International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 7s
DOI: https://doi.org/10.17762/ijritcc.v11i7s.6980
***Article Received**: 30 March 2023 **Revised**: 14 May 2023 **Accepted**: 28 May 2023*

Study on Software Defect Prediction Based on SVM and Decision Tree Algorithm

Gauri Rao[1], PG Scholar Disha Gundo Pujari[2], Assistant Professor Rohini G. Khalkar[3], Assistant Professor Vidya Atish Medhe[4]

[1]Associate Professor, Department Of Computer Engineering,
Bharati Vidyapeeth (Deemed To Be University) College Of Engineering, Pune, India
grrao@bvucoep.edu.in
[2]Department Of Computer Engineering,
Bharati Vidyapeeth (Deemed To Be University) College Of Engineering, Pune, India
dgpujaripg21-comp@bvucoep.edu.in
[3]Department Of Computer Engineering,
Bharati Vidyapeeth (Deemed To Be University)College Of Engineering, Pune, India
rgkhalkar@bvucoep.edu.in
[4]Department Of Computer Engineering,
Bharati Vidyapeeth (Deemed To Be University) College Of Engineering, Pune, India
vgavhad@bvucoep.edu.in

Abstract— Software Defect Prediction is a process of identifying the potential defects in software systems before they occur. In this approach, the dataset containing information about software attributes is used as input, and the output is the prediction of whether the software is defective or not. The input dataset is generally in the form of a CSV file, which contains various software attributes such as cyclomatic complexity, essential complexity, Design Complexity, etc. The output of the defect prediction is binary classification. It is done by using SVM (Support Vector Machine) and a decision tree algorithm. This approach can help software developers identify their systems' defects before they cause any harm or affect the system's performance.

Keywords- SVM (Support Vector Machine), Machine Learning, Software Defect Prediction, Software testing, Software Engineering, Decision Tree, Classification, Feature Selection, and Machine Learning.

I. INTRODUCTION

Defect Prediction is a Valuable Tool for Software Development that helps to reduce the cost of development, improve the quality of the software, and enhanced the overall development process. Software Defect is also known as a Software bug or an error. Defects can occur in any phase of the software development lifecycle. Software Defect Prediction Identify the bugs or errors and build better software.

There are several classifiers used in software defect prediction. These classifiers help to predict the bug. Table 1 represents the detailed idea of the previously used classifiers in software defect prediction.

TABLE 1: CLASSIFIERS USED IN DEFECT PREDICTION

CLASSIFIER	DESCRIPTION
KNN	KNN classifies a new software component in training data using K-Nearest Neighbor
Naïve Bayes	Bayes' theorem is based on Bayes' Theorem and assumes that the features are conditionally independent given the class variable.
Random Forest	In the random forest, each tree is trained on a random portion of the data, and the final prediction is formed by aggregating all the trees' predictions.
Logistic Regression	Logistic Regression is employed in the Modelling of the link between a binary outcome and a set of predictor variables.

This survey paper aims to review the literature on the use of a Support Vector Machine (SVM) and decision tree (DT) for software defect prediction using input as a CSV dataset.

SVM is a popular machine learning algorithm that is used for classification tasks. SVM has been used for software defect prediction, and studies have shown that it can accurately identify defective software.

DT (Decision Tree) is another popular machine learning algorithm that is commonly used for classification and regression tasks. It works by recursively splitting the data into subsets based on the value of a selected attribute.

CSV is a commonly used data format for storing data in a tabular form. It is easy to work with and can be easily imported

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 7s
DOI: https://doi.org/10.17762/ijritcc.v11i7s.6980
***Article Received**: 30 March 2023 **Revised**: 14 May 2023 **Accepted**: 28 May 2023*

into machine learning algorithms for analysis. In software defect prediction, CSV datasets are often used to represent attributes of software such as Volume, program length, efforts, Decision complexity, essential complexity, and time estimator.

