Software Defect Prediction Using Machine Learning Algorithms

[1]Mr Y Suresh, [2]Pikki Lovaraju, [3]T Kishore Kumar, [4]V Venkata Gopi, [5]T Rama Kotaiah, [6]T Lalas Maruthi

[1]Associate Professor, [2,3,4,5,6]BTech (IT) Students
Information Technology (IT),
Vasireddy Venkatadri Institute of Technology
Guntur, Andhra Pradesh.

***Abstract*- Software Defect Prediction [SDP] plays an important role in the active research areas of software engineering. A software defect is an error, bug, flaw, fault, malfunction or mistake in software that causes it to create a wrong or unexpected outcome. The major risk factors related with a software defect which is not detected during the early phase of software development are time, quality, cost, effort and wastage of resources. Defects may occur in any phase of software development. Booming software companies focus concentration on software quality, particularly during the early phase of the software development. Thus, the key objective of any organization is to determine and correct the defects in an early phase of Software Development Life Cycle at testing phase by using machine learning algorithms and JM1 dataset. To improve the quality of software, machine learning techniques have been applied to build predictions regarding the failure of software components by exploiting past data of software components and their defects. This project reviewed the state of art in the field of software defect management and prediction, and offered machine learning techniques.**

***Key words*: Software Defect Prediction, JM1 Dataset, Machine Learning, Random Forest, Naive Bayes, Decision Tree, Support Vector Machine (SVM), Accuracy, etc.**

I. INTRODUCTION

In today's rapidly evolving software development landscape, ensuring the quality and reliability of software systems is of paramount importance. Software defects, also known as bugs or issues, can lead to significant consequences ranging from system malfunctions to security vulnerabilities, resulting in financial losses and damage to reputation. As such, the proactive identification and mitigation of software defects have become critical objectives for software engineering teams. Traditional approaches to defect detection often rely on manual code reviews, testing, and debugging processes, which can be time-consuming, resource-intensive, and prone to human error. In recent years, however, advancements in machine learning (ML) and data mining techniques have opened up new avenues for more efficient and accurate defect prediction.

The project "Software Defect Prediction Using Machine Learning Algorithms" aims to leverage the power of ML algorithms to predict and prevent software defects early in the development lifecycle. By analyzing historical data on software defects, along with various software metrics and features extracted from code repositories, ML models can be trained to identify patterns and indicators that correlate with the likelihood of defects. This project explores the application of a variety of ML algorithms, including but not limited to decision trees, random forests, support vector machines, neural networks, and ensemble methods, to predict software defects. By comparing and evaluating the performance of these algorithms on different datasets and scenarios, we seek to identify the most effective approaches for defect prediction in diverse software projects.

II. CONCEPTS AND OVERVIEW OF SOFTWARE DEFECTS

A. Concept of software defects

Since analysts often can't distinguish between software defects and programming faults, errors, and failure, this article utilizes IEEE 729-1983 (Standard Glossary of Software Engineering Terminology) to characterize defects as, From the inside of the product, the defects are mistakes and errors in the maintenance or development of the product item. From an external perspective, a defect is the violation or failure of the framework/system to accomplish specific capacities. The description of the concepts that are easily mistaken with defects is as follows

1. **Fault**: The software doesn't perform according to the client's expectations and runs in an unsuitable internal state. We can view it as a defect that can prompt software errors and is regarded as dynamic behavior.

2. **Failure**: It refers to the outputs that the software generates at runtime, which the client doesn't accept. For instance, if the execution capacity is lost, and the client's capabilities are not met, the framework can't meet the fixed asset's execution necessities.
3. **Error**: It is introduced by individuals and changed over into faults under specific conditions. It exists in the whole software life cycle, including error information in the software design, data structure, code, requirements analysis, and other carriers.

