Software Bug Prediction using Decision Tree algorithm

Shaan Kumar
Department of Computer science engineering
IIT BHU, Varanasi
India
shaan.kumar.cse19@itbhu.ac.in

***Abstract*—Bugs in software programs have been a very crucial and challenging problem in the software development domain since a long time. These bugs can often arise due to human errors and carelessness but there may be other factors too. The losses made due to errors in critical programs and time consumed in development and maintenance of those programs is one of the major cost components of software development today hence it is of utmost importance to develop an automated methodology that can help in identifying possible bugs during the software development process. Such an model would be highly useful in reducing the costs as well as time associated with software development process.**

In this paper, we are proposing our findings and approaches in using a Decision Tree algorithm to develop a bug prediction model on the PROMISE data set. A decision tree is a flowchart-like structure where each internal node represents a test on an attribute, each branch represents the outcome of the test, and each leaf node represents a class label. We also evaluate the performance of this model based on various parameters like accuracy, F1 score, precision, recall etc. and achieved a significant performance on all metrics.

***Index Terms*—Bug prediction, Decision Tree, Software Fault, Machine Learning, Defect prediction**

I. Introduction

The vast and diverse realm of software development, along with its myriad applications, presents a formidable challenge for both software developers and end-users. The task of observing, maintaining, and managing software applications has become increasingly complex in the wake of the fourth industrial revolution, marked by the widespread incorporation of artificial intelligence within the software industry. This sector stands out as one of the most promising fields in modern times, witnessing continual transformation owing to the automation of large quantities of software technologies [1]. The continuous growth in size and complexity of contemporary software applications poses an ongoing struggle for software engineers, who grapple with defects right from the initial stages of development.

The categorization of software faults holds paramount importance in real-time scenarios; failure to do so leads to a rapid escalation in the effort and cost involved in identifying defects lurking within an application. This pressing need has spurred the development of automated fault prediction models for software defects. Detecting software defects before the software's release enables developers to pinpoint and rectify these defective modules with ease.

Machine learning techniques have emerged as the predominant solution for software fault prediction, captivating the attention of researchers and the software community alike [2]. Cutting-edge machine learning algorithms have been deployed to pinpoint defective modules within software applications, facilitating both research endeavors and practical solutions for consumers [3]. In this study, we explore six of the most popular machine learning classifiers, as recommended in a recent comprehensive literature review [4]. These selected classification techniques are applied across various real application datasets pertinent to software fault prediction. However, it is imperative to acknowledge specific data features in terms of their quality, although their correctness cannot be definitively validated. Consequently, state-of-the-art machine learning approaches [5] have been employed on fault datasets to enhance predictions. This enhancement is achieved through the elimination of superfluous features using multiple feature selection techniques and methods to balance imbalanced data.

Numerous studies have delved into fault recovery methods within software, with a focus on integrating automated recovery models within applications [6][7][8]. This study aims to scrutinize the performance of six classifiers and advocates for an automated approach to rectify software faults within applications. For our analysis, we utilized three datasets (JM1, CM1, PC1) from the PROMISE Software Engineering Repository, encompassing 22 attributes such as McCabe and Halstead metrics, along with defect information [9].

To prepare our selected data, we engaged in preprocessing activities, identifying correlated columns and addressing data imbalance issues. Various computational techniques, including PCA, Resample, and SMOTE, were employed to mitigate these challenges [10]. Consequently, this paper offers a comparative analysis of six machine learning techniques for software fault prediction within applications.

The subsequent sections of this paper are structured as follows: Section 2 outlines the datasets, classification techniques, performance metrics, and experimental setup. Section 4 presents the analysis results, while Section 5 offers conclusions and outlines directions for future research.

II. Related Work

Software fault localization and maintainability refer to the ability of a software system or its modules to be adapted for correcting faults, improving performance, conducting software and system testing, employing software development techniques, or modifying for a changed platform [11]. Predictive models for software defects empower organizations to reduce maintenance efforts, time, and overall costs in software projects [12] [13]. Ensuring the quality of software is essential, as it minimizes failures during runtime [14]. Consequently, classifying defects in software modules significantly impacts the software development process. However, real scenarios often become complex when developers modify their programs within an application, affecting other modules, potentially leading to faults and instability [15].