II. FEATURES USED IN THIS PROPOSED SYSTEM

1. *Data Collection*: The first step is to collect data related to the software project, such as code complexity, code size, etc. This data is used as input to the SVM and Decision tree algorithm.
2. *Data Pre-processing*: the collected data is pre-processed to remove the noise, inconsistencies, and irrelevant information. This step is crucial in ensuring that the data is accurate and reliable.
3. *Feature Selection*: The next step is to select the most relevant features that can help in predicting defects accurately. This step involves identifying the most significant features that have the most impact on software quality.
4. *Model Development*: In this step, SVM and Decision tree models are developed using the pre-processed data and selected features. That can help in predicting defects.
5. *Model Deployment*: Finally, the models are deployed in the software development process to predict defects.

III. RELATED WORK

There are different approaches to predicting software bugs and evaluating their effectiveness. In this system the author analyses several techniques, including simple heuristics, machine learning models, and statistical models, using a dataset of historical bug reports. They measure the accuracy of the different techniques in predicting future bugs and find that machine-learning models output perform simple heuristics and statistical models. The authors also conclude that ensemble models, which combine multiple models, tend to be more accurate than individual models.[1]

The authors explain that BBNs (Bayesian Belief Networks) can capture the relationships between different software metrics and their impact on defect-proneness. They use a dataset of metrics collected from a large software system to build a BBN, which is then used to predict defects in new software releases. The paper concludes that the BBN approach can effectively predict defects and can be used as a decision-support tool for software development teams.[2]

In this proposed system Author highlights the importance of software defect prediction and the potential benefits it can bring to software development. It also discusses the challenges associated with software defect prediction and the limitations of current approaches. The authors identify several factors that can influence the effectiveness of software defect prediction models, including the type of data used, the size of the data set, and the choice of machine learning algorithms. They also provide recommendations for future research in the area of software defect prediction.[3]

The authors use data from two large software projects and evaluate the accuracy of different machine-learning models in predicting the presence of bugs in the code. The authors compare the performance of several machine learning models, including decision trees, Bayesian networks, and support vector machines. They find that ensemble models, which combine multiple machine learning algorithms, generally perform better than individual models. They also identify the importance of feature selection and the need for careful pre-processing of data.[4]

the study found that bugs with higher severity and priority took less time to fix than those with lower severity and priority. The authors suggest that this may be due to the increased attention and resources given to higher severity and priority bugs. The study also found that the number of comments and attachments in a bug report had a positive correlation with the time taken to fix the bug. Overall, it highlights the importance of carefully selecting and using bug report fields to prioritize and manage software bugs. The findings can be useful for software developers and project managers looking to improve bug-fixing processes in their projects.[5]

The authors conducted experiments using eight different software datasets and found that ensemble methods generally outperformed single classifiers in terms of accuracy, precision, recall, and F-measure. Specifically, they found that the Bagging and Boosting ensembles consistently performed well across all datasets. The study highlights the potential benefits of using ensemble methods for software defect prediction and suggests that they can be valuable tools for software developers and quality assurance professionals. [6]

In this system, bugs found through code review had the shortest mean fixing time, followed by those found through testing and user reports. Bugs found through static analysis had the longest mean fixing time. The study also found that the severity of the bug had a significant impact on the time taken to fix it, with higher-severity bugs taking longer to fix. The findings of the study can be useful for software developers and project managers in selecting and prioritizing bug-finding techniques based on their potential impact on bug-fixing time.[7]

The authors found that each tool had strengths and weaknesses in detecting different types of bugs and that no single tool was able to detect all the bugs in the test programs. They also found that some tools produced a large number of

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 7s
DOI: https://doi.org/10.17762/ijritcc.v11i7s.6980
***Article Received**: 30 March 2023 **Revised**: 14 May 2023 **Accepted**: 28 May 2023*

false positives, which can be time-consuming for developers to investigate and fix. The study highlights the importance of carefully selecting and using static analysis tools based on the specific needs and characteristics of a software project. The findings can be useful for software developers and quality assurance professionals in selecting and using static analysis tools to improve the quality of their software.[8]

