Software Defect Prediction

Yue Ma
Computer Science Department
Northern Illinois University
DeKalb, IL, U.S.A
Z1934458@students.niu.edu

Nida Zaki
Computer Science Department
Northern Illinois University
DeKalb, IL, U.S.A
Z1938561@students.niu.edu

***Abstract*—Software Defects have always been considered a major problem in the software industry and for software engineers, early detection improves software performance and reduces faults, time and cost. The primary goal of this research is to assess the effect of Object-Oriented metrics on defect prediction performance using various classification and ensemble techniques. Classifier models have successfully been applied to the task of detecting and predicting defects in software projects. These models work by learning patterns between attributes of the software and a corresponding binary label that indicates whether or not a defect exists. In this study, the dataset is the combination of 18 individual datasets from the PROMISE Repository with more than 20 software complexity metrics. We apply various classifiers in the context of software defect prediction– Naïve Bayes, Neural Networks, Support Vector Machines, k-Nearest Neighbor, Random Forest, Decision Tree and Ensemble Learning. Imbalanced data treatment is also considered by implementing Random Under Sampling, Naive Bayes classifiers, ensemble methods and 5-fold cross validation. There are multiple evaluation indicators used to measure the classifier performance such as accuracy, precision, recall, F-1 score, MAE, AUC and ROC. The analysis of prediction performance evaluation determined whether different classification and ensemble techniques have significantly different predictive performances. The performance of detecting software faults is much better when using ensemble learning, specially Gradient Boosting.**

***Index Terms*—Classification techniques, feature selection, imbalanced data, ensemble learning, PROMISE, software defect prediction**

I. Introduction

Software defects are errors or bugs, which are detected or are possible to violate the security policies during the runtime of software applications [30]. It can cause unexpected behaviors, which will lead to either small mistakes(e.g. a login button does not work) or serious issues (e.g. Ariane 5 rocket ignited its engine after launching due to the software bug in the rocket's Inertial Reference System [29]). In hence, to avoid these cases, predicting and anticipating the defects in advance is an urgent mission. It can help to raise the awareness of developers for the safety issues, and reduce financial loss or even reduce the loss of lives. Software defect prediction is a task which uses various classifiers to predict whether certain modules or code areas in a software will loss its function or not [25]. For the classifier selection, machine learning which is a sub-field of artificial intelligence [31] provides powerful supervised classification methods. These methods are incorporated to many software designs across various fields(e.g. self driving cars [32], Medical diagnostic systems [33], Agriculture smart farming [34]). Inspired by this, we are deciding to apply machine learning classification methods on software defect prediction tasks.

However, it is challenging to apply supervised machine learning on this task due to the lack of large labeled data or even the data is common to be imbalanced [38], and the complexity of the context [25]. And previous work only tested with limited models, most of researchers chose ensemble learning([35], [36], [37]) considering about the imbalanced status, or with other limited classifier selections(e.g. regression methods [39], k nearest neighbors [40], support vector machine [41] [42]).

We will contribute to a more comprehensive comparisons which include most of the popular classification machine learning methods, as well as artificial neural networks. And discuss and compare the classifier performance with various indicators. There are also 4 methods which are used to deal with imbalanced dataset are discussed, which provides discussion and reference for future works.

Research questions which we aim to answer in this paper are as follows:

> RQ1: Whether the choice of a classifier makes a difference when attempting to perform defect detection and prediction in software systems?
>
> RQ2: Which method has the best performance dealing with imbalanced dataset?
>
> RQ3: Which software complexity metrics contributes most for the defect prediction tasks?

The paper is organized as follows: Section II addresses the limitation of the current dataset. In Section III, we elaborate the approach and the process of implementing the models. Section IV summarizes a Systematic Literature review over related work. The results of the models are described in Section V. Conclusions which include the summary answer for raised research questions are provided in Section VI.

II. Limitation

The dataset we used contains more records which do not have bugs than the records which have bugs. To overcome this limitation from the imbalanced dataset, we used the following methods to improve the classification accuracy and to provide a more convincing results.

Fig. 1. A typical Software Defect Prediction Process using Classification Methods

- Naive Bayes classifier
- Ensemble methods
- Cross validation
- Under-sampling

Naive Bayes Classifier, ensemble methods, and cross validation will be discussed in the methodology section. Under-sampling is a methods that simply cut the size of the records to make a balanced dataset. It can intuitively make the label balanced, while it will also have the disadvantages that the total size of the data is getting smaller. Other methods for dealing with imbalanced dataset like oversampling, which collects more data contains the records 'have bugs', to enlarge the dataset and to make an equal size of these binary labels, due to the time limit, and the difficulty of collecting same quality data for a given dataset, we did not do so. This will be our future work.

III. Methodology

In this paper, research is carried out using an essential approach for data mining, which is the Supervised Learning. The process of Software Defect Prediction using Classifiers is shown in the Figure 1 and is explained in the subsequent sections.

A. Data collection

The datasets used for this experiment is publicly available by the PROMISE Software Engineering Repository [2]. In order to create repeatable, disputable, and verifiable software engineering models, the PROMISE repository compiles a variety of open-source datasets. It was inspired by the popular UCI Machine Learning Repository for researchers working in the field of machine learning [52]. This repository provides the metrics that define the software artifacts from various software projects. It contains 18 individual datasets, for defect detection, where each datapoint represents a snapshot of code. The attributes for each datapoint are intended to provide an indication of whether or not the code is defective. Some of the software quality metrics that are included in the dataset are displayed in Table I.

The target/output class is the dependent attribute and the remaining attributes are known as independent attributes. The dependent attribute is predicted on the basis of independent attributes. These attributes are the quality metrics of software systems.All the independent variables except name(brief version), version, name(detailed version) are passed to the classifier models.

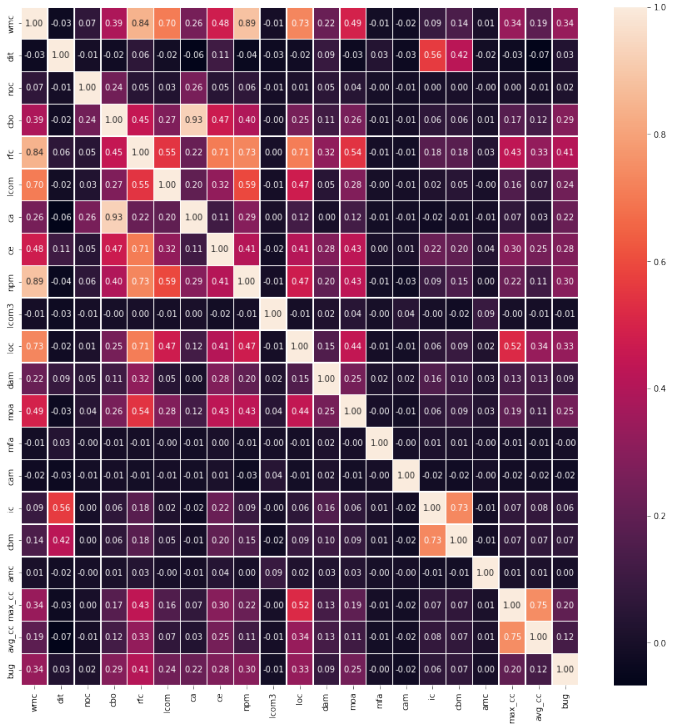

Fig. 2. Feature correlation

B. Data Preprocessing

We combined 18 individual datasets into a single dataset by the attribute names, making it convenient for further processing of the data.

