Scientific Review
ISSN(e): 2412-2599, ISSN(p): 2413-8835
Vol. 12, Issue. 1, pp: 6-22, 2026
URL: https://arpgweb.com/journal/journal/10
DOI: https://doi.org/10.32861/sr.121.6.22

Academic Research Publishing Group

Original Research **Open Access**

An -Improved Intelligent Model for Software Defect Prediction

Mahmoud Abdelmohsen Ahmed (Corresponding Author)
Department of Computer Science, Faculty of Graduate Studies for Statistical Research, Cairo University, Giza 12613, Egypt
Email: mahmoud.aamy_25@stud.fgssr.cu.edu.eg

Shahira Shaaban Azab
Department of Computer Science, Faculty of Graduate Studies for Statistical Research, Cairo University, Giza 12613, Egypt

Hesham Ahmed Hefny
Department of Computer Science, Faculty of Graduate Studies for Statistical Research, Cairo University, Giza 12613, Egypt

Article History
Received: 7 January, 2026
Revised: 21 February, 2026
Accepted: 15 March, 2026
Published: 29 March, 2026

Abstract

Software defect prediction is an important activity in every software firms, so the Software defects have severe consequences, especially in mission-critical arrangements developed by organizations like NASA. Effective techniques for early detection and prediction of defective software modules are crucial for ensuring reliability and qualifying risks. This study investigates the application of machine learning models for predicting software defects using datasets from NASA's Metrics Data Program depending on 10 datasets. Four different classification models – Support Vector Machines, Random Forests, Logistic Regression, and Ensemble model – were evaluated on their ability to classify software modules as defective or non-defective based on software metrics. The datasets exhibited significant class imbalance, with defective modules being the minority class. To address this, the Synthetic Minority Over-sampling Technique was employed, which generated synthetic examples of the minority class, leading to improved performance across all models. Also, two feature selection procedures, Recursive Feature Elimination with Cross-Validation and Information Gain, were applied and compared. RFECV generally resulted in higher accuracy and precision, while the results for recall and F1-score were mixed. Among the assessed models, the Random Forest model demonstrated the highest overall accuracy after applying SMOTE and feature selection. The research highlights the potential of machine learning, particularly ensemble methods like Random Forests, for automating software defect prediction in critical systems. By addressing trials such as class imbalance and feature selection, the performance of these models can be significantly enhanced. This study contributes to the rising field of machine learning applications in software engineering, providing insights and methods for improving the reliability and quality of software systems developed by NASA and other organizations working on mission-critical application.

Keywords: Software Defect Prediction (SDP); Ensemble Model; Feature Engineering, Recursive Feature Elimination (RFE); Support vector machines (SVMs); Information Gain (IG).

1. Introduction

The concept of software quality is often challenging to measure in numerical terms. Nevertheless, there remains plenty of research papers, books, and conferences that claim expertise in the field of measurements and metrics, there is a lack of consistent, practical data to enable characterization of the particulars of the software lifecycle where an organization can focus its improvement action. The majority of studies in software metrics pursue to identify a relationship between some set of procedures and the eventual outcome on the produced software. (Jorgensen, 1999) presents the opinion that since software is intangible, there is a need for gathering facts and opinions from different viewpoints to give a more comprehensive picture of what is really happening through the duration of a software project - in other words, to triangulate. These viewpoints can be quantitatively or qualitatively distinguished information coming from software activity, project, or result, from the perspectives of the supplier or the customer. Taking a measure of the whole, comparing the data personalities (of groups) and looking at anomalies or poor performers can imply the opportunity for the varied improvement action that organizations require. This case has presented a need for complex, flexible dataset of software defects influencing different software projects at NASA.

NASA Metrics Data Program (MDP) software defect datasets were collected by NASA for the purpose of developing predictive models capable of forecasting the number of defects which will be encountered during software development and testing, as well as the effectiveness of the software development and testing processes with respect to the discovery and removal of defects (Iqbal, et al , 2020). The dataset was gathered between 1985 and 1989 from NASA software development projects. Three of the projects were developed for the OS2 operating system, and the fourth was not associated with any particular project. Data was collected throughout various application development activities including requirements, design, code and testing. Various software development environments were used including the Flight System Development Environment (FSD), the IBM 3081 TattleTale System, and various compilers and debuggers. A variety of programming languages were used, though most of the

work was done using PL1, FORTRAN and assembly language. The data was primarily collected from project personnel working with the projects. In some cases, project managers and assistant project managers were consulted in order to obtain their perceptions of specific development activities. Finally, some of the data was extracted from project documents including requirement specifications, and weekly technical reports.

NASA's Metrics Data Program (MDP) was formulated with the intent of increasing the quality of software development at NASA. The objective of the MDP is to develop and distribute a set of metrics and models. The metrics are a standard set of measurements that can be made on software development projects. The models provide a means to understand the data and make predictions. The data sets are being made available to allow researchers to advance the art of software prediction through the use of empirical 2 data. The intent is to validate the models being developed and provide a means to measure progress. While the primary use is for NASA software, it is believed that these models can be transferred to other software development fields. Providing data from other development fields will allow comparisons to be made in order to leverage advancement in those fields. This has strong potential to impact the cost, schedule, and more importantly the quality of software in many domains. To augment and complement the software metrics data, a software system is needed to store and manage the metadata and data, thus the Metrics Data Program (MDP) has invested in a software system called the Metrics Data System (MDS). MDS is a database system that utilizes a database of NASA's software projects. MDS has developed a taxonomy to catalog and organize NASA's software projects and a process to iteratively extract software projects' metadata and data, and store them in the database. The metadata is data that describes the software data and the data's origin. The software metrics data are the measurements on the software projects. MDS can be considered the precursor to the SLE. (Petric, et al., 2016).

During the course of its activities, the MDP has developed two NASA-wide standards for software practices. These standards have been formulated into models for use in improving software development, and have been validated against data from NASA software projects. The standards and associated models define desired practices in software development, and identify" health indicators" to assess the degree to which the practices have been followed. The standards also provide guidance for selecting and applying software metrics in managing and evaluating a software project, and give examples of how the metrics can be used. The ultimate goal of the MDP is to identify and promote software practices that lead to better, more efficient development of systems that are based on the use of more advanced technology. This emphasis on" improving practice" distinguishes. the MDP from other empirical software engineering research, and reflects a strong orientation toward serving the needs of software practitioners. (Ramadhani, et al. 2024).

2. Related Works and Assessment

With the increasing impact of software applications on today's businesses and activities, software attribute prediction such as effort estimation, maintainability, defect and quality classification are gaining growing interest from both academic and industry communities. Software systems used in NASA missions and projects must meet extremely high standards of reliability and quality. Even small defects in the code can potentially lead to critical system failures or catastrophic events. As such, effective techniques for detecting software defects early in the development lifecycle are crucial for NASA. Traditional methods for software defect detection have relied heavily on code reviews and testing by human experts. However, these approaches can be time-consuming, costly, and error-prone, especially for large and complex software systems. In recent years, machine learning (ML) techniques have shown great promise in automating and enhancing various aspects of software engineering, including defect detection (Khoshgoftaar, 2003). A research paper by (Laradji et al, 2015) focuses on the increasing importance of software attribute prediction, such as effort estimation, maintainability, defect, and quality classification, due to a growing reliance on software applications. The paper highlights.

the limitations of conventional prediction algorithms such as decision trees, Bayesian methods, and artificial neural networks multilayer perceptron (ANN-MLP) when handling skewed and redundant defect datasets. The research introduces ensemble learning's voting mechanism as a solution, which assigns higher weights to successful individual classifiers, thereby mitigating the effects of feature irrelevance and redundancy. The paper's main objective is to demonstrate the positive effect of feature selection on the performance of defect classification. The findings suggest that for a dataset with 0.5% defective components, a classification accuracy of 99.5% can be achieved by classifying all components as non-defective, and an overall accuracy of 99% can be achieved by a binary classifier that classifies all data samples as the majority class. In the discussion section, the paper compares the results of the average probability ensemble (APE) model with two basic classifiers (W-SVMs and random forests). The W-SVMs classifier assigns weights inversely proportional to the class occurrence in the dataset, allowing for potentially better model fitting in the case of imbalanced datasets. Another paper by (Emmanuel, et al., 2021) discusses the process of software defect prediction, which involves identifying likely flawed sections of software. The paper introduces a model that uses a base layer of Linear Discriminant Analysis (LDA), K Nearest Neighbors (KNN), and Generalized Linear A model with Elastic Net Regularization (GLMNet) and a top layer of Random Forest (RF). The results demonstrate that this model is capable of effectively handling PROMISE datasets, known for their noisy attributes and high dimensions. This paper is anticipated to make a significant contribution to the field of software defect prediction. According to the results, the Ensemble machine learning technique offers superior prediction accuracy for software defect prediction, providing a significant insight from this study. The proposed model achieved an overall prediction accuracy of 88.56% across all experimental datasets. Furthermore, the Mean Squared Error (MSE) of the proposed model is significantly lower than other models, demonstrating that the ensemble technique effectively handles errors in the learning model caused by noise, bias, and variance.

