**International Journal of Research Publication and Reviews**

Journal homepage: www.ijrpr.com ISSN 2582-7421

# Using Machine Learning Techniques in Predicting Software Defects

***Mr. Vinesh Patel[1], Mr. Abhisek Panday[2]***

[1,2]Department of Computer Sciences and Engineering, Takshshila Institute of Engineering & Technology, Jabalpur (M.P.)

**ABSTRACT**

A In fields such as aerospace, healthcare, manufacturing, and robotics, H systems are highly reliable because they rely on a variety of software systems. Software failure prediction helps improve software reliability by identifying potential failures during software maintenance. Traditionally, the focus of software error prediction has been on the design of static code metrics that help predict the error probability of code when fed into machine learning classifiers. Machine learning techniques such as deep learning, ensembles, data mining, clustering, and classification are known to help predict the location of errors in codebases, but researchers are wondering which is the best predictive model. have not yet agreed on This white paper evaluates the performance of various predictive models using 13 software error datasets. Results show that highly accurate and consistent predictions were achieved by using the ensemble method.

**Key words:** software, software defect prediction, machine learning, classification, clustering, ensemble learning.

## 1. INTRODUCTION

As software dependency and complexity increase, so does the requirement for maintainable, high-quality, low-cost software. However, in order to reduce maintenance and improve software quality in software operation, software defect prediction systems are needed [1,2,3]. The existence of early detection systems allows for the timely correction of detected errors, thus speeding the delivery of maintainable software [4]. A number of studies have developed specific metrics that can be used as the basis of models for detecting errors that software may encounter during operation early in the software development lifecycle..

Over the past 30 years, there has been a growing interest in predicting software failures in the software engineering field. The scope of the current error prediction includes (a) classifying the error susceptibility of software components into error-prone and error-prone classes, and (b) identifying relationships between errors. and (c) estimating remaining defects. software system [5]. For the purpose of this investigation, our focus remains on the first range.

Software Defect Prediction (SDP) modules/classes can be divided into two categories: error prone and error less prone. SDP models can be built using error data and software metrics from previous software versions or similar software projects [6,7]. After building the model, it can be integrated into ongoing projects and helps classify all modules/classes as error-prone or not [8]. Based on these results, software professionals can make informed decisions to tackle all error-prone areas in the early stages of development. For example, if only 30% of your testing resources are allocated to a particular piece of software, knowing all of your error prone areas ensures that all available resources are dedicated to fixing modules/classes in those areas. can be assigned [9] . This creates a high level of quality and maintainability that is of high quality and produced within a given time frame and budget [10].

A significant part of SDP research activity focuses on detecting whether software components are error-prone by relying on the use of code-derived software metrics [11]. Various machine learning algorithms have been used to classify software components as error-prone or not, but attempts to refine the rules and patterns in the data have led to consistent failures. has never been proven. These techniques used include mixture algorithms, parametric models, machine learning techniques, and statistical techniques. However, before deciding if this problem is nearly unsolvable, we need to identify the best prediction techniques to help predict the problem based on context.

This study will rely on open source software repositories to investigate key software defect prediction models such as ensemble techniques, clustering and classification [6]. By giving clues about these models, and how they react with different datasets, we do hope that results obtained in this study will help increase confidence in them. Key findings of this study show that the use of stacking multiple classifiers can be of use to defect prediction.

## 2. LITERATURE REVIEW

There are four types of machine learning tasks: reinforcement learning, semi-supervised learning, unsupervised learning, and supervised learning. Supervised and unsupervised learning are still the most popular groups of tasks.

Supervised learning is a machine learning technique that uses labeled training data containing a variety of training examples to infer a function. Training examples consist of input objects and desired output values, and include regression and classification supervised learning tasks [12]. Regression classification tasks focus on building continuous domain models, whereas classification learning tasks focus on building predictive models that work

within a discrete domain. Examples of supervised machine learning techniques include support vector machines, neural networks, linear regression, Bayesian learning, instance-based learning, rule learning, and learning classification. [13].

Unsupervised learning allows a system to examine data and identify patterns driven by common examples without knowing the existence or number of patterns in the data set. This is also known as learning from observation, and important examples are clustering, sequential pattern mining, and association rule mining [13,14].

The rapid growth of machine learning research has led to the development of different learning algorithms that can be used in different applications [15]. Moreover, the ability of machine learning algorithms to solve real-world problems often determines their ultimate value, so replication and application of algorithms on new tasks is essential for progress in this field. However, the current state of research indicates a large number of publications on the development of software error prediction models. These can be classified based on ensembles, clustering and taxonomy methods.

### *A. Classification Methods*

A study from [6] records performance measurements of 0.8573, 0.8685 and 0.7795 when using algorithms ANFIS, ANN and SVM. Their research is based on data from the PROMISE Software Engineering Repository. The study also used McCabe software metrics. To reduce time and cost by determining the total number of bugs using the ID3 classification algorithm,

Naidu & Geethanjali [16] finally categorized bugs into time, effort, difficulty, length, program, and five parameters including estimators. Singh and Salaria [9] used the Levenberg-Marquardt (LM) algorithm-based neural network tool to develop the model and the errors in initial software testing of all data sourced from the PROMISE repository of empirical software development data. susceptibility was investigated. We then compared the accuracy of LM with that of neural networks based on polynomial functions. LM recorded 88.1% higher accuracy than neural networks based on polynomial functions.