B. Main research directions of software defects

1. **Software defect management**: Defect management mainly refers to the collection, statistics, and useful recording of defects. To improve management productivity, engineers have designed many robotized defects management devices. At present, the industry's commonly used tools are mainly JIRA dispatched by Atlassian and Bugzilla, an open-source bug tracking framework provided by Mozilla. These two tools record the transactions, attributes, statistical information of defects, and don't have a more profound investigation and explicit grouping of defects. Defect analysis and classification are significant segments of defect management. Therefore, examination and arrangement of deformities require further exploration of the data recorded in JIRA and Bugzilla.
2. **Software defect analysis**: Software designers regularly use defect analysis to better access programming quality and development quality. Software defect analysis is a strategy for characterizing imperfections and mining the reasons for defects. The motivation behind software defect analysis is to enable analysts to find, locate, evaluate, and improve test efficiency. The defects analysis methods are mostly partitioned into qualitative analysis, quantitative analysis, and attribute analysis. Qualitative analysis strategies mostly incorporate Root Cause Analysis (RCA) and Software Fault Tree Analysis (SFTA). Attribute analysis is commonly partitioned into single attribute analysis and multi-attribute analysis.
3. **Software defect classification**: Software defects are different and complex. A more grouping and conglomeration of deficiencies can assist programmers with assessing programming quality, improve analyzers' work productivity, and decrease the trouble of analysis. Classification is likewise useful to suggest repair techniques and reuse test cases. It can comprehend the distribution of defects as per the classification and analysis results, prevents frequently occurring software defects, extraordinarily improve the software development cycle, and in this way, improve the quality of software.

III. LITERATURE SURVEY

"Software Defect Prediction Using Machine Learning Algorithms" is a critical area of research within the domain of software engineering, aiming to enhance software quality and reliability by identifying potential defects in software systems early in the development lifecycle. A plethora of studies have delved into this topic, employing various machine learning techniques to construct predictive models capable of discerning patterns indicative of software defects. One prominent approach involves the utilization of historical software metrics, such as code complexity, lines of code, and code churn, as features for predictive modeling. Numerous machine learning algorithms have been explored in this context, including but not limited to Decision Trees, Random Forests, Support Vector Machines, Neural Networks, and Ensemble Methods. These algorithms are adept at handling diverse data types and exhibit varying degrees of effectiveness in defect prediction tasks. Additionally, research efforts have focused on feature selection and extraction methodologies to enhance model performance and interpretability.

Moreover, the incorporation of domain-specific knowledge and contextual information has been instrumental in refining defect prediction models, making them more tailored to the characteristics of specific software projects. Despite significant advancements, challenges persist in achieving high prediction accuracy and generalizability across different software projects and environments. Future research directions may involve the exploration of advanced machine learning techniques, such as deep learning, and the integration of novel data sources, such as code repositories and issue tracking systems, to further improve the efficacy of defect prediction models and facilitate their practical application in real-world software development scenarios.