The dependent variable, bug, is the target class in the dataset which indicates the number of bugs that occur during an instance. We categorized the attribute 'bug' based on the value of that attribute in the dataset, into two categories: "1" or "0". "1" means that the particular instance (module) is defective and "0" means it is non-defective. When the 'bug' attribute in the dataset has the value greater than or equal to 1 , we categorized it as "1" and when the 'bug' attribute in the dataset has the value 0, we categorized it as "0".

There are some missing values for the name(brief), version, and name(detailed) attributes, but not for the independent attributes that will be passed to the model. Since these attributes are of no importance to us, this will not have an influence on the overall result obtained. The data type for all the independent attributes i.e. the software quality metrics is either a float type or an integer type.

For Feature Selection, we implemented Hybrid Feature Selection method which is used to automate the process of finding the minimal and optimal set of features/attributes which reduces the computational cost and time complexity of the prediction model and increases the accuracy of the model. Correlation matrix is found for the given dataset (except for name and version attributes) as shown in the Figure 2. Correlation refers to the strength of the relationship between two attributes. As we can see from the figure, the

TABLE I
SUMMARY OF METRICS IN THE DATASET

Metric Name	Description	Metric Name	Description
wmc: weighted methods per class	It is weighted the number of methods in the class.	**moa: measure of aggregation**	The metric measures the extent of the part-whole relationship, realized by using attributes.
dit: depth of inheritance tree	It measures the inheritance levels from the object hierarchy top.	**mfa: measure of functional abstraction**	The ration of the number of methods inherited by a class to the total number of methods accessible by the member methods of the class.
noc: number of children	It measures the number of immediate descendants.	**cam: cohesion among methods**	It computes the relatedness among methods of a class based on the parameter list of methods.
cbo: coupling between object classes	It represents the number of classes coupled to a given class.	**ce: inheritance coupling**	It provides the number of parent classes to which a given class is coupled.
rfc: response for a class	It measures the number of different methods that can be executed when an object of that class receives a message.	**cbm: coupling between methods**	It measures the total number of new or redefined methods to which all the inherited methods are coupled.
locm: lack of cohesion in methods	It counts the sets of methods in a class that are not related through the sharing of some of the class fields.	**amc: average method complexity**	It measures the average method size for each class.
ca: afferent couplings	This represents the number of classes that depend upon the measured class.	**max_cc: maximum cyclomatic complexity**	This metric measures the maximum number of test cases needed to achieve full branch coverage.
ce: efferent couplings	It represents the number of classes that the measured class is depended upon.	**avg_cc: average cyclomatic complexity**	This metric measures the average number of test cases needed to achieve full branch coverage.
npm: number of public methods	The npm metric counts all the methods in a class that are declared as public.	**loc: lines of code**	It measures the size of computer program in the source code.
com3: lack of cohesion in methods	locm3 is the normalized version of the lcom metric.	**bug: number of bugs**	This metric is used as label for software defect prediction
dam: data access metric	It is the ratio of the number of private attributes to the total number of attributes declared in the class.		

correlation between the attributes is very low (indicated by the dark color in the heat map), meaning that the attributes/ features are hardly related. Features with low correlation are less linearly dependent and hence they contribute individually to the prediction model. From the figure, we check for the features with the correlation value greater than 0.95 with respect to other features, in order to remove those attributes from the dataset. As there are no features/attributes with the correlation value greater than 0.95, it means that all the attributes individually contribute to the prediction model, thus including all the independent variables in the dataset. This entire step is done to ensure that no two features have similar characteristics, as it would be unnecessary to include both of those attributes in the dataset and pass it to the prediction model as it might increase the time complexity of the model.

80% of the data is used for training and the remaining 20% is used for testing.

C. Classification Techniques

The process of predicting class or categorical data from the set of independent data is known as classification. Mathematically we can say that classification is the mapping of a function from an input variable (independent variable) to an output variable (dependent variable). In this study, we use Naive Bayes [3], Decision Tree [4], SVM [5], KNN [6], Random Forest [7], Ensemble methods [1], Sequential neural networks and MLP [8] techniques for the model classification.

1) Decision Tree: Decision Tree is a common classification method which builds a tree shaped structure with root, internal nodes, and leaf nodes [14]. Decision Trees represent only rules. The rules formed in this approach are easy to comprehend for the human brain. Decision Tree permits calculations to be done in forward and backward manner which enhances the decision correctness. Decision tree has a tree structure where every node is either a Leaf node or a Decision node. Leaf node displays the value of the attribute assigned to them whereas the Decision nodes indicate the branching, where a specific test has to be done on attribute with one branch and sub-trees as a result of the test.

Decision Tree shows its potentials on reducing the complexity for making decisions, reducing unnecessary computations, increasing the flexibility with different selections of feature subsets [15].

This algorithm uses decision tree as a predictive model in which item's target value conclusion is mapped with observation about the item and used to visually classify the decision making and the decisions. The goal of this algorithm is to build a structure that is able to predict value of target variable relying on various input variables. Every internal node represents one of the input variables (independent variable). For each possible value of these variables, there are edges to the children. Leaf in the tree represents target variable value in which the given values of the input variables can be traversed by path from root to the leaf.

For the decision tree classifier, we set the criteria as 'Gini' index, and max step as 4. Gini index is the metric used to measure how far a randomly chosen element would be identified incorrectly, that is, measures the impurity level of the splitting task [16]. The smaller the value is, the better purity it achieves. When Gini index equals to 0, it means that everything is classified into the same type. Fig. 3 shows the

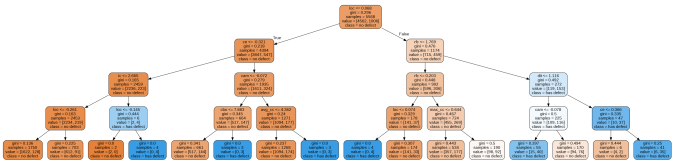

Fig. 3. Decision Tree Structure for the software defect prediction dataset

overall decision tree structure for the software metrics. We can find that the Gini index is getting smaller from the root node to the leaf nodes. And some of the Gini index for leaf nodes reached 0.0, which shows great purity.

2) Random Forest: Random forest is a bagging ensemble, which consists of several decision tree classifiers, but allows for more variance reduction compared to bagging methods [17]. It fits multiple number of decision trees on different subsets of the dataset and then uses averaging to improve the accuracy and control the over-fitting. Each decision tree in the random forest predicts the class and the class with the highest votes is selected as the classifiers' output. Random forest shows its advantages of robustness especially when dealing with data containing errors compared to other classifiers [19]. Breiman [26] demonstrated that the strength of each individual tree and the association between any two trees in the forest affect the error rates in random forests. The trees are built using the following strategy [18]:

- Each tree's root node has a sample bootstrap data which is equal to the actual data. There is a different bootstrap sample for each tree.
- Using best split method, subset of variables is randomly selected from input variables to increase the diversity among the tree.
- Then, each tree is allowed to grow as much as it can without being pruned.
- Following the construction of every tree in the forest, new instance(s) are attached to each tree, and the classification receiving the most votes is chosen as the prediction for the new instances.

We used the default 100 estimators, which means using 100 decision trees. And we also set the criterion with Gini index. A typical random forest is shown in Figure 4.

3) K-Nearest Neighbor: *K*-Nearest Neighbor (*k*-NN) is a lazy learning method, meaning the algorithm never goes through a true learning phase [20]. Simply including training examples results in the creation of a model. The model is technically prepared for testing instances at any moment throughout the addition of the training examples. Classification occurs by first measuring the distance between the test point and all other points in the data set. Euclidean distance is a logical option for many domains, but any distance metric can be utilized for this operation. The *k* nearest neighboring points to the test point are then selected, and the class is determined to be the one most common among this set of points.

