ISSN 2278-3091

Volume 9, No.4, July – August 2020

International Journal of Advanced Trends in Computer Science and Engineering

Available Online at http://www.warse.org/IJATCSE/static/pdf/file/ijatcse352942020.pdf

https://doi.org/10.30534/ijatcse/2020/352942020

# Predicting Software Defects using Machine Learning Techniques

**Mohammad Amimul Ihsan Aquil[1], Wan Hussain Wan Ishak[2]**

[1]Department of Computer and Information Sciences, Universiti Teknologi Petronas, Malaysia

aihsan.aquil @gmail.com

[2]School of Computing, Universiti Utara Malaysia, Malaysia, hussain@uum.edu.my

## ABSTRACT

A huge variety of software systems are relied upon in such domains as aviation, healthcare, manufacturing and robotics, and therefore, h systems and that they are reliable. Software defect prediction helps improve software reliability by identifying potential bugs during software maintenance. Traditionally, the focus of software defect prediction was on the design of static code metrics, which help with predicting the defect probabilities of a code when input into machine learning classifiers. While machine learning techniques such as Deep Learning technique, Ensembling, Data Mining, Clustering and Classification are known to help predict the location of defects in code bases, researchers have not yet agreed on which is the best predictor model. This paper will use 13 software defect datasets in evaluating the performance of the different predictor models. The results show that consistency in high accuracy prediction was achieved using Ensembling techniques.

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

## 1. INTRODUCTION

As the dependency and complexity of software grows, the demands for maintainable and high-quality low-cost software grows with it. However, for reduced maintenance in software operation and software quality improvement to be actualized, there is need for a software defect prediction system [1,2,3]. Having an early detection system in place will enable fast delivery of maintainable software since it will allow for timely correction of the detected faults [4]. A number of studies have developed certain metrics that can be used as the foundation of models in detecting any faults the software might encounter in operation during the initial stages of the software development life cycle.

Over the last 30 years the field of software engineering has had a growing interest in software defect prediction. The current scope of defect prediction encompasses (a) classification of software component's defect-proneness into the not defect-prone and defect-prone classes, (b) unearthing any association among defects, and (c) give an estimation of the remaining defects in software systems [5]. For the purposes of this study, our focus will remain on the first scope.

Modules/classes in Software Defect Prediction (SDP) can be categorized into two: fault-prone and not fault-prone. SDP models can be constructed using the fault data and the software metrics obtained from previous software releases or similar software projects [6,7]. After constructing the model, it can be integrated into current projects and help classify all the modules/classes as being fault-prone or not fault-prone [8]. Using these results, the software practitioners can now make an informed decision to work on all the fault prone areas during the early stages of development. For example, if only 30% of testing resources have been assigned to a certain software, having knowledge of all the fault-prone areas will ensure that all the available resources are allocated towards the correction of the modules/classes in these areas [9]. Thereby, resulting in a high quality and maintainable that is of high-quality and produced with the given time frame and budget [10].

A significant part of SDP research activity is focused in the detection of whether software components are defect prone or not by relying on using software metrics drawn from the code [11]. While different machine learning algorithms have been used in helping with the classification of software components as being defect-prone or not by trying to fine rules or patterns within data, none of them has proved to be accurate on a consistent basis. Some of these techniques used include mixed algorithms, parametric models, machine learning methods and statistical methods. However, before concluding on whether this problem is largely unsolvable, there is need for the identification of the best prediction technique to help with predicting a problem based on the 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 task which include reinforcement, semi-supervised, unsupervised and supervised learning. Though supervised and un-supervised learning remain the most popular task group.

Supervised learning is machine learning technique that involves the use of labelled training data, which houses various training examples to infer a function. The training example consists of an input object and the desired output value and includes the regression and classification of supervised learning tasks [12]. The regression classification task focuses on continuous range model building while the classification learning task focuses on building predictive model that functions within a discreet range. Example of supervised machine learning methods include support vector machine, neural network, linear regression, Bayesian learning, instance based learning, rule learning and learning classification [13].

Unsupervised learning enables systems to examine all the data to identify any patterns caused by common examples without knowledge of the presence or the number of patterns available in the data set. It also known as learning from observation and key examples include clustering, sequential pattern mining and association rule mining [13,14].

