



Optimization Accuracy of Software Defect Prediction using Feature Selection and Machine Learning Techniques

¹Praveen Prakash Ranjan, ²Dr. Ankit Temurnikar

M. Tech. Scholar, Department of Computer Science and Engineering, Madhyanchal Professional University (MPU), Bhopal, India¹

Professor, Department of Computer Science and Engineering, Madhyanchal Professional University (MPU), Bhopal, India²

Abstract: - Software defect prediction is an important research area in software engineering that aims to identify defective software modules during the software development lifecycle. The presence of software defects can reduce system reliability, increase maintenance costs, and affect overall software quality. Traditional software testing approaches often require considerable time and resources, making machine learning techniques valuable for automated defect prediction. However, the presence of irrelevant and redundant features, high-dimensional datasets, and class imbalance can negatively affect prediction accuracy. To address these challenges, this study proposes an optimized software defect prediction approach using feature selection and machine learning techniques. The proposed methodology involves collecting historical software datasets, preprocessing the data, selecting relevant software metrics, and applying machine learning algorithms for defect classification. Feature selection techniques are utilized to identify the most significant attributes, reduce data dimensionality, and improve model efficiency. Various machine learning algorithms, including Decision Tree, Random Forest, Support Vector Machine, K-Nearest Neighbors, Logistic Regression, and ensemble learning techniques, can be investigated for predicting defective and non-defective software modules. The performance of the models is evaluated using accuracy, precision, recall, F1-score, sensitivity, and specificity. The study aims to investigate the effectiveness of feature selection in improving classification performance and reducing computational complexity. The proposed approach is expected to support early defect identification, optimize software testing activities, and improve software quality and reliability.

Keywords: Software Defect Prediction, Feature Selection, Machine Learning, Accuracy Optimization, Software Metrics, Classification, Software Quality

I. INTRODUCTION

Software development has become an essential part of modern technological advancement, supporting applications in healthcare, education, banking, communication, transportation, and industrial automation. With the increasing complexity of software systems, maintaining software quality, reliability, and performance has become a significant challenge. Software defects are errors or faults in software programs that may cause incorrect outputs, system failures, security vulnerabilities, and unexpected behavior. These defects can increase development costs, delay project completion, and reduce user satisfaction. Therefore, the early identification and prevention of software defects are important objectives in software



engineering. Software defect prediction (SDP) is an effective approach that uses historical software data and software metrics to identify potentially defective modules before software deployment [1, 2].

Traditional software testing techniques mainly depend on manual inspection, code reviews, and conventional testing procedures. Although these techniques are useful for identifying defects, they may require considerable time, effort, and computational resources, particularly for large and complex software projects. Moreover, testing every software component with equal priority may not be an efficient use of available resources. Software defect prediction addresses this challenge by estimating the likelihood of defects in individual software modules. By identifying defect-prone components at an early stage, developers can prioritize testing activities, allocate resources effectively, and reduce the possibility of software failures. Historical software repositories containing defect information and software metrics provide the foundation for developing predictive models [3].

Machine learning (ML) techniques have gained considerable attention in software defect prediction because of their ability to learn patterns from historical data and perform automated classification. Various algorithms, including Decision Tree (DT), Random Forest (RF), Support Vector Machine (SVM), K-Nearest Neighbors (KNN), Logistic Regression (LR), and Naïve Bayes (NB), have been investigated for identifying defective and non-defective software modules. These algorithms analyze software metrics such as lines of code, cyclomatic complexity, coupling, cohesion, and Halstead metrics to identify relationships between software characteristics and defect occurrence. Random Forest combines multiple decision trees to improve prediction stability, while SVM identifies decision boundaries between different classes. Similarly, ensemble learning techniques combine multiple predictive models to improve classification performance. However, the effectiveness of these algorithms depends on the quality of the dataset, the relevance of selected features, and the appropriate configuration of model parameters [4, 5].