Therefore, the research suggests that ensemble machine learning models provide a robust solution for software defect prediction. Several studies have explored using supervised ML models to predict defective software modules or code components based on software metrics and other code attributes (Rathore & Gupta, 2012). For example, (Rathore and Gupta, 2012) evaluated random forests, naive Bayes, and other classifiers on defect datasets from the NASA Metrics Data Program repository, achieving high accuracy rates. Unsupervised techniques like clustering have also been applied to identify defect-prone modules without requiring labeled training data (Sharma, 2022). In addition to traditional ML, recent work has applied deep learning models like convolutional neural networks (CNNs) and recurrent neural networks (RNNs) to defect detection tasks (Hoang et al., 2019; Liu et al., 2021). These models can automatically learn semantic code representations and patterns indicative of defects directly from the source code. (Liu et al, 2021) developed a CNN-based approach for cross-project defect prediction on NASA data, outperforming traditional machine learners. Despite the promising initial results, there remain significant challenges in deploying ML for defect detection in real-world NASA software projects, including data quality issues, interpretability of models, and lack of comprehensive benchmarks. Additionally, safety-critical applications may require extra scrutiny and verification of ML-based defect detectors before adoption.

Two key challenges in applying ML to NASA defect prediction remain. The first is the imbalanced nature of the datasets, where defective modules are typically a minority compared to functioning modules (Tosun & Bener, 2009). This can lead to biased models favoring the majority class. Techniques such as resampling, cost-sensitive learning, and ensemble methods have been explored to address data imbalance. Another challenge involves identifying the most relevant software features to feed into ML models. Feature engineering requires domain expertise and can significantly impact model performance (Jureczko & Madeyski, 2010). In summary, a growing body of research has demonstrated the potential of machine learning, particularly deep learning, to improve software defect detection capabilities for NASA systems. However, more work is still needed to tackle practical constraints and transition these ML techniques into production environments at NASA and other organizations developing mission-critical software. Overcoming these challenges through further research could yield substantial benefits in software quality assurance and risk mitigation.

Extra paper provided by (Cetiner et al., 2020) deliberates the process of software defect prediction, machine learning algorithms, with Decision Tree (DT), Naive Bayes (NB), K-Nearest Neighbor (KNN), Support Vector Machine (SVM), Random Forest (RF), Extra Trees (ET), Adaboost Classifier (AC), Gradient Boosting Classifier (GBC), Bagging Classifier (BC), and Multi-Layer Perceptron (MLP). The analysis was performed on benchmark NASA datasets, specifically CM1, KC1, KC2, JM1, and PC1. The experimental results displayed that the working algorithms achieved higher average accuracy rates on the PC1 = 92.2% dataset. Among classifiers, the Random Forest learning models with the Principal Component Analysis (PCA) approach exhibited boosted average performance across the datasets. In (Wang, et al.,2021) the researchers addressed the challenge of handling a large volume of software defect reports in software development. They announced a software defect prediction (SDP) model based on LASSO–SVM to enhance prediction accuracy. This model combined feature selection using the minimum complete value compression and selection method with the support vector machine algorithm. This approach significantly enhanced prediction accuracy, with simulation results showing an accuracy of 93.25% and 66.67%, a recall rate of 78.04%, and an f-measure of 72.72%. In the point of view (Alsghaier, et al.,2020) introduced different approach is developed by integrating genetics algorithm (GA) with support vector machine (SVM) and particle swarm for software defect prediction as a stand however for improved software defect prediction technique. The developed approach is applied into 24 datasets (12-NASA MDP and 12-Java open-source projects), wherever NASA MDP is considered as a large-scale dataset and Java open-source projects are considered as a small-scale dataset. Results show that integrating GA with SVM and particle swarm algorithm improves the performance of the software defect prediction process when it is applied into large-scale and small-scale datasets and overcome the limitations in the previous studies. (Rath, et al.,2022) Suggested an algorithm based on support vector machines (SVM) and extreme learning machines (ELM) for software reliability prediction. This algorithm is explored the factors impacting prediction accuracy, such as using previous defect data and the appropriate type of incident information. this proposed a model for feature selection using ELM and SVM by addressing dataset imbalance issues through the resampling method and applying it to NASA Metrics datasets. Experimental results displayed that the ELM-based reliability prediction model achieves higher accuracy, specificity, recall, precision, and F1-measure than SVM. A research paper by (Ali, et al., 2024) focus on Software defect prediction (SDP) through model consists of a two-stages prediction process to predict defective modules. In the first stage, four supervised machine learning are hired 1-Random Forest (RF), 2-Support Vector Machine (SVM), 3-Naïve Bayes (NB), and 4-Artificial Neural Network (ANN). These algorithms are optimized through iterative parameter optimization to achieve the highest accuracy possible. In the second stage, the predictive accuracy of the individual classifiers is integrated into a voting ensemble to make the final predictions. This ensemble approach further improves the accuracy and reliability of the defect predictions. Seven past incident benchmarks extracted from the NASA MDP repository, CM1, JM1, MC2, MW1, PC1, PC3, and PC4. The results determine that each dataset's proposed intelligent system achieved accuracy by the proposed algorithm voting ensemble-based software defect prediction (VESDP) model. Another paper provided by (Ali, et al., 2024) researchers proposed a new algorithm framework as IECGA, which provides approach to enhancing the accuracy and effectiveness of the software defect prediction lifecycle. Over general experimentation on NASA benchmark, consisted of CM1, JM1, MC2, MW1, PC1, PC3 and PC4, the IECGA algorithm demonstrated its ability to achieve higher accuracy rank on the PC1 = 95.1% dataset.

3. Methodology

In this part, the methodology will contain two different parts data collection, preprocessing and brief overview of the task and models used.

3.1. Data Collection and Preprocessing

Data Collection and Preprocessing the NASA Metrics Data Program (MDP) software defect datasets were collected from various NASA software projects. These datasets encompass information on software modules each having their attributes and a binary label indicating whether the module is defective or not. The datasets were preprocessed to remove any missing values by imputing the missing values by the mean of the column and removing duplicates found in the datasets.

3.2. Brief Overview of the Task and Models Used

A common task in the field of machine learning is the binary classification which refers to the process of classifying elements into two distinct groups based on certain characteristics or features. so, the task was to predict whether a data point belongs to the majority group or the minority group. To accomplish this task, four different machine learning models were employed. These models represent a variety of algorithmic approaches and were chosen for their distinctive strengths in handling classification tasks.

i. Support Vector Machine

The Support Vector Machine (SVM) is a widely utilized algorithm in the field of machine learning, particularly for classification and regression tasks. According to (Cortes and Vapnik 1995), who first introduced the concept, SVM is a classifier which performs its task by constructing a hyperplane in a multi-dimensional space that separates data points of different classes. The The SVM model operates on the principle of maximizing the margin, which is the distance between the hyperplane (decision boundary) and the nearest data points from different classes, known as support vectors.
The hyperplane is defined as: $w \cdot x + b = 0$ where

w is the weight vector.

x is the input data vector.

and b is the bias term.

The decision function for SVM is given by:

$$f(x) = sign(w.x + b) \qquad eq(3.1)$$

where

$f(x)$predicts the class label of input vector x.

$sign(.)$ returns the sign of its argument.

In cases where the data is not linearly separable from its original feature space, SVM can use the kernel trick to implicitly map the data to a higher-dimensional space where it becomes separable see **Fig.1**.

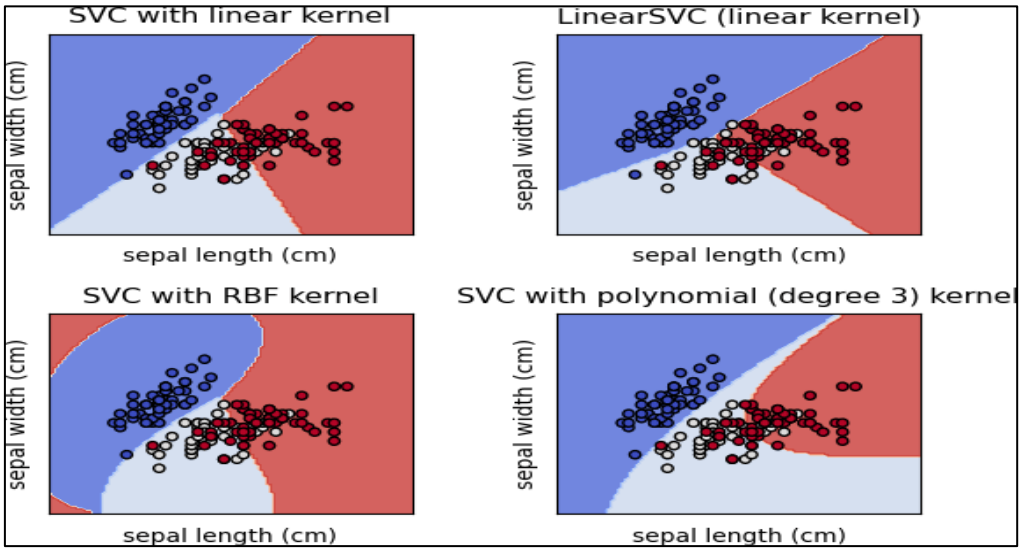

Fig-1. SVM Classifier

Linear Kernel

$$K(x_i, x_j) = x^T . x_j \qquad eq\ (3.2)$$

Explanation: The linear kernel represents the dot product between the input feature vectors x_i and x_j. This kernel is used when the data is linearly separable in the original feature space.