The literature on software fault proneness is expanding due to the growing demand for automated services [16]. Malhutra conducted a comparative study involving Artificial Neural Networks (ANN), Support Vector Machines (SVM), Decision Trees (DT), Convolutional Neural Networks (CCN), Group Method of Data Handling (GMDH), and Gene Expression Programming (GEP) to predict software fault modules. This study utilized data from 3 NASA projects in the PROMISE Software Engineering Repository. Among the classifiers, Decision Tre exhibited the highest accuracy. The study incorporated 21 CK metrics, McCabe metrics, and Halstead metrics [17]. Xu et al. introduced a defect prediction framework called KPWE, combining Kernel Principal Component Analysis (KPCA) and Weighted Extreme Learning Machine (WELM). Their study, based on 44 software projects, demonstrated KPWE's superiority over baseline methods in most cases [18]. Moeyersoms et al. conducted a comparative study using Random Forest (RF), SVM, C4.5 (DT), and Regression Tree to predict software defect modules. The results indicated that RF achieved the highest accuracy among the classifiers [19]. Yu et al. developed novel feature subset selection and classification techniques to investigate feature selection's effectiveness for cross-project defect prediction (CPDP). Their experiments revealed that CPDP feature selection approaches enhance software defect analysis performance [20]. Hotzkow discussed applications capable of learning user expectations from semantic contexts across multiple applications [21].

Researchers are keenly exploring automated fault tolerance within software systems [22]. Montani et al. described Case-based Reasoning (CBR) methods, enabling software to predict faults in distributed software applications effectively. Another study emphasized machine learning-based approaches over statistical models for software fault prediction [23][24]. Therefore, utilizing open-source datasets is crucial due to their reliability, consistency, and verifiability [25] [26].

III. Methodology

In this we first go over the dataset in detail then the decision tree based model for classification and finally the various performance metrics associated with them.

A. Dataset

The PROMISE software engineering database is a comprehensive collection of publicly available datasets designed for evaluating software engineering models and techniques. These datasets encompass various domains, including:

- **Software Fault Prediction:** These datasets provide detailed information about software modules, including metrics measuring their size, complexity, and historical data, along with indicators of whether they contain faults.
- **Software Defect Prediction:** These datasets offer insights into software defects, encompassing data on their severity, type, location, and the modules where they are found.
- **Software Cost Estimation:** These datasets contain pertinent details about software projects, such as size, complexity, effort, and actual costs incurred.
- **Software Quality Prediction:** These datasets include information on software quality attributes, such as reliability, maintainability, usability, and the specific modules contributing to these attributes.

The PROMISE software engineering database serves as an invaluable resource for both researchers and practitioners involved in the development and evaluation of software engineering models and techniques. These datasets are meticulously documented and user-friendly, enabling their seamless utilization for a wide array of model evaluations and analyses. Table I below show the list of metrics used

TABLE I: List of metrics

No	Type	Metrics Name
1	McCabe	McCabe
2	Halstead	Line of code
3	Halstead	Cyclomatic complexity
4	Halstead	Essential complexity
5	Halstead	Design complexity
6	Halstead	Halstead operators and operands
7	Halstead	Halstead volume
8	Halstead	Halstead program length
9	Halstead	Halstead difficulty
10	Halstead	Halstead intelligence
11	Halstead	Halstead effort
12	Halstead	Halstead time estimator
13	Halstead	Halstead line count
14	Halstead	Halstead comments count
15	Halstead	Halstead blank line count
16	Miscellaneous	IO code and comments
17	Miscellaneous	Unique operators
18	Miscellaneous	Unique operands
19	Miscellaneous	Total operators
20	Miscellaneous	Total operands
21	Miscellaneous	Branch count
22	Halstead	b: numeric
23	Miscellaneous	Defects
24	Miscellaneous	False or true

The term neighbors here, in the context of a particular pixel, refers to the set of pixels that are in the $D \times D$ square centered on the $Query$ pixel, where D is a hyperparameter, chosen to be a small odd integer value (5 in our case). The $Query$ pixel here is the pixel for which we want to calculate attention values for. Since we need to calculate attention values for all such pixels, we need an efficient way for calculating and representing the neighbors of all pixels, especially one that is vectorisable.

Within the PROMISE software engineering database, several sub datasets are available, each offering valuable insights into specific programming languages and their respective modules. These sub datasets, namely JM1, CM1, and PC1, provide detailed information about various software modules, including their metrics, historical data, and fault-related characteristics.

B. JM1 Dataset

The JM1 dataset encompasses data from 467 Java modules. It includes comprehensive metrics, historical records, and indications of whether these modules contain faults. Researchers and practitioners can analyze this dataset to gain a deep understanding of Java-based software development, enabling them to evaluate software engineering models and techniques specifically tailored for Java applications.

C. CM1 Dataset

In the CM1 dataset, information on 1,016 C modules is available. This dataset provides metrics, historical context, and fault-related data for C programming language modules. Software engineers and developers working with C-based applications can utilize this dataset to enhance their understanding of C module characteristics. By exploring this dataset, researchers can develop and assess software engineering methodologies tailored to C programming.

