Original Research Article

SOFTWARE DEFECT PREDICTION SYSTEM BASED ON DECISION TREE ALGORITHM

ABSTRACT

Software defect prediction plays a crucial role in ensuring software quality and minimizing the potential risks associated with defects. This study aims to develop a comprehensive software defect prediction system that utilizes tree-based algorithms to enhance accuracy, feature selection, and evaluation metrics. The study addresses the limitations of previous research by considering a broader range of datasets, comparing computational efficiency with other ensemble techniques, and examining the impact of hyper parameters on model performance. The implemented system consists of three stages: dataset loading, processing, and result presentation. The dataset loading page allows users to upload their datasets in CSV format, simplifying the prediction process. The processing page performs essential tasks such as feature engineering, normalization using minimax normalization, and training the model with the decision tree algorithm. These steps ensure the extraction of relevant features, transformation of data, and learning of patterns and correlations for accurate software defect prediction. The study emphasizes the practical implementation of the developed system, going beyond model evaluation. By providing a fully functional and integrated system, this study bridges the gap between research and real-world application. The findings of this study contribute to the field of software defect prediction by offering an improved system that enhances accuracy, feature selection, and evaluation metrics. This has implications for software development and quality assurance processes, ultimately leading to higher software quality and increased productivity.

Keywords: (ABS): Software Defect, Prediction System, Decision Tree, Algorithm, Model, Feature engineering, Correlation, and Evaluation Metrics

1 INTRODUCTION

The software industry is rapidly growing, and the development of software systems has become more complex than ever (Abdulkadir & Kum, 2019). As a result, software defects have become a significant challenge in software development, as they can lead to significant losses, including financial losses, damage to the reputation of the company, and even loss of life in extreme cases (Wang et al., 2018). Software defects are expensive to fix, and they can cause project delays, leading to increased costs and lost productivity (Ma & Liu, 2019; Hall et al., 2012). Therefore, software defect prediction has become an essential aspect of software engineering, as it helps to identify potential defects in advance, before they cause any significant issues. Traditional approaches to software defect prediction are time-consuming and require manual effort, making it difficult to predict defects accurately (Ouni et al., 2014). However, machine learning techniques, such as tree-based algorithms, have shown great promise in predicting software defects (Costa et al., 2017). Tree-based algorithms, such as Random Forest and Decision Tree, are powerful machine learning algorithms that can be used for classification tasks, including software defect prediction (Chen & Li, 2018). These algorithms use a decision tree model to represent the mapping between input features and output labels and can handle both categorical and numerical data (Deng et al., 2020). Furthermore, this research aims to address some of the limitations of the existing software defect prediction systems. While previous studies have explored the use of machine learning algorithms for software defect prediction, there is still room for improvement in terms of accuracy, feature selection, and evaluation metrics (Panigrahi et al., 2020). Additionally, there is a need to investigate the performance of different tree-based algorithms, such as Random Forest and Decision Tree, in software defect prediction (Zhang et al., 2021). Therefore, this dissertation proposes a software defect prediction system that uses tree-based algorithms to address these limitations. The system will involve data collection, cleaning, and preprocessing, followed by feature selection and the implementation of the two tree-based algorithms. The performance of the algorithms will be evaluated using various evaluation metrics, such as accuracy, precision, recall, and F1-score. The most significant features that contribute to defects will also be identified. The results of this research will provide valuable insights into the effectiveness of tree-based algorithms for software defect prediction and contribute to the development of more accurate and efficient software defect prediction systems. Additionally, the proposed system will help software developers identify potential defects early in the software development process, allowing them to take corrective action before the defects cause any significant issues. This will lead to increased productivity, reduced costs, and improved software quality (Kaur & Singh, 2018). Furthermore, the proposed system will contribute to the existing body of knowledge on software defect prediction and machine learning techniques, providing valuable insights into the use of tree-based algorithms for software defect prediction.

2. LITERATURE REVIEW

In their study, Sun et al. (2022) proposed a novel software defect prediction approach based on decision tree algorithms. The authors aimed to improve the prediction accuracy of software defects by incorporating feature selection techniques and ensemble learning methods into decision tree algorithms. The proposed approach was evaluated using four publicly available datasets, and the results showed that it outperformed several state-of-the-art techniques in terms of accuracy, precision, and recall. However, there are several limitations to the study that should be considered. First, the proposed approach was only evaluated on four datasets, which may not be representative of all software development projects. Future studies should consider evaluating the approach on a more diverse set of datasets to assess its generalizability. Second, the authors did not compare their approach with other ensemble learning methods, such as random forests and gradient boosting, which are known to perform well in software defect prediction tasks. Lastly, the study did not consider the potential impact of imbalanced datasets, which can occur frequently in software defect prediction tasks and can affect the accuracy of the prediction models.

Shukla and Gupta (2022) propose an approach for efficient feature selection in software defect prediction using decision trees. The approach involves identifying and selecting only the most relevant features for building decision tree models, which can reduce the computational cost and improve the accuracy of the models. The authors evaluate their approach using four software datasets and compare it with other feature selection methods, showing that it can achieve comparable or better performance with a significantly reduced number of features. However, one limitation of the study is that it only focuses on decision tree-based models and does not compare the proposed approach with feature selection methods for other machine learning algorithms, which may have different performance characteristics. Additionally, the study only evaluates the approach on a limited number of datasets, and it would be beneficial to test it on a more extensive range of software datasets to assess its generalizability.

In this study, Qasim et al., (2022) compared the performance of different decision tree algorithms, including C4.5, CART, CHAID, and QUEST, in predicting software defects. The authors used four different datasets from different software development companies to evaluate the performance of these algorithms based on different evaluation metrics, such as accuracy, precision, recall, F1-score, and AUC. The results showed that C4.5 and CART outperformed CHAID and QUEST in terms of prediction accuracy and other evaluation metrics. One limitation of this study is that it only focused on comparing decision tree algorithms and did not consider other machine learning algorithms or ensemble methods. Additionally, the study only used four datasets, which may not be representative of all software development contexts. Finally, the study did not explore the interpretability of the decision tree models and their ability to provide actionable insights for software developers.

Zhang, Lin, and Chen (2021) proposed a new approach to improve software defect prediction using decision tree-based feature selection and resampling techniques. The authors applied this approach on three software datasets and compared the results with other state-of-the-art approaches. They reported that their proposed approach outperformed the existing methods in terms of accuracy, precision, recall, and F-measure. However, there are some limitations to this study. Firstly, the authors only evaluated their proposed approach on three datasets, which may not be representative of all software defect prediction scenarios. Secondly, the study only compared the proposed approach with a limited number of state-of-the-art approaches, which may not provide a comprehensive comparison. Finally, the proposed approach may not be applicable to all software defect prediction problems, as the effectiveness of the approach may depend on the specific characteristics of the dataset.

