Journal of Theoretical and Applied Information Technology
15th October 2024. Vol.102. No. 19

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

A HYBRID CLASSIFIER MODEL - DR-XA FOR DEFECT PRIORITIZATION

[1*]R.ADLINE FREEDA,[2] DR.P.SELVI RAJENDRAN

[1*]Research scholar, CSE Department, Hindustan University, Padur, Chennai, Tamil Nadu, India.
[2]Professor, CSE Department, Hindustan University, Padur, Chennai, Tamil Nadu, India.
[1*]Corresponding Author Email Id: adlinefreeda2024@gmail.com

ID55494 Submission	Editorial Screening	Conditional Acceptance	Final Revision Acceptance
04-09-2204	05-09-2024	22-09-2024	05-10-2024

ABSTRACT

Fault prioritization in software testing involves determining the sequence in which identified faults should be addressed. Effective fault prioritization is crucial in software development and testing as it helps allocate resources efficiently and ensures that the most critical issues are resolved first. The criteria for prioritization may vary so by addressing the most serious flaws promptly and allocating resources effectively, software quality can be significantly enhanced. Machine learning algorithms offer powerful tools for fault prioritization by leveraging the complexity of the problem and the available data. Common machine-learning approaches for prioritization include classifier models such as Decision Trees, Random Forests, and XGBoost. This research compares the performance of these different classifier models with the proposed DR-XA hybrid prediction model. The DR-XA model, which incorporates advanced techniques for handling unbalanced data and improving prediction accuracy, has been evaluated in the context of fault prioritization. The experimental analysis demonstrates that the DR-XA hybrid model surpasses existing classifier models in prioritization accuracy, achieving superior results compared to current prioritization techniques.
Keywords: *Defect Prioritization, Software Testing, Hybrid classifier model, Prediction, Machine learning.*

1. INTRODUCTION

Software engineering is a branch of computer science that focuses on creating, testing, and maintaining software systems [1]. Software engineering requires testing to ensure the program performs as intended. Software testing is an essential part of software engineering, whereby a system or application is assessed to identify and fix flaws or issues before end users have access to it. Its main objective is to ensure that the program runs as intended, satisfies all specifications, and produces incredibly high-quality outputs [2][3]. More customer happiness, longer-term cost savings, and improved product quality are the outcomes of effective software testing. Since testing is essential to the software development life cycle, software engineers who want to build robust and dependable software applications must be proficient in testing methodologies [4]. Test cases and scripts are assembled into test suites during testing to execute them one after the other or in a big batch.

A software application's functioning or a particular feature is carefully tested using test suites. By grouping associated test cases into a single unit, they facilitate thorough testing and speed up the testing process. It's significant that while test suites are essential for identifying and organizing test cases [5] [6][7], effective test case creation is as crucial. The test cases in a test suite should have clear specifications, cover a variety of scenarios, and be created to easily recognize faults or other issues. It is an essential component of software testing processes because it ensures that software achieves its quality and functionality requirements while boosting productivity and maintainability during the testing phase.

Fault localization and fault detection are two essential software testing procedures for locating and identifying faults or defects [8]. These steps are necessary to guarantee the program's reliability and high quality. The process of locating and finding defects in an application is known as fault detection. It could be coding errors; design issues, or deviations from the requirements of the software. Fault localization is a method for determining a software bug's exact location or root cause. It looks for the specific line

of code, or module, that is the problem. Effective fault detection and localization processes are essential for defect fixing in software development [9][10] By utilizing automated techniques and [10]methods, in particular, to find and quickly correct software problems, software quality can be improved and development costs can be reduced. Thorough documentation and close communication between the development and testing teams are also essential for efficient finding of defects and localization.

Assigning a priority to found faults or defects in a software application is the practice of "fault prioritization," also known as "defect prioritization" [8]. This is a vital step in the development and testing of software because it focuses on the most critical issues first. By arranging faults in order of importance, the limited resources are used effectively, and the most critical problems are addressed with priority [9]. Software testing defect prioritization can be automated with machine learning approaches. Based on several variables, including the problem's severity, the software it affects, and historical data, they can assist in prioritizing newly found flaws or concerns in order of importance. Neutral networks, K-Nearest Neighbours, Random Forests, Decision Trees, XGBoost, and Random Forests [10].

This paper addresses the critical issue of software defect prioritization, which is essential for efficient resource allocation and timely resolution of software issues. Traditional methods often fail to provide accurate prioritization, leading to inefficiencies and potential delays in the software development lifecycle.