Polynomial Kernel

$$K(x_i, x_j) = (\gamma x^T + r_i)^d \qquad eq\ (3.3)$$

Explanation: The polynomial kernel maps the input data into a higher-dimensional space using a polynomial function. The hyperparameters *d* and *r* are user-defined positive integers that control the degree of the polynomial and a constant term, respectively. The parameter γ is a scaling factor that influences the dot product.

Gaussian (RBF) Kernel:

$$K(x_i, x_j) = exp(-\gamma \|x_i - x_j\|^2) \qquad eq\ (3.4)$$

Explanation: The Radial Basis Function (RBF) kernel measures the similarity be- tween two data points using the Gaussian distribution. It implicitly maps the data into a high-dimensional space, making it suitable for handling nonlinear relationships. The hyperparameter γ controls the width of the Gaussian kernel.

Sigmoid Kernel:

$$K(x_i, x_j) = tanh(\gamma x^T x_{ji} + r) \qquad eq\ (3.5)$$

Explanation: The sigmoid kernel uses the hyperbolic tangent function to map the data into a higher-dimensional space. This kernel can be useful in some cases, but it is generally less commonly used compared to linear, polynomial, and Gaussian kernels.

Laplacian Kernel

$$K(x_i, x_j) = exp\ (-\frac{\|x_i - x_j\|}{\gamma}) \qquad eq\ (3.6)$$

Explanation: The Laplacian kernel measures the similarity between two data points using the Laplace distribution. It is similar to the Gaussian kernel but has a sharper peak, making it robust to outliers.

This characteristic makes SVM highly effective in high-dimensional spaces and in situations where the number of dimensions is greater than the number of samples (Hearst et al. 1998). In addition to performing linear classification, SVMs can also effectively handle non-linear classification using what is called the kernel trick, implicitly mapping their inputs into high-dimensional feature spaces (Scho¨lkopf et al,1997). In terms of binary classification, the SVM model seeks a hyperplane that separates the data into two classes while maximizing the margin between the classes.

ii. Random Forest

Random Forest is a useful and well known machine-learning algorithm known. It was first introduced by (Breiman, 2001) as an extension of the decision tree algorithm. A Random Forest runs by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classification or regression of the individual trees. This is illustrated through:

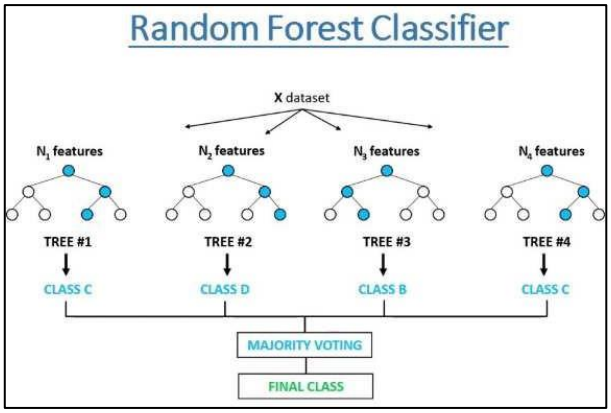

Fig-2. Random Forest

This idea of combining several models to improve the predictive performance is known as ensemble learning (Dietterich, 2000a). Random Forest attempts to mitigate the high variance problem of individual decision trees, where small changes in the training set can result in significantly different tree structures. By averaging the results of a collection of de-correlated trees, it reduces the risk of overfitting and typically provides better predic- tive performance (Sagi and Rokach ,2018). Random Forest also has an inherent feature selection mechanism, as it ranks features (variables) based on their ability to improve the purity of the node, measured by the Gini impurity or entropy (D´ıaz-Uriarte and De Andres, 2006).

iii. Logistic Regression

The Logistic Regression model is a powerful tool for binary arrangement. Logistic Regression is a supervised machine learning algorithm used for classification problems. Unlike linear regression which predicts continuous values it predicts the probability that an input belongs to a specific class (Juliana et al, 2016). One of the main advantages of Logistic Regression is its interpretability. Each feature is assigned a coefficient that describes its relative importance and direction of association with the assumption that there is a linear relationship between the log odds of the dependent variable and the independent variables. It also requires that the ob- servations be independent, and the absence of multicollinearity among the independent variables To apply logistic regression, it is required that the observations be independent and all the independent variables be independent (there is no Multicollinearity) (Peng et al, 2002). The equation of logistic regression is defined as follow eq(3.7).

$$P = \frac{1}{1 + e^{-(\beta_0+\beta_1 x)}} \qquad eq\ (3.7)$$

where
$-(\beta_0 + \beta_1 x)$ is linear function

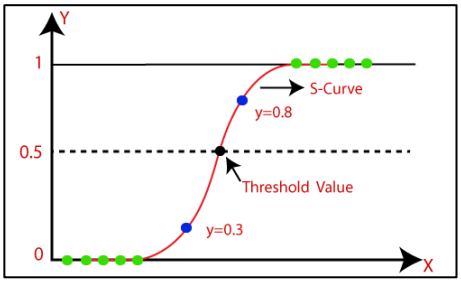

Fig-3. Logistic function

iv. Ensemble Model

Ensemble Models are powerful machine learning tools that combine multiple base models to improve prediction accuracy and robustness over a single model. The central idea behind ensemble models often referred to as the “Wisdom of the Crowd”, is that a group of weak learners can come together to form a strong (learner, D, 2000b).

There are several types of ensemble methods, including Bagging, Boosting, and Stacking. Bagging, or Bootstrap Aggregating, involves training each model in the ensemble us- ing a randomly drawn subset of the training set. Boosting is a sequential process, where each subsequent model attempts to correct the mistakes of the previous models. Stack- ing involves training a model to combine the predictions of several other models (Wolpert, 1992). Ensemble models have been shown to significantly increase accuracy on a variety of tasks, including both regression and classification problems (Opitz and Maclin , 1999).

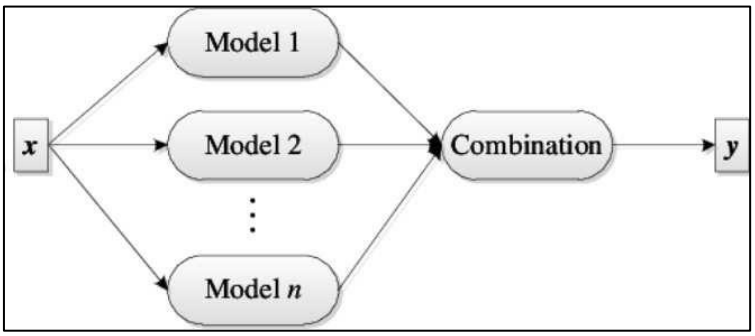

Fig-4. Ensemble Model

The Ensemble Model,,that represented by **Fig.4,** employed in this research is a type of model combination tech- nique that aims to leverage the strengths of multiple individual models to achieve im- proved predictive performance. Specifically, a Voting Classifier was used, which com- bines the predictions of several base models (Support Vector Machine, Random Forest, and Logistic Regression) through a majority vote or averaging process. The rationale behind using an ensemble approach is rooted in the principle of ”wisdom of the crowd,”

which suggests that a group of diverse models can often outperform any single model by capturing different aspects of the underlying data distribution and compensating for each other's weaknesses.

The Voting Classifier operates as follows:

- Each base model (SVM, Random Forest, Logistic Regression) is trained indepen- dently on the same training data.
- During prediction, each base model makes its individual prediction for a given input instance.
- The Voting Classifier aggregates these predictions using a majority vote scheme for classification tasks:
  - If the majority of base models predict an instance as belonging to the positive class (defective), the Voting Classifier assigns it to the positive class.
  - If the majority of base models predict an instance as belonging to the negative class (non-defective), the Voting Classifier assigns it to the negative class.
  - In case of a tie, the Voting Classifier can either choose a class based on a pre- defined preference or employ a strategy such as random selection or weighting the base models.

The final prediction of the Ensemble Model is the class label determined by the majority vote.

