# ABPET: An Automatic Software Bug Prediction Using Ensemble Learning Technique

**Nidhi Srivastava[1*], Manisha Agarwal[2], Pramod Kumar Soni[3]**

**Abstract**:

Software Bug prediction (SBP) helps in improving the quality of software to be developed. Bugs can be of different kinds such as functional, usability, security, etc, which leads to irrelevant features, data redundancy and missing samples. SBP is a very popular area in the IT world as thishelps in improving the software product.SBP is a complicated task that can be performed both manually (semi-automated) or automatically. The semi-automated task requires extensive knowledge about the product to be developed and manpower. On the other hand, automatic SBP eliminated such issues and improves the bug prediction cost significantly. In this paper, a multistage automatic SBP (ABPET) model using ensemble machine learning techniques is proposed. The proposed ABPET architecture is divided into three phases: in preprocessing stage the skewed and asymmetric data is preprocessed performed, in the second stage, ABPET classification model using ensemble ML algorithms RF and AdaBoost designed. Finally, a variety of estimated metrics such as precision, recall, accuracy F-measure, and Mathews co- relation coefficient are used to evaluate the constructed framework. The proposed multi-stage framework is tested on a publicly available dataset of the Promise repository. The experimental results indicated that AdaBoost outperforms the random forest by a 2% on accuracy and F- measure parameter.

**Keywords:** software bug prediction, random forest, AdaBoost, ensemble learning, classification technique.

**DOI Number: 10.48047/nq.2022.20.14. NQ880182** **Neuroquantology 2022; 20(14):1371-1381**



## 1Introduction

In the present world, the evolution of computational devices and technologies has transformed human life in every aspect and they are more dependent on computing systems. To maximize the utilization of computational capability of high computing machine for reducing human effort there is a need to develop software products at a faster rate, economically and reduce maintainability. While designing large systems it is necessary to find the bugs at the early stages as the cost of fixing the bug is less as compared to later stages and machine learning and its variations have the potential to ease the quality of software product development and improve businesses by making predictions and assisting them in decision-making. It is vital to understand the pros and downsides of ML applicability. ML is a part of computing technologies widely used to solve complex problems which require intelligence equivalent to human by learning from their experiences and improve on their own. Machine learning helps software researchers to keep their focus on how to improve the quality of software products, reduce maintainability and human efforts which lead to more efficient and effective software development. Predicting software flaws early in the software life cycle can help decrease product development and maintenance expenses (Khuat and Le, 2019). The primary objective of SBP techniques is to discover modules with and without bugs; the developer will then allocate appropriate testing sources and priorities testing of multiple software modules to improve software quality. The study of SBP is categorized into four classes (a) prediction of bugs present in software objects like functions, files, modules etc., by using machine learning and probability-based methods; (b) the second category involves the prediction of numberof faults by using regression analysis

***Corresponding Author**:- Nidhi Srivastava
**Address:** [1*,2]Banasthali Vidyapith University,
[3]Manipal University Jaipur

**Relevant conflicts of interest/financial disclosures:** The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest

etc. (c) In the third type of study the relationship between various types of software metrics has been covered to deduces the associations rules and finally the software products are prioritized based on the number of defects presents.

Machine learning techniques can play a key role in designing bug-free software by predicting bugs early in the software development process which in turn reduces overall development cost. In the Software bug prediction technique researchers studied many machine learning techniques like supervised (Moosavian *et al.*, 2014; Singh *et al.*, 2014a; Arar and Ayan, 2015; Sharma, Sharma and Gujral, 2015; Wang *et al.*, 2021) and unsupervised learning (Zhang, Wang and Wang, 2016; Feng *et al.*, 2021; Goyal, 2022). A fuzzy-based bug prediction model at each phase of SDLC has been proposed that minimizes the difference between predicted and actual (Yadav and Yadav, 2015). Immune theory-based fault prediction framework has been presented that uses a Naïve Bayes classifier for prediction (Soleimani and Asdaghi, 2014). In the Software bug prediction technique, software metrics can be trained by finding out the software efficiency and also checking for the software quality (Ghotra, McIntosh and Hassan, 2015). The exiting techniques for SBP suffers from issues such as The exiting techniques for SBP suffers from issues such as Class imbalance problems (Tong, Liu and Wang, 2018), overfitting issues (Zhu *et al.*, 2021), other problems with SBP are when many attributes are added to the SBP model also Facing an issues of heterogenousity.

In this paper, to over the above mention limitations, a multistage automatic SBP (ABPET) model is designed to predict the bugs from given software metrics. The basic objective of developing ABPET is to identify defective and non-defective modules, which will be provided by the developer as suitable testing sources as well as the testing of numerous software modules in order to enhance the quality of software. The remaining paper is systematized in the following manner: the previous contributions to the software bug prediction are discussed in Section 2. The description of the dataset is presented in Section 3. The designed multistage ABPET framework and the evaluation and comparative analysis are discussed in Section 4. Finally Section 5 describes the concluding remarks and upcoming directions.

## 2. Previous work

SBP is a technique for determining the bugs in software products to make better quality. The software bug prediction techniques using different metrics are further divided into categories namely static and dynamic. Instead than to use historical data, the static prediction method uses predetermined defect patterns to predict the amount of defects using metrics (without executing the code) (Tang *et al.*, 2015). The dynamic defect prediction method, on the other hand, uses the defect generation time to anticipate the distribution of system flaws over time (Jing *et al.*, 2014). The SBP systems are based on handcrafted features such as McCabe (McCabe, 1976), Hasted (Halstead, 1977), Chidamber and Kemerer (CK) metrics (Chidamber and Kemerer, 1994), object-oriented (Tang, Kao and Chen, 1999) The SBP techniques developed by various researchers are broadly categorized into following; (a) probabilistic or statistical model-based (b) machine learning-based (c) ensemble learning-based (d) deep learning-based.



### 2.1 Bug detection techniques based on the probabilistic or statistical model

Probability or statistical bug detection methods identifies bugs by analyzing the run-time behaviour of a program and are programming invariant in bug isolation and detection. The commonly used statistical bug detection methods are based on logistic regression, conditional probability (Liblit *et al.*, 2005) and hypothesis testing(Liu *et al.*, 2005) techniques. A bug detection method (Bugram) based on the probability of tokens is designed to detect bugs. The program streams are read as tokens and the probability of each token is calculated. The token having the lowest probability is marked as a potential candidate for the bug. The investigations are carried out on open source Java projects and it has been observed that Bugram has obtained better accuracy as compared to traditional rules-based detection systems (Wang *et al.*, 2016). A dynamic bug analysis (Gertler, 1988) method based on statistical inferences is designed. The designed method consists of program analysis

by predicates (Pandey and Tripathi, 2020), monitoring of predicates that are distributed using sparse random sampling and reports of each predicate. The method detects bugs and the predicates involved in the bug. A software bug isolation method based on random sampling (Lindley and Phillips, 1976) and logistic regression and predicate based technique is used to isolate both non-deterministic and deterministic bugs in large software systems (Liblit *et al.*, 2003). The statistical-based bug prediction techniques are computationally inefficient as they require a huge number of program executions.