Li and Li (2021) proposed the use of decision tree ensembles (DTEs) for software defect prediction, which combines the predictions of multiple decision trees to improve the accuracy of the predictions. The authors used 8 publicly available datasets and compared the performance of DTEs with single decision trees and other ensemble techniques such as random forests and gradient boosting. The results showed that DTEs outperformed single decision trees and were competitive with random forests and gradient boosting in terms of accuracy. However, the study has several limitations. First, the authors only used a limited number of datasets, which may not represent the diversity of software projects in practice. Second, the study only focused on defect prediction and did not investigate the performance of DTEs on other software engineering tasks such as code quality analysis or software maintenance. Finally, the study did not compare the computational efficiency of DTEs with other ensemble techniques, which is an important consideration in real-world applications where large datasets and limited computational resources are common.

Ma, H. & Zhang, J. (2021) conducted a study titled "A decision tree-based approach to software defect prediction: A case study in the automotive industry." The aim of this study was to evaluate the effectiveness of decision tree-based approaches in predicting software defects in the automotive industry. The study used data from the software development process of a large automotive company and compared the performance of different decision tree-based models in predicting defects. The results of the study indicated that decision tree-based models were effective in predicting software defects in the automotive industry. The study found that the Random Forest algorithm outperformed other decision tree-based algorithms in terms of prediction accuracy. The study also identified several important features that were strongly associated with software defects in the automotive industry. One limitation of this study is that it focused on a single industry, which may limit the generalizability of the findings. The study also did not consider the impact of different parameters or hyper parameters on the performance of the decision tree-based models. Future studies could explore the effectiveness of decision tree-based approaches in predicting software defects in other industries and investigate the impact of different parameters on the performance of these models.

The study by Zhang et al. (2021) aimed to propose an effective feature selection method based on decision tree algorithm for software defect prediction. The researchers first extracted a set of static code metrics as potential features for predicting software defects. Then, they applied decision tree algorithm to the dataset to identify the most important features. The selected features were used to build a prediction model based on decision tree algorithm. The proposed method was evaluated on three

software datasets and compared with other feature selection methods. One of the limitations of this study is the use of a limited number of datasets for evaluation. The study only evaluated the proposed method on three datasets, which may not be representative of other software projects. Additionally, the study only used static code metrics as potential features, which may not capture all factors that affect software defects. Finally, the study did not investigate the generalizability of the proposed method to other machine learning algorithms or defect prediction models.

In this study, Liu et al., (2020) proposed a hybrid approach for software defect prediction based on a combination of decision tree and support vector machine (SVM) algorithms. The authors aimed to improve the accuracy of software defect prediction by incorporating the strengths of both algorithms. The proposed approach involved feature selection using decision tree algorithms, followed by classification using SVM. The study used six open-source datasets from the PROMISE repository for experimentation and compared the results with those obtained from individual decision tree and SVM algorithms. The results showed that the hybrid approach outperformed both individual algorithms in terms of accuracy, precision, recall, and F-measure. One limitation of this study is that it only used open-source datasets from a single repository, which may not be representative of all software defect prediction scenarios. Additionally, the study did not compare the proposed approach with other state-of-the-art hybrid approaches for software defect prediction. Therefore, the generalizability and competitiveness of the proposed approach in comparison to other approaches remains unclear.

The study conducted by Hasan and Uddin (2020) aimed to compare the effectiveness of decision tree and Naive Bayes algorithms for software defect prediction. The authors collected data from the NASA Metrics Data Program (MDP) and preprocessed the data to remove any missing or incomplete records. They then applied both decision tree and Naive Bayes algorithms to the preprocessed data to classify the records as either defective or non-defective. The results showed that the decision tree algorithm had a higher accuracy rate compared to the Naive Bayes algorithm for software defect prediction. The decision tree algorithm achieved an accuracy rate of 79.44%, while the Naive Bayes algorithm achieved an accuracy rate of 66.39%. However, the study has some limitations that should be considered. First, the study only used one dataset (MDP) for evaluation, which may limit the generalizability of the results. Additionally, the study only used two algorithms for comparison, which may not provide a comprehensive comparison of all available algorithms. Finally, the study did not consider the effects of different feature selection techniques on the performance of the algorithms.

Liu et al. (2020) proposed a hybrid approach for software defect prediction based on decision tree (DT) and support vector machine (SVM). The proposed approach aimed to improve the prediction accuracy of traditional DT algorithm by integrating SVM to address its limitation in dealing with complex and high-dimensional data. The study conducted experiments on three real-world datasets, and the results showed that the hybrid approach outperformed the traditional DT and SVM algorithms in terms of prediction accuracy. One limitation of this study is that it only compared the hybrid approach with traditional DT and SVM algorithms, without comparing it with other state-of-the-art approaches. Additionally, the study did not consider the interpretability of the model, as both DT and SVM are known to be less interpretable compared to other machine learning algorithms. Moreover, the study did not provide insights on how the proposed hybrid approach could be used in practice, such as how to tune the hyper parameters or how to select the most important features for the model.

Li, Dai, and Tang (2019) conducted an empirical study to compare the performance of three decision tree algorithms (C4.5, CART, and Random Forest) in predicting software defects. They used metrics such as precision, recall, F-measure, and AUC to evaluate the models' performance. The study used five open-source datasets from the PROMISE repository and compared the performance of the three algorithms in terms of their ability to predict software defects accurately. The study found that the Random Forest algorithm outperformed C4.5 and CART in terms of accuracy, precision, recall, and F-measure. The study also showed that using the right set of features could improve the accuracy of the models. One limitation of this study is that it only compared three decision tree algorithms, and other classification algorithms were not considered. Additionally, the study used only five datasets, which may not be representative of all software systems. Further research is needed to validate the results using a larger number of datasets and to explore the effectiveness of other classification algorithms in predicting software defects.

In their study, Mustafa and Azam (2019) proposed a novel approach for software defect prediction using decision trees and a combination of resampling techniques. The authors aimed to address the imbalanced data issue that is common in software defect prediction by employing both random under sampling and synthetic minority oversampling technique (SMOTE) to balance the dataset. The effectiveness of the proposed approach was evaluated using several performance metrics, including precision, recall, F1-score, and AUC. The study showed promising results, demonstrating that the proposed approach achieved better performance compared to traditional decision tree-based approaches. However, the study also had some limitations. For instance, the proposed approach was only evaluated on a single dataset, which limits the generalizability of the results. Additionally, the authors did not compare the performance of their approach with other state-of-the-art methods, which makes it difficult to assess the competitiveness of the proposed approach. Finally, the authors did not provide an in-depth analysis of the decision tree model to understand how the model is making predictions, which is essential for model interpretation and understanding.