A few machine learning algorithms that are commonly employed for assessing fault prioritization include trees. The major contributions of this paper include,

- To develop the proposed DR-XA hybrid prediction model, features from classifier methods such as Decision Trees, Random Forests, and XGBoost were combined.
- To regulate randomly generated actions, machine learning methods and functional libraries such as scikit-learn and NumPy were employed.
- To improve the overall predictive performance of multiple fault prioritization base models, the suggested DR-XA hybrid prediction model was developed.
- To evaluate and compare the proposed model's accuracy to that of the comparative models, measures like precision, recall, and f1 score were used.

The research on fault prioritization in software testing is essential for several reasons. In short, this research plays a crucial role in effectively managing software quality, cutting costs, using resources efficiently, and meeting user expectations in today's fast-moving development world. This research makes a valuable contribution to advancing fault prioritization techniques, boosting software quality, and streamlining testing processes throughout the software development lifecycle.

The rest of the paper is organized as follows: Section 2 presents the literature review which examines existing fault prioritization methods. The proposed methodology in Section 3 describes the proposed model, the prediction process, the pseudocode, and the evaluation parameters. The results and discussion in Section 4 present and interpret the findings of the study. The threats to validity are discussed in Section 5. Finally, the conclusion and future work is discussed in section 6.

2. LITERATURE REVIEW

In the software development life cycle testing an application under test is a vital phase. The test cases assist the tester in testing all functionalities of the application to find the defects [11][12]. Some defects might be crucial which affects the core functionality of the application. Such defects are prioritized using the prioritization techniques. There a various approaches existing and proposed for defect prioritization. A few of these approaches are investigated here:

Tabassum et al. [13] proposed the development and evaluation which employed four different classifiers. The random forest model outperformed the other models with better accuracy after the models' hyperparameters were tweaked. The severely unbalanced dataset was initially generated for the justified classification, and it was balanced using the SMOTE approach. For HBR prediction, Wu et al. [14]proposed "hbrPredictor," which blends interactive machine

learning with active learning. By employing they uncertainty sampling, they were able to increase the diversity and generalizability of training samples while significantly reducing the number of bug reports required to train a prediction model.

Uddin et al. [15]surveyed automated bug prioritization, reviewing publications from 2000 to 2015. Their findings indicated that most research focused on data classification using techniques such as Random Forest, Naive Bayes, Support Vector Machine, k-nearest Neighbors, and Decision Trees. The survey revealed that the Eclipse and Mozilla datasets were extensively utilized, and the three most frequently used evaluation criteria were F-measure, Precision, and Recall. This analysis highlighted bug priority prediction as a classification problem, utilizing well-known metrics and classifiers as discussed in their study. Choudhary et al. [16] created priority prediction models with neural networks and text categorization approaches. They discovered that linguistic, temporal, author-related, severity, product, and component features affect a bug's priority.

Tian et al.'s [17] machine-learning technique, DRONE enhances linear regression by applying a threshold strategy to handle unbalanced bug reports. Their machine learning-driven system leverages information from bug reports to generate priority-level suggestions. The proposed technique outperformed existing approaches, demonstrating superior average F-measure results on a dataset comprising over 100,000 Eclipse bug reports. Goyal et al. [18] proposed a novel approach for allocating software bug priority using supervised classification on clustered bug data. Their approach is based on research suggesting that categorizing previously grouped data can significantly improve performance. It involves clustering the data based on similarity before classification. It predicts the severity of software problems which determines the performance of classifiers when clustering is performed before classification.

Behl et al. [19] proposed a bug categorization method aimed at reducing the effort required to classify and evaluate issue reports. By utilizing term frequency-inverse document frequency (TF-IDF) weights along with Naive Bayes, they introduced a bug mining technique that effectively distinguishes between security and non-security issues. This approach enabled the application of the TF-IDF-based bug mining tool to enhance classification accuracy.
Kaur and Garg et al. [20] surveyed the clustering techniques used in software engineering data mining. It focused on various data mining techniques relevant to software engineering tasks, with a particular emphasis on clustering approaches. They concluded that each clustering technique is suited to address specific challenges. Punitha et al. [21] focused on prioritizing parts of the software enabling developers to concentrate on locating faulty modules. It aimed to assist developers through data mining techniques. It has reduced software development costs and enhanced software quality.