## 2.2 Bug detection techniques based on the machine learning

ML is important for the study of computer modeling among all types of learning processes. There are many attempts to install such capabilities and to-do this ML It plays a crucial aspect in it. ML has some objectives as: (1) to improve the performance in the piece of workdevelopment and analysis of the learning system is done. (2) Simulation is done for the human learning process. (3) the theoretical exploration of likely all learning methods and algorithms independent(Carbonell, Michalski and Mitchell, 1983). ML has different methods like supervised and unsupervised. In the supervised learning method different classification models are used like SVM, NN and decision tree, KNN. Many techniques, such as decision trees (Knab, Pinzger and Bernstein, 2006), Bayesian algorithms (Kanwal and Maqbool, 2012), and ANN (Arar and Ayan, 2015) are successfully used in software defect prediction clustering, to train the classifier respectively whereas unsupervised learning clustering is gaining popularity since it does not require manually labelled data, which is particularly useful given the present rapid development of data (Mao *et al.*, 2019). A scenario-based method using k-mediods algorithm for clustering and C4.5 model for prediction is proposed for the prediction of bugs, and the size of the decision tree has been reduced by half but the performance depends on the selection of module (Li *et al.*, 2014). Machine learning's application in radiology is currently developing. The performance of machine learning-based systems is close to that of experienced radiologists in many applications.

## 2.3 Bug detection techniques based on deep learning

Deep learning architecture has grown in prominence in recent years, and it is now employed in a variety of applications such as picture categorization from remote sensing. (Sharma *et al.*, 2017) medical (Qayyum *et al.*, 2017), speech recognition (Noda *et al.*, 2015), natural language processing (Klosowski, 2018) and image processing (Liu, Lin and Shen, 2015). Various deep learning architectures have also been used in bug prediction, DeepSim model for identifying code similarity using feed-forward deep neural network based on gradient descent learning algorithms (Zhao and Huang, 2018). Ma. *et al.,* proposed a deep learning-based debugging technique that uses internal features responsible for bugs identified by differential analysis and training the model to identify bugs automatically (Ma *et al.*, 2018). Guo *et al.*, word embedding was utilized and recurrent neural network (RNN) model for automated trace link generation which are used effectively in various stages of software development such as test case selection, validation etc. The earlier model is used to learn domain knowledge which is used by the RNN model to learn the sentence semantics (Guo, Cheng and Cleland-Huang, 2017). Huo *et al.,* designed the NP-CNN model based on deep learning to delineate features helpful in locating bugs from natural and programming languages (Huo, Li and Zhou, 2016). To locate troublesome files, the architecture is constructed on two deep neural network layers that learn unifying elements from plain language and programming language. The Intra-language Feature Extraction layer used different CNNs for NLP and programming languages while Fully-CNN is used to implement cross-language Feature Fusion layers.

## 2.4 Bug detection techniques based on the Ensemble learning models

Machine learning-based solutions, on the other contrary, are poor in the case of skewed and redundant fault datasets. While in case of incomplete or irrelevant information, these methods perform worse and overlook the minority class, resulting in high false-negative. Now adays, a single predictive model (learner) is combined with other learners to form an EL model (ELM) to get better the performance of a

machine learning model. ELMs are particularly well suited to dealing with the data problems described previously. Furthermore, due to feature repetition and irrelevance, the process in EL mitigates such a long-term influence. Individual classifiers with favorable performance on the testing datasets are given higher weights, which reduces the likelihood of failure. The prediction performance is unquestionably improved by the robustness against irrelevant and duplicated features. Despite the fact that they aren't designed to directly fix data imbalances, RF outperform the above techniques in finding defective modules. Furthermore, the ensemble learning voting process reduces any residual effect due to feature irrelevance**.**

## 3. Dataset used

In this paper, research goal will be address which are specified in the previous part an ABPET model based on ensemble machine learning techniques is designed and experiments are performed on a well-known freely available dataset obtained from the promise repository (Meiliana *et al.*, 2018). An outline of dataset is described in Table 1 and 2.

**Table 1:** Description about Promise Repository Dataset.

| Dataset | No. of Modules | Defective | Defective (%) |
|---|---|---|---|
| lucene-2.0 | 195 | 91 | 46.67 |
| velocity-1.6 | 229 | 78 | 34.06 |
| log4j-1.1 | 109 | 37 | 33.94 |
| synapse-1.2 | 256 | 86 | 33.59 |
| jedit-3.2 | 272 | 90 | 33.09 |
| ant-1.7 | 745 | 166 | 22.28 |
| camel-1.6 | 965 | 188 | 19.48 |
| xerces-1.2 | 440 | 71 | 16.14 |
| data_redaktor | 175 | 27 | 15.43 |
| xerces-1.3 | 453 | 69 | 15.23 |
| xalan-2.4 | 723 | 110 | 15.21 |
| jedit-4.2 | 367 | 48 | 13.08 |
| data_arc | 225 | 29 | 12.89 |
| poi-2.0 | 314 | 37 | 11.78 |
| data_ivy-2.0 | 352 | 40 | 11.36 |
| synapse-1.0 | 157 | 16 | 10.19 |
| data_prop-6 | 644 | 61 | 9.47 |
| camel-1.0 | 339 | 13 | 3.83 |

**Table 2:** Metric used for Training ASBP model.

| Type | Attributes | | Description and Effects |
|---|---|---|---|
| CKR Metrics | lcom3 | Lack of cohesion in methods | Determines the complexity with in OOSD. classes and more complex system is soft victim for detects |
| | Ic | Inheritance coupling | |
| | Cbm | Coupling between methods | |
| | Amc | Average method complexity | |
| | Dit | Depth of Inheritance Tree | |
| | Noc | Number of Children | |
| | Cbo | Coupling Between Objects | |
| | Rfc | Response for Class | |
| | Lcom | Lack of Cohesion of Methods | |
| MT's Metrics | Ca | Afferent couplings | Describes the behavioral relationship between modules and their instances. Higher value denotes low utilization of OOP principles and pronetodefects |
| | Ce | Efferent coupling | |
| QMOOD | Npm | Number of public methods | Denotes the quality of design and behavior of object, class and lower value of quality leads to errors |
| | Dam | Data access metric | |
| | Moa | Measure of aggregation | |
| | Cam | Cohesion among methods of Class | |
| MaCabe's Metrics | max_cc | max.(McCabe's cyclomatic complexity) | Describe the complexity in decision making, more complex process is prone to errors |
| | avg_cc | avg. (McCabe's cyclomatic complexity) | |
| Size | Loc | **Lines of code** | Determine the overall size of software |

## 4. Automatic software bug prediction using ensemble Learning Techniques (ABPET)

This section describes the proposed methodology for ABPET for bug prediction using software metrics as shown in Fig. 1. The suggested framework is separated into three phases, each of which is described as preprocessing, ensemble learning-based bug prediction and evaluation of performance measure and comparative analysis on different evaluation metrics.