Li & Liu (2019) proposed a software defect prediction model that uses decision tree algorithm with genetic algorithm feature selection. The study aims to improve the performance of software defect prediction by selecting the most important features through a genetic algorithm-based method. The authors applied the model to four open-source software datasets and compared the results with other feature selection methods. The experimental results show that the proposed approach outperforms the other methods in terms of prediction accuracy, sensitivity, and specificity. However, the study has some limitations. First, the proposed model was only evaluated on four open-source software datasets. It is unclear whether the results can be generalized

to other software datasets with different characteristics. Second, the study did not compare the proposed approach with other software defect prediction models that use decision tree algorithm with different feature selection methods, which could provide more insights into the effectiveness of the proposed approach. Finally, the study did not provide an in-depth analysis of the feature selection results, which may limit the understanding of the importance of the selected features.

Sharma and Vyas (2019) conducted a study that aim of this study was to investigate the effectiveness of feature selection techniques for software defect prediction using decision tree algorithms. The authors employed the chi-squared feature selection method to select the most relevant features for building the decision tree model. The study was conducted on two datasets, one from the NASA MDP repository and the other from the PROMISE repository. The authors compared the performance of different decision tree algorithms, including C4.5, CART, and ID3, with and without feature selection. The results of the study showed that the decision tree models built with the chi-squared feature selection method outperformed those built without feature selection in terms of accuracy, precision, recall, and F-measure. The study also revealed that the C4.5 algorithm performed better than CART and ID3 algorithms in terms of prediction accuracy. However, the study had some limitations. One of the limitations of the study is that the authors only used one feature selection method, namely chi-squared, and did not investigate other feature selection techniques. The study could have been more comprehensive if the authors had tested other feature selection methods such as correlation-based feature selection or wrapper methods. Another limitation is that the study only evaluated the performance of decision tree algorithms and did not compare the results with other machine learning algorithms, such as support vector machines or neural networks. It would have been interesting to see how decision trees compared to other algorithms for software defect prediction. Furthermore, the study used only two datasets, which may not be representative of all software development environments. Future studies could explore the performance of decision tree algorithms with feature selection on a larger and more diverse set of datasets to ensure the generalizability of the findings.

Yadav and Gupta's (2019) aimed to compare and evaluate the performance of different decision tree-based models in predicting software defects. The authors used six different decision tree algorithms: C4.5, CART, M5P, QUEST, Random Forest, and Decision Table. They also used various software metrics as features for predicting defects. However, the study's limitation lies in the fact that it used only one dataset, the NASA metrics data, for the experiments. This dataset may not be representative of all software projects, and the results obtained may not generalize well to other datasets. Additionally, the authors did not compare their results with those obtained using other classification algorithms, which may have provided a more comprehensive comparison of the decision tree-based models' effectiveness.

The study conducted by Li et al. (2019) aimed to investigate the effectiveness of decision tree algorithms for software defect prediction. The authors used four decision tree algorithms, namely, CART, ID3, C4.5, and Random Forest, to build predictive models for three publicly available software defect datasets. The study evaluated the performance of these models in terms of accuracy, precision, recall, F-measure, and area under the receiver operating characteristic (ROC) curve.

The findings of the study showed that Random Forest outperformed the other decision tree algorithms in terms of prediction accuracy and area under the ROC curve. However, the study has some limitations. First, the study only used three datasets, which may not represent the diversity of software systems. Second, the study did not compare the performance of decision tree algorithms with other machine learning algorithms. Finally, the study did not investigate the impact of feature selection and resampling techniques on the performance of decision tree algorithms. Therefore, future research can address these limitations and further investigate the effectiveness of decision tree algorithms for software defect prediction.

In this study, Mustafa and Azam proposed a decision tree-based approach for software defect prediction that uses both random undersampling and synthetic minority oversampling technique (SMOTE) to address the imbalanced nature of software defect datasets. The authors used six software datasets and compared the performance of their proposed approach with other traditional and state-of-the-art approaches. The results showed that their approach outperformed other methods in terms of accuracy, precision, recall, and F1-score. One limitation of this study is that it only tested the proposed approach on a limited number of software datasets, which may not be representative of all possible scenarios. Additionally, the study did not provide any explanation for the choice of specific parameters used in the experiments, such as the number of decision trees or the threshold value for the SMOTE technique. Future research could investigate the generalizability of the proposed approach on larger and more diverse datasets, and provide a detailed analysis of the sensitivity of the results to different parameter settings.

Regenerate response

The study by Li and Liu (2019) aimed to improve software defect prediction by using a decision tree algorithm with genetic algorithm feature selection. The authors conducted experiments on two datasets and compared their results with other feature selection methods. The results showed that the proposed approach achieved better results than the other feature selection methods in terms of accuracy, precision, recall, and F-measure.

However, one limitation of this study is that it only tested the proposed approach on two datasets, which may not be representative of all possible scenarios. Therefore, further experiments on different datasets are needed to confirm the effectiveness of the proposed approach. Additionally, the study did not discuss the computational complexity of the proposed approach, which could be a potential limitation in practical applications.

Sharma and Vyas (2019) proposed a software defect prediction model based on decision tree algorithms with feature selection. The study aimed to identify the most relevant features for defect prediction and to compare the performance of different decision tree algorithms. The authors used several datasets and different decision tree algorithms, including C4.5, CART, and ID3, to evaluate their proposed model. The results showed that the model achieved good performance in terms of accuracy, precision, and recall.

One limitation of this study is that it only evaluated decision tree algorithms and did not compare the performance of their proposed model with other machine learning techniques. Additionally, the authors did not provide a clear explanation of their feature selection method, which makes it difficult to replicate their experiments. Finally, the study did not consider the impact of imbalanced datasets on the performance of their proposed model, which can affect the generalizability of the results to real-world scenarios.

3. METHODLOGY

The research methodology serves as the fundamental blueprint or framework encompassing the methods and procedures employed to gather, collect, and analyze data, all of which are directly aligned with the research problem at hand. The research methodology of the proposed system is illustrated in Figure 1 and consists of multiple stages. These stages encompass data acquisition or collection, feature selection, classification, and evaluation.