Feature selection is an important preprocessing technique for improving the performance of machine learning-based software defect prediction systems. Software defect datasets may contain numerous features, including irrelevant, redundant, or highly correlated software metrics. The inclusion of unnecessary features can increase computational complexity, introduce noise, and negatively affect the predictive performance of machine learning algorithms. Feature selection addresses these issues by identifying the most informative attributes and removing less relevant features from the dataset. Common feature selection approaches include filter methods, wrapper methods, and embedded methods. Filter methods evaluate features using statistical measures, wrapper methods assess feature subsets through model performance, and embedded methods perform feature selection during model training. Techniques such as Information Gain, Chi-Square, Recursive Feature Elimination, and feature importance-based selection can be applied to identify relevant software metrics. Selecting appropriate features can simplify the prediction model, reduce training time, and potentially improve classification accuracy [6, 7].

Despite the development of several machine learning and feature selection approaches, software defect prediction continues to face important research challenges. One major issue is class imbalance, where the number of non-defective software modules is significantly higher than the number of defective modules. This imbalance may cause machine learning models to favor the majority class, resulting in poor identification of defective components. Another challenge is the presence of redundant features and differences in software project characteristics, which can affect model generalization. Furthermore, a model trained on one software project may not achieve similar performance when applied to another project. Therefore, integrating feature selection with machine learning techniques is an important research direction for developing efficient and reliable software defect prediction systems. Appropriate preprocessing, class imbalance handling, and hyperparameter optimization can further contribute to improving predictive performance [8, 9].

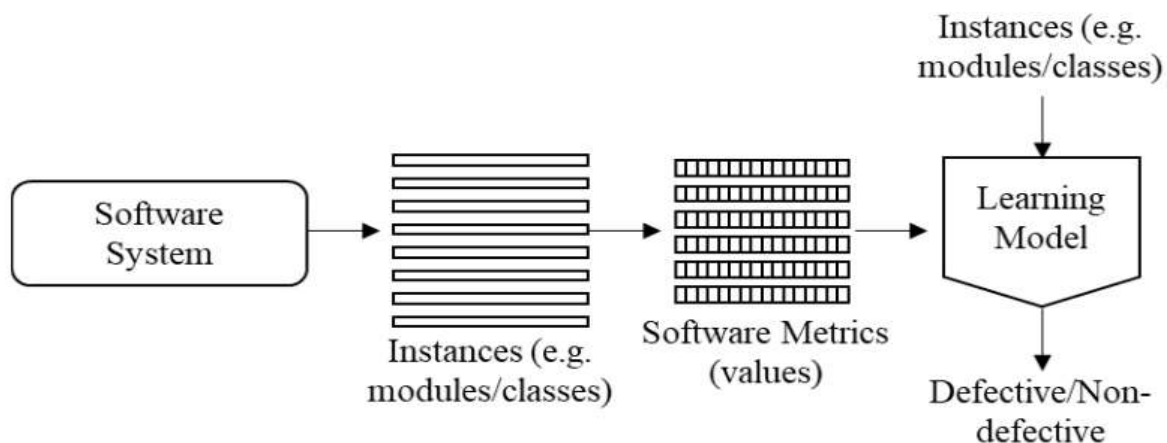


Figure 1: SD Technique

Figure 1 shows the basic design of the SDP method. To avoid programming errors, the first step is to organize a measurement data set using programming modules that contain characteristics such as module size and module complexity. These measurements are usually obtained from a variant control system (VCS). It is also important to assign the proper flag to the broken modules in case they are affected by code changes aimed at fixing a specific issue that falls under the category of deformation.

II. SOFTWARE QUALITY AND SD PREDICTION

2.1 Software Quality

Software is the study of how hardware, programs, and other data process data to extract information. Software is defined as a program organized into segments with source code. It is a collection of agents that manage conformance, traceability, complexity, testability, visibility, variability, and reliability [9]. Software is the foundation of today's modern technology. It is what constitutes or controls the goods and services, big and small, that people use every day for a variety of purposes. It is the result, the measurable attributes. According to Juran, quality is an attribute of a product that meets customer needs and



satisfies them. According to the American Society for Quality (ASQ), quality is the most important aspect of a product. It must meet quantified and derived requirements, and the software delivered to the user community must be free of errors. When evaluating the characteristics of software, complexity, consistency, code stripes, and number of function points are taken into account. To measure quality, a specific product delivery is required. When we design a product, the quality of the software should match the characteristics specified by the designer [10].