The System analyze data from four large software projects and compared the performance of nine different metrics in predicting the number of faults in the software. The authors found that the metrics that capture complexity and size were generally better predictors of fault-proneness than those that capture cohesion or inheritance. They also found that a combination of several metrics performed better than individual metrics alone in predicting fault-proneness. The study highlights the potential benefits of using object-oriented metrics to predict fault-proneness in software and suggests that they can be a valuable tool for software developers and quality assurance professionals.[9]

The authors found that their proposed method, which uses a genetic algorithm to select the best combination of classifiers, outperformed other ensemble selection methods and individual classifiers in predicting software defects. They also found that the inclusion of a diversity measure in the ensemble selection process further improved the accuracy of the predictions. The study highlights the potential benefits of using classifier ensembles for software defect prediction and suggests that the proposed method can be a valuable tool for software developers and quality assurance professionals.[10]

There are six different tools compared to their ability to detect a set of known bugs in a set of Java programs. The authors found that each tool had different strengths and weaknesses in detecting different types of bugs and that no single tool was able to detect all the bugs in the test programs. They also found that some tools produced a large number of false positives, which can be time-consuming for developers to investigate and fix. The study highlights the importance of carefully selecting and using static analysis tools based on the specific needs and characteristics of a software project. The findings can be useful for software developers and quality assurance professionals in selecting and using static analysis tools to improve the quality of their Java software.[11]

It highlights the potential benefits of Gas (Genetic Algorithms) in Software Engineering, such as the ability to explore large search spaces, handle non-linear constraints, and evolve solutions over time. However, they also note some limitations, such as difficulty choosing appropriate fitness functions and the potential for premature convergence. It provides a useful resource for researchers and practitioners interested in applying GAs in SE, as it summarizes and categorizes different approaches and highlights areas for future research.[12]

Evolutionary algorithms (EAs) improve the quality of software by automatically generating test cases that maximize code coverage and detect defects. As are a type of optimization algorithm inspired by natural selection and genetic inheritance. They work by iteratively generating candidate solutions and selecting the best ones based on a fitness function. The fittest solutions are then used to generate new candidate solutions, creating a cycle of continuous improvement. The authors argue that EAs can be applied to software testing because generating effective test cases is a challenging optimization problem. They present several case studies where EAs were used to improve code coverage and detect defects in real-world software systems, including an open-source web server and a video game. The results of the case studies show that EAs can significantly improve software quality by generating test cases that achieve higher code coverage and detect more defects than manually crafted test cases. The authors conclude that EAs have the potential to revolutionize software testing and improve the reliability and security of software systems.[13]

The authors conducted an empirical study on three large software systems and found that object-oriented metrics were more effective in predicting faults than procedure-oriented metrics. They suggest that this may be because object-oriented metrics better capture the complexity and interdependencies of modern software systems. In their conclusion, the authors state that "our study indicates that OO metrics are more effective than procedural metrics in predicting fault-proneness in software systems" (El Emam & Benlarbi, 2001, p. 249). This suggests that software developers and testers should consider using object-oriented metrics when evaluating the quality of software systems.[14]

The authors analyze over 9,000 bug reports and found that most bugs were caused by simple mistakes, such as typos or syntax errors. They also found that most bugs were fixed relatively quickly, within a few days of being reported. In addition, the authors analyze the factors that influenced the likelihood of a bug being fixed and found that factors such as the severity of the bug and the number of people assigned to fix it were important predictors. Overall, it provides valuable insights into the nature of software bugs and the factors that influence their likelihood of being fixed, which can help software developers and testers to better understand and manage the bug-fixing process.[15]

The authors developed a system called Bug Net, which continuously mines bug reports from different sources and applies natural language processing techniques to identify and