Aleem et al. [5] Investigation using various machine learning techniques with 15 datasets (KC3, KC1, CM1, AR6, AR1, etc.) followed by bagging, multi-layer perceptron (MLP), and support vector machines (SVM) achieved a high level. performance and accuracy. A study conducted by

[17] relied on a new benchmarking framework for evaluating and predicting software errors. Activities included evaluating and comparing different learning schemes with a selection and using them to create predictors containing all historical data [1]. This predictor is ready to predict errors in new data.

B. Clustering method

Using Feature Clusters to Improve the Performance of Software Predictive Models, Tan et al. [10] Improve model accuracy and performance from 73.8% and 31.6% to 91.6% and 99.2%, respectively.

According to Kaur and Sandhu [18], the k-means-based clustering approach has an accuracy of 62.4% in object-oriented programming error susceptibility. Model building relied on EM and X-Means clustering algorithms derived from AR3. AR4 and AR5 promise repository data to help predict software failures. Experiments that normalize dataset 0 to 1 and use CfsSubsetEval as the subsequently applied attribute selection algorithm yield an accuracy level of (90.48) for the X-means clustering algorithm on dataset AR3. models.

### *B. Ensemble approaches*

In an attempt to address the use of the ensemble approach in software fault prediction, Shanthini & Chandrasekaran [19] tried to use the ensemble approach to conduct model building. The data was categorized into package level, class level and method level. The metrics used in the method and class level paired with the data for the package relied on NASA KCI data using ensemble methods such as voting, staking, boosting and bagging. From the experiment, bagging was a better ensemble method compared to the rest at both the package and method level [20]. When using the AUC-curve at the method level, the performance measurement includes voting (0.63), staking (0.79), boosting (0.782) and bagging (0.809). As for the package level, the performance measurement was voting (0.76), staking (0.72), boosting (0.78) and bagging (0.82). The metric level recorded the following (0.82), staking (0.8), boosting (0.74) and bagging (0.78), although not similar to other metrics relying on the AUC-curve.

A study by Kaur & Malhotra [15] recorded an AUC of 0.81, an F measure of 75, a recall of 79%, an accuracy of 72%, and an accuracy of RF of 74.24%. In an experiment, Kaur and Malhotra relied on her JEdit open source software with object-oriented metrics in evaluating the use of random forests by open source software to predict error-prone classes.

Peng et al. [20] used an analytical hierarchical process to evaluate the use of ensemble approaches in software failure prediction. They plotted 10 public NASA MDP data sets based on 13 different power measurements. The ensemble method used is staking, boosting and bagging, recording a 92.53% result accuracy on a decision tree basis classifier.

## 3. SDP MACHINE LEARNING ALGORITHMS

This section Indicates the algorithm used for evaluation. In this study, we compared supervised learning algorithms such as Linear SVC, Maximum Vote Classifier, Wright GBM, Gaussian Naive Bayes Classifier, Passive Aggressive Classifier, Xgboost, and Extra Tree Classifier. Unsupervised learning methods included comparing clustering methods such as stacking classifiers, GMMs, and mini-batch k-means algorithms to each other. other.

### *A. Maximum Voting Classifier (MVC)*

The maximum A poll always contains a set of classifiers that make predictions and test the resulting data. The final prediction is determined by looking at who has the most votes (more than half) [21]. You can combine different classifiers to improve the accuracy of this algorithm. His

maximum votes classifier for this study works as follows way:

a) Used both ET and RF classifiers on the training data

b) Record the performance of both classifiers and come up with a comparison

c) Conduct voting with every step/observation

### *B. Extra Tree Classifier (ET)*

This algorithm It works even better by randomizing the tree construction by the number of numerical inputs when most of the variance of the induced tree falls on the selection of the best intersections [22]. Instead of finding the best truncation point for all
randomly selected K features at each node, the algorithm switches to using bootstrap copying in a random forest vase. The selection of intersection points is random. This method works best when you have a large number of different numerical properties. This method of smoothing improves accuracy while at the same time optimal cut-off points in both random forests and standard trees.

### *C. Passive Aggressive Classifier (PAC)*

Under Passive, If the model falls into the correct classification type, the model is retained. Aggressive mode requires updating each misclassification to reflect the misclassified example. Under passive circumstances, lack of sufficient information has been shown to prevent updates, whereas under proactive circumstances, better models help correct errors the previous time.

### *D. Xgboost*

Xgboost is A tool from the Distributed Machine Learning Community (DMLC), popular for its increased power and speed when it comes to gradient-enhanced decision trees. Xgboost was first designed and used by Tianqi Chen that year (1995) and proved to be a way to give machines a boost. It has gone through many iterations of various iterations developers.
In tree A boosting algorithm, eXtreme or XGBoost, is used to help utilize all available hardware and memory resources, enabling deployment in computing environments, tuning models, and improving algorithms [ 20]. There are three methods of gradient boosting in XGBoost: stochastic boosting, regularized boosting, and gradient boosting. Moreover, it is very effective in adjusting and adding regularization parameters, making optimal use of memory resources, and reducing the time spent on computational work. XGBoost can also manipulate additional data in the trained model to allow parallel structures and handle missing values. (Sparse Aware).