Waman believes that the success of a venture leads to customer loyalty. He believes that collecting customer opinions increases quality. Software quality assurance is achieved by creating test cases and using them as a control measure to ensure the desired quality. The list of events designed to evaluate a product and also help in software development and maintenance is called "software quality assurance" [11, 12].

2.2 Software Defect Prediction

Product lifecycle is a complex and critical process that enables the creation of error-free software and ensures that software is free of defects. This crucial step involves identifying and correcting defects even before the final product is delivered to the customer. Software memory contains a wealth of data that is essential to determine the quality of a product. These large data stores are essential to ensure error-free software standards and provide an optimal user experience to customers. By applying AI-based algorithms and data mining techniques to such documents, we aim to extract key information while correcting defects and improving the overall quality of the product. A product defect is an error, mistake, omission, flaw, or oversight in the programming or design of a computer system. This leads to inaccuracies or unexpected things and as a result, unintended behavior.

The impact of software bugs can be significant, affecting both the cost and the quality. Furthermore, identifying and correcting these errors can be very costly. To address this error pattern, monitoring teams should: Focus on minimizing errors and improving software quality. Additionally, some attention should be given to training and developing professional practices aimed at minimizing and correcting such errors. This important approach will bring significant benefits in terms of improved software performance and overall organizational success. Software error detection is a crucial step that helps to identify and correct faulty software modules. Ensuring high quality of software and minimizing minor errors results in a reliable and efficient product. Detecting and correcting errors at an early stage significantly reduces development costs and improves software performance. This strategy not only promotes cost efficiency, but also leads to the creation of more robust and sophisticated software, which benefits both developers and end users in the long run.

SDP can also be used as a factor in planning programs in both the intelligent world and in enterprises. Credits from previous iterations of a product are added to the static code, along with a log of the various transformations used to create models to replace damaged modules in the next delivery of the product. This makes it an area of such parts Programming that is likely to contain consequences is useful. This is useful when trying to deal with expenditures limited to the entire program structure that are too large to deal with completely. There are good indicators of adaptation, decision making to control product planning while focusing on testing products and their defective parts. SDP goes beyond the domains of business and science. One aspect of SDP is: ability to address software planning in both academic and business settings. Developers can create models to predict potential issues in subsequent versions by incorporating static code concepts that collect logs of various changes and extract

relevant information from previous software versions. This proactive methodology helps to identify and resolve imperfections in specific modules. It ultimately improves the overall quality of the product and contributes to the quality of the product.

In today's fast-paced digital environment, the importance of efficient cost and resource management in software development cannot be overemphasized. The scope of the entire software structure is often enormous, so careful planning and focus on tasks are essential to avoid burnout and succeed. Furthermore, the presence of potential defects further emphasizes the need for careful evaluation to identify flaws and defects in the product design and correct them appropriately. Focusing on a comprehensive and detailed testing approach, including careful consideration of weak areas, will ensure a solid and error-free end result.

III. PROPOSED METHODOLOGY

A system used to understand a concept and its environment using a simplified interpretation of the environment using a model is called a cognitive system. The steps taken to build a model are called inductive learning. A cognitive system can combine its experiences by creating new structures, or patterns. The models and patterns created by cognitive systems are called machine learning. A model is described as predictive because it can be used to predict the output of a function (objective function) for specific values in the function domain, whereas the information pattern is characterized by describing only the data part. Random forest is an ensemble method based on the bagging principle. It uses decision trees as base classifiers. To generate individual trees in random forest, Breiman followed these steps: If the number of records in the training set is N , then N records are selected randomly from the original data, but with replacement. This is the bootstrap sample.