In recent years, researchers have explored various approaches to enhance software defect prediction. Notably, Ramesh et al. [22] addressed class imbalance problems by employing advanced computing techniques in [4]. Their hybrid model achieved improved results by balancing class distribution and mitigating multicollinearity. Similarly, Mohd et al. [[23] proposed an S-SVM model specifically designed for imbalanced datasets, demonstrating the effectiveness of machine learning techniques in defect prediction. In Adam et al. [24] introduced a hybrid approach combining Gaussian Naive Bayes, Bernoulli Naive Bayes, Random Forest, and support vector machine (SVM) classifiers. This ensemble model effectively tackled class imbalance issues. The work explored a hybrid statistical tool and artificial neural network (ANN) approach, outperforming traditional statistical learning algorithms. Lastly, Mondel et al.[25]delved into cross-project defect prediction using a hybrid multiple models transfer approach. Their work addressed the challenge of insufficient training data for new projects. Collectively, these studies contribute valuable insights to the field of software defect prediction, emphasizing the importance of hybrid models and tailored techniques .

3. PROPOSED METHODOLOGY

This paper introduces a novel hybrid classifier model, DR-XA, which combines decision tree, random forest, and XGBoost algorithms to enhance the prediction accuracy of defect prioritization. This hybrid approach aims to leverage the unique strengths of each algorithm: decision tree for its interpretability, random forest for its ability to handle overfitting, and XGBoost for its high predictive performance.

Defect prioritization involves sorting software defects into priority orders so that the most critical problems are addressed first. Defects are assessed based on various criteria, including severity, frequency, impact on users, commercial consequences, and difficulty of repair. Common techniques for prioritizing tasks include the MoSCoW approach, the Kano model, weighted scoring, and risk-based prioritization. Effective defect prioritization requires systematic problem discovery, evaluation, rating, and communication among stakeholders to ensure that the most important defects are resolved promptly and overall software quality and user satisfaction are maximized.

In this research, Figure 1 displays the proposed DR-XA hybrid Classifier model. The flow chart depicts how defects enter the pre-processing step, where they are cleaned and structured. During pre-processing, a large number of test cases are employed to train the classifier model. The proposed classifier uses the trained model to identify defects using defect priority criteria. Once classified, prediction is done where the defects are prioritized to address the most critical issues first, improving software stability and dependability. This process involves training the proposed model on the training data using the testing data to make priority predictions. It then evaluates the model's performance by calculating accuracy generating a classification report, and displaying the results.

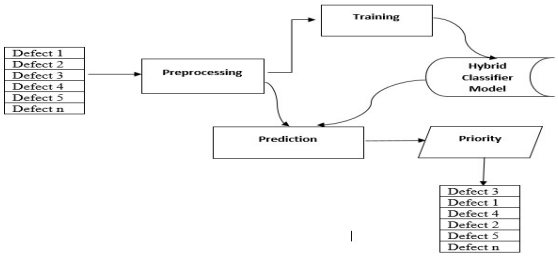

Figure 1: DR-XA Hybrid Classifier Model

The proposed approach predicts the priority as in the below-mentioned steps. The defects are given as input to the pre-processor in the order the defects are identified and received by the fault optimization model.

1. The process begins with loading and preprocessing the training and testing data.
2. The Hybrid Classifier Model is trained on the training data.
3. The testing data (x_test) is used to make predictions on priority using the proposed DR-XA Hybrid Classifier Model.
4. The model's performance is evaluated by calculating accuracy and generating a classification report.
5. The results are displayed.

Each step of the proposed approach is discussed with an illustration. A bug report of 5000 defects for an ATM application with attributes such as FaultType, Location, Severity, and Frequency.

A bug report B can be formalized as B=<t,p>
Where 't' is the textual information of 'B 'and 'p' is the assigned priority to 'B' The Hybrid model Classifier in the proposed approach predicts the priority as Low, Medium, and High based on the severity and frequency of occurrence of the bug.
A mapping f could be defined for the bug report B as shown below

$$f: B \rightarrow X, \quad X \in \{low, medium, high\},$$

where X is a suggested priority from a priority set (low, medium, high).

(i) Pre-processing: The data must normally be cleaned, transformed, and structured for it to be used effectively for training and evaluation [21]. The level of accuracy and structure of the data has a major impact on how well the machine learning models perform, making the preprocessing stage crucial. Depending on the dataset and the machine learning models used, the specific tasks required in preprocessing can change. Figure 2 shows the Preprocessing happens in the following steps:

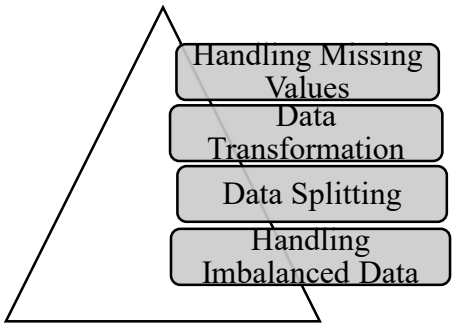

Figure 2: Pre-processing stages

(ii) Handling Missing Values: Multiple factors might cause data to have missing values. Users choose how to manage these during preprocessing, whether it be by eliminating the rows or columns with missing data or by replacing them using techniques like mean, median, and mode.