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 7s
DOI: https://doi.org/10.17762/ijritcc.v11i7s.6980
***Article Received**: 30 March 2023 **Revised**: 14 May 2023 **Accepted**: 28 May 2023*

extract relevant information about the bugs, such as their severity, category, and the affected components. The authors evaluated Bug Net on several open-source software projects and found that it was able to accurately identify common bugs and their characteristics, such as the most frequently occurring types of bugs and the components that were most frequently affected. Overall, it presents an innovative approach for mining bug reports from multiple sources and provides valuable insights into the nature of software bugs and their characteristics, which can help software developers and testers to better understand and manage the bug-fixing process.[16]

The authors trained ANNs on various software metrics, such as lines of code, McCabe complexity, and Halstead complexity, and compared their performance in predicting fault-proneness with other established techniques, such as logistic regression and decision trees. The authors found that ANNs outperformed the other techniques in terms of accuracy and generalization, suggesting that ANNs are an effective method for predicting fault-proneness in software systems. In their conclusion, the authors state that "our results suggest that ANNs are more effective in predicting fault-proneness than traditional statistical techniques" This suggests that ANNs may be a valuable tool for software developers and testers when evaluating the quality of software systems.[17]

IV. METHODOLOGY

The features of the proposed methodology are given below.

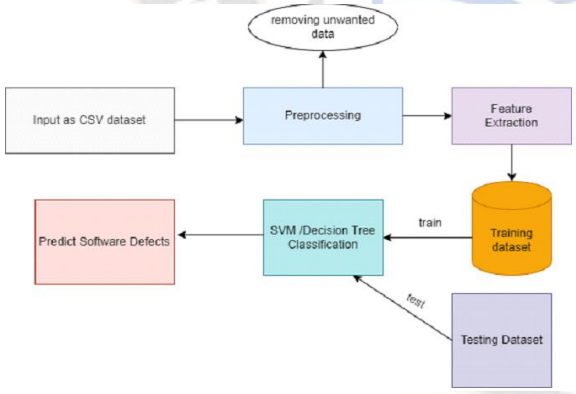

Figure 1: Proposed system overview

1. *Registration:* Users can register right here. While registering the person the device desires the primary facts of a selected user like a content number, First Name, etc.
2. *Login:* If registration is carried out then the user can log in with a username & password. It can be helpful for protection purposes. The username and password discover each user.
3. *Training:*80% training model, with SVM or DECISION TREE algorithm for software data.
4. *Testing:* 20% is the testing part. Testing will be done by giving input as software data & predict the output if the software is defective or not.
5. *Algorithm:* For training and classification, SVM or Decision Tree Algorithm is used.

V. ALGORITHM IMPLEMENTATION

Algorithm 1: SVM (Support vector machine)

Step 1. D = {(x1, y1), (x2, y2), ..., (xn, yn)},

Where xi = feature vector of the ith data point

and yi = label (defective or non-defective).

Step 2. The SVM algorithm finds the hyperplane that separates the data points into different classes

The hyperplane is defined as w*x + b = 0

Where w = weight of the vector

x = feature vector

b = bias term.

Step 3. The SVM algorithm tries to maximize the margin between the hyperplane and the closest data points from each class. The margin is defined as the distance between the hyperplane and the closest data points from each class.

The optimization problem for SVM can be represented as follows:

minimize 0.5 * ||w||^2 + C * sum (max (0, 1 - yi * (w * xi + b)))

where ||w|| represents the norm of the weight vector

C is a regularization parameter

yi is the label of the ith data point.

The objective of the optimization problem is to find the weight vector w and the bias term b that minimize the objective function while satisfying the constraints. The constraints ensure that the hyperplane separates the data points into different classes.

Step 4. Once the SVM algorithm has been trained on the labeled data, it can be used to predict the labels of new data points. The SVM algorithm predicts the label of a new data point based on which side of the hyperplane it lies.

Algorithm 2. DT (Decision Tree)

1. set of labelled data D = {(x1, y1), (x2, y2),...,(xn, yn)}

where xi = the feature vector of the ith data point

yi = its label (defective or non-defective).

