.3 Ensemble Techniques

Ensemble learning has become a successful process in classification problems. In software engineering, software error detection has been the most beneficial area when it comes to using ensemble methods rather than the other generic algorithms. Ensemble methods certainly enhance classification problems by merging various base algorithms and their corresponding features and uniqueness. These methods work very well especially on small to medium size datasets and handle the data imbalance efficiently.

Shuo Wang et al., (2013) proposed an ensemble approach named sampling-based online bagging [21]. In this paper, they tried to achieve a balanced performance through the proposed technique with negative and positive instances. However, when the class distributions alter over time, the method failed to give a straight and stable performance. Hence, Shuo Wang, et al. established Undersampling-based online bagging that is vigorous with the fluctuation of class distributions.

In the field of software engineering, well-defined research schemes are insufficient, unlike the other research areas. Monika Mangla, et al., (2022) presented a sequential ensemble model to detect the errors in the software [22]. This method was also executed on the eight datasets collected from PROMISE and ECLIPSE data repositories. The performance of the model was then assessed using different error metrics e.g. average relative error, average absolute error, and prediction. The outcome was satisfactory provided the right environment.

In software error detection, most of the classification method suffers due to a lack of correct decisions in choosing the best set of performance metrics. Yakub Kayode Saheed, et al., (2021) proposed a seven-ensemble machine learning model for software error detection. It this paper, the authors used Cat Boost, Light Gradient Boosting Machine (LGBM), Extreme Gradient Boosting (XgBoost), Boosted Cat Boost, Bagged Logistic Regression, Boosted LGBM, and Boosted XgBoost for the inspection. A separate analysis of base models of logistic regression was also initiated on six datasets. The performance of the model was measured by accuracy, AUC, precision, recall, F-measure, and Matthew Correlation Coefficient. The proposed model was able to give higher composure due to a reduction of the training time and overfitting.

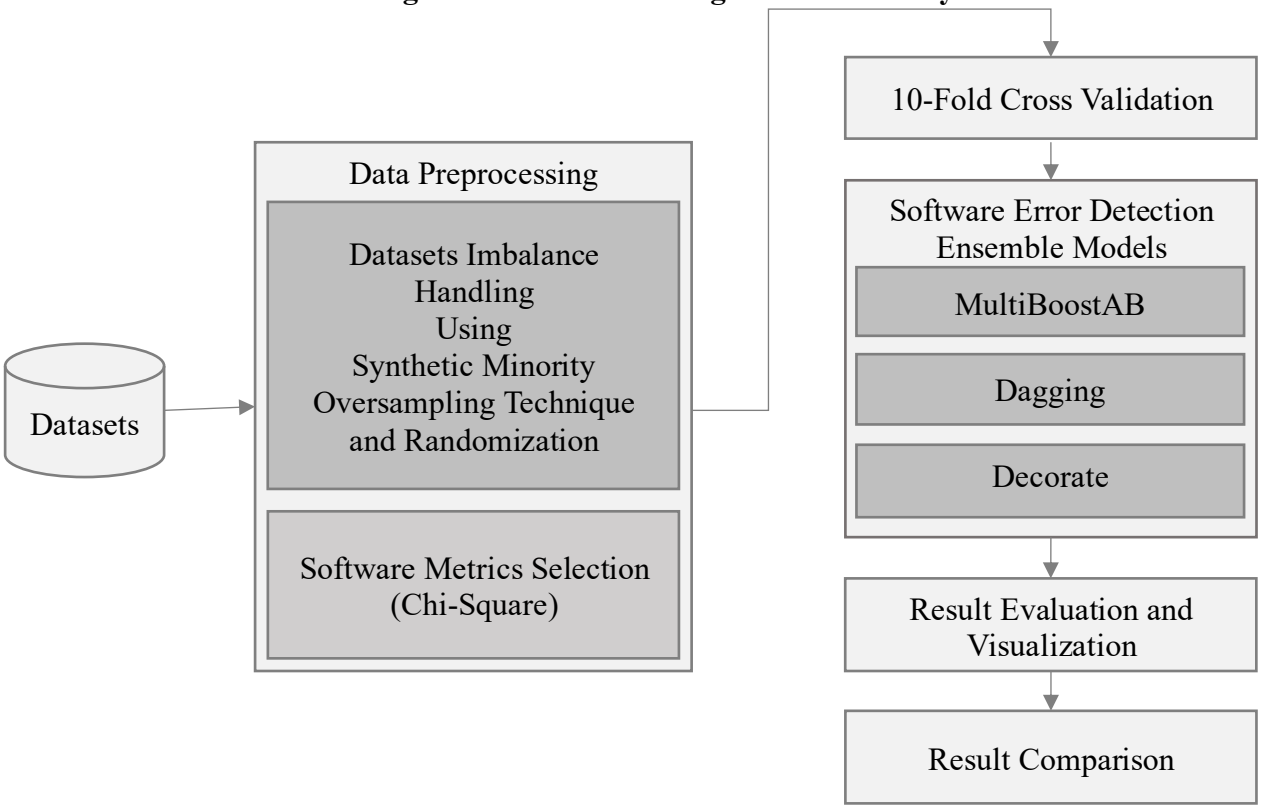
3. Research Approach

This paper mainly contributes to the software bug detection field with the help of various ensemble algorithms. Traditional studies with the base algorithms have been a common proceeding towards software error handling. Recently ensemble methods have been growing fast for their robustness, accuracy, and fast implementation in the system. In this study, three ensemble methods have been scrutinized: MultiBoostAB, Dagging, and Decorate. The main goal of this paper is to evaluate their performance on the collected datasets to find whether the modules are error-prone or not. A vast analysis of these methods will be discussed, and inspected whether their performances are statistically significant or not.