(iii) Data Transformation: Features can need to be scaled to be on the same scale, for example, by applying Z-score normalization or Min-Max scaling [16]. For algorithms like SVM or k-NN that are sensitive to the scale of the input characteristics, this is especially crucial.

(iv) Data Splitting: The data is separated into sets for training, testing, and sometimes validation to evaluate the model's performance on untested data,

(v) Handling Imbalanced Data: To balance the target variable classes, techniques like undersampling, oversampling, or the Synthetic Minority Over-sampling Technique (SMOTE) are used if one class has significantly fewer samples than the others.

(vi) Training: Training a machine learning model is the first stage in developing predictive systems. The model's ability to generalize to novel, untested data, the chosen technique, and the precision of the training data are all important factors in determining how well a machine learning application performs.

3.1 Proposed DR-XA Hybrid Classifier Model

This model is developed by combining the features of the Decision tree classifier, Random Forest, and XGBoost. The efficient hyperparameters from the existing models such as n_estimator from XGBoost and random_state from all the classifiers is considered. In ensemble learning techniques, like Random Forest and Gradient Boosting, the n_estimators hyperparameter is frequently employed. The number of base estimators (individual models or trees) to include in the ensemble is determined by this parameter. The ensemble's performance and behavior are strongly influenced by the value of n_estimators. In the proposed hybrid classifier model the value of n_estimator is set to 100.

In most of the machine learning algorithms and functions libraries like scikit-learn and NumPy, the random_state argument is a frequent input. It is used to regulate how randomly generated or reproducible certain actions are, particularly when randomness is a component of the algorithm [26]. The goal of using this hybrid technique is to improve overall predictive performance by utilizing the advantages of various base models. This model is trained with the training data for better predictions.

3.2 Prediction

The prediction process in the DR-XA model utilizes a hybrid approach, combining decision tree, random forest, and XGBoost algorithms to accurately determine the priority of software defects. This multi-algorithmic integration ensures robust prediction performance by leveraging the strengths of each individual model. By synthesizing the outputs from decision tree's simple interpretability, random forest's ability to handle overfitting, and XGBoost's superior predictive power, DR-XA achieves a more reliable and precise prioritization of software defects, enhancing the overall software quality management process.

3.3 Pseudocode for DR-XA Hybrid Classifier

```
#Import the necessary libraries
 #Import scikit-leam libraries
#Load and preprocess training and testing data
x_train, y_train_classification = load_and_preprocess_training_data()
x_test, y_test_classification = load_and_preprocess_testing_data()
#Create a list of base models
base_estimators = [] (decision tree', create decision_tree_model()),
(random_forest, create_random_forest_model()),
(xgboost, create_xgboost_model())
#Specify the meta-classifier
meta_classifier = create_meta_classifier()
#Create the Stacked Classification model
stacked_classifier=StackingClassifier(base_estimators=base_estimators,final_estimator-meta_classifier)
#Train the stacked classifier on the training data
stacked_classifier.fit(x_train, y_train_classification)
#Make predictions using the stacked classifier on the test data y_pred_stacked_classification = stacked_classifier.predict(x_test)
#Evaluate the stacked classification model
Accuracy_stacked_classification=calculate_accuracy (y_test_classification, y_pred_stacked_classification)
report_stacked_classification-generate_classification_report(y_test_classification, y_pred_stacked_classification)
#Display the accuracy and classification report
print("Accuracy (Stacked Classification):", accuracy_stacked_classification)
print("Classification Report (Stacked Classification):")
print(report_stacked_classification)
```

3.4 Evaluation Parameters

Often employed assessment measures in classification tasks are precision, recall, and F1 score. Especially in instances of binary classification, where there are positive and negative classes, they offer insights into several facets of a model's output. The outline and equations (Eq. 1-3) of each metric are stated below:

- Precision: The precision measure is the ratio of the model's total number of accurate predictions to the true positive predictions. The precision of the positive predictions is measured by it.

$$Precision = \frac{True\ Positives}{True\ Positives + fals\ \ Positives} \quad (1)$$

- Recall: Recall is the ratio between true positive predictions and real positive cases in the dataset. The model's ability to explain each positive example is evaluated.

$$Recall = True\ \frac{Positives}{True\ Positives + fal\ \ negatives} \quad (2)$$

When a model has a high recall rate, it can accurately identify a significant percentage of true positive occurrences[27]. Recall becomes critical when the cost of false negatives is high.