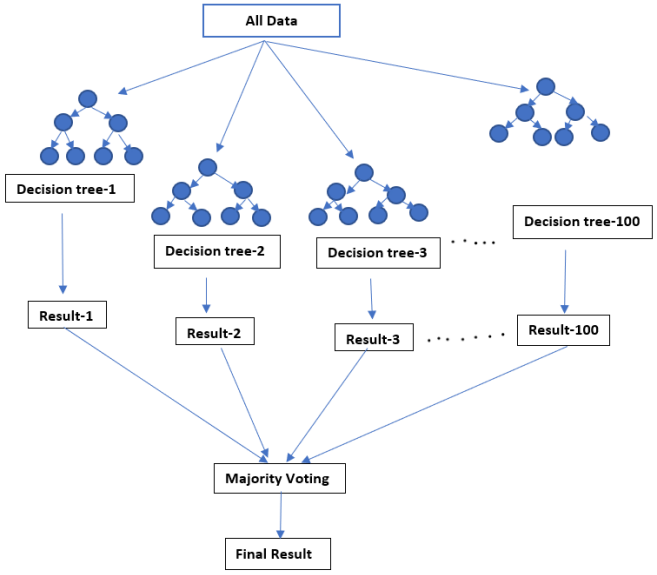

Fig. 4. Typical Random Forest with many individual decision trees

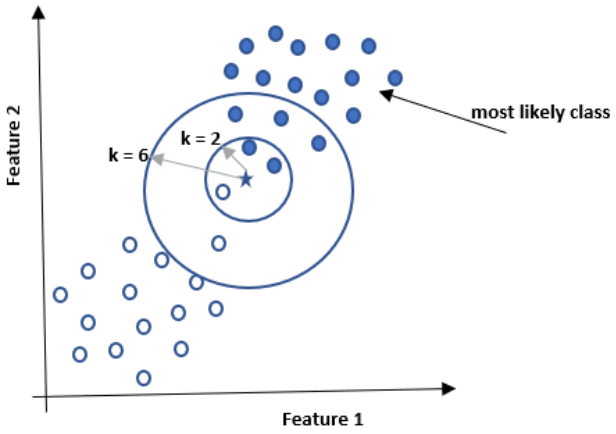

Fig. 5. Nearest Neighbor construction approximation using k = 2 and k = 6 for the data point represented by a star

A *k*-NN algorithm can be implemented in various ways. The most common data structure used for classification is the kd-tree [21]. A kd-tree divides the space into a *d*-dimensional tree in order to function. By traversing this tree, a point, x_i, can be said to be close to another point, x_j, if the point, x_i, exists within divided space of the point, x_j. Another reason for such a representation is speed and efficiency. In many cases, traversing a kd-tree is faster than using force. When the dimensionality of the feature-space is very high, a kd-tree for classification becomes ineffective. For our research, the dimensionality of the feature-space becomes large when 20 or more attributes/ features are used for classification. The number of neighbors are set to 5, with the default uniform

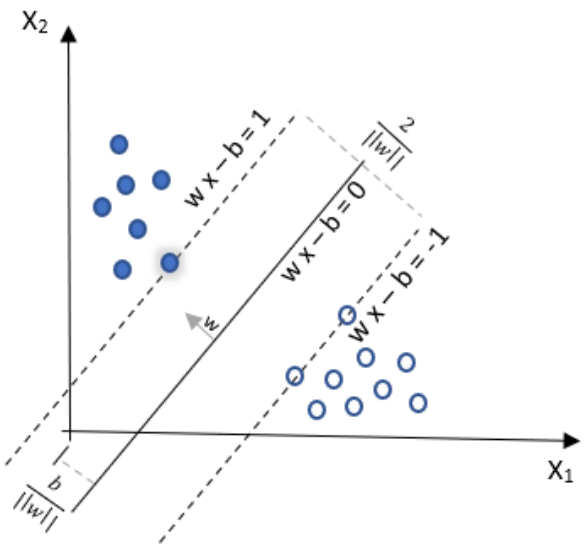

Fig. 6. A computed hyperplane (dark line) shown with a Soft Margin (area between dotted lines) separating the data points

weights, which all the points u=in each group have the same weight. An example of how *k*-NN work is shown in Figure 5.

4) Support Vector Machine: A Support Vector Machine (SVM) aims to split data into two different spaces using a hyperplane [66]. When applied to classification, the goal is to find a hyperplane that provides a maximal margin between data points in different classes, as shown in Figure 6. The data points that *support* the position of the hyperplane are known as the support vectors. The problem of deciding these points is typically solving using Quadratic Programming.

According to the categories they fall under, sample points from an SVM model are mapped into space and separated by a gap as far as possible. After being mapped into the same space, the category of the new samples is projected based on which side of the gap they fall. Since SVM is a linear classifier, meaning that it is unable to handle cases where class data overlaps. In order to take this into account, Cortes and Vapnik developed a Soft Margin Hyperplane, which permits class overlap by imposing a penalty for the degree to which a specific data point breaches the hyperplane into the feature space of the other class [23]. A slack variable is added to the Quadratic Programming formulation to more properly represent the penalty. Another way to handle non-linearly separable problems is to perform what is known as the 'kernel trick', which lifts the data into a higher dimensional space by applying a transformation to the data [24]. Our research uses this method by applying a sigmoid function as the kernel, which is one of the popular choice of kernel for SVMs. And the regularization parameter is set to 1, gamma is set to 0.5.

5) Ensemble methods: An ensemble of classifiers is proposed to address the problem of robust software defect classification. The average probability ensemble technique is the foundation of this model. Ensemble methods combine different base learners together to solve a problem. Models trained using ensemble methods typically generalize better than those trained using the standalone techniques [44]. Such models are expected to demonstrate robustness against feature redundancy and data imbalance, which typically impede software defect datasets. This robustness is guaranteed by averaging the classification performance of multiple classifiers. This averaging leads to the elimination of uncorrelated errors and, thus, enhancing overall classification performance [47].

In this paper, we study the Bagging, Adaboost, Gradient Boost (GB) and XGBoost (XGB).

Bagging (Bootstrap Aggregating) is designed to increase the stability and accuracy of machine learning algorithms. Bagging makes several predictions about the result from various training sets that are merged, either through uniform averaging or through voting [45].

Adaboost performs multiple iterations each time with different example weights, and gives a final prediction through combined voting techniques [45]. This algorithm fits a sequence of base classifiers and updates the weights of the instances by giving the wrongly classified instances a higher weight and then fit a new base learner with the updated weights. A weighted majority vote technique is used to combine the outputs from all base classifiers to get the final prediction, with each base classifier contributing and receiving a larger weight based on its performance [48]. For the estimator value, we set to 100, with decision tree as the base estimator, and 0.1 learning rate.

Gradient Boosting, proposed by Friedman, is to solve problems using a prediction model consisting of an ensemble of weak predictors [46]. This algorithm is a generalization of Adaboost ensemble that can be built using a variety of loss functions. This technique uses gradients, whereas Adaboost uses the weight of incorrectly classified cases to fit a new base classifier. Gradient boosting can improve the fit of the underlying classifiers, but it consumes too much memory and takes too long to process [49]. In our study, we used the default classifier with log_loss as the loss function, 0.1 learning rate, 100 estimators.

XGBoost (XGB) or Extreme Gradient Boosting, is a Gradient Boosting ensemble that finds the best base classifier more precisely and quickly by using the second-order derivatives of the loss function. Gradient Boosting uses gradients to fit a new base classifier whereas, XGB utilizes the second-order gradients [50]. There are linear ans tree based models can chose to be the booster. In our research, tree based models are used as boosters.

6) Naïve Bayes Classifier: A Bayesian network is a probabilistic graphical model that represents the joint probability distribution across a number of discrete random variables [12]. The model makes use of a directed acyclic graph, in which the nodes represent variables and the edges represent conditional relationships between these variables.