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 7s
DOI: https://doi.org/10.17762/ijritcc.v11i7s.6980
Article Received: 30 March 2023 Revised: 14 May 2023 Accepted: 28 May 2023

2. Select a set of features F = {f1, f2, ..., fm} that are relevant to the prediction.
3. Define a stopping criterion, such as the minimum number of data points in a subset or the maximum depth of the tree.
4. Use a metric, such as the Gini impurity or the information gain, to select the best feature for partitioning the data.
5. Create a decision node for the selected feature and partition the data into two subsets based on the value of the feature.
6. Recursively apply steps 4-5 to each subset until the stopping criteria are met.

Once the decision tree has been constructed, we can use it to predict the labels of new data points by traversing the tree from the root to a leaf node based on the values of the features. The leaf node reached by the traversal represents the predicted label of the software module.

VI. CONCLUSION

SVM is a powerful algorithm for classification tasks, and it can handle both linear and nonlinear decision boundaries. It works well with high-dimensional datasets and can handle datasets with a small number of samples. SVM can also be useful in cases where there is a class imbalance in the dataset.

On the other hand, the Decision Tree algorithm is a simpler algorithm that is easier to interpret and visualize. It can handle both categorical and continuous features, and it can be used to identify the most important features that contribute to the classification of software modules.

In terms of performance, the SVM algorithm may outperform the Decision Tree algorithm in certain cases, particularly when the dataset has a large number of features or when the decision boundary is nonlinear. However, the Decision Tree algorithm may be more suitable in cases where interpretability and simplicity are more important.

Ultimately, the choice between SVM and Decision Tree algorithms for software defect prediction will depend on the specific characteristics of the dataset and the goals of the analysis.

ACKNOWLEDGMENT

An efficient software defect prediction has been prepared by Miss Disha Pujari and Prof. Gauri Rao. Author to thank my faculty as well as my whole department, parents, and friends for their support and confidence and obtained a lot of knowledge during the preparation of this document.

REFERENCES

[1] Barr, E. T., Harman, M., & McMinn, P. (2015). A comparative study of bug prediction approaches. Proceedings of the 2015 IEEE/ACM 37th IEEE International Conference on Software Engineering, 1-11.

[2] Shen, H., Huang, J., & Zhang, H. (2006). Defect prediction using a Bayesian belief network. In International Conference on Computational Science and Its Applications (pp. 857-865). Springer, Berlin, Heidelberg.

[3] Menzies, T., Greenwald, J., & Frank, A. (2007). A systematic review of software defect prediction studies. Information and Software Technology, 49(4), 297-310.

[4] Nagappan, N., Ball, T., & Zeller, A. (2006). A Comparison of Machine Learning Techniques for Bug Prediction. In Proceedings of the International Conference on Software Engineering (ICSE), Shanghai, China, 675-684.

[5] Dr. Sandip Kadam. (2014). An Experimental Analysis on performance of Content Management Tools in an Organization. International Journal of New Practices in Management and Engineering, 3(02), 01 - 07. Retrieved from http://ijnpme.org/index.php/IJNPME/article/view/27

[6] Xia, X., Zhang, T., & Lo, D. (2014). An Empirical Study of the Effects of Bug Report Fields on Bug Fixing Time. IEEE Transactions on Software Engineering, 40(4), 373-390. doi: 10.1109/TSE.2013.47

[7] Lessmann, S., Baesens, B., & Mues, C. (2008). An empirical evaluation of classifier ensembles for software defect prediction. Information and Software Technology,50(5),462475.doi:10.1016/j.infsof.2007.09.005

[8] Zhou, Y., Zhang, H., & Kim, S. (2006). A Study of the Relationships between Bug-Fixing Time and Bug-Finding Techniques. Proceedings of the 28th International Conference on Software Engineering (ICSE'06), 943946. doi:10.1145/1134285.1134459

[9] Ernst, M. D., Cockrell, J., & Griswold, W. G. (2003). A Comparative Study of Static Analysis Tools for Bug Finding. Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA'03), 1-11. doi: 10.1145/859563.859581