### *E. Light Gradient Boost Model (LGBM)*

When When determining optimality, the algorithm assumes that k-means is optimized to generate more centroids when the k-means algorithm uses one-pass-over input data. Having to traverse a large dataset increases the cost of large computations, so multiple passes over the input data reduce execution time. In a simplified version, the algorithm uses incoming points as the basis for new clusters or assigns them to nearby clusters and uses adaptive scale to determine the distance to the closest cluster. parameter.

### *F. Gaussian Mixture Model (GMM)*

This is a parametric model used in examining the probability distribution of features or continuous measurements
in the form of a weighted sum of Gaussian component densities (parametric probability density function). The estimation of GMM parameters is done using the iterative Expectation-Maximizations (EM) algorithm from a prior model to the training data.

### *G. Stacking Classifier*

In the In the world of Netflix and other competitors, stacking has been proven as one of the Ensemling methods in machine learning [23]. The main idea of this algorithm is to use confidence values as features when combining multiple models and train a meta-classifier that helps combine predictions of multiple learners. We used three classifiers in this study, including the random forest classifier, Adaboot, and KNN. These rely on logistic regression to aid in testing and training all type of slots.

### *H. Gaussian Naïve Bayes (GNB)*

This classification It works for both multiclass and binary (two-class) classification problems, and is a fairly straightforward algorithm when explained with categorical or binary input values [16]. Naive Bayes allows an extension to real-valued attributes, also known as Gaussian Naive Bayes. Working with the normal distribution (Gaussian) is fairly straightforward. Estimate the standard deviation using the training data, mean.

### *I. Quadratic Discriminant Analysis (QDA)*

This is a A generative probabilistic method for classifying problems. The QDA conditional distribution is assumed to be a multivariate Gaussian distribution derived using the posterior distribution of Bayes' theorem, and thus is used to classify observations between different classes. Traditionally, QDA has been estimated by maximizing the joint probabilities of observations and their corresponding associated classes. labels.

## 4. METHODOLOGY

This section presents the methodological tools, steps and procedures used in achieving the study objectives.

### *Data Preparation*

The use of Machine learning techniques are very important for software reusability, maintainability and quality as they help find the root odors, ambiguities, bugs and flaws in software. Achieving this requires software failure prediction techniques based on statistical software failure techniques [24]. However, machine learning can also be used for software detection. techniques. Pre-processing It helps put data into a format that can be used by classification engines [25,26]. For example, when images are used as input data, preprocessing simplifies the feature selection process by sharpening the image or rotating the image to transform it into a standard orientation and position. If the input comes from a dataset or data vector, preprocessing may involve filtering out the input using statistical properties of the dataset or prior criteria. Key benefits of preprocessing include support for normalizing numerical data and padding missing data.

The experiment relies on the PROMISE dataset collected from real NASA software projects and involves various software modules. Benchmarks involved using publicly available datasets. This benchmarking process allows other researchers to compare their research. Code metrics used in the dataset include McCabe's cyclomatic complexity, Halstead's code size, and complexity. A description of the dataset is summarized in Table 1. Target variables in NASA MDP datasets are binary in nature, 1: yes, 0: no. Table 2 shows the performance evaluation matrix used in this study. Python programming and scikit-learn (a machine learning framework) are used with the data. examination.

**Table 1:** Description of NASA MDP DATSETS

| Variables | Description | Metrics Type |
|---|---|---|
| loc | Line count of Code | McCabe |
| v(g) | Cyclomatic Complexity | McCabe |
| ev(g) | Essential Complexity | McCabe |
| iv(g) | Design Complexity | Halstead |
| n | Total operators and Operands | Halstead |
| v | Volume | Halstead |
| l | Program Length | Halstead |
| d | Difficulty | Halstead |
| i | Intelligence | Halstead |
| e | Effort | Halstead |
| b | Number of Bugs | Halstead |
| t | Time estimator | Halstead |
| lO Code | Line Count | Halstead |
| lO Comment | Line count of Comments | Halstead |
| lO Blank | Count of Blank Lines | Halstead |
| lO Code And Comment | Lines of Comment and Code | N/A |
| Uniq_Op | Unique Operators | Halstead |
| Uniq_Opnd | Unique Operands | Halstead |
| Total_Op | Total Operators | Halstead |
| Total_Opnd | Total Operands | Halstead |
| branchCount | Flow Graph’s Branch Count | Halstead |
| defects | Reported Defects | N/A |

**Table 2:** Performance Matrices

| Performance Matrices | Formula |
|---|---|
| Accuracy | $\frac{TP + TN}{TP + TN + FP + FN}$ |
| F1 | $\frac{2 * Recall * Precision}{Recall + Precision}$ |
| MAE | \|True values-Predicted values \| |

***Feature Extraction***

Feature Extraction facilitates transforming preprocessed data into a form that can be used by pattern recognition engines. Pattern recognition algorithms exhibit varying degrees of sensitivity with respect to the form of data provided and thus the need for feature selection. In this study, we used Random Forest's Feature Importance Score to find the best features for everyone. algorithms.

