Software Defect Prediction using McCabe's and Halstead's Source Code Metrics

Edara Sai Subash
Z1948230
Northern Illinois University
Saisubash.edara@gmail.com

Ajay Kumar Kandula
Z1947305
Northern Illinois University
Kandulaajay99@gmail.com

Abstract: Software defect prediction is regarded as the most significant activity that can be utilized to maintain the quality of software in the field of software engineering. The most crucial feature of a software is its quality. Software Defect Prediction, which has gained substantial popularity in recent years, can directly affect quality. Defective software modules have a significant impact on the quality of software, causing cost overruns, delays in delivery, and much higher maintenance expenses. In this project we have analyzed the most popular and widely used Machine Learning algorithms – DT(Decision Trees), RF(Random Forest), NB(Naive Bayes) and XGB classifier. The dataset used in this project has the most serious metrics used in practice to calculate the complexity of programs. McCabe is based on the number of decision points; Halstead is based on the number of operands and operators in programs. The Project Divided into three phases. Each Phases the independent variables changes. Using these source code metrics to identify the defects in the source code that a program generates and determining which source code metrics have the greatest influence on defect prediction. The Results we demonstrated by Machine learning algorithms in terms of defect prediction accuracy and other performance metrics.

1. INTRODUCTION

A software defect is an error, flaw, failure, or fault in a computer program or system that causes it to produce an incorrect result, or to behave in unintended ways. The majority of flaws are from human oversights and mistakes in either the source code or the design of a program, as well as in the frameworks and operating systems that these programs employ. A small number of flaws are also the result of compilers producing wrong code. Models that use test data to forecast probable software flaws are referred to as software defect prediction models. The degree to which the software is prone to errors is correlated with its metrics. A software defect prediction model is made up of a dependent variable and independent variables (software metrics) that are gathered and measured during the software development life cycle. Software defect prediction is regarded as the most significant activity that can be utilized to maintain the quality of software in the field of software engineering. Software engineering, sometimes known as software quality, refers to both functional and structural software quality. Structural quality of software emphasizes non-functional requirements while functional quality of software reflects functional requirements. The quality component of the project, process, and software is the main focus. The software product is the primary focus of this paper.

The goal of software product quality engineering is to produce a product with the desired level of quality by defining quality requirements, putting them into practice, measuring the right quality attributes, and assessing the final software quality.

Forecasting the errors modules might help to increase the quality of software. The method of creating models for software defect prediction is used in the initial stages of the SDLC to find flawed modules or classes. The modules can be categorized as defect-prone or not to accomplish this. The most prominent techniques for classifying modules are Decision Trees (DT), Naive Bayesian (NB), Random Forest (RF), and XGB classifier. In the testing phase of the SDLC, the identified defect-prone modules are given higher priority, while the non-defect-prone modules are tested as time and money permit. The classification function, also referred to as the classifier, develops classification rules by examining the relationship between the attributes and the class label of the training dataset. Numerous real-world issues, which are frequently very complicated in nature, require "special" algorithms to be solved. It is unreasonable, if not impossible, to constantly create specialized algorithms to deal with such issues. Such issues can be dealt with specifically thanks to machine learning algorithms. "With respect to some task and some performance measure , a computer program is said to learn from experience if its performance on task ,improves with experience. Supervised and unsupervised machine learning algorithms are the two different categories of machine learning algorithms.

A pre-defined set of training data is supplied into supervised machine learning algorithms. The algorithm then uses the training dataset's "experience" to develop rules that predict the class label for a fresh set of data. During the learning phase, mathematical procedures are used to create and enhance the prediction function. An attribute's input value and known output value are both present in the training data used in this step. The ML algorithm's anticipated value is contrasted with the output that is already known. After that, it is changed to improve its accuracy with regard to the forecast output. Until a threshold prediction accuracy is reached or the maximum number of cycles are carried out, this is continued throughout numerous generations of training data. The value of the class label in the data produced for unsupervised machine learning algorithms is unknown. Instead, a collection of data is entered into the program, and the algorithm searches through it for patterns and connections. The relationships between the attributes within the data are the key focus. A good illustration would be locating your friends' groups on a social networking site. A quality metric for assessing the complexity of a software program is cyclomatic complexity in software testing. It is a numerical assessment of independent software program source code pathways. A software program's functions, modules, methods, or classes can be used to calculate the cyclomatic complexity of the code. An independent path is one that has at least one edge that has not previously been traveled by any other pathways. Thomas J. McCabe created this measure in 1976, and it is based on a depiction of the program's control flow. A program is represented by control flow as a graph made up of nodes and edges.