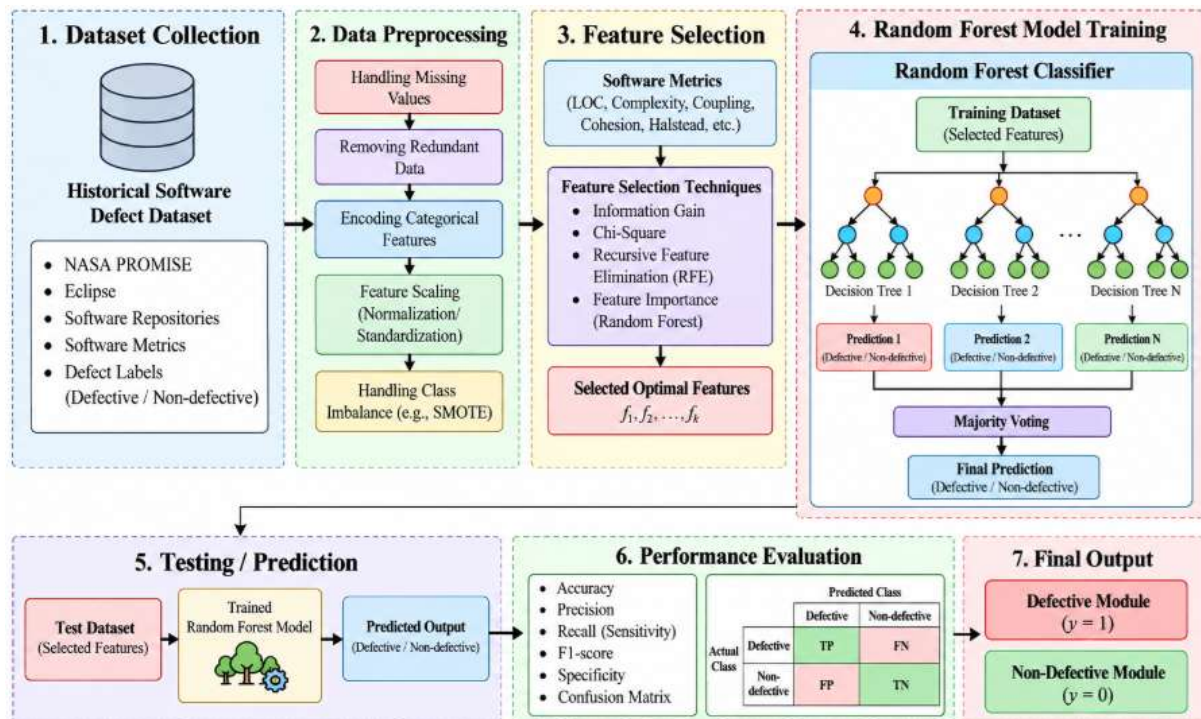


Figure 2: Flow of Methodology



Dataset Collection

The first stage involves collecting historical software defect datasets from publicly available software repositories such as NASA PROMISE, Eclipse, and other software defect prediction datasets. These datasets contain software metrics, including lines of code, cyclomatic complexity, coupling, cohesion, and Halstead metrics, along with defect labels. Each software module is represented by a set of numerical or categorical features. The target variable indicates whether the module is defective or non-defective. The collected dataset provides the foundation for training and evaluating the proposed prediction model.

Data Preprocessing

The collected dataset is preprocessed to improve data quality and ensure its suitability for machine learning algorithms. This stage involves identifying and handling missing values, removing duplicate records, detecting inconsistent data, and converting categorical attributes into numerical representations when required. Numerical features may be normalized or standardized to ensure that differences in feature scales do not negatively affect algorithms such as SVM and KNN. The dataset is also examined for class imbalance, which occurs when defective and non-defective software modules are represented unequally. Appropriate techniques, such as class weighting or SMOTE, may be applied to address this issue using only the training data.