- F1 Score: We refer to the F1 score as the harmonic mean of recall and precision. Because false positives and false negatives are taken into consideration, recall and precision are balanced.

$$F1\ Score = 2 \times \frac{Precision \times Recall}{Precision + Recall} \quad (3)$$

4. RESULTS AND DISCUSSION

The implementation of the DR-XA model demonstrated significant improvements in the accuracy and reliability of defect prioritization compared to traditional methods. Experimental results showed that DR-XA outperformed individual classifiers and other hybrid models in terms of prediction accuracy, thus validating the effectiveness of the proposed approach. The integration of these algorithms provided a comprehensive and robust solution for the prioritization of software defects, marking a notable advancement in the field.

To evaluate the performance of the proposed model which is stacked with the features of Decision tree classifier, Random Forest, and XGBoost. The hybrid approach is also compared with the performance of mentioned algorithms. We had considered 5000 defects as test cases for an ATM application. Each test case includes the following data such as fault type, Location severity, and frequency. The accuracy of the Decision tree classifier, Random Forest, XGBoost, and the stacked model is evaluated for assigning priority for the defects using metrics such as precision, recall, and f1 score. Table 1 shows the data of performance metrics for each model. The comparison of classifier models based on precision, recall, and f1-score is shown in Figures 3, 4, and 5 respectively.

Table 1: Result Summary for Defect Prioritization

Classification Model	Priority	Precision	Recall	f1-score
Decision tree classifier	0	0.81	0.56	0.66
	1	0.86	0.88	0.87
	2	0.87	0.93	0.90
Random Forest	0	0.75	0.62	0.68
	1	0.85	0.88	0.87
	2	0.90	0.92	0.91
XGBoost	0	0.75	0.62	0.67
	1	0.85	0.87	0.86
	2	0.89	0.91	0.90
DR-XA Hybrid Classifier model (Proposed)	0	0.82	0.63	0.69
	1	0.88	0.86	0.87
	2	0.88	0.93	0.91

The information obtained from the above Table 1 shows the priorities assigned to the defects as (0,1,2). The accuracy of the priorities assigned is measured using metrics such as precision, recall, and f1_score. The table shows that the DR-XA hybrid classifier model performs well as the performance parameters for priorities 0,1 and 2 are better compared to other existing approaches. The precision metric for priority '0' is 0.82, for priority '1' is 0.88, and for priority '2' is .88 in the DR-XA classifier model. The recall metric is the priorities (0,1,2) under each classifier model is analyzed and the Decision tree and DR-XA classifier models output the same recall value

Journal of Theoretical and Applied Information Technology
15th October 2024. Vol.102. No. 19

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

under priority '2'. The f1_score is also calculated and the random forest and DR-XA classifiers perform the same as 0.91 for priority '2'. The Graphical representation is shown for the performance metrics of classifier models considered for comparison.