The rapid growth of research in machine learning has resulted in the creation of different learning algorithms that can be used across different applications [15]. Additionally, the ability of machine learning algorithms to solve-real world problems will often determine its ultimate value making the reproduction and application of algorithms in new tasks critical to the field's progress. However, the current research landscape features numerous publications regarding software fault prediction model development. These can be placed into categories based on ensemble, clustering and classification methods.

### A. Classification methods

A study by [6] records performance measures of 0.8573, 0.8685 and 0.7795 when using the ANFIS, ANN and SVM algorithms. Their study is based on data obtained from PROMISE Software Engineering Repository. They also deployed McCabe software metrics in their study.
In a bid to reduce the time and costs by finding the total number of defects using the ID3 classification algorithm, Naidu & Geethanjali [16] concludes by classifying the defect into five parameters including Time, effort, difficulty, length, program and value estimator. Singh and Salaria [9] developed a model using the Levenberg-Marquardt (LM) algorithm based neural network tool in a bid to explore the fault prone of early software testing for all data drawn from the PROMISE repository of empirical software engineering data. The accuracy of the LM was then compared to that of the polynomial function-based neural network. The LM recorded higher level of accuracy compared to the polynomial function-based neural network at 88.1%.

Aleem et al [5], after suing various machine learning methods to conduct a study on 15 datasets (KC3, KC1, CM1, AR6, and AR1 etc) found out that bagging, multilayer perceptron (MLP) and support vector machine (SVM) achieved high levels of performance and accuracy.

An investigation done by [17] relied on a novel benchmark framework in evaluating and predicting software defect. The activities involved evaluating and comparing different learning schemes to the selected one and using it to build a predictor that has all the historical data [1]. This predictor is now ready to predict any defect in any new data.

### B. Clustering methods

When using the function cluster to increase the performance of the software prediction model, Tan et al [10] managed to upgrade the model's precision and performance from 73.8% and 31.6% to 91.6% and 99.2% respectively.

According to Kaur and Sandhu [18] the k-mean based clustering approach has an accuracy of 62.4% in fault proneness of object-oriented programming. The model building relied on the EM and X-means clustering algorithms drawn from the AR3. AR4 and AR5 promise repository data to aid in the prediction of software faults. The experiment, which involved normalizing data set 0 to 1 followed by the use of the CfsSubsetEval as the applied attribute selection algorithm yields an accuracy level of (90.48) in X-means clustering algorithms for dataset AR3 compared to other models.

### C. 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], records an AUC of 0.81, an F-measure of 75, a recall of 79%, a precision of 72% and an accuracy RF of 74.24%. In their experiment, Kaur & Malhotra relied on a JEdit open source software with object-oriented metrics in evaluating the use of random forest in an open source software to predict fault prone class.

Peng et al [20] evaluates the use of ensemble approaches in the prediction of software faults with an analytical hierarchical process. They drew 10 publicly NASA MDP data that relied on 13 different performance measures. The ensemble method used is staking, Boosting and Bagging and records a result accuracy of 92.53% for a decision tree base classifier.

## 3. SDP MACHINE LEARNING ALGORITHMS

This section presents the algorithms used in the evaluation. The study compared supervise learning algorithms such as Linear SVC, Maximum vote classifier, Light GBM, Gaussian Naïve Bayes classifier, Passive Aggressive classifier, Xgboost and Extra Tree Classifiers. Unsupervised learning methods involved the comparison of clustering techniques such as Stacking classifiers, GMM and mini-batch K-means algorithm against each other.

### A. Maximum Voting Classifier (MVC)

The maximum voting involves a number of classifiers constantly generating predictions and testing the resulting data. The final prediction is determined by looking onto which had the most votes (more than half) [21]. Different classifiers can be combined to increase the accuracy of this algorithm. The maximum voting classifier works in the following way:

(a) Use 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 works on randomizing a tree building much further through numerical input figures in a case where a significant part of the induced tree's variance falls under the choice of the optimal cut point [22]. The algorithm would switch to using bootstrap copies in the vase of random forests rather than finding the optimal cut off point for all the randomly chosen K features on every node. The selection of the cut of point is random. This method is ideal for cases involving large number of varying numerical features. The method's smoothing yields greater accuracy while reducing any computational challenges associated with determining the location of optimal cut-off points in both random forests and standard trees.