Research Questions which we aim to answer this paper are:

RQ1: Which techniques and algorithms have been used in the software defect prediction (SDP)?

RQ2: Which type of software metrics are more influencing to predict the Defect of source code?

2. RESEARCH METHODLOGY

Scikit-learn is probably the most useful library for machine learning in Python. The sklearn library contains a lot of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction. It features various algorithms like support vector machine, random forests, and k-neighbours, and it also supports Python numerical and scientific libraries like NumPy and SciPy. To conduct the comparative analysis investigation, various algorithms were explored. The selection was aimed at including the most well-liked and often employed machine learning algorithms. The chosen algorithms are listed below, along with a brief explanation of each:

Decision Tree: Decision trees simply represent rules. The rules created using this method are simple for the human brain to understand. Additionally, they can be used directly in database languages like SQL to get records that belong to a particular class. Decision trees enable calculations to be performed both forward and backward, improving decision accuracy. Every node in a decision tree is either a Leaf node or a Decision node, forming a tree structure. When a specific test on an attribute with one branch and a sub-tree as a consequence of the test is required, decision nodes indicate branching, and leaf nodes show the value of the attribute given to them. This learning method maps the desired value conclusion of an item with an observation about the object using a decision tree as a predictive model. This method of predictive modeling is employed in machine learning, statistics, and data mining. To graphically categorize the decision-making process and the decisions, use a decision tree. This algorithm's objective is to create a structure that can predict the value of the target variable using a variety of input variables. One of the input variables is represented by each interior node. There are edges to their children for each value of these variables. The goal variable value is represented by a leaf in the tree, and the given values of the input variables can be navigated by a path from root to leaf.

Naive Bayes: The Naive Bayes classifier is an algorithm for administered learning that is used for formation grouping with statistical methodology. This probabilistic classifier uses a statistical probability distribution to characterize a certain input over a set of classifications. As suggested by its name, Naive, this method incorrectly presumes that the attributes of a specified class are unrelated. The Bayes theorem is then used to complete the grouping process by calculating the likelihood that a circumstance falls into the correct class given a set of specified features.

These belong to a family of straightforward probabilistic classifiers built on the Bayes theorem. Features make a strong or foolish presumption of independence. Building classifiers is a straightforward process. It functions as a model that is applied to problem objects as a vector that is used to extract class labels from finite sets. Naive classifiers believe that feature value is independent of any value that has class variables. This is not only an algorithm; it is an algorithm family that is based on these concepts. For instance, an apple is a fruit with a red color, a round form, and a diameter of roughly 10 cm. Each feature's likelihood is taken into account via naive Bayes. Take an apple as an example, which doesn't matter about its color, correlation, roundness, or diameter.

Random Forest: A supervised machine learning technique called a random forest is built from decision tree algorithms. This method is used to forecast behavior and results in a variety of industries, including banking and e-commerce. In a random forest algorithm, there are many different decision trees. The random forest algorithm creates a "forest" that is trained via bagging or bootstrap aggregation. The accuracy of machine learning algorithms is increased by bagging, an ensemble meta-algorithm. Based on the predictions of the decision trees, the (random forest) algorithm determines the result. It makes predictions by averaging or averaging out the results from different trees. The accuracy of the result grows as the number of trees increases. 2The decision tree algorithm's shortcomings are eliminated with a random forest. It improves precision and lowers dataset overfitting. Without requiring numerous configurations in packages, it generates forecasts.

Gradient Boosting (XGboost) Algorithm: Gradient boosting is a method standing out for its prediction speed and accuracy, particularly with large and complex datasets. This method has delivered the greatest results across a range of applications, including Kaggle competitions and commercial machine learning solutions. Errors are a significant factor in every machine learning system, as we already know. Bias error and variance error are the two basic categories of error. The gradient boost approach aids in reducing the model's bias inaccuracy. Before looking into the specifics of this algorithm, it is important to understand the AdaBoost Algorithm, another boosting technique. This algorithm begins by creating a decision stump, after which all the data points are given equal weights. The weights for all the incorrectly classified points are then increased, while those that are simple to classify or are correctly classified have their weights decreased. These weighted data points are given their own decision stump. The purpose of this is to enhance the first stump's forecasts.

