Software Quality Prediction: Analysis Using ML Techniques

Fatima Masood
Department of Software Engineering
University of Sargodha
Sargodha
fatimamasood141@gmail.com

***Abstract*— Software Quality Assurance is a systematic process to ensure the delivery of high-quality products. To maintain software Quality, there is always a desire for a defect-free system to satisfy customers and save testing costs. Detecting faults in software before the testing phase is crucial for efficient resource allocation and high-quality software development. Software defects can be predicted using Machine Learning techniques i.e. SVM, Random Forest, Naïve Bayes, and Stacking Classifier. However, in existing approaches, ML algorithms are applied to some datasets. In this research, we aim to find the best classification model for Software Quality prediction by using ML algorithms on five different datasets of the PROMISE repository i.e. CM1, JM1, KC1, KC2, and PC1. Firstly, we applied K-means clustering for class label categorization and then applied Classification models to the selected features. We evaluated the performance of models using accuracy, precision, recall, f-measure, performance error metrics, and confusion matrices. The results indicate that SVM performs better for software quality prediction than other models. In this way, we achieved our aim to find the best model for software quality prediction.**

***Keywords*— Software quality prediction; software defect prediction; machine learning; k-means clustering; support vector machine; naïve Bayes; random forest; ensemble approach.**

I. Introduction

Software engineering is a systematic and disciplined approach to developing, designing, testing, and maintaining software systems. It applies engineering principles throughout the software development life cycle to ensure high-quality, reliable, and maintainable software[1]. Software Quality Assurance (SQA) is a set of systematic activities that ensure the process and standards are appropriate for a project and implemented systematically. In software engineering, SQA aims to verify and enhance the quality throughout its development life cycle, which involves defining processes, conducting audits, and implementing best practices[2]. At the same time, Software Defect Prediction(SDP) is a proactive approach within software quality assurance that focuses on identifying and predicting potential defects or bugs in software before they manifest. This involves using various techniques and models to analyze historical data, code metrics, and other relevant factors to predict areas of the software more likely to have defects, addressing and fixing issues early in the development process.

Machine Learning (ML) is a field of artificial intelligence that focuses on developing algorithms and models that enable computer systems to learn and make predictions or decisions without being explicitly programmed. In other words, it involves creating systems that can automatically learn patterns and relationships from data and improve their performance over time[3]Machine learning techniques involve the application of algorithms and statistical models to analyze software data and predict outcomes. In SDP, machine learning models learn patterns and relationships from historical data to predict areas of the software prone to defects. Common techniques include ensemble learning, Support Vector Machine (SVM), Decision Tree (DT) , Random Forest (RF) , Naïve Bayes (NB) , and Neural Networks (NN) [4]. Ensemble techniques are like teamwork in the world of machine learning. Instead of relying on one "expert," we combine multiple models with strengths and weaknesses. It's like having a group discussion where everyone contributes, and decisions are made by considering everyone's opinions. In machine learning, this teamwork, or ensemble, often results in more accurate and robust predictions than individual models, offering a more reliable solution[5].

The significance of SDP lies in its pivotal role in ensuring the quality and reliability of software throughout the development life cycle. SDP allows for the early identification of potential defects in the development process. By predicting areas of the software prone to issues, developers can proactively address and rectify problems before they escalate, saving time and resources. Identifying and fixing defects early in the development process is significantly more cost-effective than addressing issues in later stages or after the software has been deployed. SDP reduces cost by preventing costly defects in the final product. Prediction models guide developers in allocating their resources more efficiently. Predicting defects leads to improved software quality. By addressing potential issues early on, the overall reliability and performance of the software are enhanced, resulting in a higher-quality product. SDP aids in planning testing by providing insights into which areas of the software are more likely to have defects. This data-driven approach enables testing teams to prioritize their efforts on the riskiest components, ensuring thorough testing in critical areas. High-quality software with fewer defects contributes to better customer satisfaction. Prediction of potential issues before deployment results in a more reliable and user-friendly product, meeting or exceeding customer expectations.

It is important to focus on the most important metrics for defect prediction. The five datasets used in this research are publicly available on the PROMISE repository. It provides information on various applications that NASA (National Aeronautics and Space Administration) has investigated [6].In this research, first, we pre-process the data and select features, then K-means clustering is used to categorize output. After that, ML approaches like RF, SVM, Naïve Bayes, and Ensemble approach are applied to integrate the results. Then we apply the lazy predict classifier to compare all classifying algorithms of ML. Finally, all models were analyzed and compared using precision, accuracy, recall, f-measure, performance error metrics, and confusion matrix.

II. Motivation and Contribution

In this section, the motivation and contribution of the research are discussed.

A. Motivation

Software companies are working to create error-free software. The software's quality decreases due to defects and cannot perform tasks correctly. The most important stage of any software, which requires comprehensive testing, is software defect identification, which is the most critical aspect of the SDLC [7]. Managing defects improves the quality of solutions and encourages all software development teams to consider quality across the whole project, resulting in continuous improvement in products. To enhance the dependability and utility of the software, software flaws must be identified and fixed in time.

B. Contribution

The contributions of our research are as follows:

- Find out the best ML model to predict software defects.
- Using five different datasets to evaluate the model's accuracy, precision, recall, and f-measure.

III. RELATED WORK

In SDP, common ML approaches are clustering, classification, and deep learning. Using ML-based classifiers, researchers have suggested different SDP models. In their study, Iqbal et al. [8] introduced a feature selection-based ensemble classification framework, evaluated in 12 cleaned NASA datasets. The framework employed Bagging and Boosting with a Random Forest classifier, demonstrating superior performance compared to traditional techniques. Root cause analysis attempts to find the causes of the issue to remedy the problem. It helps us to find defects at an early stage. Different types of clustering techniques are used in the model [9]. Clustering is considered unsupervised learning, which works on data similarities. Researchers use different types of clustering techniques to find out the defect. WEKA[10] was used for the implementation of clustering techniques.

Khuat et al. [11] displayed the impact of integrating random under-sampling into ensemble learning for imbalanced software defect datasets. The balanced training data substantially improved the performance of the ensemble model and the base classifier. The combination proved effective in creating a promising classifier for predicting software faults. In 2020 [12], they investigated software defect prediction involving the use of ensemble classifiers, including Logistic Regression (LR), SVM, Multilayer Perceptron, NB, Bayesian Networks (BN), DT, and K-Nearest Neighbor, along with various sampling methods. It focuses on assessing predictive performance using metrics like accuracy, F-1 score, recall, and precision. In their 2020 research [13], Tanujit et al. used NASA Software Defect Prediction datasets from the PROMISE repository [6]. Their study applied the Hellinger Distance for tree splitting, inspired by Breiman's CART concept, and used RF and NB with log filters as training methods.

Aftab et al.[14] research centered on precisely predicting defects in software systems, addressing the significant costs associated with late defect discovery. It employed various software quality metrics, including traditional and object-oriented measures, and adopted a two-fold approach to select discriminating features. These features were incorporated into two cost-sensitive prediction schemes: CLR and DTE. Comparative analysis with recent literature demonstrated the superior performance of the proposed methodology in terms of accuracy, area under the curve, and recall. The study provided valuable insights for software practitioners.