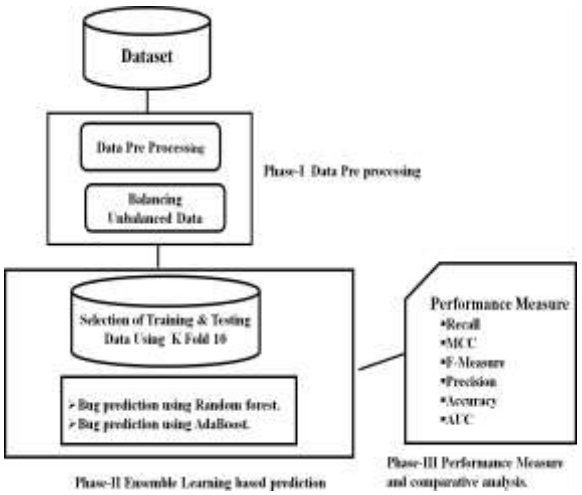



**Figure 1:** Architecture of ABPET



### 4.1 Preprocessing

Most real-world data has a skewed and asymmetric-distribution of instances available for each class. Due to the problem of asymmetric class distributions, algorithms may train unnecessarily complex models that over fit the data and are of limited use. Despite their superior performance, computational intelligence techniques are directed towards majority class instances, learning more about the majority class while learning very little or nothing about the minority class (Ball, Reay and David, 2002). The most challenging work in nowadays is with imbalanced datasets in the machine learning techniques and can be handled by synthetic minority oversampling technique (SMOTE) in which the minority class is augmented. The SMOTE approach is efficiently employed in combination with the supervised methods for bug detection and the proposed method has low computation cost and satisfactory performance (Chen *et al.*, 2019)

### 4.2 ABPET model based on Ensemble Learning.

In the domain of bug prediction, the traditional

machine learning models such ANN, SVM etc. are performs inefficiently due to their inability in reducing bias, variance and noise. All these issues are resolved by Ensemble learning models which combine the effort of various individual learning algorithms in predicting outcomes. The general framework for ensemble learning for datasets $A$ having $p$ samples and $q$ features shown in Eq. 1can are expressed and represented in
Eq. 2:

$$A=\{(x_i,y_i)\} \quad (|A|=p, x_i \in Z^q, y_i \in Z) \quad (1)$$

$$y_i = \phi(x_i) = G(f_1, f_2 \ldots f_j) \quad (2)$$

Where the $\phi$ is ensemble learning model has an aggregation function $G$ that combines $j$ learning classifiers to generate the aggregate output $y_i$ In the proposed ABPET model two ensemble learning models are used to predict the bug. The first the theory rests on the bagging (RF) and the other one is based on the boosting approach (Adaboost).The best model has been selected on the basis the different performance evaluation metrics as discussed in Section 4.4.
Before employing the Ensemble learning model in the classification step of ABPET, the data must be divided into training and testing classes, and this information is used to assess the accuracy of model.

### 4.2.1 Selection of training & Testing Samples using Fold Generation

The fold generation approach is used to pick the training and testing data which is relatively new and widely used cross-validation technique in ML and have the following steps

- Shuffle the data at random and partition into to k groups.
- For each individual group.
- Consider the some group to be a test data set. All the other groups are for training set of data.
- On the training set, build the model, then test it on the test set. Analyze the model and, based on the evaluation score, eliminate it.
- Summarize the model's fitness using the sample model validation scores (Immaculate, Begam and Floramary, 2019).

On the defect prediction models, we utilized 10-fold cross-validation to train the dataset; the input dataset was subdivided into 10 equal-sized folds. Nine out of ten folds (90%) are assigned to the training corpus, with one fold set aside for testing (10 %). The testing sample is used to evaluate model performance while the training corpus is utilized to train models using various classification strategies. This method is performed 10 times, with each fold only acting as the testing corpus once. To confirm our findings, the10-fold process ten times (Ghotra, McIntosh and Hassan, 2015).

### 4.3 Ensemble Machine learning models

Random forests are one of the most successful ensemble algorithms, exhibiting boosting and support vector machine-like performance. The method is quick, noise-resistant, does not over fit, and allows for explanation and visualization of its results. We look into different strategies for increasing or decreasing the strength of particular forest trees' correlations (Robnik-Šikonja, 2004). Whereas, a powerful ensemble of classifiers is generated by successively refitting a weak classifier to different weighted realizations of a data set in the "boosting" technique to machine learning. This simple approach has shown to be extremely effective. (Wyner *et al.*, 2017).



### Random Forest

The random forest (RF) ensemble learning model (Breiman, 2001) is a set of ensembles classifiers (trees) and is widely used classification technique in various applications like remote sensing image classification (Belgiu and Drăgu, 2016), software bug detection (Singh *et al.*, 2014b), regression problems (Zhang *et al.*, 2021) etc. RF models have the following advantages over other classifiers:

- It operates efficiently on databases having an enormous amount of data.
- Can handle a large number of features and automatically extract prominent features.
- Efficient and robust against noise and outliers.
- Provides an unbiased estimate of generalization (OOB) error.
- Computationally efficient as compared to other tree-based methods (boosting).

A RF model is a combination of tree based classifiers, $\{H(p, \Theta_i), i=1. \}$ where $p$ is the input vector and $\{\Theta_i\}$ are separately and identically distributed random vectors. The components of model contribute with a vote

for the assignation of the majority recurrent group to the input vector ($p$), $^{B}$ *majority vote* $\{\hat{c}(p)\}^{B}$, where $\hat{c}(p)$ is the class predictor of the b$^{th}$ RF tree. RF model $cRFb1b$ having a large number of different trees encouraged chances of over fitting and outliers which are mitigated by the voting processing between various ensembles. During the generation of decision trees, the best spilt is selected randomly from the set of predictors and at each spilt a fresh sample of predictors is used. The unlabeled data is tested against all the predictors of the RF model and the class membership is decided by voting majority. To obtain the global optimum, one third of training samples are selected for the validation (OOB error) of the model and the rest are used in the training process. The generation of the RF ensemble model is described (Palczewska *et al.*, 2014) as,

1. Bootstrap phase: helps in selecting training datasets at random, resulting in a local training set
for the growing tree, with the remaining samples being utilized to estimate the OOB error..
2. The growing phase involves the best dividing of training samples using the CART method(Moisen, 2008)
3. Every tree is allowed to grow to its fullest capacity(Palczewska *et al.*, 2014) as there is no pruning which makes the RF ensemble learning model makes its computationally efficient .