3. LITERATURE REVIEW

This paper [1] established a standard to facilitate common and helpful comparisons across various bug prediction strategies. The study proposed a novel approach and assessed its performance by creating a strong comparison with other approaches using the provided benchmark. It also presented a full comparison between well-known bug prediction approaches.

In this paper [2], a machine learning (ML)-based prediction model for software bugs is presented. Based on past data, three supervised ML systems have been used to forecast potential software defects. These classifiers include artificial neural networks, decision trees, and naive bayes (NB). The evaluation method demonstrated that ML algorithms may be applied successfully and accurately. Three actual testing/debugging datasets are used to perform the evaluation process. On the basis of metrics for accuracy, precision, recall, F-measure, and RMSE, experimental findings are compiled. Results show that ML techniques are effective methods for anticipating software issues in the future.

R. Malhotra in [3] presented a thorough systematic review for machine learning-based software bug prediction strategies (ML).The paper reviewed all studies conducted between 1991 and 2013; it analyzed machine learning (ML) techniques for software bug prediction models; it evaluated their performance; it compared ML to statistical techniques; it compared various ML techniques; and it summarized the strengths and weaknesses of the ML techniques.

In order to increase the overall quality of software, this paper [4] focused on software quality metrics that assist in the detection of software defects utilizing data mining approaches. The authors have examined numerous categorization methods for predicting software defects. Different product measures were investigated, including lines of code (LOC), cohesion and coupling, QMOOD metrics, Class level metrics by McCabe, Chidamber and Kemerer metrics. Additional software defect prediction methods covered in this study included regression, association rule mining, clustering, and classification. Neural Networks, Decision Trees, Naive Bayes, Support Vector Machines, and Case Based Reasoning were among the several methods examined. Four alternative classification methods—supervised learning, semi-supervised learning, unsupervised learning, and machine learning—were examined, contrasted, and analyzed.

Software metrics are used in the field of software measuring. Software metrics are the numerical statistics that are used to describe a source code's characteristics and that are used to forecast software resource needs and software quality [5, 6]. Sequential measurements of a process' quality features might serve as a good starting point for managing process improvement initiatives [5]. Quantification and modeling of software metrics are necessary for the efficient management of any software development process. McCabe and Halstead [5] made the most significant advancements in software metrics. Among the most significant metrics that boosted interest and the quantity of studies in this field were McCabe's cyclomatic complexity and Halstead's program vocabulary.

The objective of this paper [7] is to investigate realistic guidelines for the selection of training data, classifier, and metric subset of a specific project, and to evaluate the feasibility of the predictor developed with a limited metric set for software defect prediction in various situation. The methods used by the author are three types of predictors were created in three scenarios using the size of the software metric set. And validated Top-k metrics in terms of statistical methods. They tested the stability of such a minimum metric subset with one-way ANOVA tests. The results show that when compared to benchmark predictors, predictors constructed using either Top-k metrics or the minimum metric subset can deliver an acceptable result.

In this paper [8]. The author used tree based machine learning models and boost based machine learning algorithms to predict the defect in the software. The mainly used algorithms in this paper are Ada boost, Gradient Boosting, Hist Gradient Boosting, XGBoost, CatBoost, Random Forest and Extra Trees. By performing training and testing the model the author concluded that the Tree based algorithms are performing compared to the boost based algorithms. In that the Ada Boost algorithms gives bad performance than other algorithms. Random Tree and Extra Trees performs well in predicts the defects.

This Paper [9] Mainly deals with analysis of the software defect prediction models. Statistical models have been evolved by the help of the several software metrics. The majority of the different prediction models predict defects using size and complexity parameters. Model validity is threatened by fundamental statistics and data quality issues. The author recommended holistic models for the software defect prediction. Bayesian Belief Networks, as alternative approaches to the single-issue models.

4. DATASET & EVALUATION

The dataset used in this project is software prediction dataset. Data comes from McCabe and Halstead features extractors of source code. These features were defined in the 70s in an attempt to objectively characterize code features that are associated with software quality. The nature of association is under dispute. In these software prediction dataset there are 22 (5 different lines of code measure, 3 McCabe metrics, 4 base Halstead measures, 8 derived Halstead measures, a branch-count, and 1 goal field). McCabe argued that code with complicated pathways are more error-prone. His metrics therefore reflect the pathways within a code module. Halstead argued that code that is hard to read is more likely to be fault prone. Halstead estimates reading complexity by counting the number of concepts in a module; e.g. number of unique operators.