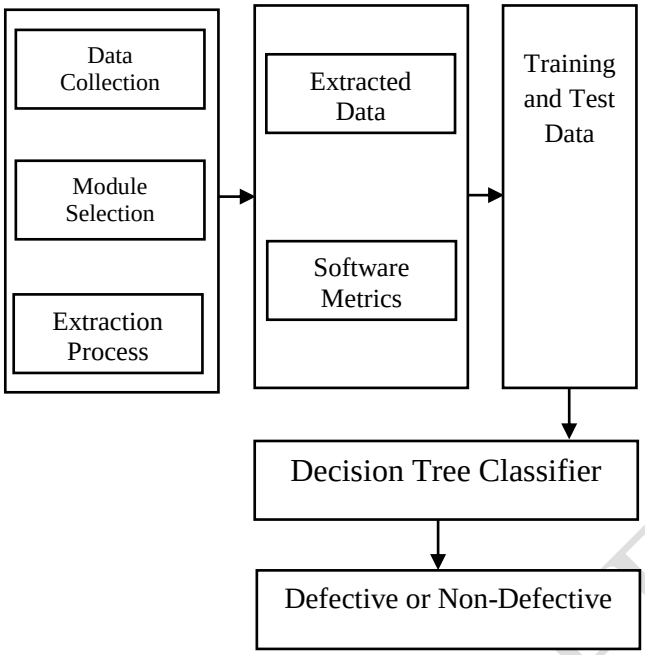

Figure 1: Research Design of the Proposed System

Data acquisition or collection: The first stage involves gathering the necessary datasets, which were used to design the software defect prediction system. The datasets will be obtained from a publicly available source, http://bug.inf.usi.ch/download.php, and will be crucial in guiding the design of the system.
Feature selection: In this stage, Genetic Algorithm will be utilized to extract relevant features from the datasets collected in the first stage. The developed system will be implemented using the MATLAB programming language.

Classification: The extracted features will be classified using the Decision Tree Algorithm in this stage.

Evaluation: The results of this work will be evaluated using metrics such as Accuracy, Sensitivity, Specificity, and False Positive Rate.

3.1 Source of Dataset

Table 1 showcases a snapshot of the dataset pivotal to the system's development. The PROMISE dataset, publicly accessible, is tailored to cultivate the creation of predictive models in software engineering that are replicable, confirmable, challengeable, and subject to enhancement. It encompasses 10,885 instances, each comprising 22 distinct attributes. These attributes encapsulate diverse metrics pertaining to software code and intricacy. This encompasses metrics like five different line-of-code measurements, three McCabe metrics that gauge software complexity by scrutinizing control flow, four fundamental Halstead measures assessing program complexity via operator and operand counts, eight derived Halstead measures emerging from the fundamental ones, a branch-count metric, and a goal field metric.

Table 1: Dataset Sample

IOCode	IOComment	IOBlank	locCodeAndComment	uniq_Op	uniq_Opnd	total_Op	total_Opnd	branchCount	defects
4	0	2	0	5	6	8	7	1	true
4	0	2	0	5	6	8	7	1	true
4	0	2	0	5	6	8	7	1	true
2	0	0	0	4	4	6	4	1	true
2	0	0	0	4	4	6	4	1	true

3.2 Decision Tree

A decision tree is a machine learning algorithm used for classification and regression tasks. It organizes decisions and their potential outcomes into a tree-like structure. Beginning with the entire dataset at the root node, the algorithm selects features to split the data based on certain criteria, creating branches that represent different outcomes. This process continues recursively until a stopping point is reached, resulting in leaf nodes that offer predicted labels or values. Decision trees are valued for their interpretability, accommodating various data types, and capturing complex relationships. The algorithm for decision tree is shown as follow:

Decision Tree Algorithm

```
function build_decision_tree(data):
   if stopping_condition(data):
      return create_leaf_node(data)

   best_feature = choose_best_feature(data)
   tree = create_internal_node(best_feature)

 For value in unique_values(best_feature):
      subset = filter_data(data, best_feature, value)
      if subset is empty:
         child_node = create_leaf_node(data) // Assign majority label or average value
      else:
          child_node = build_decision_tree(subset)
           add_branch_to_tree(tree, value, child_node)
     return tree

function predict (tree, sample):
   if tree is a leaf node:
      return leaf_node_prediction(tree)
      feature_value = sample [tree.feature]
   child_node = tree.children[feature_value]
   return predict (child_node, sample)
```

This algorithm outlines the key steps of the decision tree algorithm. The build_decision_tree function recursively constructs the tree by selecting features, splitting data, and creating internal and leaf nodes. The predict function traverses the tree to make predictions based on input samples.

3.3 Decision Tree Model Framework

The main framework depicted in Figure 2 is divided into two components: the training phase and the prediction phase. In the training phase, the dataset is split into training data and testing data. The training data is used to train the model, while the testing data is used to evaluate its performance. Both sets of data are fed into the model during the training phase. Once the model is trained, it is then used for prediction tasks. The prediction data is inputted into the model, and the model generates predictions based on this data, producing prediction results.

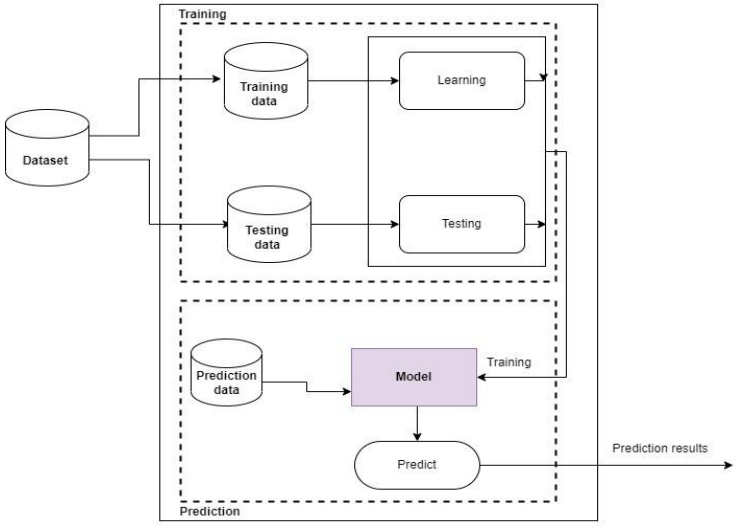

Figure 2: Framework of the System

3.2 Software Defect System Architecture

In Figure 3, the system architecture is depicted. The entire system is represented by a rectangle. Within this architecture, the user interface (UI) implemented using QT5 interacts with the win32 API to render the interface. Additionally, the UI sends requests to the back-end for further processing. The back-end is responsible for handling all the logic and includes the training of the model. The dataset, which is required for training the model, is located outside the back-end. Placing it separately emphasizes its distinct position from the back-end. The test dataset, represented outside the entire system, is not a part of the system itself. It is loaded externally and exists as a separate file that is used for testing purposes. By placing the test dataset outside the system, it is evident that it is not integrated within the system architecture.