Each node in the graph is said to be conditionally independent of any other node that is not its descendent.

Naïve Bayesian classifiers are a specialization of the model that work by forcing a particular network structure onto the graph, where the class node is the sole parent to every feature node, and the class node has no parents of its own [13]. This is consistent with the assumption that, given the class node, each feature is conditionally independent of every other feature. Due to the fact that the class probability can be calculated as a simple product of the probabilities corresponding to each characteristic, this structure makes inference extremely efficient. If this assumption is not met, 2^{n+1} values are required to express the probability distribution over binary variables. This assumption gives the classifier the 'naïve' qualifier.

Naïve Bayes is based on the Bayes theorem which is a simple mathematical formula used to calculate conditional probabilities. Conditional probability is a measure of the probability that an event will occur given that another event occurred.

In our research, we implemented Gaussian Naïve Bayes and Bernoulli Naïve Bayes. In Gaussian Naïve Bayes, it is assumed that the continuous values connected to each feature are distributed in a Gaussian manner. Bernoulli Naïve Bayes classifier is preferred when the underlying data has a multivariate Bernoulli distribution. The feature samples in this situation are presumed to be binary.

7) Neural Networks: Models of Neural Networks, which are made up of neurons and synapses, are based on biological neural networks. The synapses in the model are represented as weighted edges connecting the neurons, which are represented as nodes in a graphical network. Conceptually, synapses serve as information transfer mechanisms whereas neurons serve as processing units. This idea serves as the basis for the mathematical model and operating principles of perceptrons and Artificial Neural Network (ANN) or Sequential Neural Network (SNN) classifiers [27]. A fully-connected feed-forward ANN is comprised of layers, where the nodes in each layer are fully connected. We used sequential fully connected neural networks, the input shape is (20,), since there are 20 features(software metrics) as the network input. There are two hidden layers, they both have 32 neurons, while the output layer size is 1, since it returns the final prediction result. There are three optimizers that are tested, Adam, SGD and RMSProp. A graphical representation of Neural Network is shown in Figure 7.

In the simplest scenario, these connections represent a set of incoming weighted links that are regarded as having "high" or "low" values and are generated by a step function. The activation function of the node receives the sum of these incoming weighted values as input. An ANN typically consists of an input layer, a hidden layer, and an output layer. These layers work together to represent complex functions. The last layer of classification consists of binary class nodes, whose value is decided by the network's inputs. The most

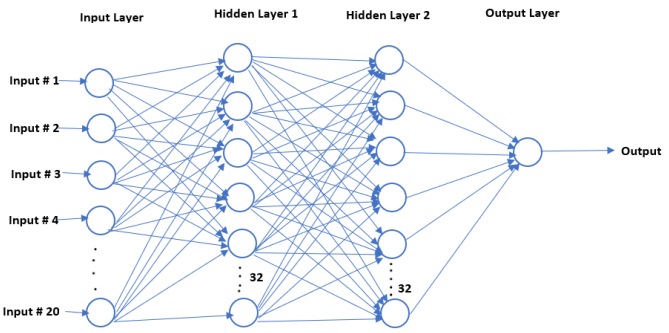

Fig. 7. An ANN with a 4 node input layer, a 5 node hidden layer and a 1 node output layer

popular method for learning edge weights is back-propagation, which modifies values in accordance with the error gradient [28]. This effectively reflects how weights influence a node's input's "downstream" behavior. Here, error is determined by incorrect classification produced in the output layer.

Multi Layer Perceptron (MLP) is a type of ANN which consists of multiple layers of feed-forward-connected computational units. These networks are considered as non-linear supervised classifiers that approximate the underlying functionality in data by minimizing the loss function. Specific data records are input into the neural network during training, and its weights are adjusted so that the output will roughly match the values in the data set. After the learning process is complete, the learnt information is represented by the weight values of the neural network. We imported MLP classifier from sklearn package neural_network module. And the default classifier is tested in this project, which with 100 hidden layer, Adam optimizer, 0.001 learning rate and other basic default settings.

8) Cross Validation: Classifiers that applied cross validations are tend to have a higher average performance compared to the classifiers without the cross validation process [61]. We applied cross_val_score function from sklearn package model_selection module. This function helps to apply cross validation and evaluate classifiers with the specified score (we used MAE score). 5-fold cross validation is chosen in the program. The data is "rotated" so that a new 20% is chosen for testing. This is done 5 times (or folds), where each fold is used exactly once as a testing dataset, and it is ensured that all the data has been tested once. To ensure that the proportion of defective and non-defective instances is approximately equal, we used this technique and repeat this procedure for the classifier models.

D. Evaluation method

Garicia et al. [63] claimed that checking the confusion matrix is the most correct way to evaluate the classifiers for imbalanced dataset. We selected a set of evaluation measures to evaluate prediction performance of the classifiers. These

measures are derived from the four possible prediction outcomes of a binary classification problem: (1) True Positives (TP) which is the number of defective instances correctly classified as defective, (2) False Positives (FP) which is the number of defective instances incorrectly classified as non-defective, (3) True Negatives (TN) which is the number of non-defective instances correctly classified as non-defective, and (4) False Negatives (FN) which is the number of non-defective instances incorrectly classified as defective. We evaluate the classifiers with the accuracy, ROC& AUC, classification report, MAE scores, and we also checked feature importance for selected methods.

In this study, the model accuracy, the precision, recall and F-1 score values are calculated as follows:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} = \frac{2 * TP}{2 * TP + FP + FN}$$

Accuracy is the ratio of correctly predicted instances to the total number of instances. It is a common measurement for evaluating classification performance, which provides an overview of the classifiers' performance. However when the dataset is imbalanced, this value may not be accurate, as the prediction results would be bias towards the majority class, in this case, it is the defective instances. In this case, we need precision and recall to provide a detailed view of the classification for each label's samples. These two indicators both prefer a larger value, as closer to 1 will be better. But they require a harmony that a classifier has a better performance when the precision and recall both have a higher value. High precision and low recall shows the classifier returns less results but most of them are correct. While low precision and high recall shows that the classifier returns many results but less of them are correct. To achieve the harmony between precision and recall, we have the F-1 score. F-1 score is calculated based on the value of precision and recall. It achieves the balance between the two important indicators, that is the precision and recall, and hence it is a suitable indicator for measuring and evaluating the imbalanced dataset [62].

Mean absolute error (MAE) is calculated according to

$$MAE = \frac{1}{n} \sum_{i=1}^{n} |x_i - y_i|$$

The absolute value is calculating error (for our project is the error of the classification, number of misclassified labels compared to its actual labels). MAE is an unambiguous, natural measurement of an average error [51], which means it is more interpretive compared to similar indicator like root mean squared error (RMSE). Hence, we chose MAE as one of indicator for classifier evaluation.

The ROC (Receiver Operator Characteristic) curve is represented by plotting the values of true positive rate (TPR), which is usually on the y-axis, against the false positive rate (FPR), which is usually on the x-axis. The diagonal line is a the line generated by random guessing, it plays the role of a baseline for the ROC curve comparison. The line above the baseline shows a better performance, while the line below the baseline shows a bad performance even worth than random guessing. AUC is the area under the ROC curve, where the larger area indicates high performance. AUC can have a value between 0 or 1, where 1 indicates the best performance and 0 is considered the worst performance.

Feature importance refers to a class of techniques for assigning scores to input features to a predictive model that indicates the relative importance of each feature when making a prediction. Its scores can provide insight into the dataset and into the prediction model used. These scores can also be used to improve the efficiency and effectiveness of the model by using it to select which features to keep (highest scores). These scores can be measured using a number of different techniques, but we use Decision Tree models and ensembles of trees such as Random Forest and XGBoost, which provides an attribute 'feature_importances_' present in scikit-learn library, when fitted.