The performance of the RF model depends on suitable feature selection technique which maximizes the randomness between various classes. The most common methods for feature selection in decision trees are gain-ratio (Quinlan, 1993), Gini-index(Breiman, 2001) and Chi-square (Mingers, 1989).The RF model $^{i}$ uses Gini-index for best split selection that measures the impurities of a given sample to the rest of the classes (Rodriguez-Galiano *et al.*, 2012). When developing an RF model, two important parameters to consider are the number of trees (Ntree) and the number of randomly picked features (Nfeatures) (Sheykhmousa *et al.*, 2020). The fewer Nfeature speed up computation but lowers the correlation between any two trees and affects the classification accuracy (Sahin, Colkesen and Kavzoglu, 2020). Ntree depends on the requirement of the specific domain but it has been shown that 500 trees are optimum (de Andrade *et al.*, 2020).

## AdaBoost

AdaBoost is an ensemble learning approach based on boosting i.e., combining weak and inaccurate rules for creating accurate prediction rules (Schapire, 2013). It has a strong theoretical foundation which is also known as the rule of thumb and this led to the introduction of a new method and design idea for learning algorithm creations(CAO *et al.*, 2013) by combining numerous weak learners whose accuracies are marginally superior to conventional guessing to create an arbitrarily accurate strong predictor (Saon and Soltau, 2012). AdaBoost and related evolution techniques are applied in a multiple of applications, including text classification (Kowsari *et al.*, 2019), text filtering (Carreras and Marquez, 2001) speech recognition(Park *et al.*, 2011; Saon and Soltau, 2012), human action recognition (Yan and Luo, 2012), object detection (Chen *et al.*, 2011) and remote sensing (McIver and Friedl, 2001). The boosting approach in machine learning is equivalent to logistic regression and is used to minimize the error of any "weak" learning algorithm.

AdaBoost is designed specifically for binary class problems and in the work presented in the paper deals with the detection of bugs which is a primarily belong to an above mentioned category and explained in Algorithm 1

## AdaBoost Algorithm

*(1)*Given a set of training samples $X(x_i, y_i)$ .....$(x_n, y_n)$ ,where $x^m_i \in \mathbb{R}$, $y_i \in \{0,1\}$ and initialize the weight vector as $w^1 = D(i), i = 1\ n$
Repeat step 2 to 6 for T iterations $t$=1.....$T$

(2)Set $$p^t = \frac{w^t}{\sum_{i=1}^{n} w_i^t}$$

(3)Obtain weak hypothesis ($h^t$) over $p$ $^t h_t = x \rightarrow [0,1]$

(4)Calculate the hypothesis error(E ) as,

$$E_t = \sum_{i=1} p_i^t \,|\, h_t(x_i) - y_i \,|$$

(5) Set $\alpha_t = \frac{E_t}{(1-E_t)}$

(6)Update weight vector as, $w^{t+1} = w^t . \alpha^{1-|h_t(x_i - y_i)|}$

(7)Finally, AdaBoost Model is developed as given below:

$$H(x)=\begin{cases}1 & if \ \sum_{t=1}^{T}\left(\log\frac{1}{\alpha_t}\right)h_t(x) \geq \frac{1}{2}\sum_{t=1}^{T}\left(\log\frac{1}{\alpha_t}\right)\\ 0 & otherwise\end{cases}$$

The bagging based techniques are based on the random selection of rules while in the boosting approach weak rules based on the rule of thumb are used. It has been observed that the classification performance of ensembles learners based on the weak rule is better as compared to random. This improvement in the performance of boosting approach is due to the reduced training error as compared to bagging based techniques which can be expressed as,

$$f(x)=\sum_t \alpha_t h_t(x) \quad (3)$$

## 4.4 Performance Measure and comparative analysis

This section contains the quantities parameters used to assess the proposed method's effectiveness of ensemble learning for bug prediction and the comparison between them.

### 4.4.1 Performance measures

The proposed model's ABPET performance is based on the ensemble learning technique is measured using different parameters obtained from error metrics like precision, recall, receiver operating characteristic curve (ROC)(Robin *et al.*, 2011), area under the ROC curve (AUC), *F-measure*(Luque *et al.*, 2019) and Mathews co-relation Coefficient (MCC) (Matthews, 1975). Precision is a measure that can be determined as the number of correct positive predictions made separated as the number of correct positive predictions made.

$$Precision=\frac{TP}{TP+FP} \quad (4)$$

Where TP stands for True positive and FP stands for false positive respectively. The recall is ameasure of how many right positive predictions were made out of all possible positive predictions and can be calculated as,

$$Recall=\frac{TP}{TP+FN} \quad (5)$$

The total number of successfully identified bugs divided by the total number of bugs is the definition of accuracy, which is calculated as,

$$Accuracy=\frac{TP+TN}{TP+TN+FP+FN} \quad (6)$$

The performance of a classification model is depicted by a ROC curve, which shows true positive rate (TPR) vs. false positive rate (FPR) at various classification thresholds. As the categorization threshold is lowered, more objects are classified as positive, increasing both FP and TP. The AUC is classification-threshold-invariant, meaning it shows how well predictions are ordered rather than their absolute values, and it assesses the quality of the model's predictions regardless of the classification threshold used.

The F-measure is calculated by weighing precision and recall equally in the harmonic mean. Itallows a model to also be evaluated using a specific value that provides for both precision and recall, which would be valuable for comparing models and discussing their performance.

$$F-Measure=\frac{(2*Precision*Recall)}{(Precision+Recall)} \quad (7)$$



MCC is also an ensemble estimation methodology that takes into account all TP and FP, as well as all TN and FN. The MCC ranges from -1 to 1, with higher values indicating greater efficiency in predicting asymmetric datasets. MCC is calculated as,

$$MCC=\frac{TP*TN-FP*FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}} \quad (8)$$

### 4.4.2 Comparative analysis

The performance of ABPET is discussed in detail on different performance on different quantitativeparameters as discussed in section 4.4.1.The comparative analysis of precision, recall and accuracy is presented in Table 3. It is being observed that when the number of defects are more in a dataset then the accuracy of AdaBoost is better as compared to the random forest as shown in Table 3 databases like Lucene 2.0,velocity 1.6, log4j1.1, synpose1.2, jedit3.2 etc. In case, when the number of the defective sample is less in a dataset the performance of AdaBoost on

accuracy is marginally less as compared to the random forest as shown in Fig. 2.