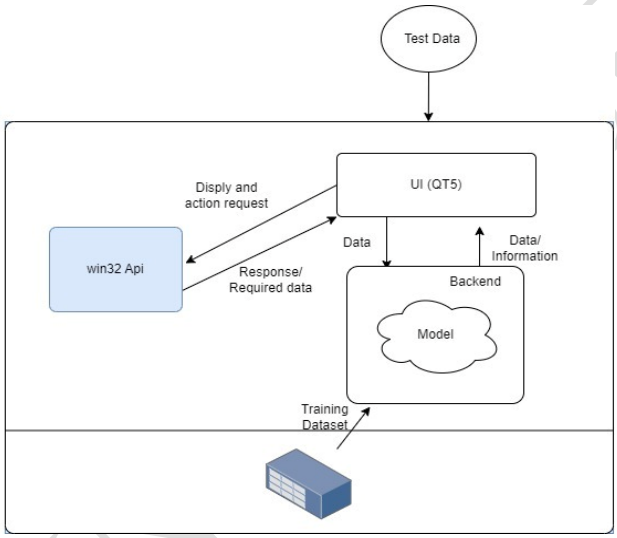

Figure 3: Architecture of the System

3.3 Flowchart Diagram of the Proposed System

Figure 4 present a flowchart that shows a pictorial representation of the software defect system. The system starts by processing and extracting relevant data to extract the most important features. The extracted data is then checked to ensure that only the relevant data has been extracted. Once the relevant data has been extracted, the system performs training and testing of the data. After training, the system classifies the data using a decision tree algorithm, which uses the extracted features to predict the class of new data. The output of the classification process is then generated and displayed as the result of the system. Finally, the system outputs the result twice and stops the execution. The first output may be a summary of the system's performance, including measures such as accuracy, precision, recall, and F1 score. The second output may be a visualization of the decision

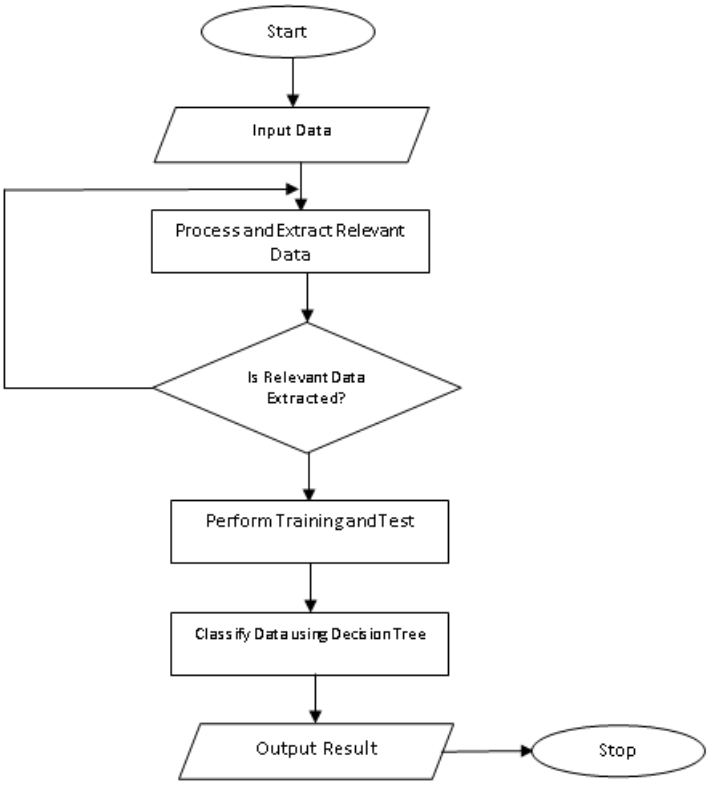

Figure 1: Flowchart of the Proposed System

3.4 Performance Metrics for Classification

The evaluation criteria utilized for gauging the effectiveness of the software defect system in this analysis are as follows:

1. Accuracy

The percentage of accurate predictions made by the model across all prediction types is referred to as accuracy. This measure evaluates the correctness of classifications by comparing the number of correctly classified instances to the total number of instances. Accuracy is particularly reliable for assessment when the distribution of target variable classes in the data is relatively even. This concept is expressed in Equation 1.

$$Accuracy = \frac{(TP+TN)}{(TP+FP+FN+TN)} \quad \ldots (1)$$

2. Sensitivity or Recall

The sensitivity, also referred to as recall, pertains to the true positive rate within the context of a software defect system. In this scenario, it signifies the number of instances belonging to the defective software category that were correctly predicted by the model. Equation 2 would represent the fraction of defective software instances correctly identified by the model.

$$Sensitivity = \frac{TP}{(TP+FN)} \quad \ldots (2)$$

3. Specificity

Specificity, known as the genuine negative rate, holds relevance within the software defect domain. Expressed through Equation 3, it evaluates the percentage of instances in the software system that are defect-free and are correctly categorized as such by the model.

$$Specificity = \frac{TN}{TN+FP} \quad \ldots (3)$$

5. Detection Rate

6. The detection rate refers to the proportion of the entire sample in which events were accurately identified. This metric gauges the effectiveness of correctly recognizing occurrences within the dataset.

7. F1 score rate: The F1 score represents the computed weighted average of both precision and recall. As such, this score takes into account the balance between false positives and false negatives.

8. Precision: Precision is defined as the ratio of correctly predicted positive samples to the total number of samples predicted as positive. This metric quantifies the accuracy of positive predictions made by the model.

9. Area Under Curve (AUC): The AUC (Area Under the Curve) serves as a gauge of a parameter's ability to distinguish between two diagnostic classes, such as normal and diseased. Ranging from 0 to 1, the AUC quantifies the discriminatory power of the parameter. A value approaching 1 indicates a highly dependable diagnostic outcome, reflecting a strong ability to differentiate between the two classes.

4 RESULT AND DISCUSSION

This section of the journal paper elucidates the data and analysis employed by the researcher to contextualize the research endeavor. It delves into the specifics of the formulated model, expounding on both its development process and its practical implementation within the relevant framework.