Feature Selection

Feature selection is an important stage of the proposed methodology because software defect datasets may contain irrelevant and redundant attributes. The objective is to identify the most informative software metrics that contribute significantly to defect prediction. Different feature selection techniques, such as Information Gain, Chi-Square, Recursive Feature Elimination (RFE), and Random Forest feature importance, can be investigated. These methods evaluate the relevance of individual features or feature subsets and eliminate less informative attributes. The selected features are then used for model training. Feature selection can reduce dimensionality, decrease computational complexity, and potentially improve classification performance. The selected feature subset will be determined using the training data to prevent information leakage from the test dataset.

Machine Learning Model Development

After feature selection, the reduced dataset is used to develop machine learning models for software defect prediction. Various classification algorithms, including Decision Tree (DT), Random Forest (RF), Support Vector Machine (SVM), K-Nearest Neighbors (KNN), Logistic Regression (LR), and Naïve Bayes (NB), can be implemented and evaluated. Each algorithm learns the relationship between software metrics and defect labels from the training dataset. The trained models classify software modules into two categories: defective and non-defective. Hyperparameter optimization techniques, such as Grid Search or Random Search, may be applied to identify suitable model configurations. The models can also be evaluated with and without feature selection to determine its contribution to prediction performance.

Model Training and Testing

The preprocessed dataset is divided into training and testing subsets. The training data are used to develop the machine learning models, while the testing data are reserved for evaluating their performance on unseen software modules. A suitable split, such as 80% training and 20% testing, may be adopted depending on dataset size. Stratified splitting can help preserve the proportion of defective and non-defective modules in both subsets. Alternatively, stratified k-fold cross-validation can be used to obtain a more reliable estimate of model performance. All preprocessing operations, feature selection, and hyperparameter optimization are performed within the training or cross-validation process to avoid data leakage.

Performance Evaluation

The performance of the developed machine learning models is evaluated using standard classification metrics. Accuracy measures the proportion of correctly classified software modules, while precision indicates the proportion of predicted defective modules that are actually defective. Recall measures the ability of the model to identify actual defective modules, and the F1-score provides a balance between precision and recall. Sensitivity and specificity are also calculated to evaluate the model's ability to identify defective and non-defective modules, respectively.

IV. SIMULATION RESULTS

Other standard classification schemes considered in this test case were SVM, LR, and RF classifiers.

Parameters:

Accuracy measures how accurately the model predicts the actual benefit compared to the overall benefit predicted by the framework. This test shows the number of real benefits that the model achieves by capturing it as a clear certainty. The F-score can provide consistency between precision and studies and is preferred over precision when there is data imbalance.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

Where,

TP = True Positive,

TN = True Negative

FP = False Positive,

FN = False Negative

Data Information:

CM1 dataset contains total 11141 entries which are given below

```
[ ] from google.colab import files
    uploaded = files.upload()

Choose Files No file chosen Upload
current browser session. Please rerun this cell to
Saving pc1_csv.csv to pc1_csv.csv

[ ] data = pd.read_csv("pc1_csv.csv")

data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10922 entries, 0 to 10921
```

Figure 3; Data Information

data.complex(10)																			
	low	v(g)	w(g)	h(g)	m	v	3	4	5	6	7	8	9	10	10count	100count	1000count	10000count	100000count
3391	4.0	1.0	1.0	1.0	0.0	13.81	0.07	1.82	7.74	17.41					2	0	0	0	0
2881	27.0	2.0	1.0	3.0	81.0	298.52	0.09	10.94	27.37	3273.82				20	0	2	0	0	
9356	255.0	20.0	22.0	17.0	1994.0	7579.72	0.04	27.34	277.10	207113.99				156	82	32	0	0	
3786	8.0	1.0	1.0	1.0	10.0	31.70	0.00	2.50	12.68	79.25				5	0	0	0	0	
707	14.0	0.0	1.0	0.0	88.0	326.49	0.16	6.32	48.26	7307.68				12	0	0	0	0	
230	81.0	17.0	11.0	0.0	245.0	1453.05	0.03	21.71	86.82	15331.52				81	12	19	0	0	
3772	4.0	1.0	1.0	1.0	10.0	31.70	0.03	3.00	16.57	95.19				2	0	0	0	0	
7278	4.0	1.0	1.0	1.0	6.0	18.61	0.00	2.00	7.75	31.82				2	0	0	0	0	
2866	63.0	10.0	1.0	2.0	0.0	0.00	0.00	0.00	0.00	0.00				0	0	0	0	0	
10450	16.0	4.0	1.0	2.0	47.0	192.00	0.14	7.33	24.50	1320.82				16	0	0	0	0	