The diversity among the base models is a crucial factor contributing to the effec- tiveness of the Ensemble Model. In this study, the base models were chosen to capture different inductive biases and learning paradigms:

- Support Vector Machines learn decision boundaries in high-dimensional feature spaces and can model complex non-linear relationships.
- Random Forests employ an ensemble of decision trees and are robust to noise and overfitting through their randomized tree construction and feature selection pro- cesses.
- Logistic Regression is a well-established statistical model that provides interpretable coefficients and can capture linear relationships between features and the output.

By combining these diverse models, the Ensemble Model aims to leverage their collec- tive strengths, mitigate individual weaknesses, and potentially achieve higher predictive accuracy and generalization capability compared to any single model alone.

Furthermore, ensemble models tend to be more robust to noise, outliers, and over- fitting, as they average out biases, reduce variance, and are not likely to model random noise in the output data (Zhou, et al, 2012). However, ensemble models can be computationally expensive and may require more time to train and predict than individual models. Also, they may suffer from complexity, which may make them harder to interpret than individ- ual models (Rokach, 2010).

3. Performance Metrics

In machine learning, the performance of a model is evaluated based on certain metrics that depend on the nature of the task whether it is a classification, regression, or clustering task. This paper focuses on the metrics used to evaluate classification models, including accuracy, precision, recall, F1-score, and Area Under the Receiver Operating Characteris- tic Curve (AUROC). Accuracy is the ratio of correctly predicted observations to the total observations is the most straightforward measure but can be misleading in the case of imbalanced classes (Powers , 2011).

$$accuracy = \frac{(TP+TN)}{(TP+TN+FP+FN)} \qquad eq(4.1)$$

Precision, the ratio of correctly predicted positive observations to the total predicted pos- itives is an essential metric when the cost of false positives is high (Sokolova and Lapalme 2009).

$$precision = \frac{TP}{(TP+FP)} \qquad eq(4.2)$$

Recall measures the ratio of correctly predicted positive observations to all observations in the actual class, and is crucial when the cost of false negatives are high (Sokolova and Lapalme ,2009).

$$Recall = \frac{TP}{(TP+FN)} \qquad eq(4.3)$$

The F1-score is the weighted average of precision and recall and is typically more useful than accuracy, particularly for uneven class distributions (Van Rijsbergen ,1979).

$$F1\ score = \frac{2TP}{(2TP+Fp+FNN)} \qquad eq(4.4)$$

Cross Validation

According to (Kohavi 1995), Cross-validation is a powerful preventative measure against overfitting. The technique is used to assess how the results of a statistical analysis will generalize to an independent data set. It is a popular method because it is simple to un- derstand and because it generally results in a less biased or less optimistic estimate of the model skill than other methods, such as a simple train/test split. In machine learning, we cannot fit the model on the training data and say the model will work well with the real-world data. We need some kind of assurance or approximation about how our model will behave in the future or unseen data. This is where Cross-validation plays a crucial role.

(Kohavi, 1995) explained that Cross-validation is a resampling procedure used to evaluate machine learning models on a limited data sample. It provides a more accurate method of measuring the performance of a model, which helps in improving the model's performance and optimizing the hyperparameters. The most common type of cross-validation, and the one often referred to by this name, is k-fold cross-validation. In k-fold cross-validation, you divide your data points into k subsets of roughly equal size. You then run k separate learning experiments:

- Randomly partition the data into k equal-sized subsets. One of the k subsets is used as the test set, and the other k-1 subsets are put together to form a training set.
- Train a machine learning model on the k-1 training sets and test the model on the test set, recording the test set error rate.
- Repeat the process k times, each time using a different subset as your test set.

Finally, compute the average of the k-test set error rates. This is the cross-validation error rate, an estimate of the model's prediction error. This process is illustrated in **Fig.5** below:

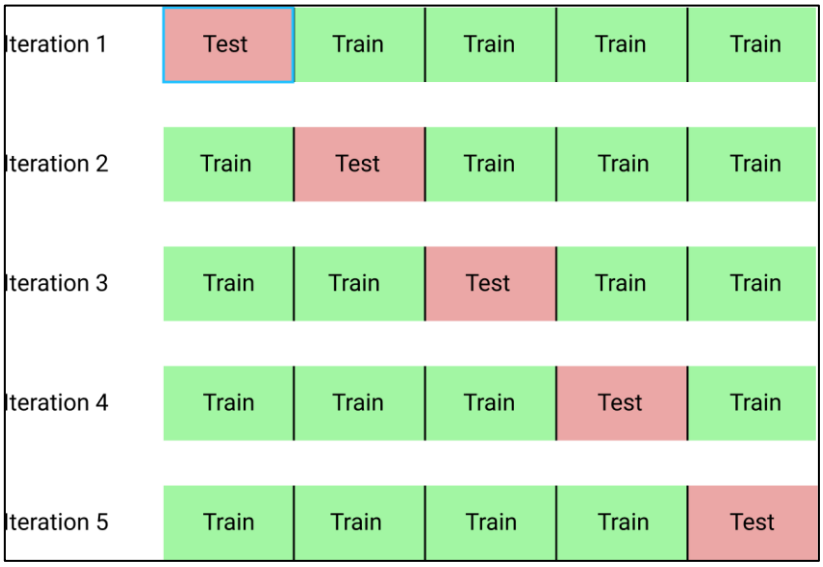

Fig-5. kfold

Cross-validation helps in making the model more robust by validating it on different subsets of data and then taking the average of all the performances which helps in improv- ing the model stability. A model's effectiveness is determined by its ability to perform well on unseen data. Cross-validation, such as k-fold cross-validation, helps ensure that the model's performance is not just tied to the way the data was split.

Hyperparameter Tuning

Hyperparameter tuning is a fundamental step in the construction of machine learning models. The performance of many machine learning algorithms depends on their hyper parameters. These are the configuration variables that govern the training process itself. This process involves adjusting the parameters of a model to improve its accuracy and performance. The optimal hyperparameters for a model can significantly differ based on the data and problem at hand, necessitating a comprehensive search for the most suitable values. For example, the learning rate for training a neural network, the C and sigma in a support vector machine. They are not learned by the model during training and must be set prior to training. Manual tuning of these hyperparameters is often tedious and suboptimal, leading to inefficient use of computational resources. Grid search is often the go-to method for hyperparameter tuning, but it suffers from the curse of dimension ality. The number of hyperparameter combinations to be searched grows exponentially with the number of hyperparameters, making grid search computationally very expensive.

Random Search Technique

Random search is a technique where random combinations of the hyperparameters are used to find the best solution for the built model. It is one of the most commonly used methods for hyperparameter optimization. The method was first proposed by (Bergstra and Bengio ,2012). Random search contrasts with grid search in that it

does not exhaus- tively try all the parameter combinations, but rather samples a fixed number of parameters sets at random from the parameter space.

The procedure of random search can be summarized as follows:

- Define a domain of possible values for each hyperparameter.
- Define a budget of the total number of hyperparameter configurations to try.
- For each iteration within the budget, sample a configuration of hyperparameters from the defined domain at random.
- Train a model with the selected hyperparameter configuration and evaluate it.
- Select the hyperparameter configuration that yields the best performance on the validation set.

The main advantage of random search is its simplicity and efficiency. It is straight forward to implement and computationally much more efficient than exhaustive search methods like grid search. Furthermore, according to Bergstra and Bengio's research, random search is more efficient for hyperparameter optimization than grid search when considering the same computational budget. However, a potential disadvantage of random search is that it might not find the optimal hyperparameters if the budget is not big enough, especially if the hyperparameter space is large. It is also possible that random search spends too much time exploring irrelevant regions of the hyperparameter space.

5. Model Performance Analysis

Four different machine learning models were evaluated in this study, which included a Support Vector Machine (SVM), a Random Forest, a Logistic Regression, and an Ensem- ble model. The performance of these models was assessed based on a binary classification task.

In conclusion, although high accuracy was demonstrated by all models, the other met- rics were low. These results indicated the presence of a class imbalance in the dataset, a common problem in machine learning tasks. Despite the high accuracy demonstrated.