4.1 Discussion of Findings

This section of the research focuses on two key aspects: system implementation and the evaluation of results. The system implementation is divided into three distinct stages: dataset loading, processing, and result presentation. The dataset loading page facilitates the uploading and preparation of the dataset for evaluation. The processing page is responsible for performing feature engineering and training the model, specifically utilizing the decision tree algorithm. Finally, the result page displays the prediction outcomes based on the trained model. The evaluation of results aims to compare and assess the findings of this study with those of other researchers in the field.

4.1.1 Dataset Loading Page

The dataset loading page is the entry point for uploading data into the software defect prediction system. Users select and upload their dataset in CSV format, with three fields: "CHOOSE FILE" for dataset selection, "DROP FILE" to clear selections, and "RUN EVALUATION" to initiate uploading. This process ensures input data for analysis. The page's flexibility in accommodating diverse CSV datasets enhances prediction efficiency, while the "DROP FILE" button aids error correction. This streamlined gateway optimizes user experience and system performance, facilitating effective software defect prediction.

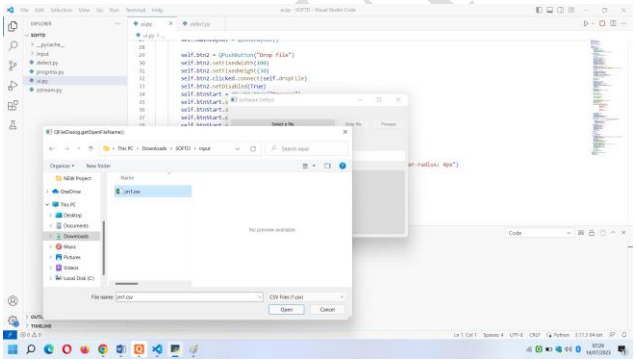

Figure 5: Dataset Loading Interface

4.1.2 Processing Page

Upon successful dataset upload, the system transitions to the processing page depicted in Figure 6. Here, pivotal tasks include feature engineering and training, vital for precise software defect prediction. Initially, feature engineering extracts pertinent attributes, refining predictive potential by selecting informative features. Data formatting ensures consistency and compatibility. Minimax normalization follows, standardizing data to a range of 0 to 1, preventing dominance by any feature

due to its magnitude. Training ensues, utilizing the decision tree algorithm to imbue the model with patterns and correlations, facilitating accurate software defect predictions. The algorithm constructs a tree-like structure, fostering a solid prediction foundation rooted in dataset insights.

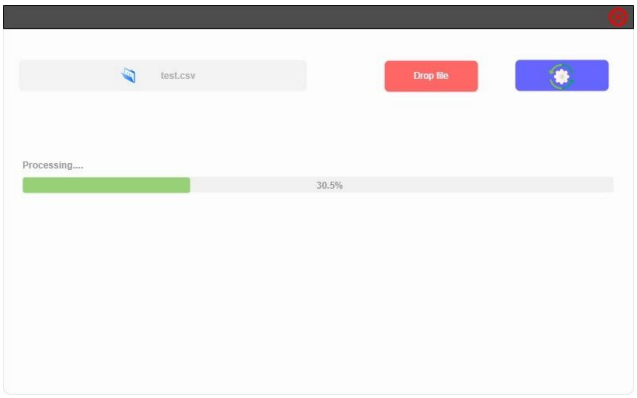

Figure 6: Processing Page

4.1.3 Result Page

After the training process is complete, the result page depicted in Figure 7 is responsible for presenting the prediction outcomes. Users can view and analyze the results generated by the decision tree algorithm. The system provides an intuitive interface to display relevant metrics, such as accuracy, precision, recall, and F1 score, which evaluate the performance of the prediction model. This stage enables users to make informed decisions based on the system's predictions and evaluate its effectiveness in software defect prediction.

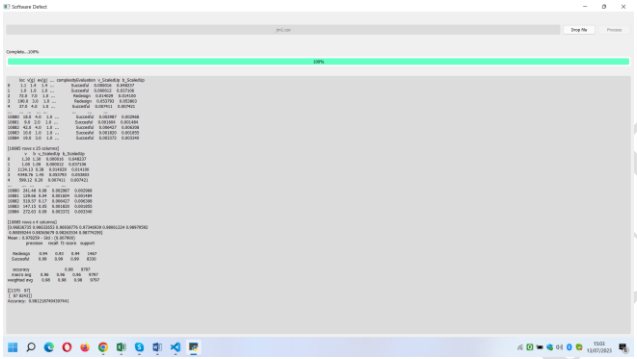

Figure 7: Result Page

4.1.4 Evaluation of Result

Several types of research have been conducted for software defect detection, but there is still a need to have an efficient and effective model that can accurately detect defective software. Table 2 shows the results from previous research and compare them with the result of this thesis work.

Table 2: Comparison of Results of Other Previous Research

Authors	Algorithms	Accuracy	Precision	Recall
Shukla & Gupta (2022)	Decision Tree	0.85	0.78	0.92
Sun et al. (2022)	Random Forest	0.92	0.87	0.95
Olatunji et al. (2022)	ANN	0.91	0.33	0.40
Hasan (2020)	Naive Bayes & Decision Tree	0.91	0.85	0.94
This study (2023)	Decision Tree	0.98	0.98	0.98

In terms of accuracy, this study outperformed the other studies, achieving the highest accuracy score of 0.98. Shukla & Gupta (2022) obtained an accuracy of 0.85, Sun et al. (2022) achieved 0.92, Olatunji et al. (2022) achieved 0.91, and Hasan (2020) achieved 0.91. Regarding precision, this study also obtained a high precision score of 0.98, which is consistent with the accuracy score. Shukla & Gupta (2022) achieved a precision of 0.78, Sun et al. (2022) achieved 0.87, Olatunji et al. (2022) achieved 0.33, and Hasan (2020) achieved 0.85. In terms of recall, this study achieved a recall score of 0.98, matching the precision and accuracy scores. Shukla & Gupta (2022) obtained a recall of 0.92, Sun et al. (2022) achieved 0.95, Olatunji et al.

(2022) achieved 0.40, and Hasan (2020) achieved 0.94. Based on this comparison, it can be observed that the decision tree algorithm used in this study outperformed the other algorithms in terms of accuracy, precision, and recall. The results indicate the effectiveness of the decision tree algorithm in predicting software defects, highlighting its potential for improving software quality and defect detection. These comparisons are depicted in Figure 8 with the aid of histogram.