Figure 4: Dataset

```
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
```

Figure 5: Classification

The performance of three machine learning classifiers, namely Support Vector Machine (SVM), Logistic Regression (LR), and Random Forest (RF), is evaluated using precision, recall, and accuracy. The SVM classifier achieved a precision of 99.34%, recall of 96.98%, and accuracy of 96.90%, demonstrating its ability to identify defective and non-defective software modules with high precision. Logistic Regression achieved a precision of 99.56%, recall of 99.24%, and accuracy of 98.56%, indicating improved classification performance compared with SVM. Among the three classifiers, Random Forest achieved the highest performance, with 100% precision, 99.89% recall, and 99.86% accuracy. These results indicate that Random Forest correctly classified nearly all software modules and identified defective modules with a very high recall. The ensemble learning mechanism of Random Forest, which combines predictions from multiple decision trees, may contribute to its strong classification performance. Overall, the results show that Random Forest outperformed Logistic Regression and SVM across all three evaluation metrics for the reported experiment. However, further validation using independent test data and cross-validation is necessary to

confirm the model's generalization capability and ensure that the high performance is not caused by data leakage or overfitting.

Table I: Comparison Result

Classifier	Precision	Recall	Accuracy
SVM	99.34%	96.98%	96.90%
LR	99.56%	99.24%	98.56%
RF	100%	99.89%	99.86%