ML-based classification is used for Software bug prediction [15]. The different classifier is applied with Machine Learning for this. Artificial Neural Networks (ANNs), NB, and DT are used in this ML-based classification, which gave good results. Five classes are the output of this technique with small dataset measurements. The experimentation is performed on three public datasets. WEKA 3.6.9[10] is used. ML techniques show good performance. Perreault et al. used five distinct classifiers to discover software defects in their investigation [16].

In this review [17], the authors discuss some software metrics and datasets for predicting defects. They are used for finding defects in the ML approach, which consists of making information from software archives. It contains messages and source codes. The Instances contain a method, class, source code, packages, and code change. The instance has some features obtained from the software archives. Metrics values characterize software complexity and development. The most used metric is the line of code metrics (loc). A defect dataset is used for predicting defects. Some recent studies show that researchers used a non-public dataset. Some publicly available datasets are NASA, SOFTLAB, PROMISE, ReLink, AEEEM, ECLIPSE 1, ECLIPSE 2, etc. Evaluation metrics for software defect prediction are Probability of defect, True Positive (TP) rate, False Negative (FN), True Negative (TN), and False Positive (FP) rate, precision, accuracy, F measure, and AUC.

The authors of [18] analyzed five public datasets from the PROMISE repository[6] using ML-based predictions and ten classifiers. Accuracy was used for evaluation. Deep learning techniques were studied for defect prediction in this survey [19]. None of the methods produced high accuracy, recall, and precision in results. SLR was employed to monitor current developments in ensemble or hybrid techniques [20]. From renowned online libraries, 46 papers were selected for shortlisting. According to a study, FS and data sampling increase outcomes. Using evaluation metrics, performance is calculated. The ensemble strategy performs better than others do.

The literature review shows that different ML techniques have been applied until now but their performance differs across datasets, and their performance is less accurate in terms of ML. Khalid et al.[21] applied RF, NB, SVM, and an ensemble approach on the CM1 dataset of the PROMISE repository[6] to select the best ML algorithm for SDP. However, their contribution is limited as they only used a single dataset and only three algorithms with their ensemble algorithm. Therefore, we want to improve accuracy by analyzing various ML techniques combined with FS and K-means clustering on five different datasets. The purpose of research is to improve accuracy concerning literature studies.

A. *Tables*

The Table 1 gives the related papers, models, datasets, and evaluation metrics used in that paper.

Table 1 Related Work

PAPER	YEAR	MODEL	Dataset	Evaluation Parameters
Software Defect Prediction Analysis Using Machine Learning Techniques [21]	2023	SVM, RF, NB, Ensemble	CM1	accuracy, precision, recall, f-measure, performance error metrics, and confusion matrices
Discriminating features-based cost-sensitive approach for software defect prediction[14]	2021	Meta cost-based Domingo's Logistic Regression, Decision tree ensemble	KC1 KC2 PC1 PC3 PC4 MC1 MC2 JM1 CM1 Mozilla4 jEdit	Accuracy Precision Recall F-measure MAE RMSE
Evaluation of sampling-based ensembles of classifiers on imbalance data for software defect prediction problems[12]	2020	Used sampling techniques and ensemble classifiers like Logistic Regression SVM MLP NB Bayesian Networks KNN Decision tree	NASA from PROMISE Repository	Accuracy Precision Recall F1-score
Hellinger net: A hybrid imbalance learning model to improve	2020	The Hellinger distance is used for the splitting tree. As training method RF	NASA SDP dataset from PROMISE	Accuracy Recall F-measure

software defect prediction[13]		and NB with log filter	Repository	
A feature selection-based ensemble classification framework for software defect prediction[8]	2019	Boosting and Bagging (Random Forest) and Feature Selection NB SVM MLP RBF KNN KSTAR ONER PART DT RF	CM1 JM1 KC1 MC1 MC2 KC3 PC1 PC2 PC3 PC4 PC5 MW1	Accuracy Precision Recall F-measure ROC MCC
Ensemble learning for software fault prediction problem with imbalanced data[11]	2019	KNN, BN, SVM, MLP, J48, Ensemble	ANT 1.7, Camel 1.6, Ivy 2.0, Tomcat, Xalan 2.4, Synapse 1.2, Pol 2.0	Precision Recall F1-score

IV. Research Methodology

This section discusses the methodology, datasets, and techniques used in our research.

A. Datasets

We have used five public datasets from the PROMISE repository of software engineering databases [6], which are freely available and part of NASA MDP (Metrics Data Program).

1) *CM1*:
In the CM1 dataset, there are 498 instances and 22 properties in Table 2.1.

2) *JM1:*
The JM1 dataset has 10885 instances with 22 attributes in Table 2.1.

3) *KC1:*
The KC1 dataset has 2109 instances with 22 attributes in Table 2.1.

4) *KC2:*
The KC2 dataset has 522 instances with 22 attributes in Table 2.1.

5) *PC1:*
The PC1 dataset has 1109 instances with 22 attributes in Table 2.1.

Table 2.1 Datasets Attributes Description

CM1,JM1,KC1,KC2, and PM1 Datasets		
McCabe Metrics	Loc	Line count of code
	v(g)	Cyclomatic complexity
	ev(g)	Essential complexity
	iv(g)	Design complexity
Halstead Metrics	N	Total operators+operands
	V	Volume
	L	Program length
	D	Difficulty
	I	Intelligence
	E	Effort to write program
	B	Delivered bugs
	T	Time estimator
	IOCode	Line count of code
	IOComment	Count of lines of comments
	IOBlank	Count of blank lines
	IOCodeAndComment	Lines of code and comments
Other Metrics	uniqOP	Unique operators
	unipOpnd	Unique operands
	totalOp	Total operators
	totalOpnd	Total operands
	branchCount	Flow of graph
	D	Module has defects or not

The data sets' properties are based on 4 McCabe metrics, 12 Halstead measurements, and some other. McCabe Metrics are Method-level Metrics, which focus on programming principles and are straightforward to gather from source code [1]. Other Halstead Metrics are numerical values and they can be gathered with any piece of software [2]. Table 2.1 lists the dataset metrics.

Table 2.2 Datasets Parameters

Title	Language	Source	Modules	Features	Defective	Defect Rate(%)
CM1	C	NASA Spacecraft Instrument	498	22	49	9.88
JM1	C	NASA Spacecraft Instrument	10885	22	8779	80.65
KC1	C++	NASA Spacecraft Instrument	2109	22	326	15.45
KC2	C++	NASA Spacecraft Instrument	522	22	105	20.5
PC1	C	NASA Spacecraft Instrument	1109	22	1032	93.05

Table 2.2 shows dataset parameters i.e. language, source, modules, features, defective features, and defect rate.

B. Model Design

ML is an important achievement in Artificial Intelligence (AI), and it is evident that a model for SDP based on ML techniques is necessary for maintaining quality and saving testing costs. We find many issues related to predicting software defects from the literature using ML algorithms. On various datasets, the authors used ML algorithms. Some are precise and others vary in their performance and accuracy or precision. The objective is to create an SDP analysis prototype by saving testing costs while increasing the proposed system's accuracy. For this purpose, we analyze ML techniques with selected features and clustering to achieve good accuracy. We intend to achieve high accuracy, which employs ML algorithms, using the CM1, JM1, KC1, KC2, and PM1 datasets. Figure 1 gives the architecture of the suggested model. We examined various well-known ML techniques on a freely available dataset to improve the datasets' accuracy. For this purpose, we employed K-means clustering for categorizing class labels. We evaluated the performance of models through precision, accuracy, recall, f-measure, performance error metrics, and a confusion matrix. In the next section, we will discuss these figure sections in detail. The aim of choosing ML techniques is to improve the accuracy of well-established ML techniques because, in the literature, the findings of these ML techniques differ and can be improved. Therefore, the objective is to improve performance or accuracy and find the best algorithm for SDP.