IV. Literature Review

A lot of research has been done to build software fault prediction models using different fault prediction techniques such as Machine Learning techniques and ensemble learning techniques. Ensemble learning is common to choose when dealing with software defect prediction tasks. Mabayoje et al. [1] did a comparison analysis between four ensemble methods (bagging, boosting, stacking and voting), and in these methods, bagging has the best performance. Sun et al. [10] proposed a ensemble learning method with three data coding schemes which aiming at dealing with imbalanced dataset. Balogun et al. [11] tested ensemble learning classifiers with various performance metrics in order to provide a comprehensive evaluation framework. Aleem et al. [65] analyzed and compared the prediction outcomes of 11 machine learning techniques such as MultiLayer Perceptron, NB, SVMs, AdaBoost, Bagging, RF, etc. using the dataset from the PROMISE repository. The result of this study shows that Bagging and SVM given better prediction performance. A study done by Elish, Aljamaan and Ahmad [68] shows that the ensemble method gives correct prediction results on the considered dataset. A study was done by Perreault et al. [67] on the datasets from PROMISE repository compared Artificial Neural Networks, LR, NB, SVMs, and *k*-NN, however, there is no clear explanation as to which strategy is the most effective. Ghotra et al. [66] established that the accuracy of a prediction model can increase or decrease upto 30% depending upon the classifier used.

For dealing with imbalanced dataset, Vuttipittayamongkol et al. [64] proposed an overlap based Under Sampling method which indicates achieve exceptional improvement

TABLE II
SUMMARY OF CLASSIFICATION REPORTS OF CLASSIFIERS

	Label	DT	RF	Bagging	AdaBoost	GB	XGB	KNN	SVM	GaussianNB	BernoulliNB	MLP
Precision	**0**	0.83	0.86	0.86	0.85	0.85	0.85	0.85	0.84	0.94	0.89	0.85
	1	0.56	0.54	0.54	0.43	0.70	0.64	0.46	0.26	0.21	0.36	0.56
Recall	**0**	0.99	0.95	0.95	0.92	0.98	0.98	0.94	0.84	0.25	0.77	0.97
	1	0.08	0.28	0.28	0.28	0.20	0.19	0.23	0.25	0.92	0.57	0.20
F1-score	**0**	0.90	0.90	0.90	0.88	0.91	0.91	0.89	0.84	0.40	0.83	0.90
	1	0.13	0.37	0.37	0.34	0.31	0.29	0.30	0.25	0.35	0.44	0.29

TABLE III
THE CLASSIFICATION REPORT AFTER APPLYING UNDER SAMPLING METHOD

	Label	DT	RF	Bagging	Adaboost	GB	XGBoost	KNN	SVM	GaussianNB	BernoulliNB	MLP
Precision	**0**	0.61	0.67	0.68	0.69	0.67	0.67	0.63	0.55	0.57	0.63	0.67
	1	0.73	0.69	0.69	0.68	0.71	0.71	0.66	0.55	0.73	0.70	0.70
Recall	**0**	0.83	0.69	0.70	0.67	0.75	0.75	0.70	0.53	0.87	0.75	0.73
	1	0.46	0.67	0.67	0.71	0.63	0.63	0.60	0.56	0.34	0.56	0.63
F1-score	**0**	0.70	0.68	0.69	0.68	0.70	0.71	0.66	0.54	0.69	0.69	0.70
	1	0.57	0.68	0.68	0.69	0.67	0.67	0.63	0.55	0.46	0.62	0.67

TABLE IV
MAE SCORE OF CROSS VALIDATION ON MULTIPLE CLASSIFIERS

	DT	RF	Bagging	Adaboost	GB	XGB	KNN	SVM	GaussianNB	BernoulliNB
Fold-1	0.175	0.181	0.180	0.187	0.173	0.171	0.186	0.267	0.810	0.244
Fold-2	0.175	0.177	0.176	0.189	0.180	0.172	0.183	0.265	0.590	0.250
Fold-3	0.185	0.184	0.187	0.198	0.186	0.194	0.197	0.263	0.633	0.246
Fold-4	0.182	0.185	0.189	0.209	0.182	0.189	0.197	0.261	0.705	0.279
Fold-5	0.174	0.176	0.179	0.195	0.173	0.167	0.166	0.261	0.266	0.253
Average	0.178	0.181	0.182	0.196	0.179	0.179	0.186	0.264	0.601	0.254

to the classifier performance. There are also other methods that shows its strong potential. SVM based activate learning strategy [53], which shows its ability of reducing the time cost as well as achieve comparative classification performance. Koziarski [54] used Under Sampling method, but with the radial based sampling strategy using mutual class potential which is proved to be beneficial to the imbalanced dataset classification.

V. RESULTS AND DISCUSSION

A. Accuracy

Fig.8 shows the overall accuracy comparisons between classifiers. Gradient Boosting which belongs to one of the ensemble methods has the highest accuracy rate 0.84. While the Gaussian Bayes has the worst performance with the accuracy rate 0.374. For sequential neural network model, the model accuracy is also tested with different optimizer selections(Adam, SGD and RMSprop). Fig. 9 shows that Adam optimizer has the best performance compared to the optimizer SGD and RMSProp during the 100 training epoches, which shows its advantages at an early stage. Adam, the gradient based optimizer is efficient in most cases and requires less memory [43] compared to other optimizers.

In summary, the accuracy score provides a basic evaluation on the classifier performance, but it still needs other evaluation

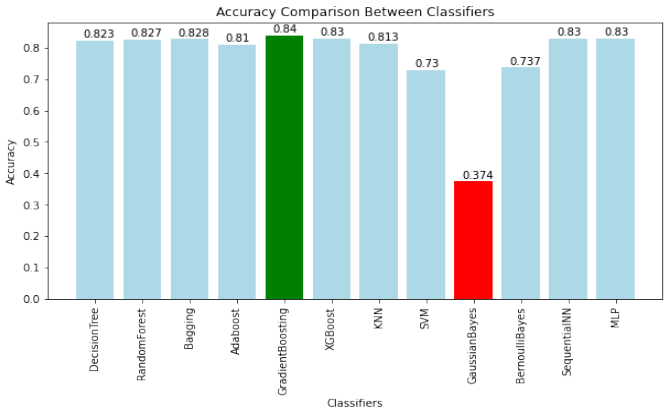

Fig. 8. Accuracy Comparisons- testing set

indicators to make a comprehensive comparison, since the imbalanced dataset may have influence on the overall accuracy result.