Table-5.1. Accuracies for each model on all datasets

Model	MC1	KC1	CM1	PC1	MC2	PC3	MW1	KC3	JM1	PC4
SVM	0.78	0.76	0.71	0.97	0.69	0.84	0.95	0.98	0.91	0.89
Random Forest	0.78	0.74	0.83	0.96	0.75	0.83	0.95	0.98	0.90	0.90
Logistic Regression	0.79	0.74	0.75	0.97	0.69	0.88	0.95	0.98	0.91	0.91
Voting Classifier	0.79	0.74	0.71	0.97	0.72	0.86	0.95	0.98	0.91	0.89

Table-5.2. Recall for each model on all datasets

Model	MC1	KC1	CM1	PC1	MC2	PC3	MW1	KC3	JM1	PC4
SVM	0.03	0.14	0.64	0.00	0.64	0.08	0.20	0.00	0.00	0.36
Random Forest	0.20	0.32	0.91	0.00	0.43	0.00	0.10	0.00	0.07	0.39
Logistic Regression	0.12	0.21	0.64	0.09	0.64	0.33	0.30	0.00	0.30	0.52
Voting Classifier	0.08	0.17	0.64	0.00	0.64	0.08	0.20	0.00	0.04	0.39

Table-5.3. Precision for each model on all datasets

Model	MC1	KC1	CM1	PC1	MC2	PC3	MW1	KC3	JM1	PC4
SVM	0.71	0.82	0.70	0.00	0.64	0.50	0.67	0.00	0.00	0.89
Random Forest	0.48	0.55	0.77	0.00	1.00	0.00	0.50	0.00	0.33	0.94
Logistic Regression	0.58	0.54	0.78	0.50	0.64	0.80	0.50	0.00	0.53	0.85
Voting Classifier	0.64	0.58	0.70	0.00	0.69	1.00	0.67	0.00	0.50	0.89

Table-5.4. F1-scores for each model on all datasets

Model	MC1	KC1	CM1	PC1	MC2	PC3	MW1	KC3	JM1	PC4
SVM	0.05	0.23	0.67	0.00	0.64	0.14	0.31	0.00	0.00	0.52
Random Forest	0.28	0.40	0.83	0.00	0.60	0.00	0.17	0.00	0.12	0.55
Logistic Regression	0.20	0.30	0.70	0.15	0.64	0.47	0.38	0.00	0.38	0.65
Voting Classifier	0.14	0.26	0.67	0.00	0.67	0.15	0.31	0.00	0.07	0.54

by all models, difficulties were encountered in effectively classifying the minority class. Strategies such as

resampling techniques are suggested to address this issue. The model's performance should not only be assessed on overall accuracy but also on its ability to identify each class accurately, especially when dealing with imbalanced datasets.

6. Proposed Method (Proposed Enhancement to Defect Prediction Model)

6.1. The Problem with Imbalanced Data Sets

Class imbalance is a common problem in machine learning classification where there is a disproportionate ratio of observations in each class. Class imbalance can be found in many different areas including medical diagnosis, spam filtering, and fraud detection. The main issue with class imbalance is that most machine learning algorithms work best when the number of samples in each class is about equal. This is because most algorithms are designed to maximize accuracy and reduce error. In the context of software defect prediction, defective modules typically constitute the minority class compared to non-defective modules. This imbalance poses a challenge for most machine learning al- gorithms, which are designed to optimize overall accuracy and may be biased towards the majority class, leading to poor performance in identifying the minority class instances (He and Garcia , 2009).

6.2. Synthetic Minority Over-sampling Technique (SMOTE)

To address the class imbalance issue, the Synthetic Minority Over-sampling Tech- nique (SMOTE) was employed in this study. Synthetic Minority Over-sampling Tech- nique, or SMOTE, is a popular algorithm to create synthetic observations of the minority class. It was presented by (Chawla et al. 2002).

SMOTE works by selecting examples that are close to the feature space, drawing a line between the examples in the feature space and drawing a new sample at a point along that line. Specifically, a random example from the minority class is first chosen. Then k of the nearest neighbors for that example are found (typically k=5). A randomly selected neighbor is chosen and a synthetic example is created at a randomly selected point between the two examples in feature space.

The SMOTE algorithm works by introducing synthetic examples along the line seg- ments joining any/all of the k minority class nearest neighbors. Specifically, the process can be summarized as **Fig.6**:

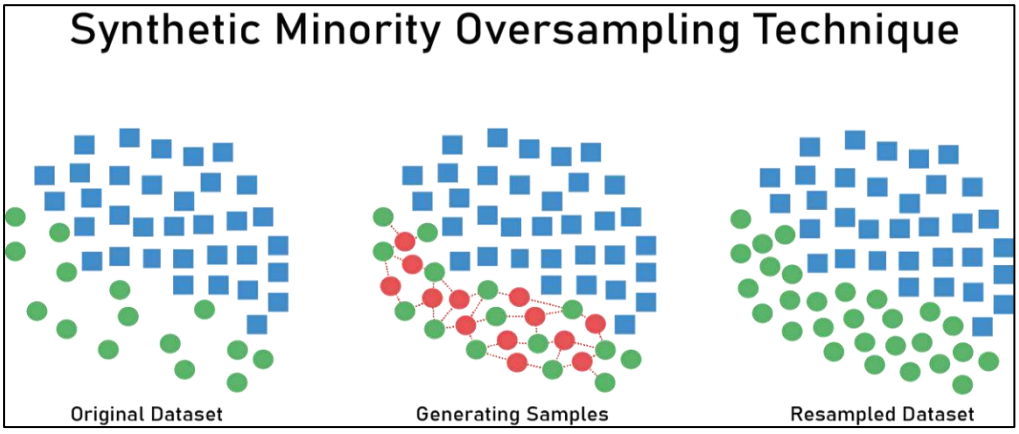

Fig-6. Schematic of the SMOTE algorithm

For each instance x_i in the minority class, compute its k nearest neighbors from the same class using a distance metric (e.g., Euclidean distance).

- Select a neighbor x_{nn} from the k nearest neighbors.
- Generate a synthetic instance x_{new} along the line segment between xi and xnn, using the equation:

$$x_{new} = x_i + \lambda \,.\, (x_{nm} - x_i) \qquad eq(6.1)$$

where λ is a random value between 0 and 1.

- Repeat steps 2 and 3 for all k nearest neighbors or until the desired number of synthetic instances is generated.

By creating synthetic instances in the neighborhood of existing minority class in- stances, SMOTE effectively forces the decision regions of the machine learning model to become more generalized, improving its ability to recognize minority class patterns (He et al. , 2008).

6.3. Efficacy and Limitations of SMOTE

According to (Ferna´ndez et al., 2018) the main advantage of SMOTE is that it can improve the performance of the minority class by generating synthetic examples that are quite similar to the existing observations in the minority class. However, one potential drawback of SMOTE is that it can increase the likelihood of overfitting since it gener- ates synthetic examples without considering the majority class. New synthetic examples could be

generated that are quite similar, or even identical, to existing examples. Further- more, synthetic examples are generated without considering the majority class, possibly resulting in ambiguous examples if there is a strong overlap for the classes.

7. Implementation

The SMOTE algorithm was utilized implemented in the imbalanced-learn library in python, and setting the number of neighbors to 5 which is the default and it's sufficient for most cases see **Fig.7**.

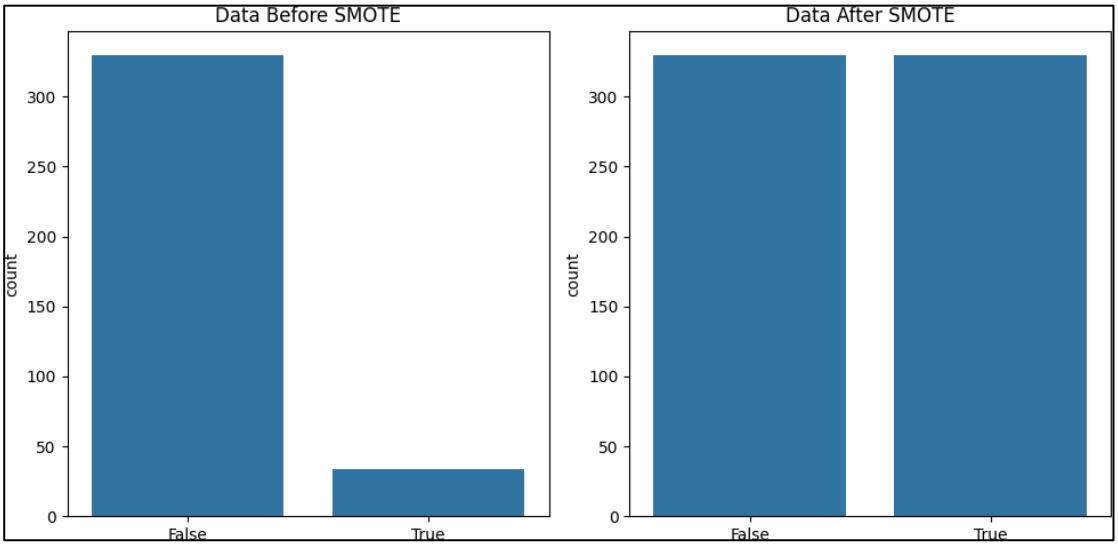

Fig-7. Data distribution before and after using SMOTE

8. Results and Discussion

After Training SVM, Random Forest, Logistic Regression, and Ensemble models on the balanced dataset and evaluated their performance. The results showed a significant improvement in the performance of all models in precision, recall and consequently the f1-score. The best performance was observed with the Random Forest model.