3.1 Functional Diagram of the Study

Numerous studies have been done on software error detection with conventional algorithms as the main focus. This paper progresses in rather a discrete way where ensemble algorithms thrive. Figure 1 displays the functional diagram of the study as follows:

Figure 1: Functional Diagram of the Study



3.2 Datasets

One of many problems in software error detection is that the datasets that exist at present tend to be very imbalanced. In this paper, eight Java projects, mined from GitHub are considered for the investigation to be carried out. Table 1 shows the datasets and how uneven the class distribution can be in the software system.

Table 1: Description of the Datasets

Project Name	No. of Instances	No. of Attributes	Data Type	No. of Defective Instances
arc	225	20	Numeric	29
camel-1.6	340	20	Numeric	13
ivy-2.0	362	20	Numeric	40
poi-2.0	314	20	Numeric	37
prop-6.0	644	20	Numeric	61
redactor	175	20	Numeric	27
synapse-1.0	157	20	Numeric	16
xerces-1.3	453	20	Numeric	69

3.3 Datasets Preprocessing

Imbalanced class distribution does not help to produce good results. Imbalanced datasets can be handled by using the Synthetic Minority Over-sampling Technique (SMOTE). Experimental studies show that most of the datasets related to software engineering are extremely imbalanced due to a small portion of modules producing errors in a software system. This means the dataset has an uneven distribution of target classes i.e. one of the class labels has a high number of observations and the other one has the opposite.

This situation makes the prediction model take the majority number class as the final prediction without considering any other variants. To alleviate this situation, we used SMOTE, one of the most recognized data sampling methods where the minority class is over-sampled by creating synthetic instances rather than using over-sampling with replacement.

Let, two data points be D_1 and D_2 and their respective coordinates are (x_1, y_1) and (x_2, y_2). D_1 is the data point under observation and D_2 is one the nearest neighbor of D_1. If the synthetic sample is coordinated at (x_3, y_3), then the following equation is used to create (x_3, y_3):

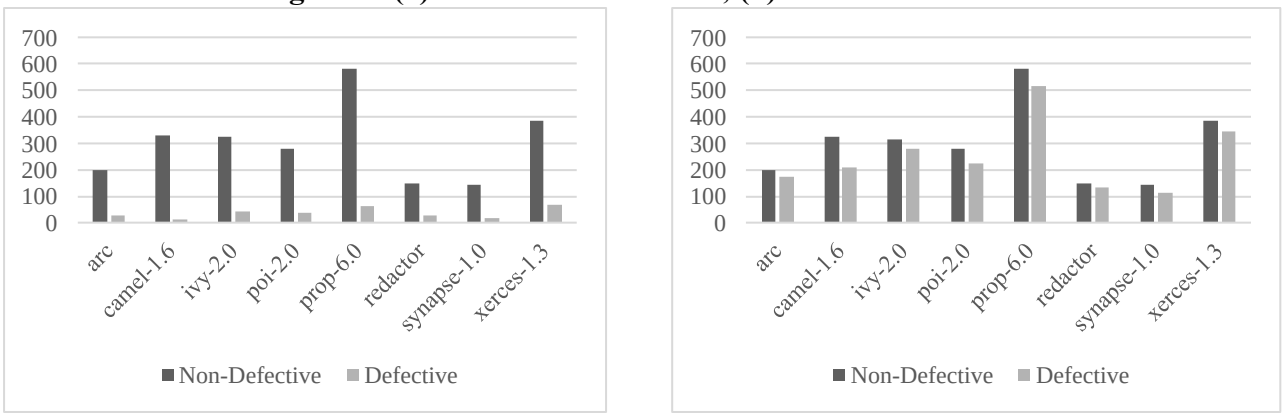
$$(x_3, y_3) = (x_1, y_1) + \text{rand}(0,1) \times \{(x_2 - x_1), (y_2 - y_1)\} \quad (1)$$

After applying SMOTE, the newly generated datasets grow with instances as the lesser class instances increase. However, the newly generated synthetic instances are appended at the end of the dataset. This creates a problem with overfitting as the newly added instances with same class stay consecutively. The new processed datasets then are randomized with randomize function to break the sequence of the same type of instances and solve the risk of overfitting. Figure 2 depicts the comparisons between uneven and even distribution of both error-prone and error-less instances.

3.4 Software Metrics Selection

Feature selection Methods are the consolidation of searching criteria that discover different subsets through all the attainable combinations of attribute subsets from an existing dataset that produces the best performance for any classification method [23]. Removing noisy, redundant, and irrelevant attributes that are hardly useful for specific classification problems, decreases the size of the datasets, and increases the possibility of generating better results. It creates possibly the finest dataset that suits the classification model best. In this paper, the features are the software metrics, and the metrics are considered very carefully as the metrics are highly dependent on the classification method used. Chi-square feature selection technique is used for dimension reduction process as there might noisy data exist.