The Project Mainly Divided into three phases. In the phase we are taking McCabe's attributes as our independent variables and training the model. In the second phase we are considering Halstead's attributes as independent variables and in the third phase combined of Halstead's and McCabe's are considered as independent variables and train the model. The programming for this project follows to the Data science pipeline design. Programming in Python is utilized. Data is first loaded into the data frame. Data cleaning is the following stage. The null values and unnecessary attributes are eliminated during the data cleaning process. Modeling the data is the subsequent stage. The data is divided into training and testing data, and the model is evaluated against it.

In our project we are taking only McCabe's and Halstead's attributes. Based on co-varience. The attributes which has highest co variance that attributes is considered as the independent variables. The primary goal of this study is to identify the measures that have the greatest impact on defect prediction. There are three phases to this project. The only independent variables in the first phase are McCabe's characteristics, the only independent variables in the second phase are Halstead's attributes, and the only independent variables in the third phase are a mixture of McCabe's and Halstead's attributes. In this project, 70% of the data were used for training and 30% were used for testing. Each model evaluated its own performance. The results were demonstrated by the ML models in terms of metrics like accuracy, precision, recall, f-1 score and ROC-AUC score.

5. RESULTS & ANALYSIS

We used a collection of widely used metrics to assess the effectiveness of applying ML algorithms to the prediction of software defects. The used evaluation measures are described in the following subsections.

Accuracy, precision, recall, and F1 score are some of the often used performance criteria for evaluating the learning algorithms mentioned above. Utilizing the confusion matrix, also known as an error matrix and a description of the predictions made for a classification task, the performance of each algorithm's model is assessed. The evaluation of the model is crucial for classification problems where the output can include two or more different classes, and the confusion matrix is one of the most popular and straightforward metrics for assessing the model's accuracy. The values of this parameter are True Positive (TP), True Negative (TN), False Positive (FP), and False Negative (FN).

Positive (P): Observation is positive (for example is an defective).

Negative (N): Observation is not positive (for example is not an defective).

True Positive (TP): The model has estimated true, and the test data is true.

False Negative (FN): The model has estimated false, and the test data is true.

True Negative (TN): The model has estimated false, and the test data is false.

False Positive (FP): The model has estimated true, and the test data is false.

Accuracy: Accuracy which is called classification rate is given by the following relation:

Accuracy = TP+TN/TP+TN+FP+FN

Recall: To get the value of Recall, correctly predicted positive observations is divided by all observations in actual class and it can be defined as below:

Recall = TP/TP+FN

Precision: Precision is the ratio of the total number of correctly classified positive examples to the number of predicted positive examples. As shown in Equation 4, As decreases the value of FP, precision increases and it indicates an example labeled as positive is indeed positive.

Precision = TP/TP+FP

F-measure/F1 score: Unlike recall and precision, this metric takes into account both false positives (FP) and false negatives (FN). F-measure is the weighted harmonic mean of the precision and recall of the test.

F1-Score = 2 * Recall * Precision /Recall + Precision

A Receiver Operating Characteristic curve, i.e., ROC curve is a graphic representation used to assess the true prediction capacity of a binary classifier system when its threshold is changed randomly. At various threshold values, the ROC curve plots the true positive rate on the x-axis and the false positive rate on the y-axis. The percentage of all actual positive values that the classifier correctly predicted is known as the true positive rate. The true positive rate and false positive rate are expressed at various probability values on the x and y axes for various intervals, and these metrics are shown. Greater accuracy of the model is shown by a higher area under the curve.

TPR = TP/TP+FN

FPR = FP/FP+TN

We obtained the results by doing these experiments empirically; the results are tabulated below. These helped us determine which one is more accurate.

As we know the project is divided into three phases. In each phase the performance metrics calculated and separately and compared the results with different phases.