Table-1. Accuracies for each model on all datasets after using SMOTE

Model	MC1	KC1	CM1	PC1	MC2	PC3	MW1	KC3	JM1	PC4
SVM (SMOTE)	0.75	0.67	0.71	0.81	0.63	0.73	0.82	0.84	0.75	0.83
Random Forest (SMOTE)	0.76	0.74	0.79	0.96	0.78	0.91	0.93	0.97	0.84	0.90
Logistic Regression (SMOTE)	0.72	0.63	0.75	0.83	0.59	0.74	0.80	0.83	0.77	0.83
Voting Classifier (SMOTE)	0.74	0.66	0.71	0.84	0.63	0.79	0.83	0.87	0.77	0.85

Table-2. Recall for each model on all datasets after using SMOTE

Model	MC1	KC1	CM1	PC1	MC2	PC3	MW1	KC3	JM1	PC4
SVM (SMOTE)	0.75	0.47	0.64	0.64	0.71	0.58	0.80	0.17	0.70	0.86
Random Forest (SMOTE)	0.76	0.44	0.82	0.09	0.71	0.42	0.30	0.00	0.33	0.66
Logistic Regression (SMOTE)	0.72	0.56	0.64	0.36	0.71	0.58	0.80	0.17	0.74	0.86
Voting Classifier (SMOTE)	0.74	0.48	0.64	0.45	0.71	0.58	0.80	0.17	0.70	0.86

Table-3. Precision for each model on all datasets after using SMOTE

Model	MC1	KC1	CM1	PC1	MC2	PC3	MW1	KC3	JM1	PC4
SVM (SMOTE)	0.43	0.40	0.70	0.09	0.56	0.30	0.20	0.02	0.23	0.49
Random Forest (SMOTE)	0.43	0.52	0.75	0.14	0.77	1.00	0.30	0.00	0.24	0.71
Logistic Regression (SMOTE)	0.40	0.38	0.78	0.06	0.53	0.32	0.18	0.02	0.25	0.49
Voting Classifier (SMOTE)	0.43	0.40	0.70	0.08	0.56	0.39	0.21	0.03	0.25	0.52

Table-4. F1-scores for each model on all datasets after using SMOTE

Model	MC1	KC1	CM1	PC1	MC2	PC3	MW1	KC3	JM1	PC4
SVM (SMOTE)	0.42	0.43	0.67	0.15	0.63	0.40	0.32	0.04	0.35	0.62
Random Forest (SMOTE)	0.38	0.48	0.78	0.11	0.74	0.59	0.30	0.00	0.28	0.68
Logistic Regression (SMOTE)	0.45	0.45	0.70	0.11	0.61	0.41	0.30	0.04	0.37	0.63
Voting Classifier (SMOTE)	0.44	0.44	0.67	0.14	0.63	0.47	0.33	0.05	0.37	0.65

8.1. Feature Selection

Feature selection is a crucial step in the machine learning pipeline. It is the process of selecting the most relevant features from the original dataset to use in model training. The aim is to enhance the performance of the machine learning model By using feature selection, we can simplify data entry, reduce size, and improve model performance in several ways according to (Guyon and Elisseeff, 2003).

- Reduce overfitting: Feature selection helps prevent overfitting by focusing on the most important features that contain the most predictive information.
- Faster training: Reducing the number of features shortens the training time of a machine learning as there is less data to process in the model.
- Enhanced interpretation: We can improve the interpretation of models by selecting the most important features. This means we can better understand which features are relevant to the model's predictions and extract insights from the model's behavior.

There are two main categories of feature selection techniques: filter methods and wrapper methods.

8.1.1. Wrapper Methods

According to (Kohavi and John, 1997) The wrapper method is a type of feature selec- tion technique that relies on the performance of a machine learning model to evaluate the importance of features. It involves training a model on a subset of features, evaluating its performance, and using that information to decide whether to add or remove features. This process repeats until it reaches an optimal set of features. The wrapper method is computationally expensive as it involves evaluating the performance of a machine- learning model for every possible combination of features. However, it often provides a better-performing feature set compared to other methods, like filter methods, because it considers the interaction of features. There are different types of wrapper methods, including forward selection, backward elimination, and recursive feature elimination. Forward selection starts with an empty set and adds one feature at a time. Backward elimination starts with all features and removes one feature at a time. Recursive feature elimination is a greedy optimization algorithm that aims to find the best-performing feature subset.

Recursive Feature Elimination with Cross-Validation (RFECV)

RFECV is a powerful technique for feature selection in machine learning that fits a model and removes the weakest feature (or features) until the specified number of features are reached as mentioned by (Guyon et al. 2002). Features are ranked and by recursively eliminating a small number of features per loop, RFECV attempts to eliminate dependencies and collinearity that may exist in the model. It combines the strengths of recursive feature elimination (RFE) and cross-validation to provide a robust and efficient method for selecting the most important features and optimizing the performance of a machine learning model. Then select the optimal number of features based on the cross-validation score. The cross-validation score generally increases with the number of features. However, the score may also increase due to overfitting. RFECV solves this problem by selecting the number of features for which the cross-validation the score is maximum. RFECV works by iteratively eliminating the least important features in a dataset while using cross.

validation to evaluate the performance of the model. The process can be broken down into the following steps:

- Initialize the model with all the features: The first step is to initialize the model with all the features in the dataset.
- Perform cross-validation: The next step is to perform cross-validation on the dataset, using the initialized model. This involves splitting the dataset into training and testing sets and evaluating the model's performance on the testing sets.
- Eliminate the least important features: Based on the performance of the model in the cross-validation step, the least important features are eliminated from the dataset.
- Repeat steps 2–3: Steps 2–3 are repeated until a stopping criterion is met, such as a maximum number of iterations or a minimum number of features.
- Evaluate the final model: The final model is evaluated on the entire dataset to determine its performance.

According to (Chen et al. 2020), RFECV has several advantages over other feature selections. RFECV can improve the accuracy of a machine learning model by selecting the most important features. It is computationally efficient, as it only requires a single pass through the dataset to eliminate the least important features. Also,

provides interpretable results, as it eliminates features one at a time, allowing for feature importance to be easily understood. However, RFECV needs a specified or calculable measure of importance provided by the estimator to perform its operations. Therefore, it doesn't work with all kinds of models. For instance, it won't work directly with models like KNN, and SVM with non-linear kernels as these models do not provide a straightforward measure of feature importance.

8.1.2. Filter Methods

Filter methods operate independently of any specific machine learning model. They rely on statistical measures to evaluate individual features and assign them a score based on their presumed relevance to the target variable(Delany et al, 2012). These measures don't involve training a model, making filter methods computationally efficient and scal- able for large datasets.

Advantages of Filter Methods:

- Computational efficiency: Filter methods are much faster than wrapper methods as they don't involve training models on multiple feature subsets.
- Scalability: They can be applied to large datasets without significant computational burden.
- Model independence: The selection process is not biased towards a specific ma- chine learning model.

Drawbacks of Filter Methods:

- Overlooking feature interactions: Filter methods don't consider how features might interact with each other, potentially missing important relationships.
- Suboptimal for specific models: The chosen features might not be optimal for the performance of a particular machine learning model.

8.1.3. Information Gain

Information Gain, commonly referred to as Mutual Information (MI), quantifies the decrease in ambiguity regarding the objective variable subsequent to the evaluation of a specific characteristic. This metric capitalizes on the principle of entropy, a measure

that gauges the level of randomness or indeterminacy connected to a stochastic variable (MacKay ,2003).

The Core Idea:

- Entropy of the Target Variable ($H(Y)$): This represents the initial uncertainty about the target variable.
- Entropy after Considering a Feature ($H(Y|X)$): This represents the remaining un- certainty about the target variable after observing a specific feature.
- Information Gain ($IG(X, Y)$): This is the difference between the initial entropy and the conditional entropy. It signifies the information gained about the target variable by considering the feature. Features with higher IG are deemed more relevant for predicting the target variable.

Benefits of Information Gain:

- Versatility: Applicable to both categorical and continuous features, making it a flexible choice for mixed datasets.
- Interpretability: Provides a clear measure of the information a feature contributes to predicting the target variable.
- Scalability: Can be efficiently applied to large datasets.

Limitations of Information Gain:

- Computational Cost: While generally efficient, it can be more computationally ex- pensive compared to simpler methods like chi-square, especially for large datasets.
- Independence Assumption: Assumes independence between features, which might not always be true in real-world data.

8.2. Comparing Results

The two feature selection techniques, Recursive Feature Elimination Cross-validation (RFECV) and Information Gain (IG), were compared to see how they affect the performance of various machine learning models. The **tables (1) : table(8)** show the accuracy, recall, precision, and F1-score of different models on several datasets after applying SMOTE (Synthetic Minority Oversampling Technique) and the two feature selection techniques. Overall, the random forest appears to perform the best based on the highest accuracy across all datasets in both feature selection techniques9. However, for recall10 and F1- score12, random forest only achieves the best results on some datasets. It's important to consider all these metrics together to choose the best model and feature selection technique depending on the specific task.