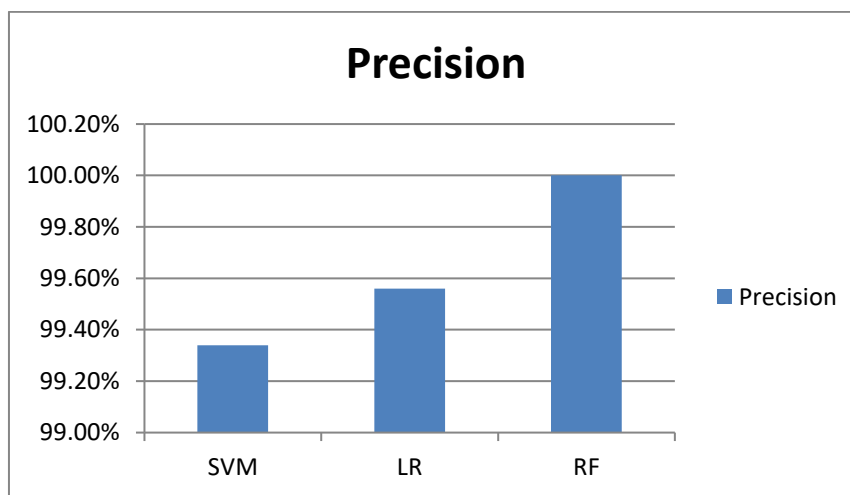


Figure 6: Graphical Represent of Precision

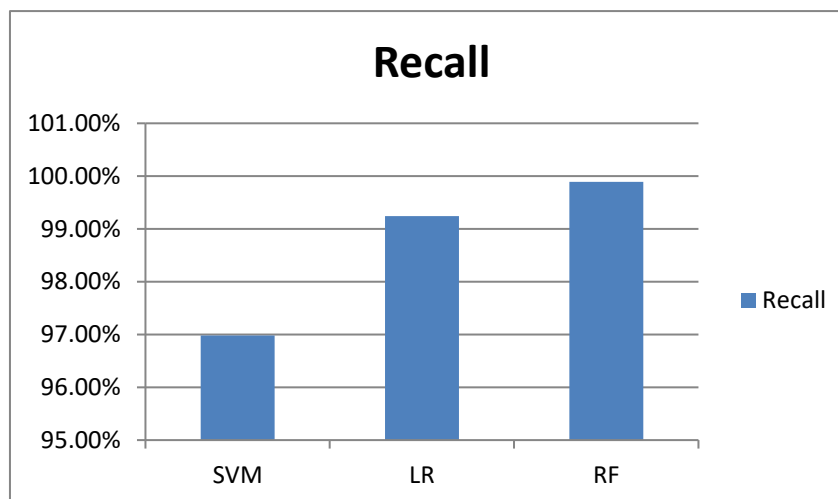


Figure 7: Graphical Represent of Recall

The above graph illustrates the recall performance of three machine learning classifiers, namely Support Vector Machine (SVM), Logistic Regression (LR), and Random Forest (RF), for software defect prediction. The SVM classifier achieved a recall of 96.98%, indicating

that it successfully identified most of the actual defective software modules. Logistic Regression demonstrated improved performance with a recall of 99.24%, showing its ability to detect a higher proportion of defective modules. Random Forest achieved the highest recall of 99.89%, indicating that it identified nearly all actual defective software modules with very few false negatives. The results show that Random Forest outperformed both SVM and Logistic Regression in terms of recall. High recall is particularly important in software defect prediction because it reduces the likelihood of overlooking defective software components. Therefore, Random Forest demonstrates strong potential for identifying defect-prone modules and supporting effective software testing and quality assurance.

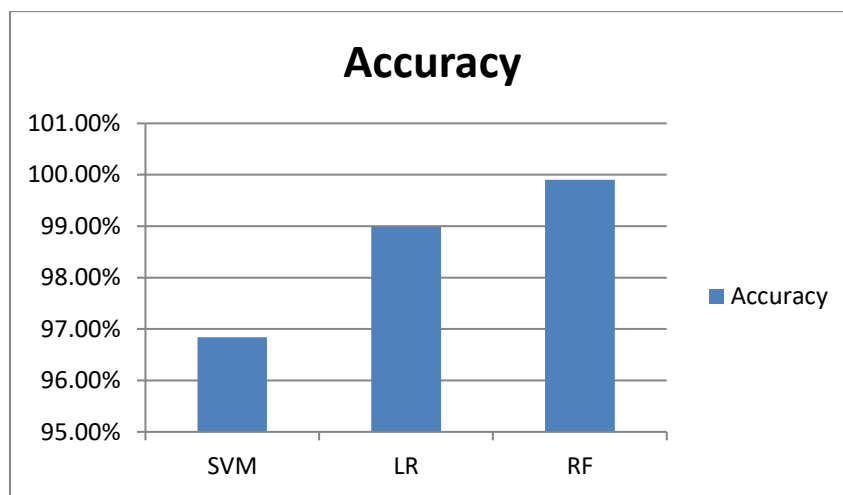


Figure 8: Graphical Represent of Accuracy

V. CONCLUSION

The study presents a machine learning-based approach for software defect prediction using feature selection and classification techniques. The performance of Support Vector Machine (SVM), Logistic Regression (LR), and Random Forest (RF) was evaluated using precision, recall, and accuracy. The experimental results demonstrate that SVM achieved 99.34% precision, 96.98% recall, and 96.90% accuracy, while Logistic Regression obtained 99.56% precision, 99.24% recall, and 98.56% accuracy. Random Forest achieved the highest performance, with 100% precision, 99.89% recall, and 99.86% accuracy. These results indicate that Random Forest effectively identifies defective and non-defective software modules in the evaluated dataset. The combination of multiple decision trees may contribute to its improved classification performance. The findings also highlight the importance of feature selection in reducing irrelevant attributes and supporting efficient software defect prediction. However, further evaluation using cross-validation and independent datasets is necessary to verify the reliability and generalization capability of the models. Future research can explore advanced feature selection methods, hybrid machine learning techniques, and optimization algorithms to improve prediction performance. Overall, the proposed approach



can support early defect identification, prioritize software testing activities, reduce maintenance efforts, and improve software quality and reliability.