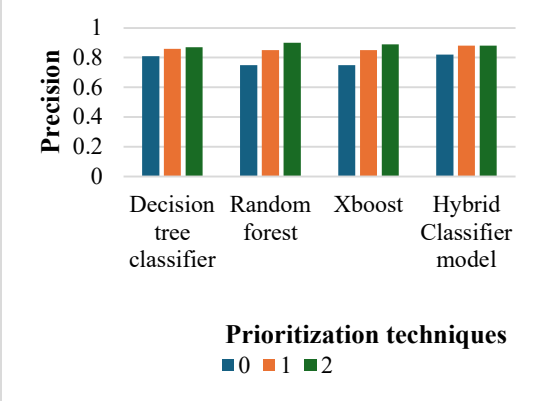

Figure 3: Precision comparison of classifier models

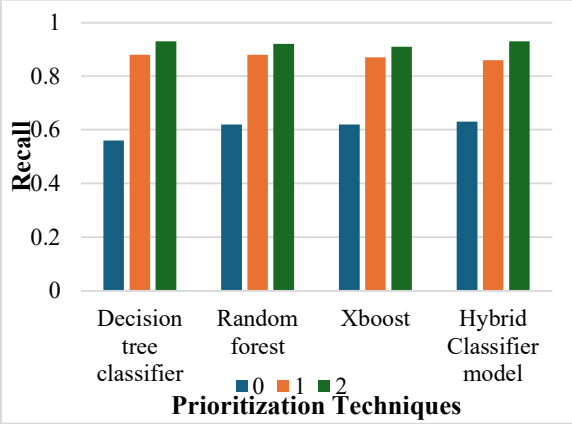

Figure 4: Recall comparison of classifier models

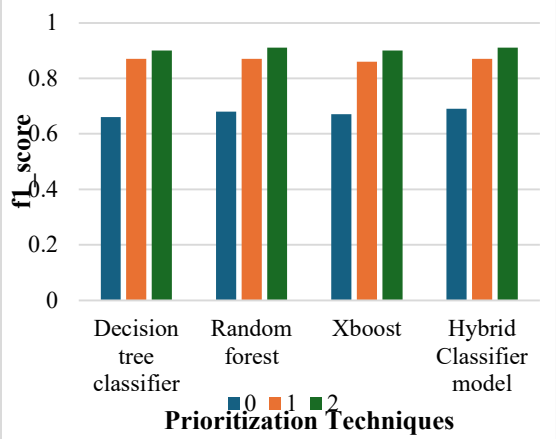

Figure 5: F1_score comparison of classifier models

Table 2: Statistical analysis of DR-XA over other approaches

Statistical Analysis	Decision tree classifier	Random Forest	XGBoost	DR-XA Hybrid Classifier (Proposed)
Mean	0.8425	0.800833	0.815833	0.800723
Median	0.855	0.875	0.87	0.843
Standard Deviation	0.0506548	0.145506	0.105525	0.04963
Minimum	0.75	0.56	0.66	0.54
Maximum	0.9	0.93	0.91	0.891

Table 2 shows the statistical analysis of the classifier models to evaluate their performance and accuracy of the classifier models. A standard deviation (std-dev), mean, median, max, and min statistical analysis were performed, and the results were compared with methodologies such as Decision tree, Random Forest, XGBoost, and Hybrid Classifier model and shown in Table 2. The standard deviation is used to measure variability from the mean value. The lower standard deviation indicates a lower deviation from the mean value. The DR-XA attains the lowest std-dev as 0.04963 of all the other approaches. Similarly, all the other measures like mean, median min, and max also attain lower values as mean=0.800, median=0.843, min=0.54, and max=0.891, which exhibits higher accuracy with low error.

Table 3: Accuracy Comparison of classifier models

Classification Model	Accuracy
Decision tree classifier	0.843
Random forest	0.855
XGBoost	0.859
DR-XA Hybrid Classifier Model (Proposed)	0.866

Table 3 compares the accuracy of four distinct classifier models for fault prioritization: Decision Tree, Random Forest, XGBoost, and the proposed

model. The Decision Tree Classifier has the lowest accuracy of all four models, at 0.843. The Random Forest model performs somewhat better, with an accuracy of 0.855. The XGBoost model performs even better, with an accuracy of 0.859. However, the proposed Hybrid Classifier Model exceeds all of them, with the greatest accuracy of 0.866. In terms of fault prioritization, the proposed model's higher accuracy indicates that it is the most successful in appropriately identifying and prioritizing failures. Its greater performance suggests a stronger ability to differentiate defects, resulting in more reliable prioritization.

The conclusions on fault prioritization in software testing are well-supported by several key points. First, the research builds on existing methods and incorporates machine learning techniques, making it more relevant to the field. Experimental results comparing the new DR-XA hybrid model with traditional classifiers add credibility to the findings. Using evaluation metrics like precision, recall, and F1 score offers a solid, objective way to measure performance. The study also tackles real-world challenges, focusing on better resource allocation and improved software quality. By acknowledging future research possibilities and limitations, the study further strengthens the validity of its conclusions. Finally, these factors underscore the significance and potential impact of the research on both theory and practice in software testing.

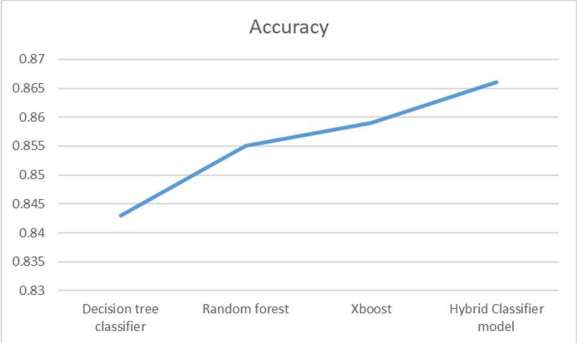

Figure 6: Accuracy comparison of classifier models

5. THREATS TO VALIDITY

Validity in implementation protocols is difficult to ensure because it is a goal rather than a guarantee [28], [29]. Validity analyses (VAs) seek to identify possible risks, discuss them, and choose the best course of action. In this context, threat validity analysis entails assessing a variety of metrics, including accuracy, precision, recall, and F1 score. Table 3 shows an accuracy comparison of the classifier models utilized in this research. Figure 6 provides a visual representation of the accuracy comparison across different models. The analysis of these measures aids in understanding the resilience and reliability of the proposed models for fault prioritization [30].