D. PC1 Dataset

The PC1 dataset comprises details from 109 Pascal modules. It includes metrics, historical information, and fault-related attributes specific to Pascal programming. This dataset is particularly valuable for researchers focusing on Pascal-based software systems. By delving into the PC1 dataset, experts can refine their software engineering models and techniques to address the unique challenges posed by Pascal applications.

These sub datasets within the PROMISE software engineering database serve as essential resources for the research community, enabling in-depth analyses and evaluations across diverse programming languages and modules.

E. Decision tree Classification

Decision trees are a type of machine learning algorithm that can be used for classification tasks. They work by constructing a tree structure, where each node represents a feature and each leaf node represents a class label. The goal of the tree is to learn a set of rules that can be used to classify new data points. To classify a new data point, the algorithm starts at the root node and traverses the tree down to a leaf node. At each node, it compares the value of the current feature to a threshold value. If the value is greater than or equal to the threshold, the algorithm goes to the left child node. Otherwise, it goes to the right child node. Once the algorithm reaches a leaf node, it predicts the class label associated with that node.

Decision trees can be used for software defect classification by training them on a dataset of known software defects. The features of the dataset can be metrics such as the number of lines of code, the number of cyclomatic complexity, and the number of bugs found in previous versions of the software. The class label can be either "defective" or "non-defective". Once the decision tree is trained, it can be used to classify new software modules. The algorithm will traverse the tree down to a leaf node and predict the defect status of the module based on the values of the features.

Decision trees are a powerful tool for software defect classification. They are relatively simple to understand and interpret, and they can be used to identify the most important features that contribute to software defects. However, decision trees can be overfitting, so it is important to use cross-validation to tune the hyperparameters of the model and to consider using ensemble methods to improve the generalization performance of the model.

IV. Results

In this section, we present the results of our experiments on software defect prediction using the Decision tree classifier. Various metrics were utilized to evaluate the model's performance, including accuracy, precision, recall, and F1-score. These metrics provide a comprehensive understanding of how well the classifier performs in different aspects.

Accuracy measures the overall correctness of the model, indicating the proportion of correctly classified instances among the total instances in the dataset. A higher accuracy score implies a better predictive performance of the model.

Precision represents the fraction of true positive instances out of all instances predicted as positive by the model. It highlights the model's ability to avoid false positives, making it a crucial metric in applications where minimizing false alarms is essential.

Recall, also known as sensitivity or true positive rate, calculates the proportion of true positive instances that were correctly identified by the model out of all actual positive instances in the dataset. Recall is vital when the goal is to capture as many positive instances as possible, even at the cost of some false positives.

F1-score is the harmonic mean of precision and recall, offering a balance between these two metrics. It provides a single value that considers both false positives and false negatives, making it useful in scenarios where there is an uneven class distribution.

TABLE II: Performance Metrics of Decision tree Classifier

Dataset	Model	Precision	Recall	Accuracy	F1-score
JM1	Decision tree	1.0	0.99	0.99	0.99
CM1	Decision tree	1.0	1.0	1.0	1.0
PC1	Decision tree	0.97	1.0	0.99	0.98

As shown in Table II, the decision tree classifier achieved exceptional performance across all three datasets, demonstrating high precision, recall, accuracy, and F1-score values. These results underscore the effectiveness of the decision tree algorithm in accurately predicting software defects in diverse contexts.

V. Conclusion

In conclusion, this study applied the decision tree classifier to predict software defects using the JM1, CM1, and PC1 datasets. The results exhibited exceptional performance across various metrics, including precision, recall, accuracy, and F1-score. The high precision values underscore the model's ability to minimize false positives, ensuring that flagged issues are likely genuine, thereby optimizing the allocation of resources for defect resolution. Moreover, the high recall values indicate the model's effectiveness in capturing most of the actual defects, making it valuable for comprehensive defect detection. The outstanding accuracy and F1-score values further validate the decision tree classifier's robustness in software defect prediction. These findings highlight the potential of machine learning techniques, specifically the decision tree algorithm, in enhancing software quality and supporting developers in building more reliable applications.

Looking ahead, future research could explore diverse avenues to further enhance software defect prediction. Investigating alternative machine learning algorithms and ensemble methods might provide comparative insights into their efficacy for defect prediction tasks. Additionally, experimenting with advanced feature engineering techniques could refine the model's accuracy by identifying the most influential features for defect prediction. Further studies on larger and more diverse datasets from various domains would help generalize the findings and validate the model's applicability across different contexts. Integrating real-time data and continuous monitoring mechanisms could enable the development of dynamic defect prediction systems, allowing developers to address emerging issues promptly. Exploring the application of deep learning models and natural language processing techniques for analyzing code comments and documentation could offer a multi-faceted approach to defect prediction, ensuring a more comprehensive evaluation of software quality.