[10] El Emam, K., Melo, W., & Cruz, P. L. S. (2001). Prediction of Fault-proneness with Object-oriented Metrics – A Comparative Study. Journal of Systems and Software, 56(3), 275-286. doi: 10.1016/S0164-1212(00)00132-5

[11] Miller, J., Evans, A., Martinez, J., Perez, A., & Silva, D. Predictive Maintenance in Engineering Facilities: A Machine Learning Approach. Kuwait Journal of Machine Learning, 1(2). Retrieved from http://kuwaitjournals.com/index.php/kjml/article/view/113

[12] Kamei, Y., Matsumoto, S., & Nakakoji, K. (2010). Software Defect Prediction using Classifier Ensemble Selection. Proceedings of the IEEE/ACM International Conference on Automated Software Engineering (ASE'10), 23-32. doi: 10.1145/1858996.1859001

[13] Ernst, M. D., Cockrell, J., & Griswold, W. G. (2004). A Comparative Study of Static Analysis Tools for Bug Finding in Java. Proceedings of the ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and

International Journal on Recent and Innovation Trends in Computing and Communication
ISSN: 2321-8169 Volume: 11 Issue: 7s
DOI: https://doi.org/10.17762/ijritcc.v11i7s.6980
***Article Received**: 30 March 2023 **Revised**: 14 May 2023 **Accepted**: 28 May 2023*

Applications (OOPSLA'04), 1-15. doi: 10.1145/1028976.1028978

[14] Harman, M., Jones, B., & Shepperd, M. J. (2001). On the application of genetic algorithms to software engineering: A survey. IEEE Transactions on Evolutionary Computation, 5(2), 97-116.

[15] Subramanyam, J., & Krishnan, S. (2003). Empirical validation of object-oriented metrics on open-source software for fault prediction. IEEE Transactions on Software Engineering, 29(8), 697-708.

[16] Harman, M., Jones, B. F., & Zhang, Y. (2012). Using Evolutionary Algorithms to Improve Software Quality. Communications of the ACM, 55(7), 68-76. doi: 10.1145/2209249.2209269

[17] Singh, A. ., & Kumar, V. . (2023). Sentiment Analysis of Customer Satisfaction Towards Repurchase Intension and the Word-Of-Mouth Advertising in Online Shopping Behavior Using Regression Analysis and Statistical Computing Techniques. International Journal of Intelligent Systems and Applications in Engineering, 11(2s), 45–51. Retrieved from https://ijisae.org

[18] El Emam, K., & Benlarbi, S. (2001). A comparison of the predictive power of object-oriented and procedure-oriented design metrics for fault prediction. IEEE Transactions on Software Engineering, 27(7), 677-686. doi: 10.1109/32.946966

[19] Mockus, A., Fielding, R. T., & Herbsleb, J. D. (2000). A study of the characteristics of bugs. In Proceedings of the 2000 International Conference on Software Engineering (pp. 294-301). ACM. doi: 10.1145/337180.337209

[20] Sahoo, D. K. . (2021). Improved Routing and Secure Data Transmission in Mobile Adhoc Networks Using Trust Based Efficient Randomized Multicast Protocol. Research Journal of Computer Systems and Engineering, 2(2), 06:11. Retrieved from https://technicaljournals.org/RJCSE/index.php/journal/article/view/25

[21] Panichella, S., Di Sorbo, A., & Visaggio, C. A. (2016). BugNet: continuously mining distributed bug reports. In Proceedings of the 38th International Conference on Software Engineering (pp. 414-425). ACM. doi: 10.1145/2884781.2884837

[22] El Emam, K., Benlarbi, S., & Goel, N. (2001). Fault-proneness estimation using artificial neural networks: A comparative study. Journal of Systems and Software, 56(3), 275-287. doi: 10.1016/S0164-1212(01)00122-6

[23] Alejandro Garcia, Machine Learning for Customer Segmentation and Targeted Marketing , Machine Learning Applications Conference Proceedings, Vol 3 2023.