**Table 3:** Comparative analysis on Precision, Recall and Accuracy.

| Dataset | Precision | | Recall | | Accuracy | |
|---|---|---|---|---|---|---|
| | RF | AdaBoost | RF | AdaBoost | RF | AdaBoost |
| Ant-1.7 | 0.853896104 | 0.844301766 | 0.908462867 | 0.908462867 | 0.808053691 | 0.798657718 |
| camel-1.0 | 0.961424332 | 0.961309524 | 0.993865031 | 0.990797546 | 0.955752212 | 0.95280236 |
| camel-1.6 | 0.832402235 | 0.828698554 | 0.958815959 | 0.958815959 | 0.811398964 | 0.807253886 |
| data_arc | 0.88317757 | 0.888888889 | 0.964285714 | 0.979591837 | 0.857777778 | 0.875555556 |
| data_ivy-2.0 | 0.907185629 | 0.90504451 | 0.971153846 | 0.977564103 | 0.886363636 | 0.889204545 |
| data_prop-6 | 0.937394247 | 0.93812709 | 0.95025729 | 0.962264151 | 0.897515528 | 0.908385093 |
| data_redaktor | 0.911392405 | 0.910828025 | 0.972972973 | 0.966216216 | 0.897142857 | 0.891428571 |
| jedit-3.2 | 0.811881188 | 0.815 | 0.901098901 | 0.895604396 | 0.794117647 | 0.794117647 |
| jedit-4.2 | 0.891117479 | 0.903225806 | 0.97492163 | 0.965517241 | 0.874659401 | 0.880108992 |
| log4j-1.1 | 0.794871795 | 0.805194805 | 0.861111111 | 0.861111111 | 0.76146789 | 0.770642202 |
| lucene-2.0 | 0.700934579 | 0.696428571 | 0.721153846 | 0.75 | 0.687179487 | 0.692307692 |
| poi-2.0 | 0.905723906 | 0.897009967 | 0.971119134 | 0.974729242 | 0.885350318 | 0.878980892 |
| synapse-1.0 | 0.986111111 | 0.986111111 | 1 | 1 | 0.987261146 | 0.987261146 |
| synapse-1.2 | 0.798913043 | 0.801075269 | 0.864705882 | 0.876470588 | 0.765625 | 0.7734375 |
| velocity-1.6 | 0.79245283 | 0.804878049 | 0.834437086 | 0.874172185 | 0.746724891 | 0.777292576 |
| xalan-2.4 | 0.867540029 | 0.870072993 | 0.972267537 | 0.972267537 | 0.850622407 | 0.853388658 |
| xerces-1.3 | 0.898795181 | 0.886524823 | 0.971354167 | 0.9765625 | 0.883002208 | 0.874172185 |

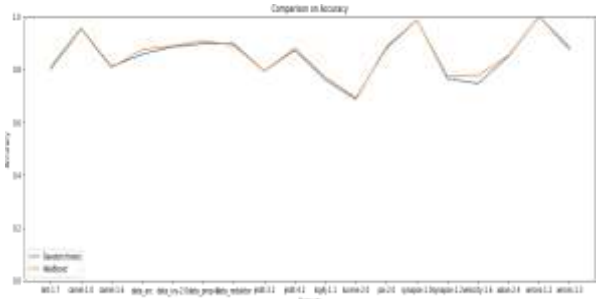


**Figure 2 :** Comparative analysis on Accuracy.

In terms of F Measure, it depicts taking into account both precision and recall using a single score to represent the model's performance. The performance of AdaBoost is better as compared to the random forest in terms of *F_Measure* when the defective % there'll be more as described in Table 4having datasets like log4j-1.1, lucene-2.0, synapse-1.2, velocity-1.6 etc. The comparative analysis of both ensemble learning techniques on *F_Measure* on whole bug datasets of promise repository is shown in Fig 3.

**Table 4:** Dataset comparative analysis on *F_Measure*, MCC and AUC.

| Dataset | F_Measure | | MCC | | AUC | |
|---|---|---|---|---|---|---|
| | RF | AdaBoost | RF | AdaBoost | RF | AdaBoost |
| Ant-1.7 | 0.880334728 | 0.875207987 | 0.402845158 | 0.364487476 | 0.802260857 | 0.795009052 |
| camel-1.0 | 0.977375566 | 0.975830816 | -0.015383775 | -0.01886922 | 0.688414346 | 0.704695611 |
| camel-1.6 | 0.891148325 | 0.88902148 | 0.245755167 | 0.219144964 | 0.709161669 | 0.706929954 |
| data_arc | 0.92195122 | 0.932038835 | 0.158834876 | 0.259919329 | 0.662737509 | 0.680682618 |
| data_ivy-2.0 | 0.938080495 | 0.93990755 | 0.282614627 | 0.27899851 | 0.800761218 | 0.785697115 |
| data_prop-6 | 0.943781942 | 0.950042337 | 0.366224118 | 0.40444719 | 0.760270506 | 0.75127239 |
| data_redaktor | 0.941176471 | 0.937704918 | 0.554306665 | 0.532367484 | 0.768143143 | 0.787537538 |
| jedit-3.2 | 0.854166667 | 0.853403141 | 0.515413668 | 0.516729748 | 0.8495116 | 0.845970696 |
| jedit-4.2 | 0.931137725 | 0.933333333 | 0.286103217 | 0.365362379 | 0.808842738 | 0.814524556 |
| log4j-1.1 | 0.826666667 | 0.832214765 | 0.449956827 | 0.47383946 | 0.798611111 | 0.801989489 |
| lucene-2.0 | 0.710900474 | 0.722222222 | 0.370446112 | 0.379759364 | 0.732882502 | 0.735259932 |
| poi-2.0 | 0.93728223 | 0.934256055 | 0.305412371 | 0.221545024 | 0.820031223 | 0.761440141 |
| synapse-1.0 | 0.993006993 | 0.993006993 | 0.924461805 | 0.924461805 | 0.9990161033 | 1 |
| synapse-1.2 | 0.830508475 | 0.837078652 | 0.456416823 | 0.472862391 | 0.833510534 | 0.828077975 |
| velocity-1.6 | 0.812903226 | 0.838095238 | 0.423165667 | 0.487636621 | 0.8040414 33 | 0.810833758 |
| xalan-2.4 | 0.916923077 | 0.918335901 | 0.23941393 | 0.262631884 | 0.815267685 | 0.818760196 |
| xerces-1.3 | 0.933667084 | 0.92936803 | 0.470080449 | 0.405907476 | 0.853128774 | 0.836371528 |

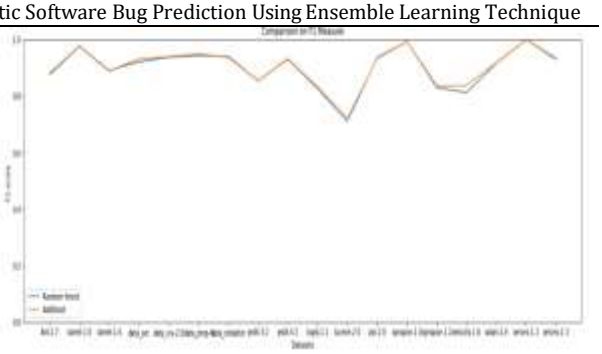


**Figure 3:** Comparative analysis on F-Measure.

Whereas, MCC is considered to be the best performance measure for binary setting class prediction for imbalance class. AdaBoost comes out to be the better algorithm as compared to the random forest as shown in Table 4 in a different dataset like log4j-1.1, lucene-2.0, synapse-1.2, velocity-1.6 etc. The comparative analysis of both ensembles learning techniques on MCC on whole bug datasets of promise repository is shown in Fig. 4.