Figure 2: (a) Imbalanced Datasets, (b) Balanced Datasets



(a) (b)

Based on the dependency on the inductive algorithm that eventually will operate on the subset, the feature selection methods can be divided into two extensive categories, filter and wrapper. The Chi-Square is a filter feature selection method. Any specific dependent and insightful feature with a high Chi-square value is selected for final classification and vice versa. The values are evaluated using the following equation:

$$X^2 = \sum_i \frac{(O_i - E_i)^2}{E_i} \quad (2)$$

Here, X^2 denotes the chi-square value for every selected feature. Afterward, O_i and E_i represent the observed and expected values respectively. Chi-square is an effective feature selection method in software defect prediction as it gives statistically significant values to the attributes that are most relevant and compatible with the distinct classification field. The Chi-square technique is utilized for its robustness towards the distribution of the data, its effective use where parametric assumptions cannot be made, and its simplicity of calculations.

Table 2: Software Metrics Used in the Datasets

Sl.	Software Metrics	Description	Sl.	Software Metrics	Description
1.	WMC	Weighted method per class	11.	LOC	Lines of code
2.	DIT	Depth in inheritance tree	12.	DAM	Data access metrics
3.	NOC	Number of children	13.	MOA	Measure of aggression
4.	CBO	Coupling between objects	14.	MFA	Measure of functional abstraction
5.	RFC	Response for a class	15.	CAM	Cohesion among methods of class
6.	LCOM	Lack of cohesion in methods	16.	IC	Inheritance coupling
7.	CA	Afferent coupling	17.	CBM	Coupling between methods
8.	CE	Efferent coupling	18.	AMC	Average method complexity
9.	NPM	Number of public methods	19.	MAX_CC	Maximum McCabe's cyclomatic complexity
10.	LCOM3	Lack of cohesion	20.	AVG_CC	Average McCabe's cyclomatic complexity

3.5 Cross Validation

Cross validation is a process of utilizing the training data in a better way to train and evaluate any model. It is better than traditional train-test approach in machine learning as cross validation allows to use the whole dataset as training data. This technique is also known as k-fold cross validation. The dataset is divided into k subsets or folds. Then the model is trained k times with each subset. The results are averaged to calculate the model's generalized performance. This way the datasets in this paper are properly utilized during the training stage to prepare the model for evaluating unknown data. In this paper, 10-fold cross validation process is considered where every dataset gets divided into 10 different folds, and each subset is used to train the three ensemble models used separately.

4. Ensemble Methods for Software Error Detection

The ensemble method is much more convenient and efficient in classification problems such as defect detection in software. In this study, three ensemble algorithms are utilized to find whether the modules in the datasets are faulty or not. A brief detail on each method is given below:

4.1 MultiBoostAB

This ensemble method is an extension of AdaBoost algorithm combined with wagging, where the base algorithm is C4.5 decision tree. Superior parallel execution and fewer errors are noticed than AdaBoost

and wagging. Originally, wagging and boosting together create an ensemble classifier where wagging assigns distinct weights to data samples and boosting assigns equal weights. Boosting is an iterative process whereas bagging forms the classifier independently. Thus, in MultiBoostAB algorithm, weights are assigned randomly using continuous poisson probability distribution, giving more accurate results than the other classifiers. Below is the algorithm:

1. Start and assign random weight (w_i) to data sample (D_i) using continuous poisson distribution.
2. For each 'k' iteration
{
Call C4.5 algorithm to build a model for D_i.
Compute error 'E' for all D_i and calculate reweighting for all D_i.
Keep W_i unchanged for false positive.
Create set of D_{LW} for low weight and D_{HW} for high weight.
Build classifier C(x) for reweighted D_I and classify D_{HW} correctly.
Combine C(x) output using weighted vote to form a prediction with-
$W_V = -\log(E \div (1-E))$
Add max. vote classifiers weights by choosing class with greatest sum.
If E is close to zero
{
C(x) receive a D_{HW} shows C(x) performing well
}
If E is close to 0.5
{
C(x) receive a D_{LW} shows C(x) performing poorly
}
Do the final prediction of C(x)
3. Exit

4.2 Dagging

Dagging is one of the ensemble methods that is being used widely in numerous research areas for its robust outcomes and enhanced accuracy in classification operations than most of the classification algorithms that exist. It was presented by Ting and Witten [42] in 1997. Dagging is related to Bagging ensemble method and named as it is obtained from the clause "disjoint Bagging". In Bagging, the same data can be repeated in multiple subsets. In contrast, Dagging is a meta classifier that construct a number of separate and stratified folds from the data making the subsets more unique. The main difference between bagging and dagging can now be illustrated as follows:

- Bootstrap samples: Randomly take samples from training dataset, T_D with replacement into K subsets of size N, where $KN \leq N'$.
- Disjoint samples: Randomly take samples from training dataset, T_D without replacement (also known as stratified cross-validation) into K subsets of size N, where $KN \leq N'$.