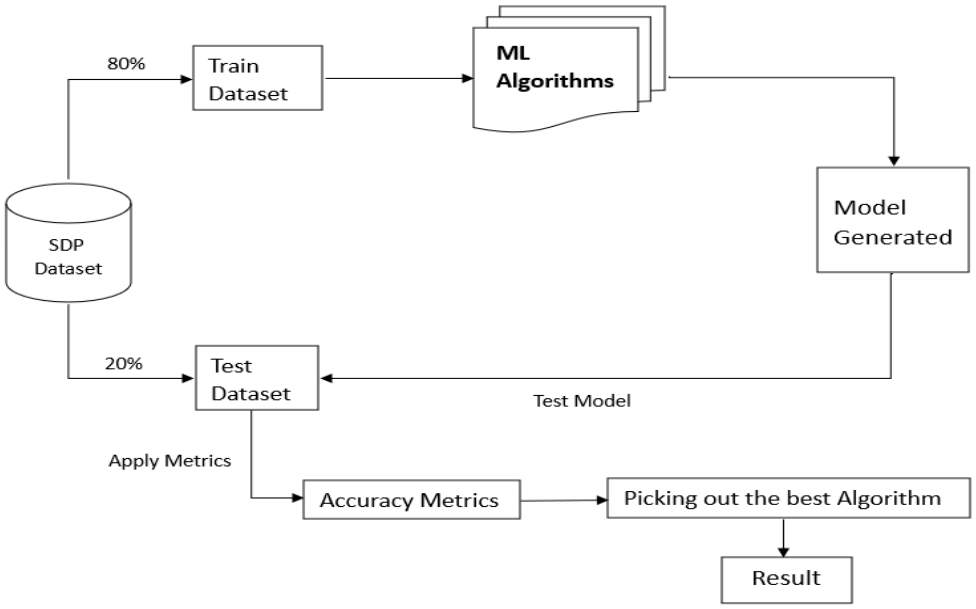
IV. PROPOSED SYSTEM

The proposed system for "Software Defect Prediction Using Machine Learning Algorithms" aims to develop a robust framework capable of predicting defects in software projects. The system will entail several key components: data collection and preprocessing, feature selection, model training, evaluation, and deployment. Initially, historical data on software defects will be gathered from various repositories. Preprocessing techniques will be employed to clean and transform the data into a suitable format for analysis. Feature selection methodologies will then be applied to identify the most relevant variables affecting software defects. Machine learning algorithms, including but not limited to logistic regression, decision trees, random forests, and support vector machines, will be trained on the selected features to predict the likelihood of defects in future software releases. The system's performance will be evaluated using metrics such as accuracy, precision, recall, and F1-score. Finally, the trained model will be deployed into the software development

pipeline to provide real-time defect prediction and aid developers in proactively addressing potential issues. Regular updates and maintenance will ensure the system's effectiveness and adaptability to evolving software landscapes.

V. SYSTEM ARCHITECTURE

A System Architecture is the conceptual model that defines the structure, behavior,and more views of a system. An architecture description is a formal description and representation of a system, organized in a way that supports reasoning about the structuresand behaviors of the system. A System Architecture can consist of system components and the sub-systems developed, that will work together to implement the overall system.



- **Data Collection** The first step involves collecting a dataset which will be used to train the model. The data is split into two parts: the training dataset (80%) and the testing dataset (20%).
- **Training the Model** The training dataset is used to train the machine learning model. The model learns by identifying patterns in the data.
- **Testing** The testing dataset is used to assess the performance of the trained model. This is done by applying the model to the testing data and evaluating how well it performs.
- **Evaluation** Metrics are used to measure the accuracy of the model's predictions.
- **Model Selection** Based on the evaluation results, you can choose the best performing model.

VI. DESIGN METHODOLOGY

Design Methodology of this project includes various processes which can be analyzing the raw data or JM1 dataset into meaningful data by the following steps

1.Problem Definition and Scope Identification: Defining the objective of predicting software defects using machine learning algorithms is crucial. This involves specifying the types of defects targeted (e.g., bugs, vulnerabilities) and determining the scope of the project, including the software systems to be analyzed.

2.Data Collection and Preparation: There are several datasets available for software defect detection. This research used a dataset, namely JM1. The dataset is obtained from the NASA promise dataset repository. **TABLE 1** describes the dataset in detail.

TABLE 1. CHARACTERISTICS OF THE DATASETS

Dataset	No. of attributes	No. of instances	% of faulty instances
JM1	22	10,885	19.35

We performed an in-depth analysis of dataset to understand every feature's effect on the output prediction. It was found that many features had a high correlation with others, and this led to a little overfitting. Hence, these features were not considered at the time of final training.

4.Model Selection: Identify suitable machine learning algorithms for defect prediction tasks. Evaluate various models including Decision Trees, Random Forest, Support Vector Machines (SVM), Naive Bayes, and Logistic Regression. Consider employing ensemble methods to potentially enhance predictive performance.

5.Model Training and Evaluation: Split the dataset into training, validation, and testing sets. Train the selected machine learning models using the training set and fine-tune hyperparameters through techniques like cross-validation. Evaluate model performance using metrics such as accuracy, precision, recall, F1-score, and ROC-AUC on the validation set. Validate models on the testing set to assess generalization performance accurately.

VII. RESULTS AND DISCUSSION

A. Experimental Setup

The ML model was implemented using the Python programming language. Python is commonly used in predictive analytics and data science projects involving both qualitative and quantitative data. The Python packages pandas, numpy, seaborn, matplotlib, sklearn and pyswarm were used to build the ML predictor.

B. Experimental Results

The results of the experiments on the JM1 dataset using various ML techniques are reported in the form of table. These offer a comparison of several techniques. We will assess all findings by comparing all strategies with and without optimization to make the notion obvious and understood. TABLE 2 show the evaluation metrics for all ML approaches studied without and with optimization, respectively. All analyzed models have the best results but SVM and optimized SVM have better performance in both cases than others. Here, the SVM model accurately predicted output class 1 as 1 so precision is 100%, while one output class 0 is predicted as 1, which is why recall is lesser. Moreover, NB does not work well as it works well on the high-dimensional dataset. RF and ensemble have almost the same result because Rf is also an ensemble approach that combines the large collection of trees. The RF model achieves a precision of 100% and an accuracy of 98.7%. This model also predicts 100% accurately the output class 1 and misses two values in the 0 class.