The precision, recall, f-1 score and Roc-Auc score of all four models as shown below:

	ROC-AUC Score	Precision	Recall	F1-score
Decision tree	0.71	0.628	0.128	0.212
Random Forest	0.727	0.6	0.076	0.134
XGBoost Classifier	0.696	0.53	0.18	0.269
Naive Bayes	0.631	0.525	0.196	0.285

Tab1: performance metrics for the McCabe's Attributes

By Analyzing Phase1 Results. The Decision tree model has the highest precision and naïve bayes has the highest recall and F1-score. Decision tree has highest ROC-AUC curve as shown below figure:

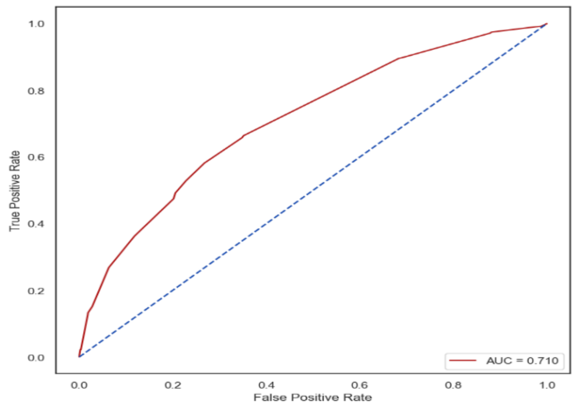

Fig1: Roc-Auc curve for McCabe's attributes for Decision tree model

	ROC-AUC Score	Precision	Recall	F1-score
Decision tree	0.695	0.55	0.017	0.034
Random Forest	0.712	0.75	0.038	0.072
XGBoost Classifier	0.698	0.492	0.139	0.216
Naive Bayes	0.641	0.48	0.169	0.25

Tab2: performance metrics for the Halstead's Attributes

By Analyzing Phase2 Results. The Random Forest model has the highest precision and naïve bayes has the highest recall and F1-score. Random Forest has highest ROC-AUC curve as shown below figure:

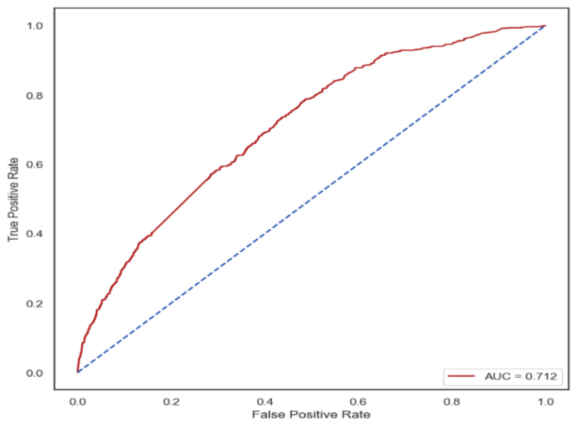

Fig2: Roc-Auc curve for Halsted's attributes for random Forest model

	ROC-AUC Score	Precision	Recall	F1-score
Decision tree	0.7	0.533	0.076	0.133
Random Forest	0.729	0.697	0.084	0.149
XGB Classifier	0.639	0.524	0.186	0.275
Naive Bayes	0.702	0.508	0.192	0.279

Tab3: performance metrics for the combined attributes

By Analyzing Phase3 Results. The Random Forest model has the highest precision and naïve bayes has the highest recall and F1-score. Random Forest has highest ROC-AUC curve as shown below figure:

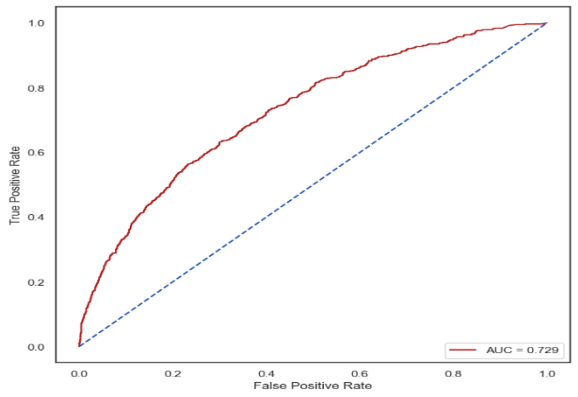

Fig3: Roc-Auc curve for combined attributes for Random Forest model

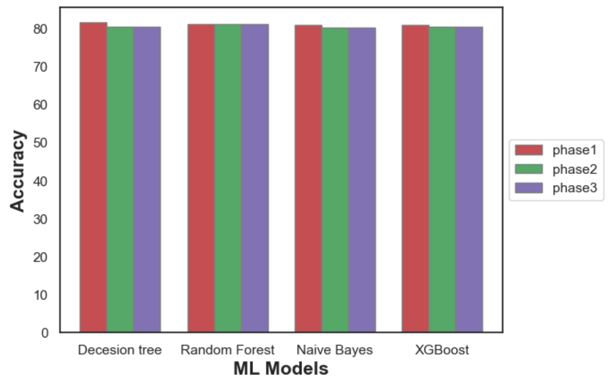

Fig4: Graphical Representation of Accuracy for ML Models

The above Bar graph shows that accuracies of the machine learning models in three phases. We can see that the accuracy is slightly increased in phase1 compared to phase2 and phase3. But for the random forest model the accuracy are almost similar in every phase. By this we concludes that McCabe's attributes are contributing more in the prediction of software defect.