### C. Passive Agressive Classifier (PAC)

Under Passive, the model is retained if it falls under the right type of classification. In aggressive mode, any incorrect classification should be updated to help account for the misclassified example. While under passive it shows the lack of sufficient information prevents updates. Under aggressive terms, having a better model will help you modify any mistakes since the last time you were wrong.

### D. Xgboost

Xgboost is a tool belonging to the Distributed Machine Learning Community (DMLC) and is popular for its increased performance and speed when it comes to gradient-boosted decision trees. Xgboost was first designed and used by Tianqi Chen in the year (1995) and has worked as a way to provide a boost to machines. It has gone through a number of iteration by various developers.

In tree boosting algorithms, eXtreme or XGBoost are used to aid in the exploitation of all hardware and memory resources available, allowing for its deployment in computing environments, tuning the model and enhancing the algorithm [20]. There are three techniques of gradient boosting in XGBoost, include: stochastic boosting, Regularized Boosting and Gradient Boosting. Additionally, it is very effective in the tuning and adding regularization parameters, optimal use of memory resources and reducing the time used in computing activities. XGBoost can also perform on the trained model's added data, allow parallel structures and takes care of any missing value (Sparse Aware).

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

When determining optimality, this algorithm assumes that when the K-Means algorithm uses the one-pass over input data, it optimizes the K-Means to increase the production of the centroids. Multiple passes over input data will reduce the running time since costs in largescale computations will increase after having to go through a large data set. In a simplified version, this algorithm uses incoming points as basis for a new cluster or assigns them to a nearby cluster and makes a decision regarding distances to the closest cluster using an adaptive scale parameter.

### F. Gaussian Mixture Model (GMM)

This is a parametric model used in examining the probability distribution of features or continuous measurements in a biometric system 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 world of Netflix and other competition, stacking has proved beneficial as one of the ensembling methods in machine learning [23]. The main idea with this algorithm is to use the confidence scores as features in combining multiple models and training a meta-classifier to help combine the predictions of multiple learners. The study employed three classifiers which include a Random Forest Classifiers, Adaboot and KNN, which rely on logistic regression to help test and train all type of slots.

### H. Gaussian Naïve Bayes (GNB)

This classification algorithm that works with both multi-class and binary (two class) classification problems and can be quite simple to understand when described using categorical or binary input values [16]. Naïve Bayes allows extension to real value attributes, which is also known as the Gaussian Naïve Bayes. Working with the Normal distribution (Gaussian) is very simple, all one has to do is use the training data to estimate the standard deviation and mean.

### I. Quadratic Discriminant Analysis (QDA)

This is a generative probabilistic method used in classifying problems. QDA's conditional distributions are used in the classification of observations across different classes as they are assumed to be multivariate Gaussian derived using the posterior distributions in the Bayes theorem. Traditionally, the estimation of QDA was done through the maximization of joint likelihood of observations together with their corresponding associate class labels.

## 4. METHODOLOGY

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

### 3.1 Data Preparation

The use of Machine-learning techniques is critical in achieving software reusability, maintainability and quality since it helps with finding the bas smell, ambiguity, fault and defect in software. Accomplishing this requires software fault prediction techniques, which rely on statistical techniques to any software defects [24]. However, software detection can also be done through machine learning techniques. Pre-processing helps shape the data into a form that the classification engine can use [25,26]. For example, when using an image as the input data, pre-processing will simplify the feature selection processing by sharpening the image or rotating and translating the image into a standard orientation and position. In cases where the input is made of records or vectors of data, pre-processing might involve using the dataset's statistical property or priori criteria to filter out inputs. Key benefits of pre-processing include normalizing numeric data and helping fill in missing data.

The experiments relied on datasets drawn from the PROMISE data repository collected from real NASA software projects and entail various software modules. The benchmarking involved using public domain datasets. This benchmarking procedure allows other researchers compare their studies. some of the code metrics used in the datasets include McCabe's cyclomatic complexity, code size and Halstead's complexity among others. The description of Datasets is summarized in the Table 1. The target variables in the NASA MDP data sets are binary in nature, 1: Yes, 0: No. Table 2 shows the performance evaluator matrices that are used in this study. Python programming and Scikit-learn (machine learning framework) is used in 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 |
| lOCode | Line Count | Halstead |
| lOComment | Line count of Comments | Halstead |
| lOBlank | Count of Blank Lines | Halstead |
| lOCodeAndComment | 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 \| |

### 3.2 Feature extraction

Feature extraction facilitates the conversion of preprocessed data into a form that can be used by the pattern recognition engine. Pattern recognition algorithms express different levels of sensitivity regarding the form of data provided and therefore the need for a feature selection. In this study, Random forest feature importance score was used for finding best features for all algorithms.