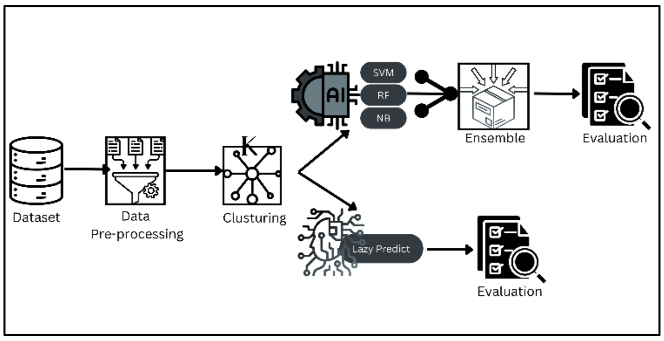

Figure 1 Proposed Method

C. *Techniques*

1) *Data Pre-processing*

From the analysis of the datasets, it is concluded that all five datasets need to be transformed to a standard format before applying any ML models, as there are 22 features in the datasets. In each column of the datasets, there is a wide range of values, for example in the CM1 dataset the column 'e' has a maximum value of 2,153,690.63 and a minimum value of 0.0. Additionally, the column 't' has a maximum value of 119,649.48 and a minimum value of 0.0. The column 'I' has a maximum value of 1.30 and a minimum value of 0.0. There is a huge difference between the columns and the other columns. The mean value of column 't' is 1938.056124 and the standard deviation is 7453.591519, while column 'I' has a mean value of 0.146325 and a standard deviation of 0.159337. Thus, due to this, a standard scaling technique is used for standardizing the data set. It arranges data in a standard normal distribution. Mathematically, we can determine Z as:

$$Z = \frac{(x - u)}{x}$$

Where x is an observation, u is the mean of the training sample, and s is the standard deviation of the training sample.

2) *Feature Selection*

The feature selection is to reduce the features utilized in a predictive model's training and testing. The correlation method and variance inflation factor approach are utilized in this research to identify the significance of values and multi-collinearity of a feature after preparing the dataset (promise). Two features are in a positive correlation if an increase in the value of one feature causes an increase in the value of another feature and a decrease in the value of one feature causes a decrease in the value of another feature(Neil). If the change in the value of one feature does not affect the value of another feature, these two features do not correlate. If an increase in the value of one feature causes a decrease in the value of another feature, the two features are negatively correlated. Figures (2.1-2.5) show the correlation between all features of the data sets i.e. CM1, JM1, KC1, KC2, and PC1. We select the top 10 attributes that are negatively associated or have no association based on relevance and high importance without class labels. We reduce the number of features to reduce computational costs and to prevent overfitting difficulties to increase model performance. In this research, for categorizing the output class labels we utilized unsupervised ML (clustering), and supervised ML (classification) for prediction purposes.