In summary, this research not only demonstrates the effectiveness of the decision tree classifier in software defect prediction but also opens avenues for future investigations, paving the way for more accurate and efficient approaches to software quality assurance.

References

1) H. B. Bolat, G. T. Temur, and IGI Global, Agile approaches for managing projects in the fourth industrial revolution.
2) G. P. Bhandari and R. Gupta, "Machine learning based software fault prediction utilizing source code metrics," in 2018 IEEE 3rd International Conference on Computing, Communication and Security (ICCCS), 2018, pp. 40–45.
3) R. Malhotra, "A systematic review of machine learning techniques for software fault prediction," Appl. Soft Comput., vol. 27, pp. 504–518, Feb. 2015.
4) L. Son et al., "Empirical Study of Software Defect Prediction: A Systematic Mapping," Symmetry (Basel)., vol. 11, no. 2, p. 212, Feb. 2019.
5) C. W. Yohannese and T. Li, "A Combined-Learning Based Framework for Improved Software Fault Prediction," Int. J. Comput. Intell. Syst., vol. 10, no. 1, p. 647, Dec. 2017.
6) R. Malhotra and S. Kamal, "An empirical study to investigate oversampling methods for improving software defect prediction using imbalanced data," Neurocomputing, vol. 343, pp. 120–140, May 2019.
7) S. Montani and C. Anglano, "Achieving self-healing in service delivery software systems by means of case-based reasoning," Appl. Intell., vol. 28, no. 2, pp. 139–152, Apr. 2008.
8) C. Catal and B. Diri, "A systematic review of software fault prediction studies," Expert Syst. Appl., vol. 36, no. 4, pp. 7346–7354, May 2009.
9) V. U. B. CHALLAGULLA, F. B. BASTANI, I.-L. YEN, and R. A. PAUL, "EMPIRICAL ASSESSMENT OF MACHINE LEARNING BASED SOFTWARE DEFECT PREDICTION TECHNIQUES," Int. J. Artif. Intell. Tools, vol. 17, no. 02, pp. 389–400, Apr. 2008.
10) J. Moeyersoms, E. Junqué de Fortuny, K. Dejaeger, B. Baesens, and D. Martens, "Comprehensible software fault and effort prediction: A data mining approach," J. Syst. Softw., vol. 100, pp. 80–90, Feb. 2015.
11) Q. Yu, J. Qian, S. Jiang, Z. Wu, and G. Zhang, "An Empirical Study on the Effectiveness of Feature Selection for Cross-Project Defect Prediction," IEEE Access, vol. 7, pp. 35710–35718, 2019.
12) J. Hotzkow and Jenny, "Automatically inferring and enforcing user expectations," in Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis - ISSTA 2017, 2017, pp. 420–423.
13) A. A. Hudaib, H. N. Fakhouri, A. A. Hudaib, and H. N. Fakhouri, "An Automated Approach for Software Fault Detection and Recovery," Commun. Netw., vol. 08, no. 03, pp. 158–169, Jul. 2016.
14) S. Montani and C. Anglano, "Achieving self-healing in service delivery software systems by means of case-based reasoning," Appl. Intell., vol. 28, no. 2, pp. 139–152, Apr. 2008.
15) C. Catal and B. Diri, "A systematic review of software fault prediction studies," Expert Syst. Appl., vol. 36, no. 4, pp. 7346–7354, May 2009.
16) V. U. B. CHALLAGULLA, F. B. BASTANI, I.-L. YEN, and R. A. PAUL, "EMPIRICAL ASSESSMENT OF MACHINE LEARNING BASED SOFTWARE DEFECT PREDICTION TECHNIQUES," Int. J. Artif. Intell. Tools, vol. 17, no. 02, pp. 389–400, Apr. 2008.
17) E. Shihab et al., "Studying re-opened bugs in open source software," Empir. Softw. Eng., vol. 18, no. 5, pp. 1005–1042, Oct. 2013.
18) M. Shepperd, Q. Song, Z. Sun, and C. Mair, "Data Quality: Some Comments on the NASA Software Defect

Datasets," IEEE Trans. Softw. Eng., vol. 39, no. 9, pp. 1208–1215, Sep. 2013.

19) T. Menzies, J. Distefano, A. O. S, and R. M. Chapman, "Assessing Predictors of Software Defects."

20) M. Jureczko and L. Madeyski, "Towards identifying software project clusters with regard to defect prediction," in Proceedings of the 6th International Conference on Predictive Models in Software Engineering - PROMISE '10, 2010, p. 1.