6. CONCLUSION AND FUTURE WORK

Fault prioritization in software testing faces several challenges that can limit its effectiveness. One key issue is the dependence on historical data, which might not accurately predict future fault patterns, especially in fast-changing software environments. Additionally, existing models often struggle with unbalanced data, where some fault types are underrepresented, affecting the accuracy of prioritization. However, there is significant potential for improvement. Integrating machine learning techniques can enhance predictive capabilities by analyzing fault patterns, while hybrid models that combine different prioritization approaches could offer more robust solutions. This research introduces a DR-XA hybrid prediction model designed to effectively predict defect priorities, using a dataset of 5,000 defects from an ATM application. By comparing the performance of Decision Trees, Random Forests, and XGBoost, this work found that this DR-XA hybrid model delivers significantly better accuracy. Evaluating the model using precision, recall, and F1 scores, we determined that the DR-XA Hybrid Classifier Model is the best option for accurate fault prioritization, resulting in more efficient defect management and improved software quality. Incorporating real-time data from ongoing development processes can help make prioritization more adaptive. Additionally, research into multi-objective optimization techniques could help balance priorities like code coverage and execution time, improving fault detection while reducing resource use. Addressing these limitations will be crucial for advancing fault prioritization in software testing.

CONFLICTS OF INTEREST

The authors declare no conflicts of interest.

DATA AVAILABILITY STATEMENT

Not Applicable

REFERENCES

[1] P. Terdchanakul, H. Hata, P. Phannachitta, and K. Matsumoto, 'Bug or not? Bug Report classification using N-gram IDF', in Proceedings - 2017 IEEE International Conference on Software Maintenance and Evolution, ICSME 2017, Institute of Electrical and Electronics Engineers Inc., Nov. 2017, pp. 534–538. doi: 10.1109/ICSME.2017.14.

[2] C. Beyer, G. Krempl, and V. Lemaire, 'How to select information that matters: A comparative study on active learning strategies for classification', in ACM International Conference Proceeding Series, Association for Computing Machinery, Oct. 2015. doi: 10.1145/2809563.2809594.

[3] F. Farooq and A. Nadeem, 'A Fault Based Approach to Test Case Prioritization', in Proceedings - 2017 International Conference on Frontiers of Information Technology, FIT 2017, Institute of Electrical and Electronics Engineers Inc., Jul. 2017, pp. 52–57. doi: 10.1109/FIT.2017.00017.

[4] S. Biswas, A. Bansal, P. Mitra, and R. Mall, 'Fault-Based Regression Test Case Prioritization', IEEE Trans Reliab, Sep. 2022, doi: 10.1109/TR.2022.3205483.

[5] Ra. Freeda and D. Rajendran, 'An Overview of Efficient Regression testing prioritization techniques based on Genetic algorithm'.

[6] A. Gonzalez-Sanchez, É. Piel, R. Abreu, H. G. Gross, and A. J. C. Van Gemund, 'Prioritizing tests for software fault diagnosis', in Software - Practice and Experience, Sep. 2011, pp. 1105–1129. doi: 10.1002/spe.1065.

[7] H. A. Alsattar, A. A. Zaidan, and B. B. Zaidan, 'Novel meta-heuristic bald eagle search optimisation algorithm', Artif Intell Rev, vol. 53, no. 3, pp. 2237–2264, Mar. 2020, doi: 10.1007/s10462-019-09732-5.

[8] S. Singhal, N. Jatana, A. F. Subahi, C. Gupta, O. I. Khalaf, and Y. Alotaibi, 'Fault Coverage-Based Test Case Prioritization and Selection Using African Buffalo Optimization', Computers, Materials and Continua, vol. 74, no. 3, pp. 6755–6774, 2023, doi: 10.32604/cmc.2023.032308.

[9] R. Mukherjee and K. S. Patnaik, 'A survey on different approaches for software test case prioritization', Nov. 01, 2021, King Saud bin Abdulaziz University. doi: 10.1016/j.jksuci.2018.09.005.

[10] S. Nayak, C. Kumar, and S. Tripathi, 'Enhancing Efficiency of the Test Case Prioritization Technique by Improving the Rate of Fault Detection', Arab J Sci Eng, vol. 42, no. 8, pp. 3307–3323, Aug. 2017, doi: 10.1007/s13369-017-2466-6.

[11] S. Asim, A. Shah, Z. Sultan, R. Abbas, S. N. Bhatti, and S. Asim, 'Analytical Review on Test Cases Prioritization Techniques: An Empirical Study', 2017. [Online]. Available: www.ijacsa.thesai.org

[12] C. Ni, X. Xia, D. Lo, X. Chen, and Q. Gu, 'Revisiting Supervised and Unsupervised Methods for Effort-Aware Cross-Project Defect Prediction'.