B. ROC and AUC

Fig.10 (A) shows the AUC value comparisons between multiple classifiers. Gradient boosting classifier has the best performance for the AUC score with the value 0.772, while support vector machine has the worst value 0.515. And we compared the ROC curve with figures (B) gradient boosting

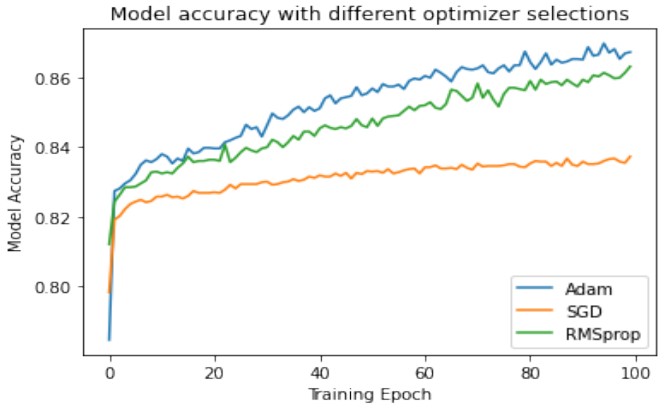

Fig. 9. Sequential neural network optimizer comparison during training

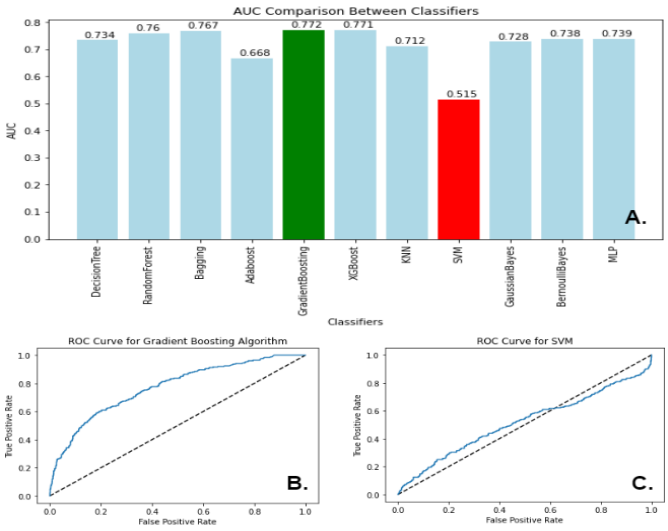

Fig. 10. AUC and ROC

and the (C) SVM. We can see it shows a great difference of the ROC curve for these two classifiers. The ROC curve for SVM classifier is even worse than random guessing baselines when the false positive rate reaches to 0.6.

C. Classification Report

This classification report analysis will be compared and discussed between the imbalanced dataset (Table II), and the balanced dataset (Table III) which applied undersampling methods.

Table II summarizes the classification report(Precision, Recall and F1-score) of classifiers. In general. the performance of all the classifiers have a better performance on the prediction on the data with label 0 (no defect) than label 1 (has defect). It is due to the differences between the quantity value of each type of labels in the dataset. As we introduced in the limitation sections, to overcome imbalanced data, Naive Bayes classifier(GaussianNB and BernoulliNB in the table) and ensemble methods(Bagging, ADaboost, GB and XGB in the table) are used. Which is worth to notice, Gradient Boosting (GB) has the highest Precision value on predicting label 1 with the value 0.70. Gaussian Naive Bayes has the best recall value(0.92) for classifying label 1, even outperforms its value for label 0. Bernoulli Naive Bayes has the highest F-1 score 0.44 compared to other classifiers for label 1. Apparently, ensembles methods (especially boosting methods), and the Naive Bayes classifiers have a slightly better performance dealing with the classification tasks of label 1 compared to other classifiers.

We also tested with an undersampling method to achieve a balanced label quantity. In our project, label 0 has more record, so we simply deleted the extra data, and matched the record size of label 1. For the balanced dataset, we re-run all the classifiers and to check their performance. The classification report for the classifiers after applying the undersampling method is in Table III. The overall scores between label 0 and label 1 have less difference than before. And it is interesting to see, the scores for label 1 are all got improved compared to the classifier performance without balancing the dataset(Table II). Examples are like the decision tree classifier, which has the least F-1 score 0.13 compared to other classifiers, has increased to 0.57. This is a over triple fold increase. However, the scores for label 0 for all the evaluation indicators are decreased. For example, the Gaussian Naive Bayes which has 0.94, the highest precision score for label 0 classification task, has dropped to 0.57. The reduction of sample size for label 0 is one of the reason for the decreased score. Hence, it is able to find that in general, undersampling method has positive impact to the category type has less quantity values, and has negative impact to the category type that has more quantity values but got cut off. Another observation from the comparisons is that Gradient Boosting and XGBoost show its stability for the precision score on label 1 without substantial increase or decrease.

D. MAE score with cross validation method

To deal with the imbalanced dataset, we applied cross validation method. And MAE scores are used as the indicator to evaluate the classifier performance. Table IV summarized the overall MAE score for all the classifiers with cross validation method. Since MAE represents the average error, which means the value smaller the better. Gaussian Naive Bayes has the highest average MAE score 0.601, while decision tree has the best performance of 0.178 MAE score in average. There is also an observation that all the other classifiers has a rather stable performance across 5 folds, except the Gaussian Naive Bayes. The value has a great difference across different folds, for example, the MAE score in fold 1 is 0.810, it dropped to 0.590 in fold 2, but the value got increased in fold 3 (0.633).

E. Feature importance

Fig 11 shows the feature importance of three classifiers. The top 5 features for decision tree classifier are loc, rfc, ce, cam and max_cc. The top 5 features for random forest are loc, rfc, cam, amc, and cbo. For XGBoost, the top 3 features are loc, rfc, max_cc. In summary, loc, rfc and cam are the most

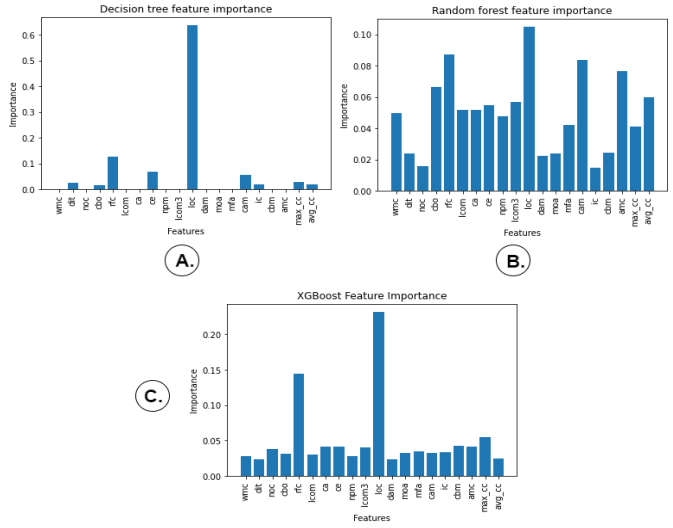

Fig. 11. Feature importance of three classifiers (A: Decision tree, B: Random Forest, C: XGBoost)

important features compared to other features for these three models.

loc, which is lines of codes, is one of the most common software metric when talking about complexity. loc measures the size of software, and indicated the time and effort requirement for developing a software product [55]. It shows that the more lines of code, more time and related cost will be spent on the software development. Sathanandam et al. [56] shows that programs are tend to involve more error when the lines of codes are increased, but at the same time, there will be a higher rate that the performance of the program will also get raised. There also many research evidence indicates the highly correlation between lines of code and software program defects [57]. rfc, which is the response for a class. The response set in a class consists of all the local methods [59]. It indicates that the larger the response set for a class, the program tends to have a higher complexity. It will also bring more difficulties for the software maintaining, since it involves detailed checking for the local methods to figure out where is the bug come from [58]. These two metrics shows a significant importance towards the classification tasks for all three classifiers. While as for cam, which is the cohesion among methods. It computes the relatedness among methods of a class based on the parameter list of methods. cam shows its strong ability on measuring cohesions on the design stage of programs [60]. It also indicates that the higher relatedness between methods will cause more bugs, since the bug in one method may highly relate to the use of another method or function, which also brings difficulty to the software maintenance. This metric has a relatively higher importance compared to other software complexity metrics. In summary, these three metrics depict the complexity of a program from the general lines level (loc), program function level (cam) to the general class level (rfc), which increases the confidence in the result.

VI. Conclusion

This work compared the performance of various machine learning classifiers and sequential neural networks on software defect prediction task. We summarize our conclusion by answering the previous raised research questions.