The steps of dagging algorithm are:

1. Divide D into t strata.
2. Set up the most complete list of each t comprising strata.

3. For t from 1 to T:
4. {
   Evaluate the percentage instances P_t to construct t^{th} stratum
   with respect to dataset D.
   Choose simple random sample to select instances from t_{th}
   stratum to build sample D^t regarding P_t.
   Learn weak classifier on D^t to generate H_t.
   }
5. Calculate $H_{vote} = \text{argmax}\sum_{i=1}^{T} H_i(x, y)$.
6. Exit.

4.3 Decorate

Diverse Ensemble Creation by Oppositional Relabeling of Artificial Training Examples, or Decorate is an ensemble learning algorithm proposed by Prem Melville et al., (2005) [24]. Using the artificial training instances, Decorate generates differing classifiers. The main difference makes Decorate from Bagging and Adaboost superior is that Bagging and Adaboost uses the training dataset to create various classifiers. Decorate gives us a new perspective by yielding base classifiers on the augmented training dataset which breaks the constraints of only using provided training sets. Initially, the base algorithm is launched to train a classifier C_1 on the training set and the first ensemble is formed. Then Decorate generates classifiers repeatedly in the following way: A set of artificial datasets is made according to the characteristics of the original dataset. Each artificial instance is assigned by a class label that should be different as much as possible than the predictor of the initially created ensemble. Then the artificial datasets are appended to the original dataset thus, creating an augmented dataset which is then used on the base algorithm to train. The generated classifier then again put into the current ensemble. If its accuracy is lower than that of the original ensemble without classifier, then the classifier is pruned. These steps repeat until a desired number of classifiers are produced or iteration is reached to its maximum number. The Decorate algorithm is illustrated below:

- Input:
  BaseLearn: a base learning algorithm;
  D_{tr}: the original training set, $D_{tr} = \{(x_1,y_1),(x_2,y_2),\ldots,(x_N,y_N)\}$;
  R: the proportion of artificial training set with respect to D_{tr};
  M: the desired ensemble size
  I: the maximum iteration time;
- Training phase:
  - Initialization
    (1) i = 1, iteration = 1;
    (2) C_i = BaseLearn(D_{tr});
    (3) Initialize the ensemble, C = $\{C_i\}$;
    (4) Compute the training accuracy of C: accu = $\frac{1}{N}\sum_{t=1}^{N} I(C(x_t) = y_t)$;
  - Iteration process
    While i < M and iteration < I
    (5) Generate [N×R] artificial training examples D_{arti} according to the data distribution characteristic of the original training set D_{tr};
    (6) Label each example in D_{arti} according to te principle that the

assigned labels differ maximally fom those predicted by the current ensemble C;

(7) Add the labeled artificial training data D_{arti} to the original training set D_{tr}: $D_{augm} = D_{tr} \cup D_{arti}$;

(8) Train a classifier on the augmented training set C_i = BaseLearn(D_{augm});

(9) Put the generated classifier into the current ensemble, $C = C \cup \{C_i\}$;

(10)Based on the original training set D_{tr}, compute the training accuracy accrcy of the expanded ensemble C as in step (4);

(11)If accrcy ≥ accu

(12) i = i + 1, accu = accrcy;

(13)Else

(14) C = C\{C_i};

(15)iteration = iteration + 1;

(16)Exit;

5. Experiment Results Analysis and Discussion

The main goal of this paper is to scrutinize the performances of the above-mentioned ensemble methods and how they are statistically significant to each other. A thorough evaluation has been made of the classification performance metrics to clarify their executions The performance metrics considered in this study are listed below:

Table 3: Classification Performance Metrics

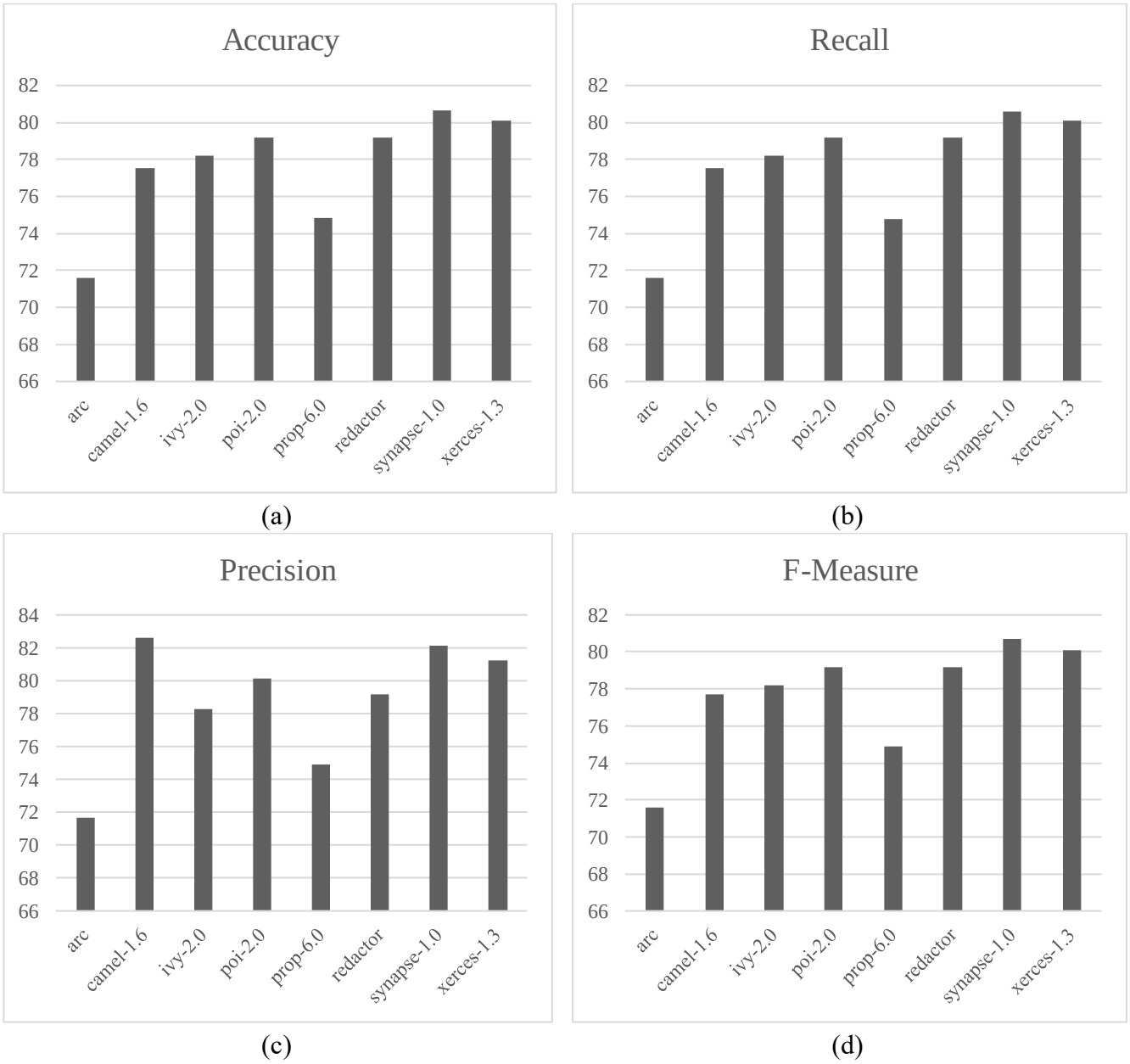
Metrics	Formula	Focused Area
Accuracy	$\frac{tp + tn}{tp + tn + fp + fn}$	Ratio of accurately predicted instances out of all instances
Recall	$\frac{tp}{tp + fn}$	Ratio of the accurately identified True Positives data and total of positive data
Precision	$\frac{tp}{tp + fp}$	Ratio of correctly predicted data and total number of positive predictions
F-Measure	$\frac{tp}{tp + 1/2(fp + fn)}$	Harmonic mean of precision and recall, where β = 1
ROC Curve		Plot between true positive rate and false positive rate
tp = true positive, tn = true negative, fp = false positive, fn = false negative		

5.1 Performance Analysis of MultiBoostAB

MultiBoostAB classifier has the learning curve of 89.40% for datasets xerces-1.3, followed by camel-1.6 dataset which has 89.30%. In most cases, this classifier performs better on dataset xerces-1.3 in every category of performance metrics averaging above 80 % all the time. With 82.60% of precision on camel-1.6, MultiBoostAB learns well for every dataset but the outcome is not good as expected. The lowest

values achieved by the classifier on dataset arc are 71.62%, 71.70%, 71.60% and 71.60% for accuracy, pprecision, recall and F-measure correspondingly. Figure 3 illustrates the performance of the classifier through charts.