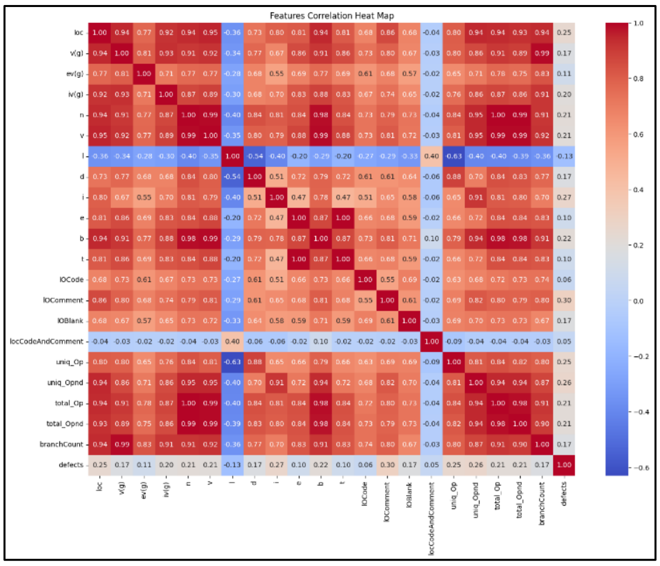

Figure 2.1 CM1 Features Correlation

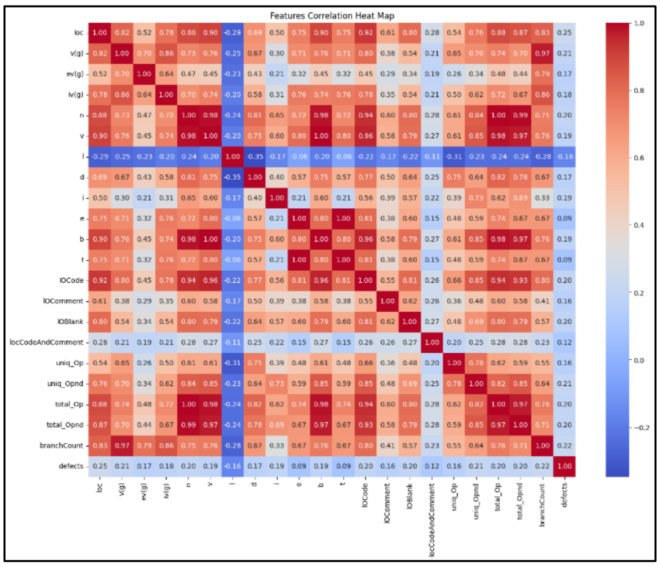

Figure 2.2 JM1 Feature Correlation

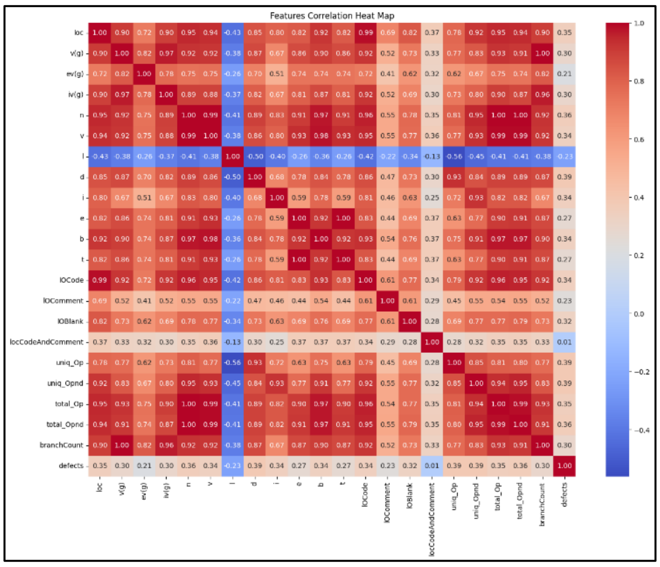

Figure2.3 KC1 Features Correlation

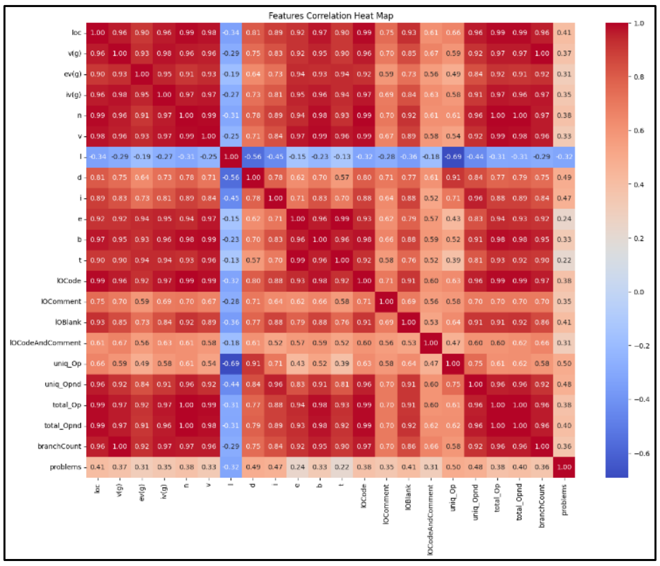

Figure 2.4 KC2 Features Correlation

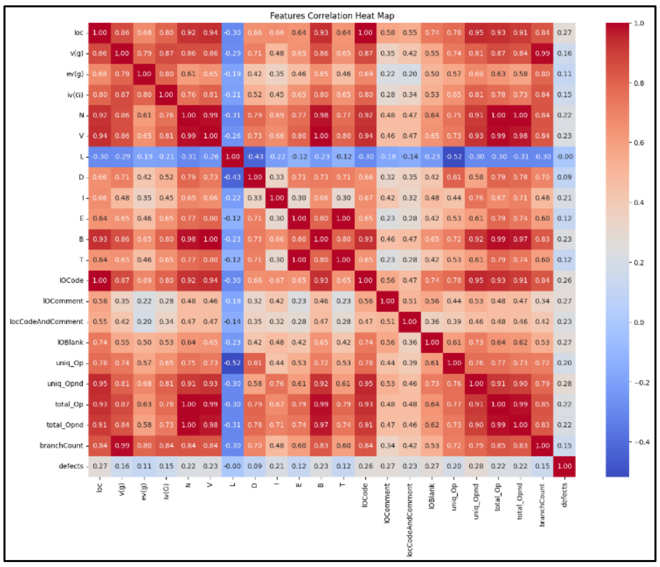

Figure 2.5 PC1 Features Correlation

3) Clustering

Next, we use K-means clustering to discover class labels and inertia and the elbow approach to select the best number of clusters (K). In this method, the number of clusters is denoted by the K. Inertia is for how effectively a data collection is clustered and is obtained by squaring the distance between each data point and its centroid, then adding the squares throughout one cluster. The elbow approach uses the Within Cluster Sum of Squares idea (WCSS). To calculate the distance between data points and the centroid, the WCSS employs the distance formula.

