Towards an automated classification phase in the software maintenance process using decision tree

Sahar Alturki and Sarah Almoaiqel

Department of Software Engineering, King Saud University, Riyadh, Saudi Arabia

ABSTRACT

The software maintenance process is costly, accounting for up to 70% of the total cost in the software development life cycle (SDLC). The difficulty of maintaining software increases with its size and complexity, requiring significant time and effort. One way to alleviate these costs is to automate parts of the maintenance process. This research focuses on the automation of the classification phase using decision trees (DT) to sort, rank, and accept/reject maintenance requests (MRs) for mobile applications. Our dataset consisted of 1,656 MRs. We found that DTs could automate sorting and accepting/rejecting MRs with accuracies of 71.08% and 64.15%, respectively, though ranking accuracy was lower at 50%. While DTs can reduce costs, effort, and time, human verification is still necessary.

Subjects Artificial Intelligence, Software Engineering
Keywords Decision tree, Machine learning, Maintenance request, Software engineering, Software maintenance

Submitted 12 February 2024
Accepted 10 July 2024
Published 30 August 2024

Corresponding author
Sahar Alturki,
sahar.alturki1@gmail.com

Academic editor
Antonio Jesus Diaz-Honrubia

Additional Information and Declarations can be found on page 13

DOI **10.7717/peerj-cs.2228**

INTRODUCTION

Software engineering aims to produce reliable, maintainable, and high-quality software within a specified timeframe. One phase in the SDLC is the maintenance phase, which is both costly and time-consuming. Software maintenance involves correcting and improving software after its release (*Baqais, Alshayeb & Baig, 2013*; *Sharawat, 2012*). It is essential for the continued use of software, as neglected software becomes obsolete (*Stojanov & Stojanov, 2016*). Maintenance can account for up to 70% of project costs (*Ikram et al., 2020*). As software increases in size and complexity, maintenance becomes more challenging.

Automating parts of the maintenance phase can reduce associated costs, time, and effort. The IEEE model outlines seven phases in the software maintenance life cycle (SMLC), beginning with classification. This phase involves receiving requests, assigning maintenance types, priorities, and unique numbers. Subsequent phases include analysis, design, implementation, testing, acceptance testing, and delivery, as shown in Fig. 1 (*Ren et al., 2011*). Our study focuses on the classification phase, which involves collecting MRs, sorting them by type, ranking them by priority and severity, and deciding on acceptance or rejection. We evaluate the applicability and accuracy of DTs for this purpose (*Jo, 2021*).

Most research has focused on sorting, severity, priority, or acceptance/rejection of MRs using various machine learning techniques such as support vector machines (SVM), random forest (RF), linear SVC, KNN, naive Bayes (NB), and logistic regression. However,