***Classification***

Solving Transformation issues have enabled the creation of numerous classification algorithms that can be adapted to handle error streams, fragments, or source code tokens. Each classifier has different strengths and weaknesses aimed at meeting specific needs. Finally, the performance of the aforementioned algorithms is measured using performance metrics. Table 2.

## 5. RESULTS

This section discusses the results of the different ML techniques for defect prediction using various datasets are shown in Table 3, 4,5,6,7 and 8. The training was performed based on 10-fold cross validation.

**Table 3:** Performance of Supervised and Unsupervised Learning Algorithms

| | Supervised Learning | | | | Unsupervised Learning | | |
|---|---|---|---|---|---|---|---|
| Dataset | Perceptron | PAC | QDA | GNB | MiniBatch K-mean | KNN | GMM |
| AR1 | 92.63 | 89.29 | 91.79 | 77.56 | 90.06 | 84.42 | 84.17 |
| AR6 | 74.66 | 72.73 | 87.69 | 84.35 | 60.02 | 78.4 | 63.55 |
| CM1 | 57.01 | 76.87 | 84.73 | 81.32 | 68.05 | 81.92 | 87.176 |
| JM1 | 58.43 | 71.15 | 79.93 | 80.44 | 37.63 | 78.89 | 80.66 |
| KC1 | 69.44 | 76.45 | 81.25 | 82.34 | 83.96 | 78.89 | 68.7 |
| KC2 | 45.65 | 62.32 | 82.29 | 82.85 | 68.37 | 60.37 | 79.51 |
| KC3 | 67.96 | 65.02 | 86.43 | 86.24 | 83.75 | 84.95 | 90.62 |
| MC1 | 97.64 | 92.64 | 97.18 | 97.91 | 32.4 | 97.18 | 97.69 |
| MC2 | 48.99 | 67 | 75.21 | 72.7 | 58.8 | 60.31 | 64.4 |
| MW1 | 97.64 | 92.64 | 97.18 | 92.91 | 32.4 | 97.18 | 97.69 |
| PC1 | 67.65 | 78.02 | 90.27 | 89.19 | 55.1 | 89.01 | 93.06 |
| PC2 | 97.58 | 93.3 | 97.86 | 91.5 | 86.66 | 95.17 | 88.39 |
| PC3 | 62.7 | 52.9 | 47.34 | 20.09 | 80.71 | 80.13 | 87.56 |
| PC4 | 79.14 | 76.89 | 50.59 | 86.01 | 55.9 | 79.88 | 68.77 |
| PC5 | 60.07 | 95.67 | 95.53 | 97.14 | 59.67 | 95.65 | 97 |
| Mean | 71.81 | 77.53 | 83.02 | 81.50 | 63.57 | 82.82 | 83.26 |

**Table 4:** Performance of Ensemble Learning Algorithms

| | Ensemble Learning | | | | | |
|---|---|---|---|---|---|---|
| Dataset | RF | ET | XgBoost | LGBM | STC | MVC |
| AR1 | 87.63 | 91.79 | 92.56 | 92.3 | 91.8 | 92.56 |
| AR6 | 85.67 | 82.74 | 84.56 | 72.73 | 85.67 | 84.56 |
| CM1 | 86.24 | 85.66 | 85.01 | 81.82 | 84.7 | 85.02 |
| JM1 | 80.84 | 79.14 | 78.1 | 80.8 | 78.1 | 78.1 |
| KC1 | 85.38 | 83.2 | 82.87 | 81.51 | 84.06 | 82.87 |
| KC2 | 81.27 | 78.78 | 79.24 | 66.03 | 81.73 | 79.24 |
| KC3 | 83.41 | 80 | 90.36 | 89.13 | 89.72 | 90.36 |
| MC1 | 97.69 | 97.99 | 97.84 | 97.49 | 97.84 | 97.84 |
| MC2 | 72.71 | 72.7 | 71.49 | 70.59 | 72.11 | 71.49 |
| MW1 | 97.56 | 97.99 | 97.84 | 97.49 | 97.84 | 97.84 |
| PC1 | 92.97 | 93.43 | 92.26 | 92.79 | 93.33 | 92.26 |
| PC2 | 97.46 | 97.86 | 97.45 | 97.33 | 97.72 | 97.45 |
| PC3 | 87.46 | 87.84 | 85.19 | 87.01 | 87.65 | 87.01 |
| PC4 | 89.2 | 90.37 | 90.75 | 85.27 | 90.13 | 90.75 |
| PC5 | 97.18 | 96.95 | 96.64 | 96.97 | 97.09 | 96.64 |
| Mean | 88.18 | 87.76 | 88.14 | 85.95 | 88.63 | 88.27 |