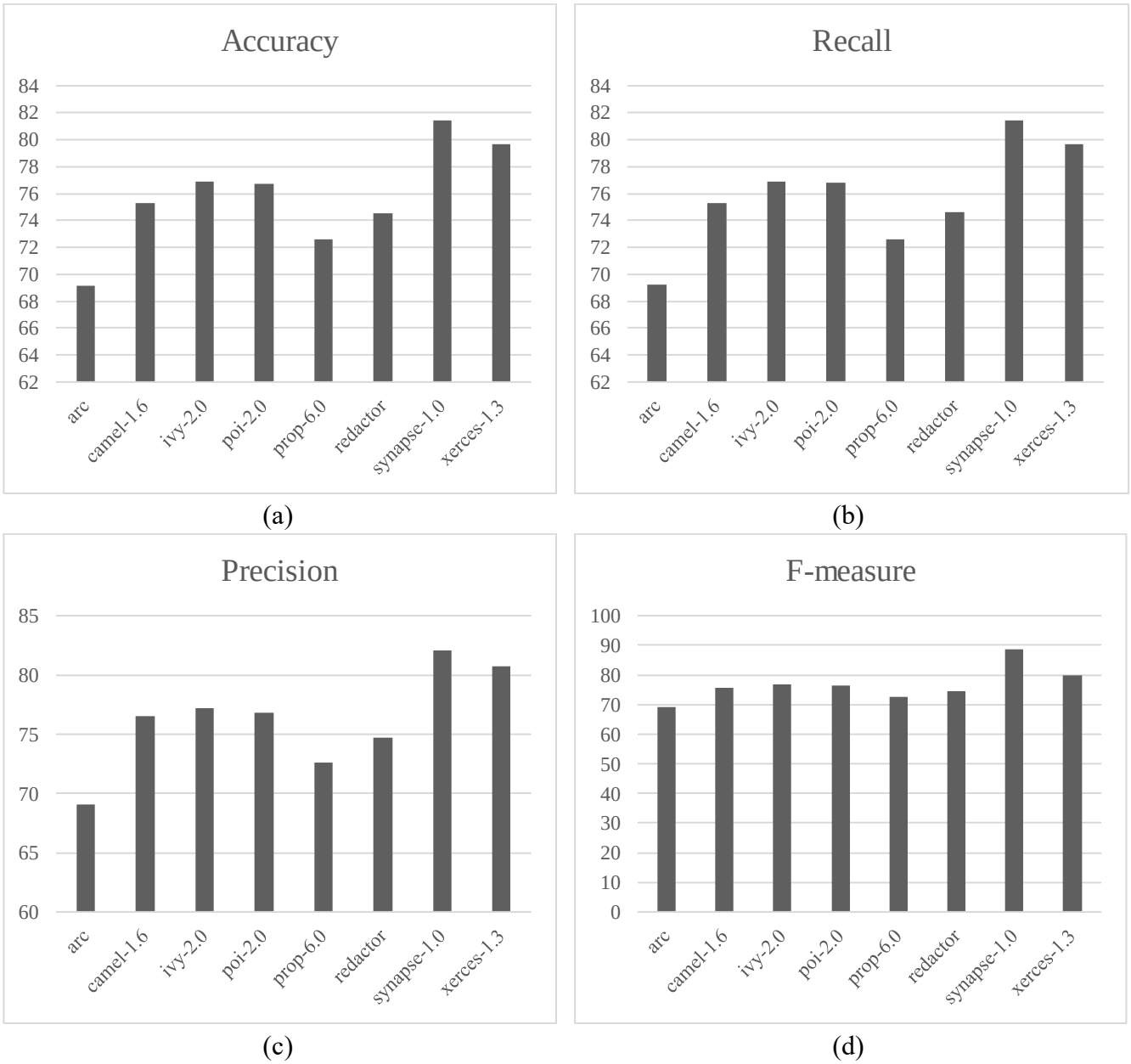
Figure 3: Performance Analysis of MultiBoostAB



5.2 Performance Analysis of Dagging

The stratified cross fold validated Dagging performs better on dataset synapse-1.0 with the highest achievement 88.70% on F-measure category and overall, it has over 80% success rate on accuracy, precision, and recall on synapse-1.0. It has the lowest performance on the arc dataset with little over 69% in all performance categories. The learning curve is a little less than MultiBoostAB with only two exceptions of poi-2.0 and synapse-1.0 datasets, having 88.70% and 84.90%, where MultiBoostAB gains 83.80% and 83.40% respectively. The minimum values generated by Dagging is with dataset arc, averaging 70.20%.

Figure 4: Performance Analysis of Dagging



5.3 Performance Analysis on Decorate

Among the three ensembles explained in this paper, Decorate is the most promising and accurate of them all according to the performance metrics in Figure 5. Decorate learns the best from the training data compared to the other two. With the maximum ROC curve of 97.90% on camel-1.0 dataset, Decorate produces the best criteria of balancing between generating false alarm and misses, averaging the value of more than 91%. The highest values gained by Decorate in on dataset camel-1.0 with the accuracy of 96.25%, 96.30% on precision, 96.30% on recall and 96.30% on F-measure. But with a careful observation, it can be seen that the MultiBoostAB and Dagging performed well on dataset synapse-1.0, where Decorate performed less on synapse-1.0 dataset compared to its exhibition on other datasets provided. The classifier produces the least amount of performance on the classification performance metrics with 86.22% on accuracy and 86.20% on the other three metrics respectively.