OUTPUT

```
file_path = "/content/gh.txt"
output = RunModels(file_path)

Decision Tree Algorithm
Defects  False
ACC:  0.919652570499064

Naive Bayes Algorithm
Defects  False
ACC:  0.9098585877675985

Random Forest Algorithm
Defects  False
ACC:  0.9436705996776653

SVM Algorithm
Defects  False
ACC:  0.8371679932849946
```

TABLE 2. EVALUATION METRIC OF CLASSIFIERS

Dataset	Evaluation Measures	SVM	Naive Bayes	Random Forest	Decision Tree
JM1	Accuracy	99.80	93.80	99.50	97.60
	Precision	99.70	100	100	99
	Recall	100	92.90	99.50	97.50
	F-measure	96	67.30	91.10	84.80

The outcomes of all ML models, as well as the ensemble approach, without optimization. The precision of all algorithms is the greatest among the measures, as seen in the graph. This is because all models have predicted the one class as 1. The NB model does not perform well in the case of recall and f-measure because it works on a large quantity of data or a high-dimensional dataset. The RF model and ensemble model have almost the same results. However, the SVM model outclasses in all evaluating metrics.

VIII. CONCLUSION

In this paper, ML techniques are utilized with feature selection and K-means clustering techniques for software defect prediction. We examined various well-known ML techniques and optimized ML techniques on a freely available dataset to improve the accuracy of the dataset in comparison with previous research. The results are also analyzed by utilizing ML with the PSO method and ensemble approach. All ML models are trained and tested through Python programming language using Google Co Lab. The analysis aimed to improve the accuracy performance of Machine Learning Algorithms on the JM1 dataset.

REFERENCES:

1. 2005, [online] Available: http://promise.site.uottawa.ca/SERepository/datasets/kc1-class-level-numericdefect.arff.

2. M. Alenezi, S. Banitaan and Q. Obeidat, "Fault-proneness of open source systems: An empirical analysis", *Synapse*, vol. 1, pp. 256, 2014.
3. Ahmed Iqbal et al., "Performance Analysis of Machine Learning Techniques on Software Defect Prediction using NASA Datasets", *Int. J. Adv. Comput. Sci. Appl*, vol. 10, no. 5, 2019.
4. Malkit Singh and Dalwinder Singh Salaria, "Software defect prediction tool based on neural network", *International Journal of Computer Applications*, vol. 70, no. 22, 2013.
5. M. Ahmad, S. Aftab and S. S. Muhammad, "Machine Learning Techniques for Sentiment Analysis: A Review", *Int. J. Multidiscip. Sci. Eng.*, vol. 8, no. 3, pp. 3, 2017.
6. I. A and A. Saha, "Software Defect Prediction: A Comparison Between Artificial Neural Network and Support Vector Machine", *Adv. Comput. Commun. Technol*, pp. 51-61, 2017.
7. Abdullah Alsaeedi and Mohammad Zubair Khan, "Software Defect Prediction Using Supervised Machine Learning and Ensemble Techniques: A Comparative Study", *Journal of Software Engineering and Applications*, vol. 12, no. 5, pp. 85-100, 2019.
8. Shuo Wang and Xin Yao, "Using class imbalance learning for software defect prediction", *IEEE Transactions on Reliability*, vol. 62, no. 2, pp. 434-443, 2013.
9. Divya Tomar and Sonali Agarwal, "Prediction of defective software modules using class imbalance learning", *Applied Computational Intelligence and Soft Computing*, vol. 2016, 2016.
10. Qinbao Song, Guo Yuchen and Martin Shepperd, "A comprehensive investigation of the role of imbalanced learning for software defect prediction", *IEEE Transactions on Software Engineering*, vol. 45, no. 12, pp. 1253-1269, 2018.
11. Lipika Goel et al., "Cross-project defect prediction using data sampling for class imbalance learning: an empirical study", *International Journal of Parallel Emergent and Distributed Systems*, pp. 1-14, 2019.