By comparing all the performance metrics for all the models. Random forest model gives better performance when compared to all the models used in this project. In all phases the phase1 performance metrics are Better to the remaining phases. So McCabe's metrics are influencing in predicting the software defect for this dataset

6. Conclusion and Future Work

The Research Questions laid out in the beginning of this paper are answered as follows:

RQ1: Which techniques and algorithms have been used in the Software Defect Prediction(SDP)?

The machine learning models used in this project are Software Defect Prediction are Decision Trees(DT), Random Forest(RF), Naive Bayes(NB) and Gradient Boosting (XGB).

RQ2: Which type of software metrics are more influencing to predict the Defect of source code?

The most serious metrics used in practice to calculate the complexity. of source code. McCabe is based on the number of decision points; Halstead is based on the number of operands and operators in programs. When compared amongst all the Halsted's and McCabe's attributes, the models with only Mc Cabe as its independent variables did a better job with respect to the performance metrics obtained. So, to detect whether there is a presence of defects in a program only the Mc Cabe metrics are required for better results.

These Results can be further refined by using a greater number of algorithms. Also comparison can be done amongst more number of algorithms. Most popular and widespread algorithms were considered in this project. Hopefully new techniques will arise in future and could be included in the comparative analysis, providing new and improved results.

References:

[1]. D'Ambros, Marco, Michele Lanza, and Romain Robbes. "An extensive comparison of bug prediction approaches." Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on. IEEE, 2010.

[2]. Awni Hammouri, Mustafa Hammad, Mohammad Alnabhan, Fatima Alsarayrah. “Software Bug Prediction using Machine Learning Approach”. International Journal of Advanced Computer Science and Applications, Vol. 9, No. 2, 2018.

[3]. Malhotra, Ruchika. "A systematic review of machine learning techniques for software fault prediction." Applied Soft Computing 27 (2015): 504-518.

[4]. M. CM Prasad, L. Florence and A. Arya, “A Study on Software Metrics based Software Defect Prediction using Data Mining and Machine Learning Techniques”, International Journal of Database Theory and Application, Vol. 8, pp.179-190,2015

[5] Florac, A., E. Park and D. Carleton, “Practical Software Measurement: Measuring for Process Management and Improvement”, Software Engineering Institute, Carnegie Mellon University, April, 1997.

[6] Fenton, N., P. Krause and M. Neil, “Software Measurement: Uncertainty and Casual Modeling”, IEEE Software, July/August 2002.

[7]. Peng He, Bing Li, Xiao Liu, Jun Chen, Yutao Ma “An empirical study on software defect prediction with a simplified metric set”. https://doi.org/10.1016/j.infsof.2014.11.006.

[8]. Hamoud Aljamaan, Amal Alazba. “Software Defect Prediction using Tree-Based Ensembles”. PROMISE ’20, November 8–9, 2020, Virtual, USA.

[9]. Norman E. Fenton, Member, IEEE Computer Society, and Martin Neil, Member, IEEE Computer Society. “A Critique of Software Defect Prediction Models”. IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, VOL. 25, NO. 5, SEPTEMBER//OCTOBER 1999

[10]. Evren Ceylan, F Onur Kutlubay, and Ayse B Bener. Software defect identification using machine learning techniques. In 32nd EUROMICRO Conference on Software Engineering and Advanced Applications (EUROMICRO’06), pages 240–247. IEEE, 2006.

[11]. Jinsheng Ren, Ke Qin, Ying Ma, and Guangchun Luo. On software defect prediction using machine learning. Journal of Applied Mathematics, 2014, 2014.

[12]. J. Tian, and M.V. Zelkowitz. Complexity measure evaluation and selection, IEEE Transactions on Software Engineering, 21(8), 641-650, 1995

[13]. S. Aleem, L.F. Capretz, and F. Ahmed. Benchmarking machine learning technologies for software defect detection. International Journal of Software Engineering & Applications (IJSEA), 6(3), pp. 11-23, 2015.