A. Answer for Q1: Summary of the classifier performance

The previous section measures the performance of classifiers from various aspects. It is able to answer that different classifiers will have different performance dealing with given tasks. We can find that Gradient Boosting has the highest accuracy score 0.84, the best ROC curve with the AUC score 0.772, and it even has a better ability to predict the label type which has the less samples in the dataset compared to other classifiers. For the MAE score, it also has a lower value 0.179, just 0.001 difference with the value for decision tree classifier. In summary, gradient boosting has the best performance dealing with defect prediction tasks given the dataset in this project.

B. Answer for Q2: Summary of the methods for imbalanced dataset

Dealing with imbalanced data, there are 4 popular methods we applied and tested, Naïve Bayes classifier, Ensemble methods, Cross Validation, and Under-Sampling.

For Naïve Bayes classifier, in the classification report (Table II), it shows high recall but low precision, which indicates that it returns many results but actually in these results, very few of them are correctly labeled compared to the sample's real label. However, gradient boosting shows an opposite status with high precision and low recall, which shows it returns less results but most of them are correct. None of it is the ideal classifier which could achieve high precision and high recall. It shows Naïve Bayes classifier and ensembles still have limitations when processing imbalanced data. We applied cross validation to the dataset and returned the MAE score for evaluating the classifier performance. Other classifiers have similar performance, while Gaussian Naive Bayes shows that its performance is not stable when processing different folds of data. Under-sampling significantly raised the performance of classification on label 1 for all the classifiers while sacrificing the classification accuracy for label 0.

C. Answer for Q3: Summary for feature importance

There are three classifiers' feature importance are checked in this research, decision tree, random forest and XGBoost. loc (lines of code), rfc (response for a class) and cam (cohesion among methods) are the most important features which contributes most to the overall prediction.

For future work, there are more methods for processing imbalanced dataset can be tested. What is also important is there is an urgent need to collect more labeled data and expand the dataset for providing a better classifier evaluation for future classification tasks.

REFERENCES

[1] Mabayoje, M., Balogun, A., Bajeh, A. & Musa, B. Software defect prediction: effect of feature selection and ensemble methods. *FUW Trends Sci. Technol. J*. **3** pp. 518-522 (2018)

[2] Aggarwal, D. Software Defect Prediction Dataset. (2021,1), https://doi.org/10.6084/m9.figshare.13536506.v1

[3] Rish, I. An Empirical Study of the Naïve Bayes Classifier. *IJCAI 2001 Work Empir Methods Artif Intell*. **3** (2001,1)

[4] Zhao, Y. & Zhang, Y. Comparison of decision tree methods for finding active objects. *Advances In Space Research*. **41**, 1955-1959 (2008), https://www.sciencedirect.com/science/article/pii/S027311770700796X

[5] Malhotra, R. & Bansal, A. Use of Support Vector Machine to Check Whether Process Metrics are as Good as Static Code Metrics. *Topical Drifts In Intelligent Computing*. pp. 35-42 (2022)

[6] Cover, T. & Hart, P. Nearest neighbor pattern classification. *IEEE Transactions On Information Theory*. **13**, 21-27 (1967)

[7] Masetic, Z. & Subasi, A. Congestive heart failure detection using random forest classifier. *Computer Methods And Programs In Biomedicine*. **130** pp. 54-64 (2016), https://www.sciencedirect.com/science/article/pii/S0169260715303369

[8] Kotsiantis, S. Supervised Machine Learning: A Review of Classification Techniques. *Proceedings Of The 2007 Conference On Emerging Artificial Intelligence Applications In Computer Engineering: Real Word AI Systems With Applications In EHealth, HCI, Information Retrieval And Pervasive Technologies*. pp. 3-24 (2007)

[9] Dietterich, T. Ensemble Methods in Machine Learning. *Multiple Classifier Systems*. pp. 1-15 (2000)

[10] Sun, Z., Song, Q. & Zhu, X. Using coding-based ensemble learning to improve software defect prediction. *IEEE Transactions On Systems, Man, And Cybernetics, Part C (Applications And Reviews)*. **42**, 1806-1817 (2012)

[11] Balogun, A., Bajeh, A., Orie, V. & Yusuf-Asaju, W. Software defect prediction using ensemble learning: an ANP based evaluation method. *FUOYE J. Eng. Technol*. **3**, 50-55 (2018)

[12] Koller, D. & Friedman, N. Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning. (The MIT Press,2009)

[13] Duda, R., Hart, P., Hart, P., Hart, P., Stork, D., Library, E. & Sons, J. Pattern Classification. (Wiley,2001), https://books.google.com/books?id=YoxQAAAAMAAJ

[14] Song, Y. & Ying, L. Decision tree methods: applications for classification and prediction. *Shanghai Archives Of Psychiatry*. **27**, 130 (2015)

[15] Safavian, S. & Landgrebe, D. A survey of decision tree classifier methodology. *IEEE Transactions On Systems, Man, And Cybernetics*. **21**, 660-674 (1991)

[16] Tangirala, S. Evaluating the impact of GINI index and information gain on classification using decision tree classifier algorithm. *International Journal Of Advanced Computer Science And Applications*. **11**, 612-619 (2020)

[17] Wager, S., Hastie, T. & Efron, B. Confidence intervals for random forests: The jackknife and the infinitesimal jackknife. *The Journal Of Machine Learning Research*. **15**, 1625-1651 (2014)

[18] Jiang, Y., Cuki, B., Menzies, T. & Bartlow, N. Comparing Design and Code Metrics for Software Quality Prediction. *Proceedings Of The 4th International Workshop On Predictor Models In Software Engineering*. pp. 11-18 (2008), https://doi.org/10.1145/1370788.1370793

[19] Ao, Y., Li, H., Zhu, L., Ali, S. & Yang, Z. The linear random forest algorithm and its advantages in machine learning assisted logging regression modeling. *Journal Of Petroleum Science And Engineering*. **174** pp. 776-789 (2019)

[20] N. S. Altman An Introduction to Kernel and Nearest-Neighbor Nonparametric Regression. *The American Statistician*. **46**, 175-185 (1992),

[21] Bentley, J. Multidimensional Binary Search Trees Used for Associative Searching. *Commun. ACM*. **18**, 509-517 (1975,9), https://doi.org/10.1145/361002.361007

[22] Andriansyah, M., Suhendra, A., Wicaksana, I. & Wicaksana, S. Comparative Study: The Implementation of Machine Learning Method for Sentiment Analysis in Social Media. A Recommendation for Future Research. *Advanced Science Letters*. **20** (2014,5)

[23] Cortes, C. & Vapnik, V. Support-Vector Networks. *Mach. Learn.*. **20**, 273-297 (1995,9), https://doi.org/10.1023/A:1022627411411

[24] Aizerman, M. Theoretical Foundations of the Potential Function Method in Pattern Recognition Learning. *Automation And Remote Control*. **25** pp. 821-837 (1964)

[25] Akimova, E., Bersenev, A., Deikov, A., Kobylkin, K., Konygin, A., Mezentsev, I. & Misilov, V. A survey on software defect prediction using deep learning. *Mathematics*. **9**, 1180 (2021)

[26] Breiman, L. Random Forests. *Machine Learning*. **45** pp. 5-32 (2001,10)

[27] Minsky, M. & Papert, S. Perceptrons: An Introduction to Computational Geometry. (MIT Press,1969)

[28] Rumelhart, D., Hinton, G. & Williams, R. Learning representations by back-propagating errors. *Nature*. **323** pp. 533-536 (1986)

[29] LYNCH, J. The Worst Computer Bugs in History: The Ariane 5 Disaster.