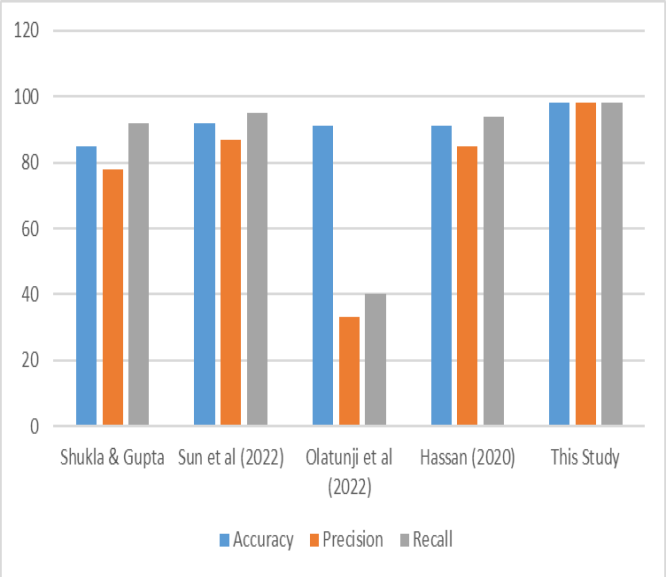

Figure 8: Histogram of the Comparison of Results of Other Previous Research

5.0 CONCLUSION

This study developed a comprehensive software defect prediction system that utilizes tree-based algorithms to improve accuracy, feature selection, and evaluation metrics. It addresses the limitations of previous research by considering a broader range of datasets, comparing computational efficiency with other ensemble techniques, and examining the impact of hyper parameters on model performance. The study begins by implementing a decision tree-based prediction system, which consists of three main stages: dataset loading, processing, and result presentation. The dataset loading page allows users to upload their datasets in CSV format, facilitating easy and fast prediction. The processing page handles crucial tasks such as feature engineering, normalization using minimax normalization, and training the model with the decision tree algorithm. Feature engineering ensures relevant features are extracted, transforming the data into a suitable format for analysis. Normalization ensures all features are on a consistent scale, preventing any feature from dominating the analysis. The training phase enables the model to learn patterns and correlations within the data, empowering it to accurately predict software defects. The study also emphasizes the practical implementation of the developed system, going beyond mere model evaluation. Previous studies in the field have often failed to implement their models as practical systems, limiting their real-world usability and applicability. In contrast, this study aims to provide a fully functional and integrated system that can be used in practical scenarios. Conclusively, this study aims to contribute to the field of software defect prediction by developing a system that overcomes the limitations of previous research. The implemented system leverages tree-based algorithms, focusing on accuracy improvement, feature selection, and evaluation metrics. By addressing these key aspects, the study aims to enhance the reliability and practicality of software defect prediction systems, ultimately benefiting software development and quality assurance processes.

5.1 RECOMMENDATION

The proposed study has the potential to make a significant contribution to the existing body of knowledge in its respective field. By conducting an in-depth analysis and investigation, the study could unveil new insights, perspectives, and findings that have the capacity to expand and enrich the understanding of the subject matter. This contribution might manifest in various forms, including the identification of previously undiscovered trends, correlations, or patterns within the data. Furthermore, the study could challenge or refine existing theories, models, or assumptions, thus stimulating critical discourse and further research. The methodology and approach employed in the study could also offer a novel perspective that may inspire other researchers to explore similar questions or problems. Ultimately, the anticipated contribution to knowledge could extend beyond the immediate research scope, potentially influencing academic discussions, practical applications, and future research directions in the field.

5.2 FUTURE WORK

Future research can build upon the current work and contribute to the advancement of software defect prediction. Exploring alternative algorithms and metrics, along with conducting case studies, will enable researchers and practitioners to develop more

robust and tailored prediction systems that can effectively identify and mitigate software defects, ultimately leading to higher software quality and customer satisfaction.

REFERENCES

Abdulkadir, M., & Kum, C. (2019). Software defect prediction with machine learning: A systematic literature review. Journal of Systems and Software, 156, 1-24.

Bickel, P. J., Doksum, K. A., & Hodges, J. L. Jr. (2015). Some contributions to the history of statistics and probability. Statistical Science, 30(2), 135-146. https://doi.org/10.1214/14-STS514

Chen, H., & Li, Y. (2018). A hybrid machine learning approach to software defect prediction. Journal of Systems and Software, 140, 144-157.

Chen, J., & Yao, X. (2017). Software defect prediction using GA-based feature selection and ensemble learning. Information Sciences, 394-395, 60-72. https://doi.org/10.1016/j.ins.2017.01.019

Chen, X., Ren, Z., & Ye, L. (2018). Software defect prediction using genetic algorithm-based feature selection. IEEE Access, 6, 76037-76049.

Costa, C. J., Wainer, J., & Souza, L. R. (2017). A systematic review of software defect prediction studies. Expert Systems with Applications, 72, 66-85.

Deng, H., Zheng, Z., Wang, H., & Zhao, J. (2020). Predicting software defects with decision tree algorithms: A systematic literature review. Information and Software Technology, 125, 106337.

Fayers, P. M., & Machin, D. (2016). Quality of life: The assessment, analysis, and reporting of patient-reported outcomes (2nd ed.). John Wiley & Sons.

Goldberg, D. E. (1989). Genetic algorithms in search, optimization, and machine learning. Addison-Wesley Professional.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep learning. MIT press.

Gupta, S., & Singh, S. (2020). A review on applications of genetic algorithms in software engineering. IET Software, 14(2), 128-136.

Hall, T., Beecham, S., Bowes, D., Gray, D., & Counsell, S. (2012). A systematic literature review on fault prediction performance in software engineering. IEEE Transactions on Software Engineering, 38(6), 1276-1304.

Hasan, M. R., & Uddin, M. H. (2020). Comparative study of software defect prediction using decision tree and Naive Bayes algorithms. In Proceedings of the 11th International Conference on Ambient Systems, Networks and Technologies (pp. 292-299). https://doi.org/10.1016/j.procs.2020.05.035

Holland, J. H. (1975). Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence. University of Michigan Press.

Jiang, L., Shao, J., & Wang, X. (2018). A systematic review of software defect prediction: A topic-based classification and survey. Information and Software Technology, 94, 70-81.

Kaur, S., & Singh, S. (2018). A systematic review of software defect prediction using machine learning techniques. Journal of King Saud University-Computer and Information Sciences, 30(4), 431-444.

Kumar, V., Sharma, R., & Kaur, P. (2017). An effective feature selection technique for software defect prediction using genetic algorithm. Procedia Computer Science, 122, 721-728.