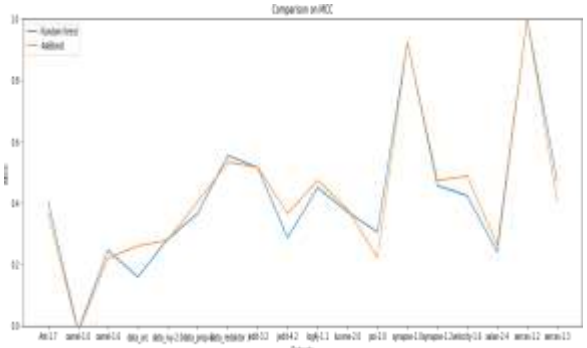


**Figure 4:** Comparative analysis on MCC.

Whereas, AUC is the performance measure whose ability is to classify and distinguish between classes and also gives the summary of the ROC curve. So after using AUC it is being found that AdaBoost is comes out to be the better algorithm as compared to the random forest as shown in Table4 and Fig 5.

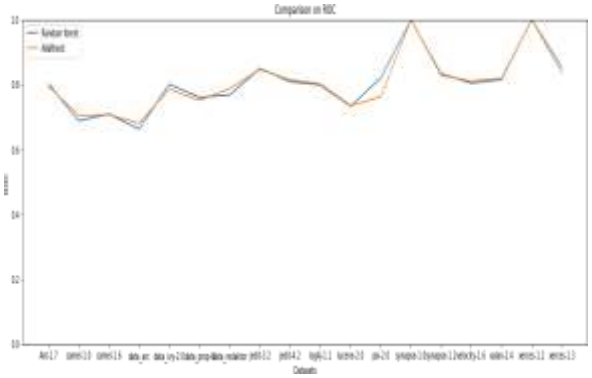


**Figure 5**: Comparative analysis on ROC.

## 5. Conclusion

In this paper, a multi-stage stage automatic software bug prediction is proposed using ensemble machine learning techniques. The proposed framework is consisting of three stages, in the first stage; the pre-processing of data is performed for eliminating the issues of imbalanced data, missing data etc. In the second phase, the two ensemble machine learning models based on the bagging (random forest) and boosting (AdaBoost) approach are used to predict the bugs. While the training and testing samples are selected using 10-fold cross-validation. In the third phase, the performance of two ensemble learning models has been evaluated based on different quantitative parameters like F-measure, MCC, AUC etc. The well-known promise repository bug dataset has been used to perform the experiments. After the extensive experimental results, it has been observed that the performance of AdaBoost is better as compared to the random forest when the detective percentage is higher in terms of accuracy, F-measure, MCC and AUC. While in the case of the dataset having less defective percentage, the performance of both classifiers is almost identical. In future work, instead of selecting classifiers after observing the performance the best classifiers for bug detection can be dynamically selected and a deep learning-based architecture can be designed to improve the prediction performance.

## References

de Andrade, B. M. et al. (2020) „Comparison of the performance of multiclass classifiers in chemical data: Addressing the problem of overfitting with the permutation test", Chemometrics and Intelligent Laboratory Systems, 201, p. 104013. doi: 10.1016/j.chemolab.2020.104013.

Arar, Ö. F. and Ayan, K. (2015) „Software defect prediction using cost-sensitive neural network",
Applied Soft Computing. Elsevier, 33, pp. 263–277. doi: 10.1016/J.ASOC.2015.04.045.

Ball, S. J., Reay, D. and David, M. (2002) „"Ethnic choosing": Minority ethnic students, social class and higher education choice", Race Ethnicity and Education. Routledge, 5(4), pp. 333–357.
doi: 10.1080/1361332022000030879.

Belgiu, M. and Drăgu, L. (2016) „Random forest in remote sensing: A review of applications and future directions", ISPRS Journal of Photogrammetry and Remote Sensing. Elsevier, pp. 24–31. doi: 10.1016/j.isprsjprs.2016.01.011.

Breiman, L. (2001) „Random forests", Machine Learning, 45(1), pp. 5–32. doi:10.1023/A:1010933404324.

CAO, Y. et al. (2013) „Advance and Prospects of AdaBoost Algorithm", Acta Automatica Sinica. The Chinese Association of Automation and The Institute of Automation, Chinese Academy of Sciences, 39(6), pp. 745–758. doi: 10.1016/s1874-1029(13)60052-x.

Carbonell, J. G., Michalski, R. S. and Mitchell, T. M. (1983) „an Overview of Machine Learning", in Michalski, R. S., Carbonell, J. G., and Mitchell, T. M. B. T.-M. L. (eds) Machine Learning. San Francisco (CA): Morgan Kaufmann, pp. 3–23. doi: 10.1016/b978-0-08-051054-5.50005-4.

Carreras, X. and Marquez, L. (2001) „Boosting Trees for Anti-Spam Email Filtering", arXiv preprint cs/0109015. Available at:
http://arxiv.org/abs/cs/0109015.

Chen, S. et al. (2011) „Boosting part-sense multi-feature learners toward effective object detection", Computer Vision and Image Understanding, 115(3), pp. 364–374. doi: 10.1016/j.cviu.2010.11.006. Chen, X. et al. (2019) „Software defect number prediction: Unsupervised vs supervised methods", Information and Software Technology, 106, pp. 161–181. doi: 10.1016/j.infsof.2018.10.003.

Chidamber, S. R. and Kemerer, C. F. (1994) „A Metrics Suite for Object Oriented Design", IEEE Transactions on Software Engineering, 20(6), pp. 476–493. doi: 10.1109/32.295895.

Feng, S. et al. (2021) „Investigation on the stability of SMOTE-based oversampling techniques in software defect prediction", Information and Software Technology. Elsevier, 139, p. 106662. doi: 10.1016/j.infsof.2021.106662.

Gertler, J. J. (1988) „Survey of Model-Based Failure Detection and Isolation in Complex Plants", IEEE Control Systems Magazine, 8(6), pp. 3–11. doi: 10.1109/37.9163.

Ghotra, B., McIntosh, S. and Hassan, A. E. (2015) „Revisiting the impact of classification techniques on the performance of defect prediction models", in Proceedings - International Conference on Software Engineering, pp. 789–800.
doi: 10.1109/ICSE.2015.91.