**Table 5:** F-measure Performance of Supervised and Unsupervised Learning Algorithms

| | Supervised Learning | | | | Unsupervised Learning | | |
|---|---|---|---|---|---|---|---|
| Dataset | Perceptron | PAC | QDA | GNB | MiniBatch K-mean | KNN | GMM |
| AR1 | 89.01 | 87.84 | 88.67 | 79.94 | 90.02 | 85.43 | 81.15 |
| AR6 | 69.1 | 71.6 | 82.08 | 81.22 | 59.35 | 76.6 | 55.72 |
| CM1 | 50.09 | 63.44 | 82.23 | 81.29 | 67.82 | 81.54 | 81.2 |
| JM1 | 56.74 | 63.12 | 76.6 | 75.91 | 35.61 | 71.3 | 72.02 |
| KC1 | 67.98 | 72.12 | 81.36 | 81.84 | 82.02 | 78.6 | 71.41 |
| KC2 | 38.38 | 50.35 | 79.52 | 80.85 | 62.4 | 62.8 | 70.43 |
| KC3 | 64.01 | 73.18 | 84.58 | 86.8 | 84.26 | 84.48 | 86.17 |
| MC1 | 96.52 | 95.12 | 96.8 | 94.18 | 34.14 | 97.02 | 96.54 |
| MC2 | 34.15 | 65.21 | 70.28 | 67.98 | 48.13 | 59.3 | 51.05 |
| MW1 | 96.52 | 95.12 | 96.8 | 94.18 | 34.14 | 97.02 | 96.54 |
| PC1 | 63.73 | 80.75 | 89.81 | 89.37 | 56.03 | 88.89 | 89.71 |
| PC2 | 96.7 | 94.44 | 96.8 | 93.46 | 86.552 | 95.42 | 87 |
| PC3 | 58.86 | 58.66 | 54.56 | 15.96 | 80.57 | 79.9 | 81.75 |
| PC4 | 72.5 | 75.67 | 56.45 | 83.9 | 51.08 | 79.74 | 63.14 |
| PC5 | 65.73 | 95.75 | 95.9 | 96.64 | 59.73 | 95.77 | 95.52 |
| Mean | 68.00 | 76.16 | 82.16 | 80.23 | 62.12 | 82.25 | 78.62 |

**Table 6:** F-measure Performance of Ensemble Learning Algorithms

| | Ensemble Learning | | | | | |
|---|---|---|---|---|---|---|
| Dataset | RF | ET | XgBoost | LGBM | STC | MVC |
| AR1 | 86.22 | 89.18 | 90.32 | 91.3 | 88.67 | 89.1 |
| AR6 | 81.02 | 79.1 | 81.77 | 71.89 | 82.63 | 81.6 |
| CM1 | 81.69 | 81.2 | 82.66 | 80.86 | 84.71 | 80.89 |
| JM1 | 74.97 | 75.59 | 75.96 | 73.84 | 75.7 | 73.13 |
| KC1 | 81.69 | 81.19 | 81.47 | 80.68 | 81.65 | 80.2 |
| KC2 | 81.27 | 78.78 | 78.5 | 72.46 | 78.78 | 80.49 |
| KC3 | 85.83 | 87.57 | 88.96 | 90.73 | 89.72 | 85.94 |
| MC1 | 96.54 | 97.63 | 97.24 | 93.5 | 97.84 | 96.54 |
| MC2 | 68.74 | 70.82 | 69.73 | 75 | 72.11 | 62.61 |
| MW1 | 96.58 | 97.63 | 97.24 | 93.5 | 97.23 | 96.54 |
| PC1 | 55.03 | 92.1 | 91.78 | 89.32 | 91.6 | 89.71 |
| PC2 | 96.6 | 96.8 | 96.59 | 84.93 | 96.73 | 96.8 |
| PC3 | 82.93 | 84.42 | 85.3 | 90.15 | 84.24 | 81.42 |
| PC4 | 86.56 | 88.85 | 90.21 | 94.07 | 90.13 | 80.77 |
| PC5 | 96.55 | 96.7 | 96.46 | 97.06 | 96.63 | 96.27 |
| Mean | 83.48 | 86.50 | 86.95 | 85.29 | 87.22 | 84.80 |

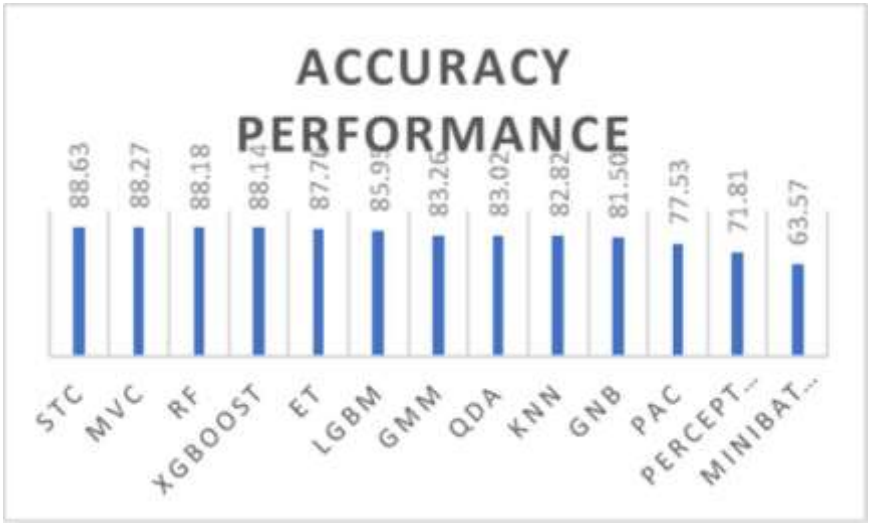
**Table 7:** MAE Performance of Supervised and Unsupervised Learning Algorithms