Here's a more detailed comparison:

- Accuracy: RFECV resulted in higher accuracy for most models compared to IG.
- Recall: The results are mixed, with both techniques achieving similar recall scores for most models.
- Precision: RFECV resulted in higher precision for most models compared to IG.
- F1-score: Similar to recall, the results are mixed with both techniques achieving similar F1-scores for most models.

Table-5. Accuracies for each model on all datasets after using SMOTE and Feature Selection

Model	MC1	KC1	CM1	PC1	MC2	PC3	MW1	KC3	JM1	PC4
SVM (SMOTE RFECV)	0.75	0.67	0.79	0.80	0.59	0.81	0.82	0.84	0.76	0.82
Random Forest (SMOTE RFECV)	0.76	0.74	0.88	0.96	0.78	0.91	0.93	0.96	0.84	0.90
Logistic Regression (SMOTE RFECV)	0.72	0.63	0.79	0.82	0.63	0.83	0.82	0.83	0.76	0.83
Voting Classifier (SMOTE RFECV)	0.75	0.66	0.88	0.82	0.63	0.82	0.83	0.87	0.78	0.84
SVM (SMOTE IG)	0.77	0.67	0.67	0.79	0.66	0.83	0.73	0.74	0.78	0.78
Random Forest (SMOTE IG)	0.77	0.75	0.83	0.96	0.69	0.88	0.91	0.96	0.81	0.90
Logistic Regression (SMOTE IG)	0.73	0.62	0.71	0.83	0.66	0.84	0.72	0.75	0.76	0.84
Voting Classifier (SMOTE IG)	0.76	0.66	0.67	0.84	0.66	0.83	0.75	0.75	0.78	0.86

Table-6. Recall for each model on all datasets

Model	MC1	KC1	CM1	PC1	MC2	PC3	MW1	KC3	JM1	PC4
SVM (SMOTE RFECV)	0.43	0.47	0.82	0.64	0.71	0.58	0.80	0.17	0.78	0.89
Random Forest (SMOTE RFECV)	0.34	0.44	0.91	0.09	0.71	0.42	0.40	0.00	0.37	0.68
Logistic Regression (SMOTE RFECV)	0.52	0.56	0.73	0.55	0.71	0.75	0.80	0.17	0.74	0.84
Voting Classifier (SMOTE RFECV)	0.47	0.48	0.82	0.55	0.71	0.67	0.80	0.17	0.74	0.86
SVM (SMOTE IG)	0.38	0.44	0.64	0.45	0.50	0.75	0.80	0.50	0.63	0.75
Random Forest (SMOTE IG)	0.37	0.48	0.82	0.09	0.57	0.33	0.50	0.00	0.52	0.70
Logistic Regression (SMOTE IG)	0.43	0.62	0.55	0.55	0.50	0.83	0.90	0.33	0.56	0.73
Voting Classifier (SMOTE IG)	0.37	0.47	0.64	0.45	0.50	0.75	0.80	0.33	0.63	0.73

Table-7. Precision for each model on all datasets

Model	MC1	KC1	CM1	PC1	MC2	PC3	MW1	KC3	JM1	PC4
SVM (SMOTE RFECV)	0.43	0.40	0.75	0.08	0.53	0.41	0.20	0.02	0.25	0.48
Random Forest (SMOTE RFECV)	0.44	0.52	0.83	0.14	0.77	1.00	0.36	0.00	0.26	0.71
Logistic Regression (SMOTE RFECV)	0.39	0.38	0.80	0.08	0.56	0.47	0.20	0.02	0.25	0.49
Voting Classifier (SMOTE RFECV)	0.43	0.40	0.90	0.08	0.56	0.44	0.21	0.03	0.27	0.51
SVM (SMOTE IG)	0.47	0.40	0.64	0.06	0.64	0.47	0.14	0.04	0.24	0.41
Random Forest (SMOTE IG)	0.47	0.54	0.82	0.11	0.67	0.80	0.29	0.00	0.25	0.69
Logistic Regression (SMOTE IG)	0.40	0.38	0.75	0.09	0.64	0.50	0.14	0.03	0.21	0.50
Voting Classifier (SMOTE IG)	0.45	0.40	0.64	0.08	0.64	0.47	0.15	0.03	0.24	0.56

Table-8. F1-scores for each model on all datasets

Model	MC1	KC1	CM1	PC1	MC2	PC3	MW1	KC3	JM1	PC4
SVM (SMOTE RFECV)	0.43	0.43	0.78	0.15	0.59	0.48	0.31	0.04	0.38	0.62
Random Forest (SMOTE RFECV)	0.39	0.48	0.87	0.11	0.78	0.59	0.38	0.00	0.31	0.70
Logistic Regression (SMOTE RFECV)	0.45	0.45	0.76	0.14	0.63	0.58	0.32	0.04	0.37	0.62
Voting Classifier (SMOTE RFECV)	0.45	0.44	0.86	0.14	0.63	0.53	0.33	0.05	0.39	0.64
SVM (SMOTE IG)	0.42	0.42	0.64	0.11	0.66	0.58	0.24	0.08	0.35	0.53
Random Forest (SMOTE IG)	0.41	0.51	0.82	0.10	0.69	0.47	0.37	0.00	0.34	0.70
Logistic Regression (SMOTE IG)	0.41	0.47	0.63	0.15	0.66	0.63	0.25	0.05	0.31	0.59
Voting Classifier (SMOTE IG)	0.41	0.43	0.64	0.14	0.66	0.58	0.25	0.05	0.35	0.63

9. Conclusion

Different machine learning techniques for software defect prediction is applied in this research using datasets from NASA's Metrics Data Program. Several challenges were encountered, including class imbalance in the datasets and the need for effective feature range. To address the class imbalance issue, the Synthetic Minority Oversampling Technique (SMOTE) was applied, which generated synthetic examples of the minority class. This meaningfully improved the performance metrics like precision, recall, and F1-score across different models, including Support Vector Machines, Random Forests, Logistic Regression, and Ensemble models. Furthermore, two feature selection techniques, Recursive Feature Elimination with Cross-Validation (RFECV) and Information Gain (IG), were implemented and compared. RFECV generally resulted in higher ac- curacy and precision, while the results for recall and F1-score were mixed across models and datasets. Among the evaluated models, the Random Forest model demonstrated the highest overall accuracy after applying SMOTE and feature selection techniques. For Example, of achievement: CM1 dataset achieved 88%, JM1 dataset achieved 84%, PC3 dataset achieved 91% and KC3, PC1 dataset achieved 96%. However, the choice of the best model and feature selection approach depends on the specific requirements and priorities, such as whether recall or precision is more critical for the given task.

This study shows the possibility of machine learning, particularly ensemble methods like Random Forests, for automating software defect prediction in critical systems like those developed by NASA. By addressing data imbalance and feature selection challenges, the performance of these models can be expressively improved. The best-performing model varied across datasets, but in general, Random Forest and Voting Classifier, especially when combined with SMOTE and feature selection, showed promising results. These models achieved competitive performance across all metrics, making them appropriate choices for handling imbalanced multivariate time series regression datasets.

Future research could discover more advanced techniques for handling class imbalance, such as cost-sensitive learning or hybrid oversampling methods. Additionally, investigating the interpretability and explain ability of the trained models could be valuable for understanding the factors contributing to software defects and informing the soft- ware development process. Overall, this research contributes to the growing field of ma- chine learning applications in software engineering, providing insights and techniques for improving the reliability and quality of software systems, which is crucial for mission- critical applications in aerospace and beyond.

Data Availability Statement

The dataset used in evaluation up on request.

Declarations

Conflict of Interest Statement The authors of this article declared that they have no conflict of interest.

References

Ali, Misbah, Mazher, T., Arif, Y., Al Otibi, S., Ghadi, Y, Shahzad, T.(2024). "Software defect prediction using an intelligent ensemble-based model." IEEE Access, 12., 20376-20395.

Ali, Misbah, Mazhar, T., Al- Rasheed, A., Shahzad, T, Ghadi, Y.,Khan, M.(2024). "Enhancing software defect prediction: a framework with improved feature selection and ensemble machine learning." PeerJ Computer Science 10 (2024): e1860.

Alsghaier, Hiba, and Mohammed Akour. (2020) "Software fault prediction using particle swarm algorithm with genetic algorithm and support vector machine classifier." Software: Practice and Experience 50.4 .407-427.

Bergstra , J. and Bengio ,Y. (2012) 'Random Search for Hyper-Parameter Optimization '. Journal of Machine Learning Research 13. 281-305.

Breiman, L.(2001). 'Random forests. Machine Learning' .45(1).5–32.

Chawla, N. V., Bowyer, K. W., Hall, L. O., and Kegelmeyer, W. P.(2002) 'Smote: synthetic minority over-sampling technique' . Journal of artificial intelligence research. 16. 321–357.