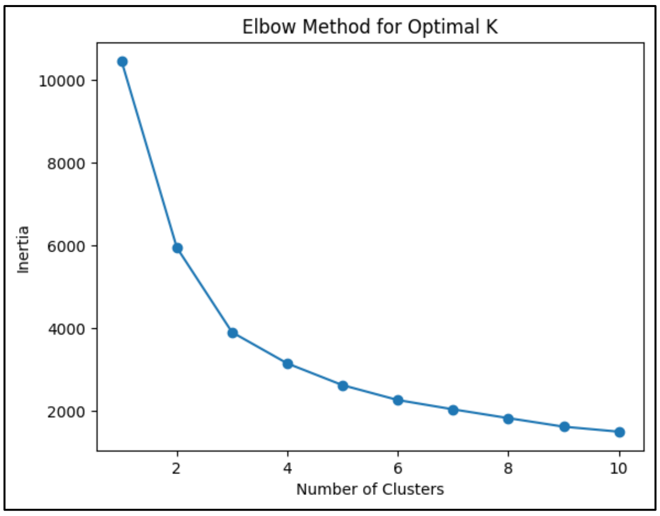

Figure 3.1 CM1 Relation of Inertia with Cluster

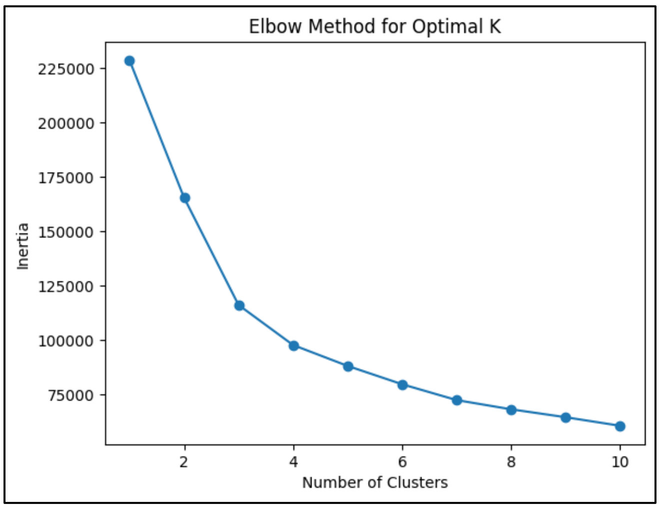

Figure 3.2 JM1 Relation of Inertia with Cluster

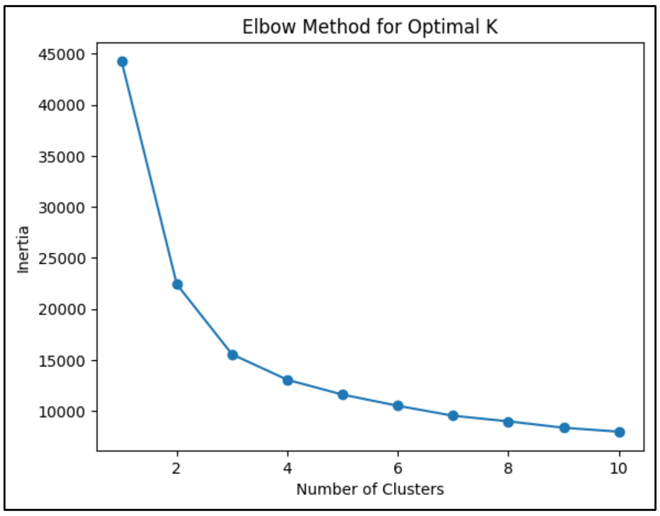

Figure 3.3 KC1 Relation of Inertia with Cluster

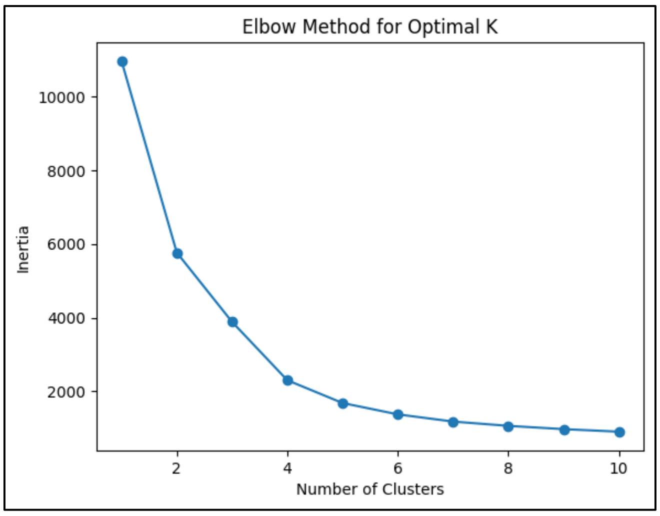

Figure 3.4 KC2 Relation of Inertia with Cluster

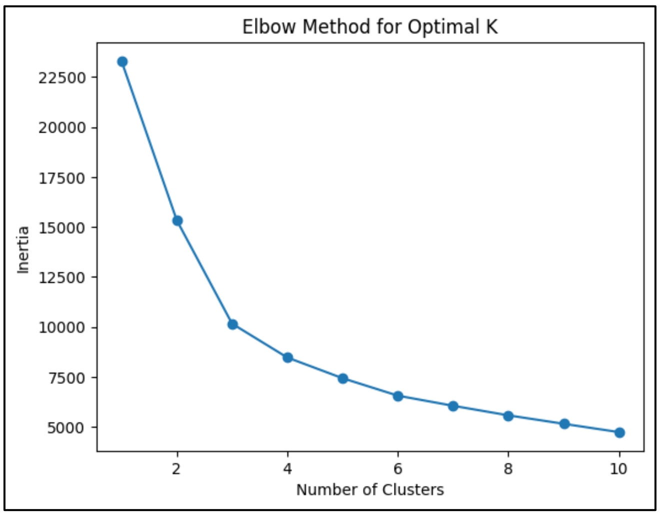

Figure 3.5 PC1 Relation of Inertia with Cluster

4) *Classification*

For Supervised ML, we divide the dataset in two portions i.e. training and testing. The training dataset makes up 67% of the total and the testing dataset makes up just 33%. First, the data with output class labels is used for training purposes, and then unseen data with no output class labels are used for testing. First, we applied three ML classification models for analysis, i.e., SVM as Linear SVC classifier, NB as Gaussian NB classifier, and RF as Random Forest classifier with ensemble method as Stacking classifier to analyze and combine the results of all three models by using NB as a base model and SVM and RF as member models. Then, we applied the lazy predict with 26 classification models.

5) *SVM*

SVM is a supervised ML model mostly applied to data for binary classification. SVM models perform better with high speed for limited datasets. The SVM model makes a decision boundary between the data to classify each class. The best hyperplane in an SVM model is when there are maximum margins from both positive and negative classes. The SVM model is used in this work to predict the output class label and it obtains the best results.

6) *NB*

The NB algorithm is a Bayes algorithm-based categorization method. It calculates the probabilities against likelihoods. When there is no association between the properties of a data set, the NB method works well. The classifier used is a Gaussian NB.

7) *RF*

The RF employs ensemble learning, which entails combining numerous classifiers to improve the outcome. The RF model comprises several DTs applied to subsets of the data set and then averaged to determine performance measures. The number of trees has an important impact on accuracy and other measures. However, the model improvement becomes constant after a certain number of trees. Knowing the right amount of trees is important for training purposes. As a classifier, Random Forest is utilized.

8) *Ensemble*

In ML, the ensemble technique combines numerous prediction models to provide the best results. In the ensemble method, one model is the basis, while the others are members. Ensembling is performed in a variety of ways. We use the stacking classification approach. The NB model is the base model, and the SVM and RF models are member models.

9) *Lazy Predict*

In ML, Lazy Predict helps build many basic models without much code and helps understand which models work better without parameter tuning. In this research, we used lazy prediction to get the results of 26 classification models.

V. Results And Discussions

This section gives experimental setup, experimental results, figures, and dicussions.

A. *Experimental Setup*

The ML classification experiments in this research were performed on a Windows laptop with an Intel(R) Core(TM) i5-4300U CPU @ 1.90GHz 2.50, 8 GB of primary storage, and 128 GB of secondary storage. The ML models were implemented using the Python programming language. Python is commonly used in predictive analytics and data science projects involving qualitative and quantitative data. The Python packages i.e. pandas, numpy, seaborn, matplotlib, lazypredict, and sklearn were used to build the ML predictor. We performed coding using Google Colab.

B. *Experimental Results*

The results of the experiments on five datasets using various ML techniques are represented in the form of graphs and tables. These graphs give a comparison of applied techniques. We will assess all results by comparing all strategies. Tables 3.1 shows the evaluation metrics for all ML approaches All analyzed models have the best results but SVM on three datasets (CM1, KC1, KC2), RF on 1 dataset (PC1), and ensemble on one dataset (JM1) have better performance than others.

Table 3.1 Comparison of Evaluation Measures

Dataset	Evaluation Measures	SVM	NB	RF	Ensemble
CM1	Accuracy	0.87	0.85	0.87	0.85
	Precision	0.79	0.82	0.82	0.82
	Recall	0.87	0.85	0.87	0.85
	F-measure	0.83	0.83	0.84	0.83
JM1	Accuracy	0.80	0.81	0.81	0.81
	Precision	0.75	0.77	0.78	0.79
	Recall	0.80	0.81	0.81	0.81
	F-measure	0.72	0.77	0.78	0.79
KC1	Accuracy	0.86	0.81	0.85	0.82

	Precision	0.84	0.80	0.83	0.81
	Recall	0.86	0.81	0.85	0.82
	F-measure	0.82	0.80	0.84	0.81
KC2	Accuracy	0.83	0.82	0.80	0.82
	Precision	0.81	0.81	0.78	0.81
	Recall	0.83	0.83	0.80	0.82
	F-measure	0.81	0.81	0.79	0.81
PC1	Accuracy	0.92	0.87	0.93	0.88
	Precision	0.84	0.87	0.91	0.89
	Recall	0.92	0.87	0.93	0.88
	F-measure	0.88	0.87	0.90	0.88

The major goal and objective of analyzing different ML models is to improve the accuracy of classifiers so that outcomes can be predicted as precisely as feasible. The expected model analysis is compared to recognized benchmark metrics such as Mean Absolute Error (MAE) and Root Mean Square Error (RMSE) to confirm that the models are adaptive. The following Table 3.2 shows the Performance error values of models. In performance error evaluation, the analyzed models have a lower error rate.

Table 3.2 Comparison of Error Metrics

Dataset	Performance Error Metrics	SVM	NB	RF	Ensemble
CM1	MAE	0.13	0.15	0.13	0.15
	RMSE	0.36	0.39	0.36	0.39
JM1	MAE	0.20	0.19	0.18	0.18
	RMSE	0.44	0.44	0.43	0.43
KC1	MAE	0.14	0.19	0.15	0.18
	RMSE	0.37	0.44	0.38	0.43
KC2	MAE	0.17	0.17	0.20	0.18
	RMSE	0.40	0.42	0.45	0.42
PC1	MAE	0.08	0.12	0.07	0.12
	RMSE	0.23	0.35	0.27	0.35

Tables 4.1-4.5 give the result of the Lazy Predict with 26 classifiers and their results. i.e. DT Classifier, Label Propagation (LP), Label Spreading (LS), LGBM Classifier, Bernoulli NB, Extra Tree Classifier (ET), XGB Classifier, Extra Trees Classifier, Perceptron, Passive Aggressive Classifier (PA), Gaussian NB, RF Classifier, Bagging Classifier (BC), Nearest Centroid (NC), Dummy Classifier (DC), SVC, K Neighbors Classifier (KN), Ridge Classifier CV (RCCV), Ridge Classifier (RC), Linear SVC, Quadratic Discriminant Analysis (QDA), Logistic Regression (LR), Linear Discriminant Analysis (LDA), Calibrated Classifier CV (CCCV), Ada Boost Classifier (ABC), and SGD Classifier.