| Supervised Learning | | | | | Unsupervised Learning | | |
|---|---|---|---|---|---|---|---|
| Dataset | Perceptron | PAC | QDA | GNB | MiniBatch K-mean | KNN | GMM |
| AR1 | 0.07 | 0.11 | 0.08 | 0.22 | 0.10 | 0.16 | 0.16 |
| AR6 | 0.25 | 0.27 | 0.12 | 0.16 | 0.40 | 0.22 | 0.36 |
| CM1 | 0.43 | 0.23 | 0.15 | 0.19 | 0.32 | 0.18 | 0.13 |
| JM1 | 0.42 | 0.29 | 0.20 | 0.20 | 0.62 | 0.21 | 0.19 |
| KC1 | 0.31 | 0.24 | 0.19 | 0.18 | 0.16 | 0.21 | 0.31 |
| KC2 | 0.54 | 0.38 | 0.18 | 0.17 | 0.32 | 0.40 | 0.20 |
| KC3 | 0.32 | 0.35 | 0.14 | 0.14 | 0.16 | 0.15 | 0.09 |
| MC1 | 0.02 | 0.07 | 0.03 | 0.02 | 0.68 | 0.03 | 0.02 |
| MC2 | 0.51 | 0.33 | 0.25 | 0.27 | 0.41 | 0.40 | 0.36 |
| MW1 | 0.02 | 0.07 | 0.03 | 0.07 | 0.68 | 0.03 | 0.02 |
| PC1 | 0.32 | 0.22 | 0.10 | 0.11 | 0.45 | 0.11 | 0.07 |
| PC2 | 0.02 | 0.07 | 0.02 | 0.09 | 0.13 | 0.05 | 0.12 |
| PC3 | 0.37 | 0.47 | 0.53 | 0.80 | 0.19 | 0.20 | 0.12 |
| PC4 | 0.21 | 0.23 | 0.49 | 0.14 | 0.44 | 0.20 | 0.31 |
| PC5 | 0.40 | 0.04 | 0.04 | 0.03 | 0.40 | 0.04 | 0.03 |
| Mean | 0.28 | 0.22 | 0.17 | 0.18 | 0.36 | 0.17 | 0.17 |

**Table 8:** MAE Performance of Ensemble Learning Algorithms

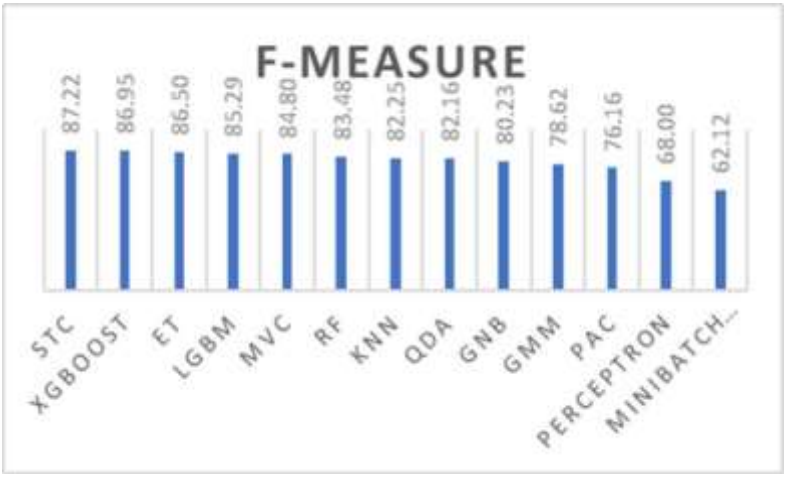
| | Ensemble Learning | | | | | |
|---|---|---|---|---|---|---|
| Dataset | RF | ET | XgBoost | LGBM | STC | MVC |
| AR1 | 0.12 | 0.08 | 0.07 | 0.08 | 0.08 | 0.07 |
| AR6 | 0.14 | 0.17 | 0.15 | 0.27 | 0.14 | 0.15 |
| CM1 | 0.14 | 0.14 | 0.15 | 0.18 | 0.15 | 0.15 |
| JM1 | 0.19 | 0.21 | 0.22 | 0.19 | 0.22 | 0.22 |
| KC1 | 0.15 | 0.17 | 0.17 | 0.18 | 0.16 | 0.17 |
| KC2 | 0.19 | 0.21 | 0.21 | 0.34 | 0.18 | 0.21 |
| KC3 | 0.17 | 0.20 | 0.10 | 0.11 | 0.10 | 0.10 |
| MC1 | 0.02 | 0.02 | 0.02 | 0.03 | 0.02 | 0.02 |
| MC2 | 0.27 | 0.27 | 0.29 | 0.29 | 0.28 | 0.29 |
| MW1 | 0.02 | 0.02 | 0.02 | 0.03 | 0.02 | 0.02 |
| PC1 | 0.07 | 0.07 | 0.08 | 0.07 | 0.07 | 0.08 |
| PC2 | 0.03 | 0.02 | 0.03 | 0.03 | 0.02 | 0.03 |
| PC3 | 0.13 | 0.12 | 0.15 | 0.13 | 0.12 | 0.13 |
| PC4 | 0.11 | 0.10 | 0.09 | 0.15 | 0.10 | 0.09 |
| PC5 | 0.03 | 0.03 | 0.03 | 0.03 | 0.03 | 0.03 |
| Mean | 0.12 | 0.12 | 0.12 | 0.14 | 0.11 | 0.12 |