### 3.3 Classification

Solving the translation problem allowed the creation of numerous classification algorithms, which can be customized in line to flows to defect, fragments or machining source code tokens. Each classifier comes with different strength and weaknesses aimed to fit specific needs. Finally, the performance of the mentioned algorithms is measured based on the performance metrics in 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 |

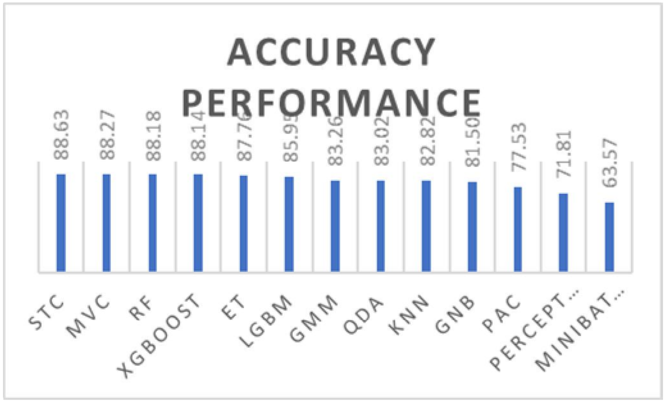

**Figure 1:** Accuracy Chart of Different algorithms

Based on Accuracy chart (Figure 1), it can be clearly depicted that the proposed stacking classifier (STC) proposed in this study scored better comparing to other algorithms. All the ensemble classifiers performed better in accuracy measure than other supervised and unsupervised learning. In classification algorithm, QDA scored better than other algorithms and GMM performed better than other clustering algorithms. Between classification and clustering algorithms, clustering algorithms performed relatively well.

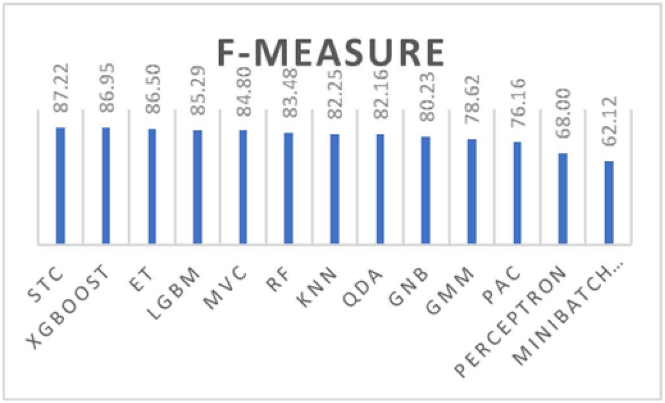

**Figure 2:** F-Measure of Different Algorithms

The above chart (Figure 2) represents the average F-measure of machine learning algorithms in all 13 datasets. Based on F-measure, STC remained in top list and all ensemble classifiers are consistently preformed higher than other algorithms. Among all the supervised learning algorithms, QDA performed relatively higher followed by GNB. As for unsupervised learning algorithms, KNN scored higher than other clustering algorithms. Unsupervised algorithm outperformed supervised algorithm in F-measure in terms of highest relative scores.

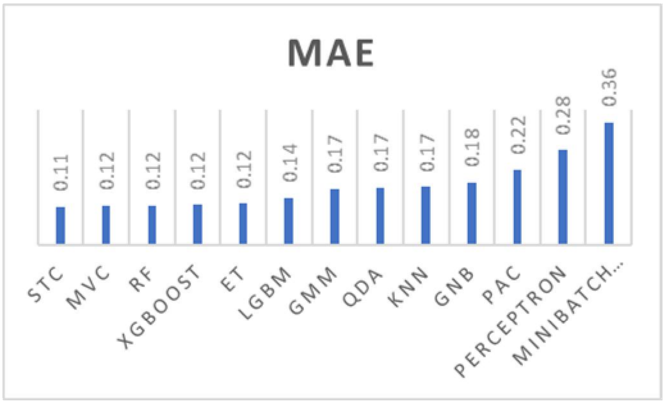

**Figure 3:** MAE Performance of Different Algorithms

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.