Table 4.1 CM1 Lazy Predict Results

Model	Accuracy	AUC ROC	F Score
DT	0.87	0.63	0.86
LP	0.88	0.57	0.86
LS	0.88	0.57	0.86
LGBM	0.87	0.56	0.85
BernouliNB	0.78	0.56	0.80
ET	0.86	0.56	0.85
XGB	0.88	0.54	0.85
ExtraTrees	0.88	0.54	0.85
Perceptron	0.86	0.53	0.84
PA	0.85	0.53	0.84
NB	0.85	0.52	0.83
RF	0.87	0.51	0.84
BC	0.87	0.51	0.84
NC	0.81	0.50	0.81
DC	0.89	0.50	0.84
SVC	0.89	0.50	0.84
KN	0.88	0.49	0.83
RCCV	0.87	0.49	0.83
RC	0.87	0.49	0.83
LinearSVC	0.87	0.49	0.83
QDA	0.87	0.49	0.83
LR	0.87	0.49	0.83
LDA	0.87	0.49	0.83
CCCV	0.87	0.49	0.83
ABC	0.87	0.49	0.83
SGD	0.85	0.48	0.82

Table 4.2 JM1 Lazy Predict Results

Model	Accuracy	AUC ROC	F Score
BernouliNB	0.73	0.63	0.74
NC	0.76	0.63	0.76
DT	0.75	0.60	0.75
Perceptron	0.78	0.60	0.76
LP	0.80	0.60	0.77
RF	0.82	0.60	0.78
LS	0.80	0.60	0.78
BC	0.81	0.60	0.78
ET	0.75	0.59	0.75
ExtraTrees	0.81	0.59	0.78
XGB	0.80	0.58	0.77
KN	0.80	0.58	0.77
LGBM	0.81	0.58	0.77
NB	0.81	0.58	0.77
QDA	0.80	0.56	0.76
PAC	0.80	0.56	0.76
SGD	0.81	0.55	0.76
LDA	0.81	0.55	0.76
CCCV	0.81	0.54	0.75
ABC	0.80	0.54	0.75
LR	0.81	0.54	0.75
LinearSVC	0.81	0.53	0.75
SVC	0.81	0.53	0.75
RCCV	0.81	0.53	0.74
RC	0.81	0.53	0.74
DC	0.80	0.50	0.72

Table 4.3 KC1 Lazy Predict Results

Model	Accuracy	AUC ROC	F Score
ET	0.82	0.68	0.82
BernoulliNB	0.73	0.67	0.76
Perceptron	0.76	0.66	0.78
NC	0.79	0.66	0.80
DT	0.82	0.65	0.82
QDA	0.82	0.65	0.82
ExtraTrees	0.86	0.64	0.84
RF	0.85	0.64	0.84
LDA	0.85	0.64	0.84
BC	0.85	0.63	0.83
XGB	0.84	0.63	0.83
LGBM	0.84	0.63	0.83
NB	0.81	0.61	0.80
LS	0.82	0.60	0.81
LP	0.81	0.60	0.81
LR	0.85	0.60	0.83
LinearSVC	0.85	0.60	0.83
RCCV	0.86	0.59	0.83
CCCV	0.85	0.59	0.82
ABC	0.83	0.59	0.81
RC	0.85	0.59	0.82
KN	0.83	0.58	0.81
SVC	0.85	0.57	0.81
SGD	0.83	0.56	0.80
PAC	0.83	0.52	0.78
DC	0.84	0.50	0.77

Table 4.4 KC2 Lazy Predict Results

Model	Accuracy	AUC ROC	F Score
BernoulliNB	0.79	0.75	0.80
XGB	0.82	0.69	0.81
LGBM	0.81	0.68	0.80

RCCV	0.84	0.66	0.82
RC	0.84	0.66	0.82
RF	0.80	0.66	0.80
NC	0.80	0.66	0.80
ET	0.77	0.66	0.77
SGD	0.83	0.66	0.81
NB	0.83	0.66	0.81
BC	0.81	0.66	0.80
KN	0.79	0.66	0.79
ABC	0.80	0.65	0.79
PAC	0.80	0.65	0.79
LR	0.83	0.65	0.81
SVC	0.82	0.64	0.80
LinearSVC	0.82	0.64	0.80
CCCV	0.83	0.64	0.81
LDA	0.80	0.64	0.79
LP	0.77	0.62	0.77
ExtraTrees	0.78	0.62	0.77
QDA	0.81	0.61	0.78
DT	0.74	0.61	0.74
LS	0.77	0.61	0.76
Perceptron	0.72	0.58	0.72
DC	0.80	0.50	0.71

Table 4.5 PC1 Lazy Predict Results

Model	**Accuracy**	**AUC ROC**	**F Score**
DT	0.91	0.63	0.90
ET	0.91	0.63	0.90
XGB	0.93	0.61	0.91
ABC	0.93	0.60	0.91
RF	0.93	0.60	0.91
KN	0.93	0.60	0.91
BC	0.93	0.60	0.91
ExtraTrees	0.93	0.60	0.91
LDA	0.92	0.59	0.90
LGBM	0.92	0.59	0.90
NB	0.87	0.58	0.87
NernouliNB	0.70	0.58	0.77
LP	0.91	0.57	0.89
LS	0.91	0.57	0.89
SGD	0.91	0.57	0.89
LinearSVC	0.92	0.55	0.90
LR	0.92	0.55	0.89
NC	0.80	0.54	0.83
QDA	0.90	0.52	0.88
SVC	0.92	0.52	0.89
DC	0.92	0.50	0.88
Perceptron	0.92	0.50	0.88
CCCV	0.92	0.50	0.88
RC	0.92	0.50	0.88
RCCV	0.92	0.50	0.88
PAC	0.90	0.49	0.87

C. *Figures*

Figures 4.1- 4.5 compare accuracy, precision, recall, and an f-score of SVM, NB, RF, and Ensemble Classifier on all five datasets i.e. CM1, JM1, KC1, KC2, and PC1 respectively.