**Figure 1:** Accuracy Chart of Different algorithms



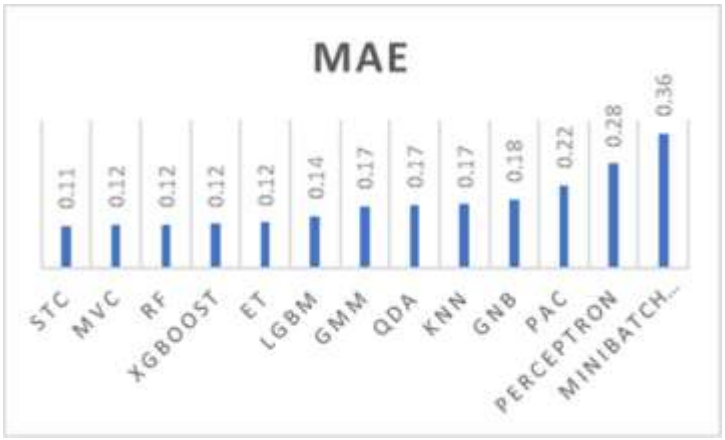

Based on Accuracy charts (Fig. 1) clearly show that the Stacking Classifier (STC) proposed in this study performs better compared to other algorithms. All ensemble classifiers outperformed other supervised and unsupervised learning methods in accuracy measures. For classification algorithms, QDA performed better than other algorithms, and GMM performed better than other clustering algorithms. Relatively good performance of clustering algorithms between classification and clustering algorithms well.

**Figure 2:** F-Measure of Different Algorithms




The above The graph (Fig. 2) represents the average F-measures of the machine learning algorithms across all 13 datasets.Based on the F-measures, STC remains on the top list, while all ensemble classifiers consistently It performs higher than other algorithms. Among all supervised learning algorithms, QDA performed relatively well, followed by GNB. For unsupervised learning algorithms, ANN performed better than other clustering algorithms. The unsupervised algorithm outperformed the supervised algorithm for the highest relative F measure. scores.

**Figure 3:** MAE Performance of Different Algorithms




Based on MAE score chart (Figure 3), all ensemble classifiers have lowest MAE score where STC is on top. QDA acquired the lowest score among all supervised learning algorithms. KNN and GMM both achieved same lowest score among all other unsupervised learning. Unsupervised algorithm has lower MAE score than Supervised Algorithm based on relative lowest minimum scores.

Overall, STC performed well in all 3 performance measures and outperformed all other algorithms. Ensemble algorithms performed relatively well than individual classification and clustering algorithms. In supervised learning, QDA showed promising performance. In unsupervised learning, GMM and KNN both performed well in all 3 performance measures.

## CONCLUSION

Recent years have seen a growth in the development of software-based systems even though the quality of the system has to be guaranteed before delivery to the end-users. Software quality can be enhanced through several quality metrics such as ISO standards, CMM, and software testing. The need for software testing grows with each day, and its efficiency can be improved by using software defect prediction. The objective of this study was to investigate different software defect prediction models, which were identified as the ensemble, clustering, and classification techniques. The findings of this study show that stacking multiple classifiers can be used to defect prediction. It is our hope that these results will help increase the confidence in these models. In the future, more time ought to be spent on time and resources when dealing with error-prone modules.

**References**

1. J. Tian and M.V. Zelkowitz. Evaluation and Selection of Complexity Measures, IEEE Transactions on Software Engineering, 21(8), 641-650, 1995. https://doi.org/10.1109/32.403788
2. R. B. Jadhav, S. D. Joshi, U. G. Thorat und A.S. Joshi. Learning and Analysis of Software Defects Using Regression Methods for Quality

Software Development, International Journal of Advanced Trends in Computer Science and Engineering, 8(4), 1275