Figure 5: Performance Analysis on Decorate

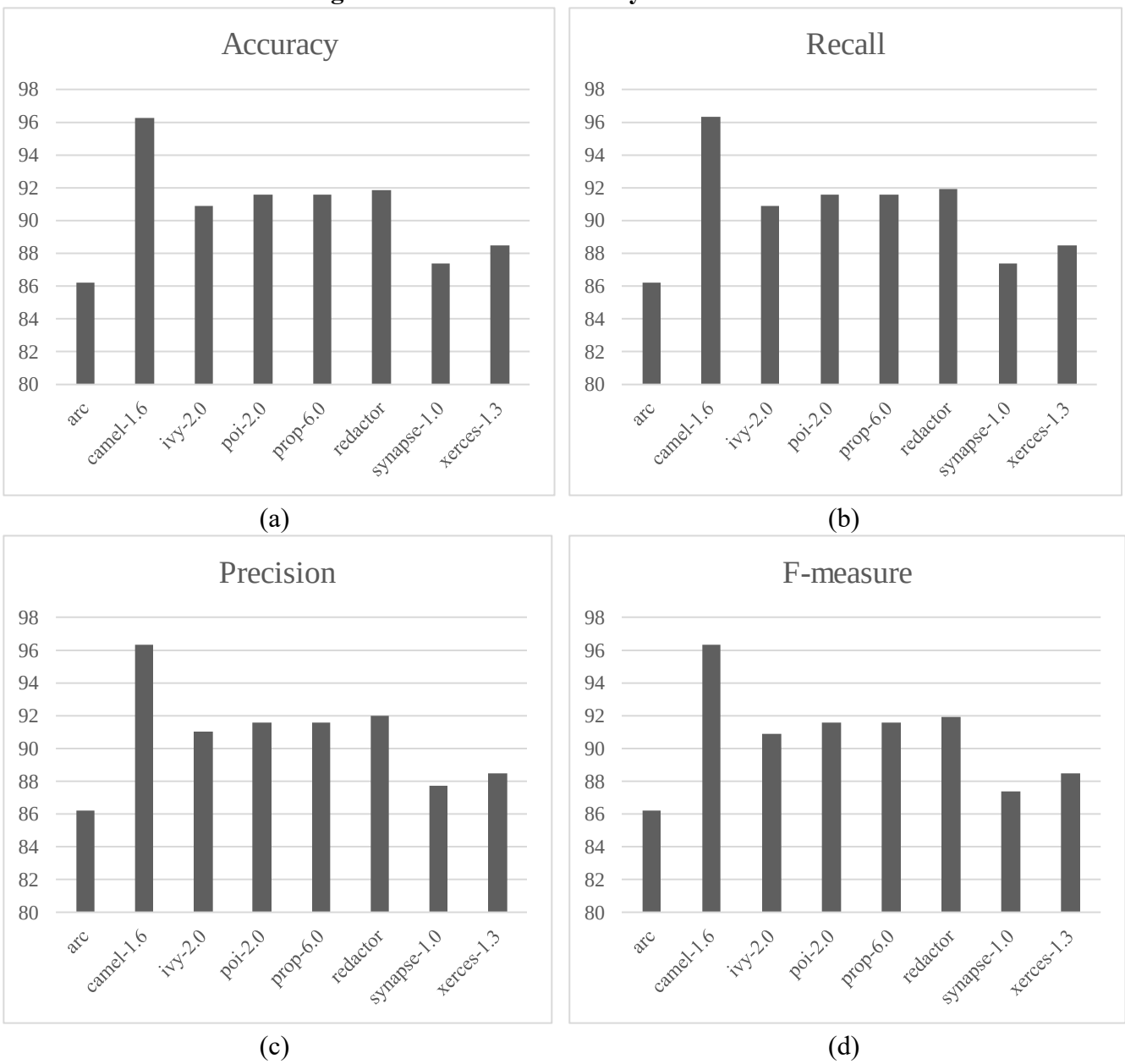
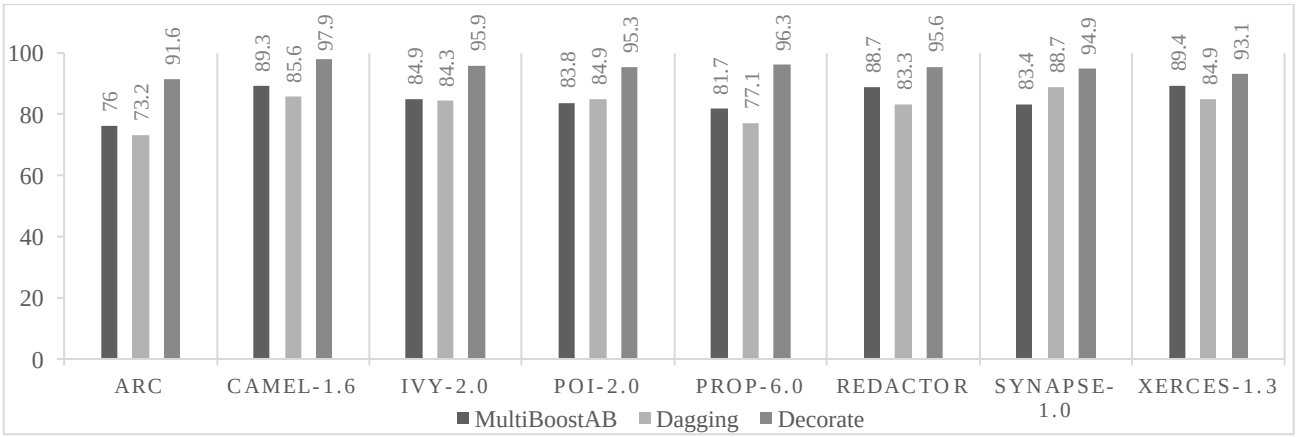


Figure 6: ROC Curve Comparisons on Ensemble Algorithms



From Figure 6, it is clear, that among the three ensembles, Decorate algorithm thrives the most and outperforms the other two in term of ROC Curve. In all the other performance criteria, Decorate dominates for all the datasets considered in this study.

6. Threads to Validity

There are quite a few threads that might have issues on the outcome of this study. Both internal and external threads are discussed in brief below.

Internal threads are related to the selection of the software metrics. This paper has used the metrics that are only available in the datasets. Other software metrics might perform better to error detection in other literatures.

Though the datasets are all experimented by WEKA, there is a chance that the corresponding results can vary in other languages as a threat of generalized outcome. As the datasets were resampled to avoid overfitting and 10-fold cross validation was used instead of simply splitting them into train and test data, it still might depend on environmental changes to produce different results.

The study concludes here is based on the selected system. This system may not ne aligned alongside the industrial perspective as datasets related to software engineering tend to be very imbalanced. So there always remain a possibility that the findings of the study are not good fit for production stage.