Goyal, S. (2022) „Handling Class-Imbalance with KNN (Neighbourhood) Under-Sampling for Software Defect Prediction", Artificial Intelligence Review, 55(3), pp. 2023–2064. doi: 10.1007/s10462-021- 10044-w.

Guo, J., Cheng, J. and Cleland-Huang, J. (2017) „Semantically Enhanced Software Traceability Using Deep Learning Techniques", in 2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE), pp. 3–14. doi: 10.1109/ICSE.2017.9.

Halstead, M. H. (1977) Elements of software science library., Operating and programming systems series. Elsevier Science Inc.

Huo, X., Li, M. and Zhou, Z. H. (2016) „Learning unified features from natural and programming languages for locating buggy source code", in IJCAI International Joint Conference on Artificial Intelligence, pp. 1606–1612.

Immaculate, S. D., Begam, M. F. and Floramary, M. (2019) „Software Bug Prediction Using Supervised Machine Learning Algorithms", in 2019 International Conference on Data Science and Communication, IconDSC 2019, pp. 1–7.
doi: 10.1109/IconDSC.2019.8816965.

Jing, X.-Y. et al. (2014) „Dictionary Learning Based Software Defect Prediction", in Proceedings of the36th International Conference on Software Engineering.





New York, NY, USA: Association for Computing Machinery (ICSE 2014), pp. 414–423. doi: 10.1145/2568225.2568320.

Kanwal, J. and Maqbool, O. (2012) „Bug Prioritization to Facilitate Bug Report Triage", Journal of Computer Science and Technology, 27(2), pp. 397–412. doi: 10.1007/s11390-012-1230-3.

Khuat, T. T. and Le, M. H. (2019) „Ensemble learning for software fault prediction problem with imbalanced data", International Journal of Electrical and Computer Engineering, 9(4), pp. 3241–3246. doi: 10.11591/ijece.v9i4.pp3241-3246.

Klosowski, P. (2018) „Deep learning for natural language processing and language modelling", in Signal Processing - Algorithms, Architectures, Arrangements, and Applications Conference Proceedings, SPA, pp. 223–228.
doi: 10.23919/SPA.2018.8563389.

Knab, P., Pinzger, M. and Bernstein, A. (2006) „Predicting Defect Densities in Source Code Files with Decision Tree Learners", in Proceedings of the 2006 International Workshop on Mining Software Repositories. New York, NY, USA: Association for Computing Machinery (MSR '06), pp. 119–125. doi: 10.1145/1137983.1138012.

Kowsari, K. et al. (2019) „Text classification algorithms: A survey", Information (Switzerland).
doi:10.3390/info10040150.

Li, B. et al. (2014) „A Scenario-Based Approach to Predicting Software Defects Using Compressed C4.5 Model", in 2014 IEEE 38th Annual Computer Software and Applications Conference, pp. 406– 415. doi: 10.1109/COMPSAC.2014.64.

Liblit, B. et al. (2003) „Bug isolation via remote program sampling", ACM SIGPLAN Notices. NewYork, NY, USA: Association for Computing Machinery, 38(5), pp. 141–154. doi: 10.1145/780822.781148.

Liblit, B. et al. (2005) „Scalable Statistical bug isolation", ACM SIGPLAN Notices. New York, NY,USA: Association for Computing Machinery, 40(6), pp. 15–26. doi: 10.1145/1064978.1065014.

Lindley, D. V. and Phillips, L. D. (1976) „Inference for a bernoulli process (A bayesian view)", American Statistician. Taylor & Francis, 30(3), pp. 112–119. doi: 10.1080/00031305.1976.10479154. Liu, C. et al. (2005) „Sober", in Proceedings of the 10th European Software Engineering Conference Held Jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering. New York, NY, USA: Association for Computing Machinery (ESEC/FSE-13), p. 286. doi: 10.1145/1081706.1081753.

Liu, F., Lin, G. and Shen, C. (2015) „CRF learning with CNN features for image segmentation", PatternRecognition, 48(10), pp. 2983–2992. doi: 10.1016/j.patcog.2015.04.019.

Luque, A. et al. (2019) „The impact of class imbalance in classification performance metrics based on the binary confusion matrix", Pattern Recognition, 91, pp. 216–231. doi: 10.1016/j.patcog.2019.02.023. Ma, S. et al. (2018) „MODE: Automated neural network model debugging via state differential analysis and input selection", in ESEC/FSE 2018 - Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. New York, NY, USA: Association for Computing Machinery (ESEC/FSE 2018), pp. 175– 186. doi: 10.1145/3236024.3236082.

Mao, X. et al. (2019) „Extractive summarization using supervised and unsupervised learning", Expert Systems with Applications, 133, pp. 173–181. doi: 10.1016/j.eswa.2019.05.011.

Matthews, B. W. (1975) „Comparison of the predicted and observed secondary structure of T4 phage lysozyme", BBA - Protein Structure, 405(2), pp. 442–451. doi: 10.1016/0005-2795(75)90109-9. McCabe, T. J. (1976) „A Complexity Measure", IEEE Transactions on Software Engineering, SE-2(4), pp. 308–320. doi: 10.1109/TSE.1976.233837.

McIver, D. K. and Friedl, M. A. (2001) „Estimating pixel-scale land cover classification confidence using nonparametric machine learning methods", IEEE Transactions on Geoscience and Remote Sensing, 39(9), pp. 1959–1968. doi: 10.1109/36.951086.

Meiliana, M. et al. (2018) „Software metrics for fault prediction using machine learning approaches: A literature review with PROMISE repository dataset", in 2017 IEEE International Conference on Cybernetics and Computational Intelligence, CyberneticsCOM 2017 - Proceedings, pp. 19–23. doi: 10.1109/CYBERNETICSCOM.2017.8311708.

Mingers, J. (1989) „An Empirical Comparison of Pruning Methods for Decision Tree Induction",
Machine Learning, 4(2), pp. 227–243. doi: 10.1023/A:1022604100933.

Moisen, G. G. (2008) Classification and Regression Trees, Encyclopedia of Ecology, Five-Volume Set. Routledge. doi: 10.1016/B978-008045405-4.00149-X.

Moosavian, A. et al. (2014) „Support vector machine and K-nearest neighbour for unbalanced fault detection", Journal of Quality in Maintenance Engineering. Emerald Group Publishing Limited, 20(1), pp. 65–75. doi: 10.1108/JQME-04-2012-0016.

Noda, K. et al. (2015) „Audio-visual speech recognition using deep learning", Applied Intelligence, 42(4), pp. 722–737. doi: 10.1007/s10489-014-0629-7.

Palczewska, A. et al. (2014) „Interpreting random forest classification models using a feature contribution method", in Bouabana-Tebibel, T. and Rubin, S. H. (eds) Advances in Intelligent Systems and Computing. Cham: Springer International Publishing, pp. 193–218. doi: 10.1007/978-3-319-04717-1_9.