3. - 1282, 2019. https://doi.org/10.30534/ijatcse/2019/38842019
4. K.S. Kavya und Y. Prasanth. An Ensemble DeepBoost Classifier for Software Defect Prediction、International Journal of Advanced Trends in Computer Science and Engineering、9(2)、2021 – 2028、2020。
5. 4.M.K. Albzeirat, M.I. Hussain、R. Ahmad、F.M. Al-Saraireh und I. Ahmad。 New mathematical logic for improvement using lean manufacturing techniques. Journal of Advanced Manufacturing Systems, 17(03), 391-413, 2018. 5. S. Aleem, L.F. Capretz and F. Ahmed. Benchmark machine learning technology for detecting software bugs. International Journal of Software Engineering & Applications (IJSEA), 6(3), pp. 11-23, 2015. 6. E.Erturk and E.A.Cesar. A comparison of several soft computing techniques for predicting software failures. Expert Systems with Applications, 42(4), 1872-1879, 2015.
6. 7. M.K. Albzeirat, M.I. Hussain, R. Ahmad, F.M. Al-Saraireh, A. Salahuddin, N. Bin-Abdun. Applications of nanofluids in nuclear power plants as part of our future vision. International Journal of Applied Engineering Research, 13(7), 5528-5533, 2018.
7. 8. S. Lessmann、B. Baesens、C. Muse and S. Peach. A Benchmark Classification Model for Software Failure Prediction: A Proposed Framework and New Insights. IEEE Transactions on Software Engineering, 34(4), 485-496, 2008. M. Singh and D.S. Salaria. A software error prediction tool based on neural networks. International Journal of Computer Applications, 70(22), 2013. X Tan, X Peng, S Pan, W Zhao. Assessing software quality through program clustering and error prediction. 18th Reverse Engineering Workshop, pp. 244-248, 2011. Li, M. Shepperd, Y. Guo. A Systematic Review of Unsupervised Learning Techniques for Predicting Software Errors, Information and Software Technology, Vol. 3, No. 122, 106287, 2020 https://doi.org/10.1016/j.infsof.2020.106287
8. 12. MA hare. Correlation-Based Feature Selection for Discrete and Numeric Class Machine Learning, Proceedings of the Seventeenth International Conference on Machine Learning, S. 359-366, 2000. 13. S. Karim, H.L.H.S. Warners, F.L. Prison, E. Abdurachman and B. Soewito. Software Metrics for Failure Prediction Using Machine Learning Approaches: A Literature Review Using the PROMISE Repository Dataset. IEEE International Conference on Cybernetics and Computational Intelligence (CyberneticsCom), pp. 19-23, 2017.
9. T. Menzies, J. Greenwald, and A. Frank. Data mining static code attributes to learn defect predictors. IEEE transactions on software engineering, 33(1), 2-13, 2006.
10. A. Kaur and R. Malhotra, "Applying Random Forests in Predicting Failure-Prone Classes," International Conference on Advanced Computer Theory and Engineering, S. 37-43, 2008, doi: 10.1109/ICACTE .2008.204.
11. MS Naidu and N Geethanjali. Classification of software bugs using decision tree algorithms. International Journal of Engineering Science and Technology, 5(6), 1332, 2013.
12. M. Shepperd, D. Bowes, and T. Hall. Researcher bias: the use of machine learning in predicting software errors. IEEE Transactions on Software Engineering, 40(6), 603-616, 2014. Sandu.
13. A k-means Based Approach for Prediction of Level of Faults of Faults in Software, Proceedings of International Conference on Intelligent Computational Systems, 2011, Quelle: http://psrcentre.org/images/extraimages/71. Jaspreet _papier.pdf
14. A. Shantini and R.M. Chandrasekaran. Analyzing the effect of Bagged Ensemble Approach for Software Fault Prediction in Class Level and Package Level metrics, International Conference on Information Communication and Embedded Systems (ICICES2014),
15. Chennai, S. 1-5, 2014, doi: 10.1109/ICICES.2014.7033809. 15. Y. Pen, G. Kou, G. Wang, W. Wu, Y. Shi. An Ensemble of Software Failure Predictors: AHP-Based Scoring Method. International Journal of Information Technology and Decision Making, 10(01), 187-206, 2011. https://doi.org/10.1142/S0219622011004282
16. Q. Song, Z. Zia, M. Shepard, S. Ying and J. Liu. A general framework for predicting software failures. IEEE Transactions on Software Engineering, 37(3), 356-370, 2010.
17. A Porter and R.W. Selby. Experience-based software development using metric-based classification trees. IEEE Software, 7(2), 46-54, 1990. Srinivasan and D. Fisher.
18. A machine learning approach for estimating software development effort. IEEE Transactions on Software Engineering, 21(2), 126-137, 1995. Z. Li and M. Reformat.
19. A practical method of software error prediction. 2007 IEEE International Conference on Information Reuse and Integration, p. 659-666, 2007
20. . W.H.W. Ishak, K.R.K. Mahamud, and N.M. Norwawi. Modelling of Human Expert Decision Making in Reservoir Operation, Journal Teknologi, 77(22), 1-5, 2015.
21. W.H.W. Ishak, K.R.K. Mahmud and N.M. Norwawi. An Intelligent Decision Support Model Based on Neural Networks for Reservoir Water Release Decision Support,
22. J.M. Zain et al. (ed): ICSECS 2011, Part I, Communications in Computer and Information Science (CCIS) 179, pp. 365-379, 2011. https://doi.org/10.1007/978-3-642-22170-5_32