7. Conclusion

The main aim of this study is focused on detecting error-prone modules in a software. In software error detection field, it is a universal truth that, if a product is to be productive, developers must proof check further after a software product is completed making before it is handed over to the user or to the market. Any software, with bugs, is bound to get demoted from its demand drastically. So, this is a compelling matter to yield an environment and decrease the error from happening. Obviously, there are tons of researches on how to achieve it and ensemble methods are thriving against the traditional classification algorithms in this prospect. In this paper, three ensemble algorithms are studied. Eight datasets from GitHub have been utilized in the process. Synthetic Minority Oversampling Technique was used to all the datasets to solve the unevenness problem. A 10-fold cross validation technique was used rather than train-test split process as the data instances is more randomized in cross validation that gives better insight of the dataset. The study is carries out in WEKA data mining tool. The detection results were verified by the classification accuracy, recall, precision, F-measure and ROC Curve. This study shows that, Decorate dominates in all criteria marking ROC Curve with 97.90%. This paper, in the future, can be extended with increased number of datasets, using other ensemble algorithms and deep learning as well.

References

1. Dominik R. A., Stanislav C., Bruno R., “Source Code Metrics for Software Defects Prediction”, In Proceedings of the 38th ACM/SIGAPP Symposium on Applied Computing, March 2023, 1469-1472.
2. Issam H. L., Mohammad A., Lahouari G., “Software defect prediction using ensemble learning on selected features”, Information and Software Technology, 2015, 58, 388-402.
3. Sweta M., Sridhar K. P., “Improved prediction of software defects using ensemble machine learning techniques”, Neural Computing and Applications, 2021, 33(16), 10551-10562.
4. Ishani A., Vivek T., Anju S., “Open issues in software defect prediction”, Procedia Computer Science,

46, 906-912.
5. Halstead M. H., “Elements of Software Science”, Operating and programming systems series, Elsevier Science Inc, 1997.
6. Chidamber S. R., Kemerer C. F., “A metrics suite for object oriented design,” IEEE Transactions on software engineering, 1994, 20(6), 476-493.
7. Henderson B., Sellers, “The mathematical validity of software metrics,” ACM SIGSOFT Software Engineering Notes, 1996, 21(5), 89-94.
8. Fernando B. A., “Object-oriented software engineering: Measuring and controlling the development process,” In Proceedings of the 4th international conference on software quality, October, 1994, 186.
9. Jagdish B., Carl G., “A hierarchical model for object-oriented design quality assessment,” IEEE Transactions on software engineering, 2002, 28(1), 4-17
10. Surendra M. N., Geethanjali N., “Classification of defects in software using decision tree algorithm”, International Journal of Engineering Science and Technology, June, 2013, 5(6), 1332.
11. Logan P., Seth B., Clemente I, Johm S., "Using classifiers for software defect detection." 26th International conference on software engineering and data engineering. 2017.
12. Ömer F., A., Kürşat A., "A feature dependent Naive Bayes approach and its application to the software defect prediction problem", Applied Soft Computing, 2017, 59, 197-209.
13. Hiba A., Mohammed A., "Software fault prediction using particle swarm algorithm with genetic algorithm and support vector machine classifier", Software: Practice and Experience, 2020, 50.4, 407-427.
14. Mohammad A., Yousef E., Ali B., N., Lefteris A., "Examining the performance of kernel methods for software defect prediction based on support vector machine", Science of Computer Programming, 2023, 226, 102916.
15. Xuelian D., Yuqing L., Jian W., Jilian Z., "Feature selection for text classification: A review", Multimedia Tools and Applications, 2019, 78.3, 3797-3816.
16. Atikul I., Soumita S., Tapas B., Saurav M., Arup R., Aimin L., Manash S., "Feature Selection, Clustering and IoMT on Biomedical Engineering for COVID-19 Pandemic: A Comprehensive Review.", Journal of Data Science and Intelligent System, 2023.
17. Kehan G., Taghi M., K., Huanjing W., Naeem S., "Choosing software metrics for defect prediction: an investigation on feature selection techniques", Software: Practice and Experience, 2011, 41.5, 579-606.
18. Kun Z., Shi Y., Nana Z., Dandan Z., "Software defect prediction based on enhanced metaheuristic feature selection optimization and a hybrid deep neural network", Journal of Systems and Software, 2021, 180, 111026.
19. Lakshmi P., C., Shivakumar N., "Software defect prediction using machine learning techniques", 4th International Conference on Trends in Electronics and Informatics (ICOEI), IEEE, 2020, 48184
20. Tumar I., Yousef H., Hamza T., Thaer T., "Enhanced binary moth flame optimization as a feature selection algorithm to predict software fault prediction", IEEE Access 8, 2020, 8041-8055.
21. Wang S., Leandro L. M., Xin Y., "Online class imbalance learning and its applications in fault detection", International Journal of Computational Intelligence and Applications, 2013, 12.04, 1340001.
22. Mangla M., Nonita S., Sachi N. M., "A sequential ensemble model for software fault prediction", Innovations in Systems and Software Engineering, 2022, 18.2, 301-308.

23. Jain, S., Anju S., "Improving performance with hybrid feature selection and ensemble machine learning techniques for code smell detection", Science of Computer Programming, 2021, 212, 102713.
24. Prem M, Raymond J. M., "Creating diversity in ensembles using artificial data", Information Fusion, 2005, 6.1, 99-111.