Chen, Q., Meng, Z., and Su, R.(2020)' Werfe: A gene selection algorithm based on recursive feature elimination and ensemble strategy'. Frontiers in Bioengineering and Biotechnology. 8.

Cortes, C. and Vapnik, V.(1995).'Support-vector networks. Machine Learning'. 20(3). 273–297.

Cetiner, Murat, and Ozgur Koray Sahingoz.(2020) "A comparative analysis for machine learning based software defect prediction systems." 2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT). IEEE.

D´ıaz-Uriarte, R. and De Andres, S. A. (2006) 'Gene selection and classification of microarray data using random forest' . BMC Bioinformatics. 7(1).1–13.

Delany, S., Buckley, M., and Greene, D.(2012) 'Sms spam filtering: methods and data.Expert Systems With Applications'. 39. 9899–9908.

Dietterich, T. G.(2000a)' Ensemble methods in machine learning'. 1–15.

Dietterich, T. G.(2000b)' Ensemble methods in machine learning'. Multiple classifier systems. 1857. 1–15.

Emmanuel Gbenga Dada, David Opeoluwa Oyewola, Stephen Bassi Joseph and Ali Baba Dauda (2021) Ensemble Machine Learning Model for Software Defect Prediction. Adv Mach Lear Art Inte, 2(1): 11-21.

Fernandez, A., Garc ´ ´ıa, S., Herrera, F., and Chawla, N. V. (2018)' Smote for learning from imbalanced data: progress and challenges' .marking the 15-year anniversary. Journal of artificial intelligence research. 61. 863–905.

Guyon, I. and Elisseeff, A.(2003)' An introduction to variable and feature selection'. Journal of machine learning research..3 .1157–1182.

Guyon, I., Weston, J., Barnhill, S., and Vapnik, V.(2002) 'Gene selection for cancer classification using support vector machines' . Machine Learning. 46. 389–422.

He, H. and Garcia, E. A.(2009) 'Learning from imbalanced data. IEEE Transactions on knowledge and data engineering'..21.1263–1284.

He, H., Bai, Y., Garcia, E. A., and Li, S.(2008) ' Adasyn: Adaptive synthetic sampling approach for imbalanced learning' . IEEE international joint conference on neural networks (IEEE world congress on computational intelligence).1322–1328.

Hearst, M., Dumais, S., Osuna, E., Platt, J., and Scholkopf, B.(1998) ' Support vector machines'. IEEE Intelligent Systems and their Applications. 13. 18–28.

Hoang, A. D. (2019). 'Fantastic Educational Gaps and Where to Find Them: LERB-A Model to Classify Inequity and Inequality'.Journal of International Education and Practice.2(4).

Hosmer Jr, D. W., Lemeshow, S., and Sturdivant, R. X. (2013)' Applied logistic regression'. John Wiley & Sons..

Iqbal, A. and Aftab, S. (2020)'A classification framework for software defect prediction using multi-filter feature selection technique and MLP'. International Journal of Modern Education and Computer Science,.12. 18–25.

Jørgensen, M., (1999)"Software quality measurement'. Adv. Eng. Softw, 30, 907–912.

Juliana, T.; Meurer, W.,J. (2016)"Logistic Regression Relating Patient Characteristics to Outcomes". JAMA. (2016).316 (5): 533–4.

Jureczko, M., Madeyski,L. (2010)'Towards identifying software project clusters with regard to defect prediction' he 6th International Conference on Predictive Models in Software Engineering,9., 1-10.

Khoshgoftaar, M., T.(2003)' Fault Prediction Modeling for Software Quality Estimation: Comparing Commonly Used Techniques' . Empirical Software Engineering. 8(3):255-283.

Kohavi, R. (995)'A study of cross-validation and bootstrap for accuracy estimation and model selection' .

Kohavi, R. and John, G. H.(1997)'Wrappers for feature subset selection.'.Artificial Intelligence. 97(1-2).273–324.

Laradji, I. H., Alshayeb, M., and Ghouti, L. (2015)'Software defect prediction using ensemble learning on selected feature'. Information and Software Technology.58. 388–402.

Liu, Y., Esan, O. C., Pan, Z., & An, L. (2021). Machine Learning for

Liu, Y., Esan, O. C., Pan, Z., and An, L(2021)'Machine Learning for Advanced Energy Materials'. Energy and AI, 3, 100049.

MacKay, D. J. (2003)'Information theory, inference and learning algorithms' . Cambridge university press.

Opitz, D. and Maclin, R. (1999)'Popular ensemble methods: An empirical study'. Journal of artificial intelligence research. 11. 169–198.

Peng, C.-Y. J., Lee, K. L., and Ingersoll, G. M. (2002)' An introduction to logistic regression analysis and reporting'. The journal of educational research.96(1). 3–14.

Petrić, J., Bowes, D., Hall, T., Christianson,B. and Baddoo,N. (2016)'The Jinx on the NASA software defect data sets' ACM International Conference Proceeding Series, New York, 1–5.

Powers, D. M. (2011)' Evaluation: from precision, recall and f-measure to roc'.informedness, markedness and correlation. arXiv preprint arXiv:2010.16061.

Ramadhani,A.,P, Nugroho,R.,A Faisal,M.,R., Abadi,F., Herteno, R.(2024)' The Impact of Software Metrics in NASA Metric Data Program Dataset Modules for Software Defect Prediction' TELKOMNIKA Telecommunication Computing Electronics and Control, 22, 846-853.

Rathore, S., Kumar, D., Singh, J. and Gupta, S. (2012) 'Homotopy Analysis Sumudu Transform Method for Nonlinear Equations'. International Journal of Industrial Mathematics, (2012), 4, 301-314.

Rokach, L.(2010) 'Ensemble-based classifiers. Artificial Intelligence Review' .33(1-2):139.

Rath, S. K.(2022) "A comparative analysis of SVM and ELM classification on software reliability prediction model." Electronics 11.17 : 2707.

Sagi, O. and Rokach, L.(2018)' Ensemble learning: A survey. Wiley Interdisciplinary Reviews'.Data Mining and Knowledge Discovery. 8(4):e1249.
Scholkopf, B., Smola, A., and M ¨ uller, K.-R.(1997)'Kernel principal component analysis'.583–588.
Sharma, H. (2022)' Logistic regression python implementation from scratch without using sklearn'.
Sokolova, M. and Lapalme, G.(2009)' A systematic analysis of performance measures for classification tasks' Information Processing & Management, 45(4).427–437.
Tosun, A. , Bener, B.,A., Turhan, B.(2009)' Feature weighting heuristics for analogy-based effort estimation models'.Expert Systems with Applications, 36. 10325-10333.
Van Rijsbergen, C. J. (1979)' Information retrieval. Butterworth-Heinemann'.
Wolpert, D. H. (1992)' Stacked generalization.' Neural networks, 5(2). 241–259.
Wang, K.(2021)"Software defect prediction model based on LASSO–SVM." Neural Computing and Applications 33 : 8249-8259.
Zhou, Z.-H.' Tsang, w., I.(2012)' Efficient Optimization of Performance Measures by Classifier Adaptation'.IEEE Transactions on Pattern Analysis and Machine Intelligence, 35(6).

Appendix

Authors	Year	Model	Classifiers	Dataset	Accuracy%
Cetiner, et al.	2020	PCA	DT, NB, KNN, SVM, RF, ET, AC, GBC, BC and MLP	NASA	PC1 = 92.2% CM1 = 87.8% KC1 = 85.0% KC2 = 82.3% JM1 = 80.3%
Alsghaier, et al.	2020	GASVM, PSO-SVM, GAPSO-SVM,	GA, SVM, swarm	12-NASA, 12-Java	GA with SVM = from 79% to 99% PSO-SVM = from 68.8% and 99.5% swarm algorithm and SVM = from 67% to 99%
Wang, et al.	2021	LDA, CA, BPNN, LR, LASSO–SVM	SVM	NASA	CM1 = 93.25%
Emmanuel, et al.	2021	Stacking,	KNN, GLMNet, LDA, RPART, RF,	NASA	CM1 = 87.69% JM1 = 81.11% PC3 = 90.70% KC3 = 94.74%
Rath, et al.	2022	SVM, ELM	SVM, ELM	NASA	ELM = 84.61% SVM = 78.68%
Ali, et al.	2024	voting(VESDP)	RF, SVM, NB, ANN	NASA	CM1 = 86.87% JM1 = 79.92% MC2 = 68.42% MW1 = 89.33% PC1 = 92.16% PC3 = 87.97% PC4 = 87.14%
Ali, et al.	2024	Voting	RF, SVM, NB	NASA	CM1 = 85.86 % JM1 = 79.66% MC2 = 71.05% MW1 = 89.33% PC1 = 95.1% PC3 = 88.29 % PC4 = 89.76 %