[13] N. Tabassum, A. Namoun, T. Alyas, A. Tufail, M. Taqi, and K. H. Kim, 'Classification of Bugs in Cloud Computing Applications Using Machine Learning Techniques', Applied Sciences (Switzerland), vol. 13, no. 5, Mar. 2023, doi: 10.3390/app13052880.

[14] X. Wu, W. Zheng, X. Chen, Y. Zhao, T. Yu, and D. Mu, 'Improving high-impact bug report prediction with combination of interactive machine learning and active learning', Inf Softw Technol, vol. 133, May 2021, doi: 10.1016/j.infsof.2021.106530.

[15] J. Uddin, R. Ghazali, M. M. Deris, R. Naseem, and H. Shah, 'A survey on bug prioritization', Artif Intell Rev, vol. 47, no. 2, pp. 145–180, Feb. 2017, doi: 10.1007/s10462-016-9478-6.

[16] P. A. Choudhary and S. Singh, 'Neural Network Based Bug Priority Prediction Model using Text Classification Techniques', International Journal of Advanced Research in Computer Science, vol. 8, no. 5, [Online]. Available: www.ijarcs.info

[17] Y. Tian, D. Lo, and C. Sun, 'DRONE: Predicting priority of reported bugs by multi-factor analysis', in IEEE International Conference on Software Maintenance, ICSM, 2013, pp. 200–209. doi: 10.1109/ICSM.2013.31.

[18] J. Kacprzyk, 'Advances in Intelligent Systems and Computing Volume 320 Series editor'. [Online]. Available: http://www.springer.com/series/11156

[19] Behl Diksha ; Handa Sahil ;Arora Anuja;, A Bug Mining Tool to Identify and Analyze Security Bugs using Naive Bayes and TF-IDF.

Institute of Electrical and Electronics Engineers, 2014.

[20] M. Kaur, S. K. Garg, M. Gobindgarh, and F. Sahib, 'Survey on Clustering Techniques in Data Mining for Software Engineering', 2014.

[21] K. Punitha and S. Chitra, 'Software Defect Prediction Using Software Metrics-A survey'. [Online]. Available: http://www.softwaretestingtimes.com

[22] R. Ponnala and C. R. K. Reddy, 'Software Defect Prediction using Machine Learning Algorithms: Current State of the Art'. [Online]. Available: www.solidstatetechnology.us

[23] M. Mustaqeem and T. Siddiqui, 'A hybrid software defects prediction model for imbalance datasets using machine learning techniques: (S-SVM model)', Journal of Autonomous Intelligence, vol. 6, no. 1, pp. 1–19, 2023, doi: 10.32629/jai.v6i1.559.

[24] H. Adam, A. Muhammad, and A. A. Aboaba, 'Design of a Hybrid Machine Learning Base-Classifiers for Software Defect Prediction', International Journal of Innovative Research and Development, Oct. 2022, doi: 10.24940/ijird/2022/v11/i10/oct22020.

[25] S. Mondal and R. Nasre, 'Hansie: Hybrid and consensus regression test prioritization', Journal of Systems and Software, vol. 172, Feb. 2021, doi: 10.1016/j.jss.2020.110850.

[26] A. S. Yaraghi, M. Bagherzadeh, N. Kahani, and L. Briand, 'Scalable and Accurate Test Case Prioritization in Continuous Integration Contexts', Sep. 2021, doi: 10.1109/TSE.2022.3184842.

[27] N. Gupta, A. Sharma, and M. K. Pachariya, 'Multi-objective test suite optimization for detection and localization of software faults', Journal of King Saud University - Computer and Information Sciences, vol. 34, no. 6, pp. 2897–2909, Jun. 2022, doi: 10.1016/j.jksuci.2020.01.009.

[28] M. G. Epitropakis, S. Yoo, M. Harman, and E. K. Burke, 'Empirical evaluation of Pareto efficient multi-objective regression test case prioritisation', in 2015 International Symposium on Software Testing and Analysis, ISSTA 2015 - Proceedings, Association for Computing Machinery, Inc, Jul. 2015, pp. 234–245. doi: 10.1145/2771783.2771788.

[29] S. Yoo and M. Harman, 'Regression testing minimization, selection and prioritization: a survey', Software Testing, Verification and Reliability, vol. 22, no. 2, pp. 67–120, Mar. 2012, doi: 10.1002/stvr.430.

[30] Grattan, N., da Costa, D. A., & Stanger, N. (2024). The need for more informative defect prediction: A systematic literature review. Information and Software Technology, 107456.