REFERENCES

- [1] Z. Zhou, Y. Zhu, Q. Yu and J. Li, "Software Defect Prediction Method Based on Multi-Feature Fusion," *2025 25th International Conference on Software Quality, Reliability and Security (QRS)*, Hangzhou, China, 2025, pp. 529-539.
- [2] Siddika, Ayesha, Momotaz Begum, Fahmid Al Farid, Jia Uddin, and Hezerul Abdul Karim. 2025. "Enhancing Software Defect Prediction Using Ensemble Techniques and Diverse Machine Learning Paradigms" *Eng* 6, no. 7: 161.
- [3] S. Setia, K. K. Ravulakollu, K. Verma, S. Garg, S. K. Mishra and B. Sharan, "Software Defect Prediction using Machine Learning," *2024 11th International Conference on Computing for Sustainable Global Development (INDIACom)*, New Delhi, India, 2024.
- [4] Albattah, Waleed, and Musaad Alzahrani. 2024. "Software Defect Prediction Based on Machine Learning and Deep Learning Techniques: An Empirical Approach" *AI* 5, no. 4: 1743-1758.
- [5] Alsghaier, H. & Akour, M. Software fault prediction using particle swarm algorithm with genetic algorithm and support vector machine classifier. *Sofw. Pract. Exp.* 50(4), 407–427, 2020.
- [6] Yu, X., Bennin, K. E., Liu, J., Keung, J. W., Yin, X., & Xu, Z. An empirical study of learning to rank techniques for effort-aware defect prediction. In *SANER 2019 - Proceedings of the IEEE 26th International Conference on Software Analysis, Evolution, and Reengineering*, pp. 298–309, 2019.
- [7] Bach, T., Andrzejak, A., Pannemans, R. & Lo, D. The impact of coverage on bug density in a large industrial software project. *ACM/ IEEE Int. Symp. Empirical Sofw. Eng. Meas. (ESEM)*, 307–313, 2017.
- [8] Mohammed, R., Rawashdeh, J., & Abdullah, M. Machine Learning with Oversampling and Undersampling Techniques: Overview Study and Experimental Results. In *11th International Conference on Information and Communication Systems (ICICS)*, pp. 243–248, 2020.
- [9] Aljamaan, H.; Alazba, A. Software defect prediction using tree-based ensembles. In *Proceedings of the 16th ACM International Conference on Predictive Models and Data Analytics in Software Engineering, Virtual USA*, 8–9 November, pp. 1–10, 2020.
- [10] Li, R.; Zhou, L.; Zhang, S.; Liu, H.; Huang, X.; Sun, Z. Software Defect Prediction Based on Ensemble Learning. In *Proceedings of the 2nd International Conference on Data Science and Information Technology*, Seoul, Republic of Korea, 19–21, pp. 1–6, 2019.
- [11] Alsaeedi, A.; Khan, M.Z. Software Defect Prediction Using Supervised Machine Learning and Ensemble Techniques: A Comparative Study *J. Softw. Eng. Appl.* 12, 85–100, 2019.
- [12] Tua, F.M.; Sunindyo, W.D. Software Defect Prediction Using Software Metrics with Naïve Bayes and Rule Mining Association Methods. In *Proceedings of the 5th*



International Conference on Science and Technology (ICST), Yogyakarta, Indonesia, pp. 1–5, 2019.

- [13] Manjula, C.; Florence, L. A Deep neural network based hybrid approach for software defect prediction using software metrics. *Clust. Comput.*, 22, 9847–9863, 2019.
- [14] Manjula, C.; Florence, L. Hybrid approach for software defect prediction using machine learning with optimization technique. *Int. J. Comput. Inf. Eng.*, 12, 28–32, 2018.
- [15] Paramshetti, P.; Phalk, D.A. Software defect prediction for quality improvement using hybrid approach. *Int. J. Appl. Innov. Eng. Manag.*, 4, 99–104, 2015.