Pandey, S. K. and Tripathi, A. K. (2020) „BCV-Predictor: A bug count vector predictor of a successive version of the software system", Knowledge-Based Systems, 197, p. 105924. doi: 10.1016/j.knosys.2020.105924.

Park, J. et al. (2011) „The efficient incorporation of MLP features into automatic speech recognition systems", Computer Speech & Language, 25(3), pp. 519–534. doi: https://doi.org/10.1016/j.csl.2010.07.005.

Qayyum, A. et al. (2017) „Medical image retrieval using deep convolutional neural network",
Neurocomputing, 266, pp. 8–20. doi: 10.1016/j.neucom.2017.05.025. Quinlan, J. R. (1993) {C4}.5 - Programs for Machine Learning. Elsevier.

Robin, X. et al. (2011) „pROC: An open-source package for R and S+ to analyze and compare ROC curves", BMC Bioinformatics, 12(1), p. 77. doi: 10.1186/1471-2105-12-77.

Robnik-Šikonja, M. (2004) „Improving Random Forests BT - Machine Learning: ECML 2004", inBoulicaut, J.-F. et al. (eds). Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 359–370.





Rodriguez-Galiano, V. F. et al. (2012) „An assessment of the effectiveness of a random forest classifierfor land-cover classification", ISPRS Journal of Photogram-metry and Remote Sensing, 67(1), pp. 93– 104. doi: 10.1016/j.isprsjprs.2011.11.002.

Sahin, E. K., Colkesen, I. and Kavzoglu, T. (2020) „A comparative assessment of canonical correlation forest, random forest, rotation forest and logistic regression methods for landslide susceptibility mapping", Geocarto International. Taylor & Francis, 35(4), pp. 341–363. doi: 10.1080/10106049.2018.1516248.

Saon, G. and Soltau, H. (2012) „Boosting systems for large vocabulary continuous speech recognition",

Speech Communication, 54(2), pp. 212–218. doi: 10.1016/j.specom.2011.07.011.

Schapire, R. E. (2013) „Explaining adaboost", in Schölkopf, B., Luo, Z., and Vovk, V. (eds) Empirical Inference: Festschrift in Honor of Vladimir N. Vapnik. Berlin, Heidelberg: Springer Berlin Heidelberg,pp. 37–52. doi: 10.1007/978-3-642-41136-6_5.

Sharma, A. et al. (2017) „A patch-based convolutional neural network for remote sensing image classification", Neural Networks, 95, pp. 19–28. doi: 10.1016/j.neunet.2017.07.017.

Sharma, G., Sharma, S. and Gujral, S. (2015) „A Novel Way of Assessing Software Bug Severity UsingDictionary of Critical Terms", Procedia Computer Science. Elsevier, 70, pp. 632–639. doi: 10.1016/J.PROCS.2015.10.059.

Sheykhmousa, M. et al. (2020) „Support Vector Machine Versus Random Forest for Remote Sensing Image Classification: A Meta-Analysis and Systematic Review", IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, 13, pp. 6308–6325. doi: 10.1109/JSTARS.2020.3026724.

Singh, K. et al. (2014a) „Big Data Analytics framework for Peer-to-Peer Botnet detection using Random Forests", Information Sciences. Elsevier, 278, pp. 488–497. doi: 10.1016/J.INS.2014.03.066. Singh, K. et al. (2014b) „Big Data Analytics framework for Peer-to-Peer Botnet detection using Random Forests", Information Sciences. Elsevier, 278, pp. 488–497. doi: 10.1016/j.ins.2014.03.066. Soleimani, A. and Asdaghi, F. (2014) „An AIS based feature selection method for software fault prediction", in 2014 Iranian Conference on Intelligent Systems, ICIS 2014, pp. 1–5. doi: 10.1109/IranianCIS.2014.6802598.

Tang, H. et al. (2015) „Enhancing Defect Prediction with Static Defect Analysis", in Proceedings of the 7th Asia-Pacific Symposium on Internetware. New York, NY, USA: Association for Computing Machinery (Internetware '15), pp. 43–51. doi: 10.1145/2875913.2875922.

Tang, M.-H., Kao, M.-H. and Chen, M.-H. (1999) „An empirical study on object-oriented metrics", in Proceedings Sixth International Software Metrics Symposium (Cat. No.PR00403), pp. 242–249. doi: 10.1109/METRIC.1999.809745.

Tong, H., Liu, B. and Wang, S. (2018) „Software defect prediction using stacked denoising autoencoders and two-stage ensemble learning", Information and Software Technology, 96, pp. 94–111. doi: 10.1016/j.infsof.2017.11.008.

Wang, K. et al. (2021) „Software defect prediction model based on LASSO–SVM", Neural Computing and Applications, 33(14), pp. 8249–8259. doi: 10.1007/s00521-020-04960-1.

Wang, S. et al. (2016) „Bugram: Bug detection with N-gram language models", in ASE 2016 - Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering. NewYork, NY, USA: Association for Computing Machinery (ASE 2016), pp. 708–719. doi: 10.1145/2970276.2970341.

Wyner, A. J. et al. (2017) „Explaining the success of adaboost and random forests as interpolating classifiers", Journal of Machine Learning Research. JMLR. org, 18(1), pp. 1–33.

Yadav, H. B. and Yadav, D. K. (2015) „A fuzzy logic based approach for phase-wise software defects prediction using software metrics", Information and Software Technology. Elsevier, 63, pp. 44–57. doi: 10.1016/j.infsof.2015.03.001.

Yan, X. and Luo, Y. (2012) „Recognizing human actions using a new descriptor based on spatial- temporal interest points and weighted-output classifier", Neurocomputing, 87, pp. 51–61. doi: 10.1016/j.neucom.2012.02.002.

Zhang, W. et al. (2021) „Assessment of pile drivability using random forest regression and multivariate adaptive regression splines", Georisk: Assessment and Management of Risk for Engineered Systems and Geohazards. Taylor & Francis, 15(1), pp. 27–40. doi: 10.1080/17499518.2019.1674340.

Zhang, W., Wang, S. and Wang, Q. (2016) „KSAP: An approach to bug report assignment using KNN search and heterogeneous proximity", Information and Software Technology. Elsevier, 70, pp. 68–84. doi: 10.1016/j.infsof.2015.10.004.

Zhao, G. and Huang, J. (2018) „DeepSim: Deep learning code functional similarity", in ESEC/FSE 2018

- Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. New York, NY, USA: Association for Computing Machinery (ESEC/FSE 2018), pp. 141–151. doi: 10.1145/3236024.3236068.

Zhu, K. et al. (2021) „Software defect prediction based on enhanced metaheuristic feature selection optimization and a hybrid deep neural network", Journal of Systems and Software, 180, p. 111026. doi: 10.1016/j.jss.2021.111026.