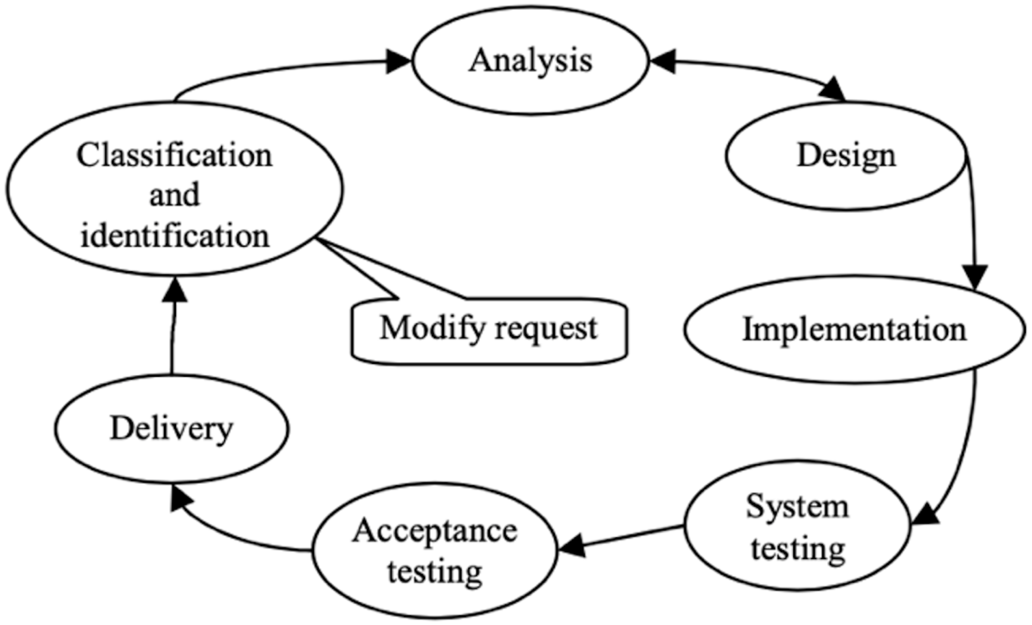


Figure 1 IEEE software maintenance life cycle model.
Full-size DOI: 10.7717/peerjcs.2228/fig-1

no research has comprehensively automated all activities in the classification phase. Our research aims to fill this gap by using DTs for sorting MRs based on maintenance types (corrective, adaptive, and perfective), ranking them by severity and priority, and deciding on acceptance or rejection. We focus on mobile applications, given their current prominence and need for rapid maintenance (*Phetrungnapha & Senivongse, 2019*).

This research addresses the following question: Is the DT learning technique suitable for sorting, ranking, and accepting/rejecting MRs with acceptable accuracy? Our study contributes to solving significant problems in the software maintenance process, such as reducing time, effort, and cost, and speeding up the maintenance process. Automating the classification process allows maintenance teams to focus on other phases, reduces human intervention, and directs future research toward further automation in the SMLC.

This article is divided into literature review, methodology, data collection, data processing, feature extraction, decision tree, experimental results, discussion, and conclusion.

LITERATURE REVIEW

Sorting maintenance requests

Machine learning techniques have been used to sort and categorize MRs. *Phetrungnapha & Senivongse (2019)* developed an approach to categorize user reviews into feature requests or bug reports using several machine learning techniques, including DT, linear SVC, KNN, NB, logistic regression, and ensemble methods. *Ciurumelea, Panichella & Gall (2018)* introduced the AUREA tool to help developers analyze user feedback and plan maintenance activities. *Ekanata & Budi (2018)* used machine learning to categorize user reviews automatically, finding logistic regression to be the most effective. *Levin & Yehudai (2017)* classified commitments in maintenance activities using J48, GBM, and RF, achieving an accuracy of 76%.

Otoom, Al-jdaeh Hammad & Hammad (2019) built a classifier to distinguish software bug reports into corrective or perfective reports using keyword frequency and classification algorithms, achieving an average accuracy of 93.1% with SVM. *Pandey et al. (2017)* analyzed bug reports using various algorithms, including RF and SVM, finding high performance with RFs and SVMs. *Ahmed, Bawany & Shamsi (2021)* introduced the CaPBug framework, using NLP and machine learning to classify and prioritize error reports, achieving class prediction accuracy of 88.78% with RF.

Ranking maintenance requests

Researchers have also addressed ranking MRs using machine learning techniques. *Srewuttanapitikul & Muengchaisri (2016)* proposed prioritizing software flaws based on severity, priority, and user reports. *Guzman, Ibrahim & Glinz (2017)* surveyed researchers and practitioners to rank and prioritize tweets for software development. *Ekanayake (2021)* proposed using the RAKE algorithm for keyword extraction and NB, DT, and logistic regression for prioritizing error reports. *Alenezi & Banitaan (2013)* predicted bug priority using DTs, NB, and RF, concluding that DTs and RFs outperformed NB.

Researchers have explored various machine learning techniques for ranking maintenance requests based on severity and priority. *Srewuttanapitikul & Muengchaisri (2016)* suggested using a natural language processing approach combined with an analytic hierarchy process to prioritize MRs by severity, priority, and the number of users reporting the same defect. *Guzman, Ibrahim & Glinz (2017)* conducted a survey of 84 software engineering practitioners to rank tweets for software development, emphasizing the importance of audience-based requirements engineering. *Ahmed, Bawany & Shamsi (2021)* introduced the CaPBug framework, which uses natural language processing and machine learning to classify and prioritize error reports into six categories and five priority levels. The CaPBug framework achieved class prediction accuracy of 88.78% with the RF classifier and priority prediction accuracy of 90.43%.