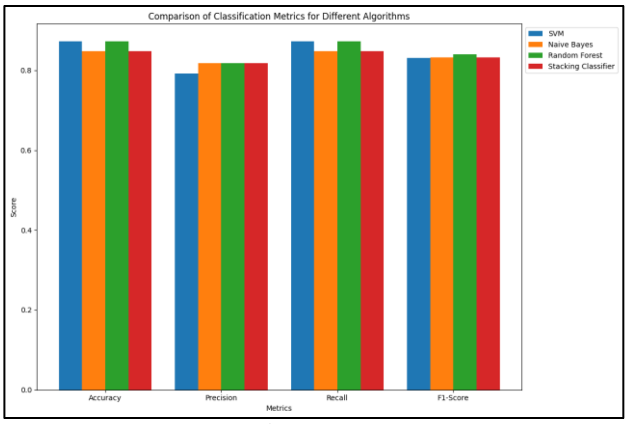

Figure 4.1 CM1 Comparison of Evaluation Metrics

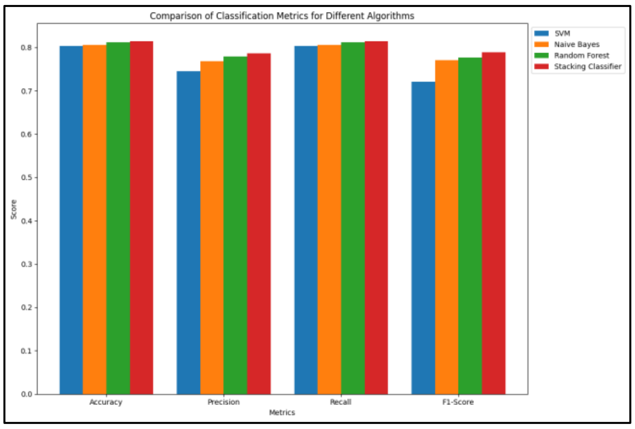

Figure 4.2 JM1 Comparison of Evaluation Metrics

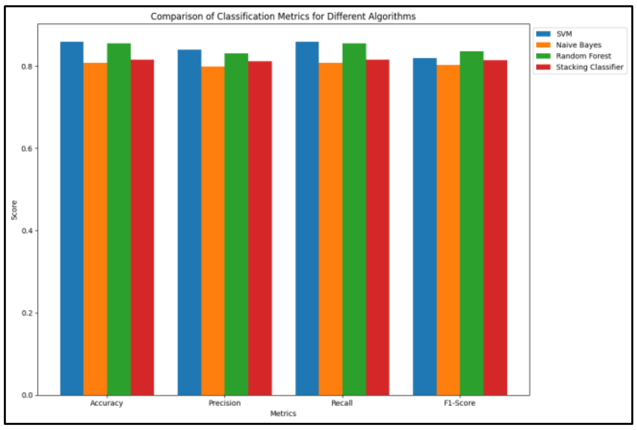

Figure 4.3 KC1 Comparison of Evaluation Metrics

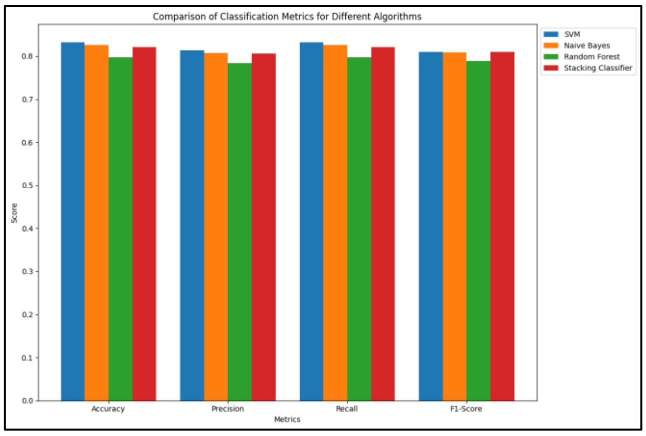

Figure 4.4 KC2 Comparison of Evaluation Metrics

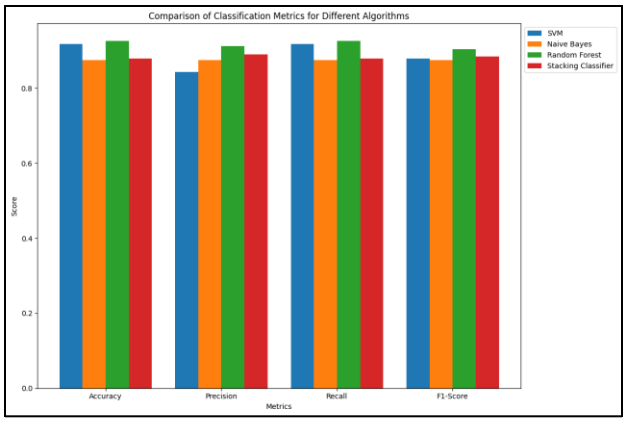

Figure 4.5 PC1 Comparison of Evaluation Metrics

Figures 5.1- 5.5 compare error metrics, MAE, and RMSE of all five datasets i.e. CM1, JM1, KC1, KC2, and PC1 respectively.