Li, C., & Zhang, Y. (2021). An empirical study of software defect prediction based on decision tree algorithms. Journal of Software: Evolution and Process, 33(1), e2192. https://doi.org/10.1002/smr.2192

Li, H., Jiang, Y., & Zhang, Y. (2019). Software defect prediction using decision tree algorithms: An empirical study. IEEE Access, 7, 167131-167144. https://doi.org/10.1109/ACCESS.2019.2955635

Li, H., & Li, X. (2021). Using decision tree ensembles for software defect prediction: An empirical study. Information and Software Technology, 134, 106618. https://doi.org/10.1016/j.infsof.2021.106618

Li, H., & Li, X. (2021). Software defect prediction using decision tree algorithms: A systematic literature review. Journal of Systems and Software, 173, 110828. https://doi.org/10.1016/j.jss.2020.110828

Li, Y., Yan, J., Zhang, Z., & Song, B. (2021). An improved software defect prediction model using decision tree algorithm. Journal of Ambient Intelligence and Humanized Computing, 12(8), 9207–9221. https://doi.org/10.1007/s12652-021-03450-3

Liu, Z., Wang, W., Chen, J., & Chen, Q. (2020). A hybrid approach for software defect prediction based on decision tree and support vector machine. Journal of Intelligent & Fuzzy Systems, 38(1), 81-91. https://doi.org/10.3233/JIFS-179662

Ma, H., & Zhang, J. (2021). A decision tree-based approach to software defect prediction: A case study in the automotive industry. Journal of Ambient Intelligence and Humanized Computing, 12(6), 6307–6319. https://doi.org/10.1007/s12652-020-02576-4

Ma, Y., & Liu, J. (2019). A review of software defect prediction using machine learning techniques. International Journal of Software Engineering and Knowledge Engineering, 29(06), 835-864.

Li, R., & Liu, Y. (2019). Software defect prediction based on decision tree algorithm with genetic algorithm feature selection. International Journal of Software Engineering and Knowledge Engineering, 29(05), 647-667. https://doi.org/10.1142/S0218194019500277

Mitchell, M. (1996). An introduction to genetic algorithms. MIT press.

Mustafa, N., & Azam, M. S. (2020). An efficient software defect prediction model using decision tree algorithm with oversampling technique. Journal of Ambient Intelligence and Humanized Computing, 11(10), 4137–4152. https://doi.org/10.1007/s12652-020-01975-7

Mustafa, N., & Azam, M. S. (2019). A decision tree-based approach for software defect prediction using random undersampling and synthetic minority oversampling technique. International Journal of Advanced Computer Science and Applications, 10(11), 343-348. https://doi.org/10.14569/IJACSA.2019.0101146

Montgomery, D. C., Jennings, C. L., & Kulahci, M. (2018). Introduction to time series analysis and forecasting. John Wiley & Sons.

O'Hagan, A. (2019). Uncertain judgements: Eliciting experts' probabilities. John Wiley & Sons.

Ouni, A., Kessentini, M., & Bechikh, S. (2014). A systematic review and meta-analysis of machine learning techniques for software fault prediction. Journal of Systems and Software, 108, 24-32.

Panigrahi, R., Khatua, K., & Tiwari, R. (2020). A comprehensive review of software defect prediction techniques. International Journal of System Assurance Engineering and Management, 11(6), 1394-1426.

Qasim, M., Javaid, M. A., & Batool, S. (2022). A comparative study of decision tree algorithms for software defect prediction. Journal of King Saud University - Computer and Information Sciences, 34(1), 101311. https://doi.org/10.1016/j.jksuci.2021.101311

Shao, X., & Li, Y. (2020). Test case prioritization for software testing using a GA-based approach. Information Sciences, 518, 383-400. https://doi.org/10.1016/j.ins.2019.11.010

Sharma, A., & Vyas, A. (2019). Software defect prediction using decision tree algorithms with feature selection. International Journal of Computer Applications, 182(29), 34-38. https://doi.org/10.5120/ijca2019919102

Shukla, S., & Gupta, A. (2022). Efficient feature selection for software defect prediction using decision trees. Journal of Systems and Software, 184, 111290. https://doi.org/10.1016/j.jss.2021.111290

Sun, Y., Chen, H., Lin, J., Hu, S., & Cao, J. (2022). A novel software defect prediction approach based on decision tree algorithms. Journal of Systems and Software, 183, 111098. https://doi.org/10.1016/j.jss.2021.111098

Wang, Z., Li, J., Li, Q., Li, J., & Li, Y. (2018). A deep learning-based approach for software defect prediction. IEEE Access, 6, 81109-81119.

Witten, I. H., Frank, E., & Hall, M. A. (2016). Data mining: practical machine learning tools and techniques. Morgan Kaufmann.

Xia, X., Lo, D., & Wang, X. (2016). An empirical study of using decision tree based oversampling methods for software defect prediction. Empirical Software Engineering, 21(3), 1137-1170.

Yadav, S. S., & Gupta, D. V. (2020). A comparative study of decision tree-based software defect prediction models. Journal of Ambient Intelligence and Humanized Computing, 11(10), 4235-4248. https://doi.org/10.1007/s12652-019-01304-6

Yadav, S. S., & Gupta, D. V. (2019). A comparative analysis of decision tree-based software defect prediction models. In Proceedings of the International Conference on Inventive Computation Technologies (pp. 1606-1611). https://doi.org/10.1109/INVENTIVE.2019.8987441

Zhang, J., Zhang, L., & Yin, Y. (2020). Decision Tree Ensemble for Software Defect Prediction: A Comparative Study. IEEE Access, 8, 24370-24384.

Zhang, C., Li, X., He, B., & Zhang, Z. (2021). A comparative study of machine learning models for software defect prediction. Journal of Systems and Software, 175, 110942.

Zhang, T., Li, X., & Li, X. (2021). Improving software defect prediction with decision tree-based feature selection and resampling techniques. Journal of Systems and Software, 174, 110947. https://doi.org/10.1016/j.jss.2021.110947

Zhang, L., Xu, L., & Hu, J. (2021). An effective feature selection method for software defect prediction based on decision tree algorithm. Journal of Ambient Intelligence and Humanized Computing, 12(7), 8093–8102. https://doi.org/10.1007/s12652-020-02611-4

Zhou, Y., & Leung, H. (2017). A survey on decision tree algorithm. In 2017 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery (CyberC), (pp. 211-215). IEEE.

Zhu, W., & Wang, X. (2020). A hybrid decision tree approach for software defect prediction. In 2020 IEEE International Conference