[14]. N. Li, M. Shepperd, and Y. Guo. A systematic review of unsupervised learning techniques for software defect prediction, Information and Software Technology, Vol. 122, 106287, 2020

[15]. S. Karim, H. L. H. S. Warnars, F.L. Gaol, E. Abdurachman, and B. Soewito. Software metrics for fault prediction using machine learning approaches: A literature review with PROMISE repository dataset. IEEE International Conference on Cybernetics and Computational Intelligence (CyberneticsCom), pp. 19-23, 2017

[16]. M.S. Naidu, and N. Geethanjali. Classification of defects in software using decision tree algorithm. International Journal of Engineering Science and Technology, 5(6), 1332, 2013.

[17]. M. Shepperd, D. Bowes, and T. Hall. Researcher bias: The use of machine learning in software defect prediction. IEEE Transactions on Software Engineering, 40(6), 603-616, 2014.

[18]. K. Srinivasan, and D. Fisher. Machine learning approaches to estimating software development effort. IEEE Transactions on Software Engineering, 21(2), 126-137, 1995.

[19]. Xia, Y., Yan, G., Jiang, X., & Yang, Y. (2014, May). A new metrics selection method for software defect prediction. In 2014 IEEE International Conference on Progress in Informatics and Computing (pp. 433-436). IEEE.

[20]. Rhmann, W., Pandey, B., Ansari, G., & Pandey, D. K. (2020). Software fault prediction based on change metrics using hybrid algorithms: An empirical study. Journal of King Saud University-Computer and Information Sciences, 32(4), 419-424.

[21]. H. Lu, B. Cukic, and M. Culp, "Software defect prediction using semisupervised learning with dimension reduction," in Automated Software Engineering (ASE), 2012 Proceedings of the 27th IEEE/ACM International Conference on. IEEE, 2012, pp. 314–317.

[22]. Sharadhi A. K, Vybhavi Gururaj, Keerti R. Umadi, Mushkan Kumar, Sahana P. Shankar, Deepak Varadam, "Comprehensive Survey of different Machine Learning Algorithms used for Software Defect Prediction", *2022 International Conference on Decision Aid Sciences and Applications (DASA)*, pp.425-430, 2022.

[23]. Ma. José Hemández-Molinos, Ángel J. Sánchez-García, R. Erandi Barrientos-Martínez, "Classification Algorithms for Software Defect Prediction: A Systematic Literature Review", *2021 9th International Conference in Software Engineering Research and Innovation (CONISOFT)*, pp.189-196, 2021.

[24]. Md Nasir Uddin, Bixin Li, Md Naim Mondol, Md Mostafizur Rahman, Md Suman Mia, Elizabeth Lisa Mondol, "SDP-ML: An Automated Approach of Software Defect Prediction employing Machine Learning Techniques", *2021 International Conference on Electronics, Communications and Information Technology (ICECIT)*, pp.1-4, 2021.

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

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

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

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

[29]. Rathore, S.S. and Kumar, S. (2016) A Decision Tree Regression Based Approach for the Number of Software Faults Prediction. ACM SIGSOFT Softw Are Engineering Notes, 41, 1-6.

[30]. Wang, T. and Li, W. (2010) Naive Bayes Software Defect Prediction Model. 2010 International Conference on Computational Intelligence and Software Engineering, 10-12 December 2010, Wuhan, 1-4.

[31]. Aleem, S., Capretz, L. and Ahmed, F. (2015) Benchmarking Machine Learning Technologies for Software Defect Detection. International Journal of Software Engineering & Applications, 6, 11-23

[32]. Jindal, R., Malhotra, R. and Jain, A. (2014) Software Defect Prediction Using Neural Networks. Proceedings of 3rd International Conference on Reliability, Infocom Technologies and Optimization, 8-10 October 2014, Noida, 1-6

[33]. Deep Singh, P. and Chug, A. (2017) Software Defect Prediction Analysis Using Machine Learning Algorithms. 2017 7th International Conference on Cloud Computing, Data Science Engineering-Confluence, 12-13 January 2017, Noida, 775-781

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

[35]. Hammouri, A., Hammad, M., Alnabhan, M. and Alsarayra, F. (2018) Software Bug Prediction Using Machine Learning Approach. International Journal of Advanced Computer Science and Applications, 9, 78-83.