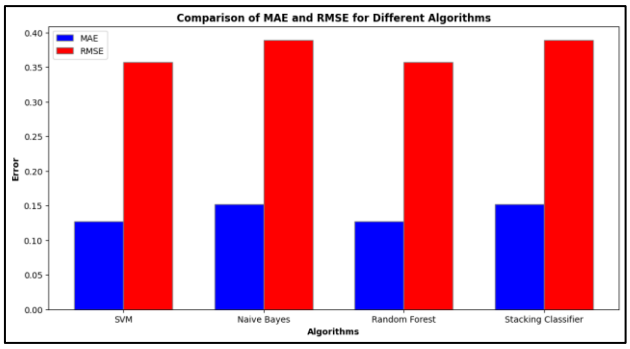

Figure 5.1 CM1 Comparison of Error Metrics

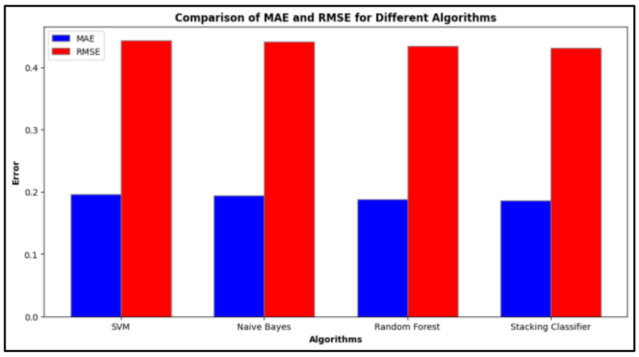

Figure 5.2 JM1 Comparison of Error Metrics

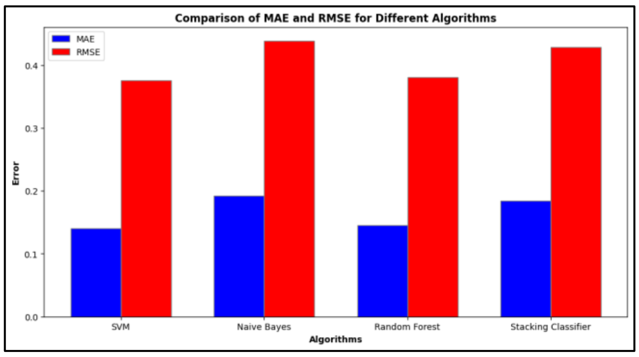

Figure 5.3 KC1 Comparison of Error Metrics

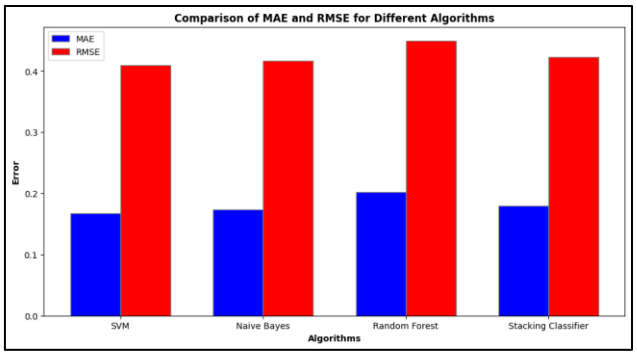

Figure 5.4 KC2 Comparison of Error Metrics

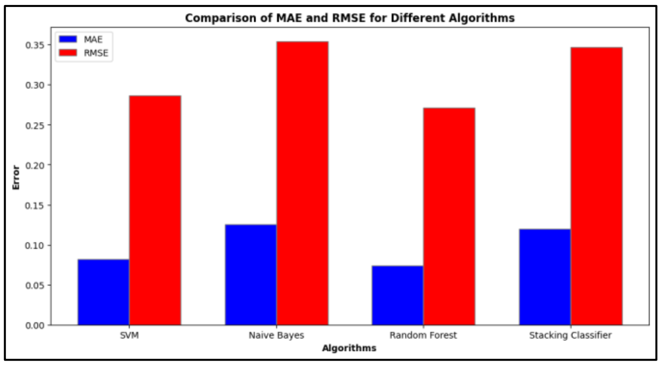

Figure 5.5 PC1 Comparison of Error Metrics

D. *Discussion*

According to the results of the SVM, NB, RF, and ensemble approach applied to CM1, JM1, KC1, KC2, and PC1. We can decide that on CM1, KC1, and KC2, the best algorithm for SDP is SVM, while on JM1, the ensemble approach gives the best results, and on PC1, RF outperforms other algorithms. According to the results of the Lazy Predict on CM1 the DC and SVC give the best results with 89% accuracy, on JM1 the RF gives the best result with 82% accuracy, on KC1 the Ridge Classifier CV gives the best result with 86% accuracy, on KC2 the Ridge Classifier CV (RCCV) and the Ridge Classifier (RC) gives best results with accuracy 84%, and on PC1 the XGB Classifier, Extra Trees Classifier, RF Classifier, Bagging Classifier (BC), K Neighbors Classifier (KN) and Ada Boost Classifier (ABC) outperforms other algorithms with 93% accuracy.

VI. Conclusion and Future Directions

This section concludes and gives future directions for this research.

A. *Conclusion*

In this research, ML techniques are used with feature selection and K-means clustering techniques for SDP. We examined various well-known ML on a freely available five datasets to find the best model for SDP. All ML models are trained and tested through Python programming language using Google Colab. The analysis aimed to find the best ML Model on the CM1, JM1, KC1, KC2, and PC1 datasets. The results indicate that all the ML models achieve the maximum results; however, the

SVM models outperformed with the highest achieved accuracy.

B. *Future Work*

In this article, three ML classification models and their ensemble approach is used to find bugs in software. These models are applied to five datasets. In the future, we will increase the size of the dataset and try to analyze different types of ensemble classifiers after applying data balancing techniques because due to balancing technique we can improve error measure as well to get maximum results.

ACKNOWLEDGMENT

Special thanks to all the teachers, colleagues, and supporters.

REFERENCES

[1] "What_Every_Engineer_Should_Know_about_So".

[2] "Software Quality Assurance – Software Engineering." Accessed: May 16, 2024. [Online]. Available: https://www.geeksforgeeks.org/software-engineering-software-quality-assurance/

[3] Linda Tucci and Ed Burns, "What is machine learning and how does it work? In-depth guide," CIO/IT Strategy. Accessed: May 16, 2024. [Online]. Available: https://www.techtarget.com/searchenterpriseai/definition/machine-learning-ML

[4] A. Iqbal *et al.*, "Performance Analysis of Machine Learning Techniques on Software Defect Prediction using NASA Datasets," *International Journal of Advanced Computer Science and Applications*, vol. 10, pp. 300–308, May 2019, doi: 10.14569/IJACSA.2019.0100538.

[5] K. L. Chiew, C. L. Tan, K. Wong, K. S. C. Yong, and W. K. Tiong, "A new hybrid ensemble feature selection framework for machine learning-based phishing detection system," *Inf Sci (N Y)*, vol. 484, pp. 153–166, 2019, doi: https://doi.org/10.1016/j.ins.2019.01.064.

[6] J. Sayyad Shirabad and T. J. Menzies, "The PROMISE Repository of Software Engineering Databases." 2005. [Online]. Available: http://promise.site.uottawa.ca/SERepository

[7] B. Grishma and C. Anjali, "Software root cause prediction using clustering techniques: A review," May 2015, pp. 511–515. doi: 10.1109/GCCT.2015.7342714.

[8] A. Iqbal, S. Aftab, I. Ullah, S. Bashir, and M. A. Saeed, "A Feature Selection based Ensemble Classification Framework for Software Defect Prediction," *International Journal of Modern Education and Computer Science*, vol. 11, pp. 54–64, May 2019, doi: 10.5815/ijmecs.2019.09.06.

[9] B. Grishma and C. Anjali, "Software root cause prediction using clustering techniques: A review," May 2015, pp. 511–515. doi: 10.1109/GCCT.2015.7342714.

[10] E. Frank, M. A. Hall, and I. H. Witten, *The WEKA Workbench. Online Appendix for "Data Mining: Practical Machine Learning Tools and Techniques,"* Fourth. Morgan Kaufmann, 2016.

[11] T. Khuat and L. Thi My Hanh, "Ensemble learning for software fault prediction problem with imbalanced data," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 9, p. 3241, May 2019, doi: 10.11591/ijece.v9i4.pp3241-3246.

[12] T. Khuat and L. Thi My Hanh, "Evaluation of Sampling-Based Ensembles of Classifiers on Imbalanced Data for Software Defect Prediction Problems," *SN Comput Sci*, vol. 1, May 2020, doi: 10.1007/s42979-020-0119-4.

[13] T. Chakraborty and A. K. Chakraborty, "Hellinger Net: A Hybrid Imbalance Learning Model to Improve Software Defect Prediction," *IEEE Trans Reliab*, vol. 70, no. 2, pp. 481–494, 2021, doi: 10.1109/TR.2020.3020238.

[14] A. Ali, N. Khan, M. Abu-tair, J. Noppen, S. McClean, and I. McChesney, "Discriminating features-based cost-sensitive approach for software defect prediction," *Automated Software Engineering*, vol. 28, May 2021, doi: 10.1007/s10515-021-00289-8.

[15] A. Hammouri, M. Hammad, M. Alnabhan, and F. Alsarayrah, "Software Bug Prediction using Machine Learning Approach," *International Journal of Advanced Computer Science and Applications*, vol. 9, May 2018, doi: 10.14569/IJACSA.2018.090212.

[16] L. Perreault, S. Berardinelli, C. Izurieta, and J. W. Sheppard, "Using Classifiers for Software Defect Detection," 2016. [Online]. Available: https://api.semanticscholar.org/CorpusID:21679278

[17] Z. Li, X.-Y. Jing, and X. Zhu, "Progress on approaches to software defect prediction," *IET Software*, vol. 12, May 2018, doi: 10.1049/iet-sen.2017.0148.

[18] M. Cetiner and O. Sahingoz, "A Comparative Analysis for Machine Learning based Software Defect Prediction Systems," May 2020, pp. 1–7. doi: 10.1109/ICCCNT49239.2020.9225352.

[19] E. N. Akimova, A. Y. Bersenev, K. S. Deikov Artem A and Kobylkin, A. V Konygin, I. P. Mezentsev, and V. E. Misilov, "A survey on software defect prediction using deep learning," *Mathematics*, vol. 9, no. 11, p. 1180, May 2021.

[20] F. Matloob *et al.*, "Software Defect Prediction Using Ensemble Learning: A Systematic Literature Review," *IEEE Access*, vol. 9, pp. 98754–98771, May 2021, doi: 10.1109/ACCESS.2021.3095559.

[21] A. Khalid, G. Badshah, N. Ayub, M. Shiraz, and M. Ghouse, "Software Defect Prediction Analysis Using Machine Learning Techniques," *Sustainability*, vol. 15, no. 6, p. 5517, 2023, doi: 10.3390/su15065517.