## 6. 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. **Complexity measure evaluation and selection,** *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, and A.S. Joshi. **A Software Defect Learning and Analysis Utilizing Regression Method for Quality Software Development**, *International Journal of Advanced Trends in Computer Science and Engineering*, 8(4), 1275 - 1282, 2019. https://doi.org/10.30534/ijatcse/2019/38842019
3. K.S. Kavya, and 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. https://doi.org/10.30534/ijatcse/2020/173922020
4. M.K. Albzeirat, M.I. Hussain, R. Ahmad, F.M. Al-Saraireh, and I. Ahmad. **A novel mathematical logic for improvement using lean manufacturing practices.** *Journal of Advanced Manufacturing Systems*, 17(03), 391-413, 2018.
5. 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.
6. E. Erturk, and E.A. Sezer. **A comparison of some soft computing methods for software fault prediction**. *Expert systems with applications*, 42(4), 1872-1879, 2015. https://doi.org/10.1016/j.eswa.2014.10.025
7. M.K. Albzeirat, M.I. Hussain, R. Ahmad, F.M. Al-Saraireh, A. Salahuddin, and N. Bin-Abdun. **Applications of Nano-Fluid in Nuclear Power Plants within a Future Vision**. *International Journal of Applied Engineering Research*, 13(7), 5528-5533, 2018.

8. S. Lessmann, B. Baesens, C. Mues, and S. Pietsch. **Benchmarking classification models for software defect prediction: A proposed framework and novel findings.** *IEEE Transactions on Software Engineering*, 34(4), 485-496, 2008.
9. M. Singh, and D.S. Salaria. **Software defect prediction tool based on neural network**. *International Journal of Computer Applications*, 70(22), 2013.
10. X. Tan, X. Peng, S. Pan, and W. Zhao. **Assessing software quality by program clustering and defect prediction**. *18th working conference on Reverse Engineering*, pp. 244-248, 2011.
11. 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 https://doi.org/10.1016/j.infsof.2020.106287
12. M.A. Hall. **Correlation-based feature selection of discrete and numeric class machine learning**, *Proceedings of the Seventeenth International Conference on Machine Learning*, pp. 359-366, 2000.
13. 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.
14. 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.
15. A. Kaur, and R. Malhotra, **Application of Random Forest in Predicting Fault-Prone Classes**, *International Conference on Advanced Computer Theory and Engineering*, pp. 37-43, 2008, doi: 10.1109/ICACTE.2008.204.
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. J. Kaur, and P.S. Sandhu. **A k-means Based Approach for Prediction of Level of Severity of Faults in Software System**, *Proceedings of International conference on Intelligent Computational Systems*, 2011, Source: http://psrcentre.org/images/extraimages/71.%20jaspreet_paper.pdf
19. A. Shanthini, 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)*, Chennai, pp. 1-5, 2014, doi: 10.1109/ICICES.2014.7033809.
20. Y. Peng, G. Kou, G. Wang, W. Wu, and Y. Shi. **Ensemble of software defect predictors: an AHP-based evaluation method**. *International Journal of Information Technology & Decision Making*, 10(01), 187-206, 2011. https://doi.org/10.1142/S0219622011004282
21. Q. Song, Z. Jia, M. Shepperd, S. Ying, and J. Liu. **A general software defect-proneness prediction framework**. *IEEE transactions on software engineering*, 37(3), 356-370, 2010.
22. A.A. Porter, and R.W. Selby. **Empirically guided software development using metric-based classification trees**. *IEEE software*, 7(2), 46-54, 1990.
23. K. Srinivasan, and D. Fisher. **Machine learning approaches to estimating software development effort.** *IEEE Transactions on Software Engineering*, 21(2), 126-137, 1995.
24. Z. Li, and M. Reformat. **A practical method for the software fault-prediction**. *In 2007 IEEE International Conference on Information Reuse and Integration*, pp. 659-666, 2007.
25. 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.
26. W.H.W. Ishak, K.R.K. Mahamud, and N.M. Norwawi. **Intelligent Decision Support Model Based on Neural Network to Support Reservoir Water Release Decision,** In J.M. Zain et al. (Eds.): 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