[30] Thota, M., Shajin, F., Rajesh, P. & Others Survey on software defect prediction techniques. *International Journal Of Applied Science And Engineering*. **17**, 331-344 (2020)

[31] Muhammad, I. & Yan, Z. SUPERVISED MACHINE LEARNING APPROACHES: A SURVEY.. *ICTACT Journal On Soft Computing*. **5** (2015)

[32] Soni, A., Dharmacharya, D., Pal, A., Srivastava, V., Shaw, R. & Ghosh, A. Design of a machine learning-based self-driving car. *Machine Learning For Robotics Applications*. pp. 139-151 (2021)

[33] Wazery, Y., Saber, E., Houssein, E., Ali, A. & Amer, E. An efficient slime mould algorithm combined with k-nearest neighbor for medical classification tasks. *IEEE Access*. **9** pp. 113666-113682 (2021)

[34] Sharma, A., Jain, A., Gupta, P. & Chowdary, V. Machine learning applications for precision agriculture: A comprehensive review. *IEEE Access*. **9** pp. 4843-4873 (2020)

[35] Laradji, I., Alshayeb, M. & Ghouti, L. Software defect prediction using ensemble learning on selected features. *Information And Software Technology*. **58** pp. 388-402 (2015)

[36] Wang, H., Khoshgoftaar, T. & Napolitano, A. A comparative study of ensemble feature selection techniques for software defect prediction. *2010 Ninth International Conference On Machine Learning And Applications*. pp. 135-140 (2010)

[37] Petrić, J., Bowes, D., Hall, T., Christianson, B. & Baddoo, N. Building an ensemble for software defect prediction based on diversity selection. *Proceedings Of The 10th ACM/IEEE International Symposium On Empirical Software Engineering And Measurement*. pp. 1-10 (2016)

[38] Huda, S., Liu, K., Abdelrazek, M., Ibrahim, A., Alyahya, S., Al-Dossari, H. & Ahmad, S. An ensemble oversampling model for class imbalance problem in software defect prediction. *IEEE Access*. **6** pp. 24184-24195 (2018)

[39] Prykhodko, S. Developing the software defect prediction models using regression analysis based on normalizing transformations. *Modern Problems In Testing Of The Applied Software"(PTTAS-2016), Abstracts Of The Research And Practice Seminar, Poltava, Ukraine*. pp. 6-7 (2016)

[40] Mabayoje, M., Balogun, A., Jibril, H., Atoyebi, J., Mojeed, H. & Adeyemo, V. Parameter tuning in KNN for software defect prediction: an empirical analysis. *Jurnal Teknologi Dan Sistem Komputer*. **7**, 121-126 (2019)

[41] Gray, D., Bowes, D., Davey, N., Sun, Y. & Christianson, B. Using the support vector machine as a classification method for software defect prediction with static code metrics. *International Conference On Engineering Applications Of Neural Networks*. pp. 223-234 (2009)

[42] Shuai, B., Li, H., Li, M., Zhang, Q. & Tang, C. Software defect prediction using dynamic support vector machine. *2013 Ninth International Conference On Computational Intelligence And Security*. pp. 260-263 (2013)

[43] Kingma, D. & Ba, J. Adam: A method for stochastic optimization. *ArXiv Preprint ArXiv:1412.6980*. (2014)

[44] Zhou, Z. Ensemble Learning. *Encyclopedia Of Biometrics*. (2009)

[45] Wang, T., Li, W., Shi, H. & Liu, Z. Software defect prediction based on classifiers ensemble. *Journal Of Information And Computational Science*. **8** pp. 4241-4254 (2011,12)

[46] Friedman, J. Stochastic gradient boosting. *Computational Statistics and Data Analysis*. **38**, 367-378 (2002), https://EconPapers.repec.org/RePEc:eee:csdana:v:38:y:2002:i:4:p:367-378

[47] Dietterichl, T. Ensemble Learning. *The Handbook Of Brain Theory And Neural Networks*. pp. 405-408 (2002)

[48] Freund, Y. Boosting a Weak Learning Algorithm by Majority. *Information And Computation*. **121**, 256-285 (1995), https://www.sciencedirect.com/science/article/pii/S0890540185711364

[49] Friedman, J. Greedy Function Approximation: A Gradient Boosting Machine. *The Annals Of Statistics*. **29**, 1189-1232 (2001), http://www.jstor.org/stable/2699986

[50] Chen, T. & Guestrin, C. XGBoost: A Scalable Tree Boosting System. *Proceedings Of The 22nd ACM SIGKDD International Conference On Knowledge Discovery And Data Mining*. pp. 785-794 (2016), https://doi.org/10.1145/2939672.2939785

[51] Willmott, C. & Matsuura, K. Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance. *Climate Research*. **30**, 79-82 (2005)

[52] Sayyad Shirabad, J. & Menzies, T. The PROMISE Repository of Software Engineering Databases.. (School of Information Technology and Engineering, University of Ottawa, Canada,2005), http://promise.site.uottawa.ca/SERepository

[53] Ertekin, S., Huang, J., Bottou, L. & Giles, L. Learning on the border: active learning in imbalanced data classification. *Proceedings Of The Sixteenth ACM Conference On Conference On Information And Knowledge Management*. pp. 127-136 (2007)

[54] Koziarski, M. Radial-based undersampling for imbalanced data classification. *Pattern Recognition*. **102** pp. 107262 (2020)

[55] Bhatia, S. & Malhotra, J. A survey on impact of lines of code on software complexity. *2014 International Conference On Advances In Engineering & Technology Research (ICAETR-2014)*. pp. 1-4 (2014)

[56] Aswini, S. & Yazhini, M. An assessment framework of routing complexities using LOC metrics. *2017 Innovations In Power And Advanced Computing Technologies (i-PACT)*. pp. 1-6 (2017)

[57] El Emam, K., Benlarbi, S., Goel, N. & Rai, S. The confounding effect of class size on the validity of object-oriented metrics. *IEEE Transactions On Software Engineering*. **27**, 630-650 (2001)

[58] Li, W. & Henry, S. Object-oriented metrics that predict maintainability. *Journal Of Systems And Software*. **23**, 111-122 (1993)

[59] Chidamber, S. & Kemerer, C. Towards a metrics suite for object oriented design. *Conference Proceedings On Object-oriented Programming Systems, Languages, And Applications*. pp. 197-211 (1991)

[60] Bonja, C. & Kidanmariam, E. Metrics for class cohesion and similarity between methods. *Proceedings Of The 44th Annual Southeast Regional Conference*. pp. 91-95 (2006)

[61] Schaffer, C. Selecting a classification method by cross-validation. *Machine Learning*. **13**, 135-143 (1993)

[62] Kamalov, F., Thabtah, F. & Leung, H. Feature Selection in Imbalanced Data. *Annals Of Data Science*. pp. 1-15 (2022)

[63] Garcia, S. & Herrera, F. Evolutionary undersampling for classification with imbalanced datasets: Proposals and taxonomy. *Evolutionary Computation*. **17**, 275-306 (2009)

[64] Vuttipittayamongkol, P. & Elyan, E. Improved overlap-based undersampling for imbalanced dataset classification with application to epilepsy and parkinson's disease. *International Journal Of Neural Systems*. **30**, 2050043 (2020)

[65] Aleem, S., Capretz, L. & Ahmed, F. Benchmarking Machine Learning Technologies for Software Defect Detection. *CoRR*. **abs/1506.07563** (2015), http://arxiv.org/abs/1506.07563

[66] Ghotra, B., McIntosh, S. & Hassan, A. Revisiting the Impact of Classification Techniques on the Performance of Defect Prediction Models. (2015,5)

[67] Perreault, L., Berardinelli, S., Izurieta, C. & Sheppard, J. Using Classifiers for Software Defect Detection. (2016)

[68] Elish, M., Aljamaan, H. & Ahmad, I. Three empirical studies on predicting software maintainability using ensemble methods. *Soft Computing*. **19** pp. 2511-2524 (2015)