Ekanayake (2021) proposed a model using the RAKE algorithm to extract keywords from error reports, converting them into attributes for prioritizing MRs with NB, DT, and logistic regression. The model achieved logistic regression accuracy of 0.86, with DT and NB accuracies of 0.81 and 0.79, respectively. *Alenezi & Banitaan (2013)* suggested using machine learning algorithms to predict bug priority, concluding that DTs and RFs outperformed NB. *Tian et al. (2015)* proposed a multi-factor analysis approach for predicting bug report priority, achieving a relative improvement of 209% in the average F-measure. *Umer, Liu & Sultan (2018)* introduced an emotion-based approach for predicting bug report priority using natural language processing to identify emotional words and a supervised machine learning classifier. Their approach outperformed the latest technologies, improving the F-measure by more than 6%. *Ramay et al. (2019)* proposed a deep neural network-based approach for predicting bug report severity, which outperformed existing methods and improved the F-measure by 7.90%.

Accepting/rejecting maintenance requests

Various machine learning techniques have been used to predict the acceptance or rejection of MRs. *Nizamani et al. (2018)* proposed a NB polynomial approach for this task, achieving an accuracy of 89.25%. *Umer, Liu & Sultan (2019)* used sentiment analysis to predict approval, achieving 77.90% accuracy. *Cheng et al. (2021)* used deep learning for approval prediction, achieving 90.56% accuracy. *Nyamawe et al. (2020)* recommended refactorings based on feature requests, achieving 83.19% accuracy. *Nizamani et al. (2018)* proposed a NB polynomial approach for predicting the acceptance or rejection of improvement requests, using data from Bugzilla. Their method achieved an accuracy of 89.25%. *Umer, Liu & Sultan (2019)* developed a sentiment-based approach to predict the approval of enhancement reports, achieving 77.90% accuracy and a significant improvement in the F-measure to 74.53%.

Cheng et al. (2021) proposed a deep learning approach to predict the approval of enhancement reports, achieving 90.56% accuracy, 80.10% recall, and 85.01% F-measure. *Nafees & Rehman (2021)* used SVM for predicting the acceptance or rejection of improvement reports, comparing it with logistic regression and multinomial NB. Their study found that SVM outperformed other algorithms with high accuracy. *Nyamawe et al. (2020)* suggested a machine learning approach for predicting software refactorings based on feature requests, achieving 83.19% accuracy. *Arshad et al. (2021)* proposed a deep learning technique for predicting the resolution of enhancement reports, using Word2Vec and a deep learning classifier to learn semantic and grammatical relations between words. Their approach enhanced performance and demonstrated effective prediction accuracy.

METHODOLOGY

We used an experimental methodology, collecting data from records, documents, and help desk services of mobile applications. Statistical analysis was used to evaluate the efficiency of DTs for sorting, ranking, and deciding on MRs. The stages included data collection, data processing, feature extraction, model building, testing, and analysis, as shown in Fig. 2.

Collecting data

To collect data, we contacted several researchers and developers, seeking a dataset consisting of MR texts and their sorting, ranking, and acceptance/rejection classes. Unfortunately, we did not find a suitable dataset. Therefore, we turned to online data collection sites, such as Mendeley Data, which provide datasets used in scientific research. We used two datasets: the Commit dataset from *Levin & Yehudai (2017)* and the Pan dataset from *Al-Hawari, Najadat & Shatnawi (2021)*. The Commit dataset classifies MRs as perfective, adaptive, and corrective, while the Pan dataset categorizes user reviews into problem discovery, feature request, information seeking, and information giving. We reclassified the Pan dataset to match the Commit dataset's categories. The final dataset consisted of 1,656 MRs, as shown in Fig. 3. The demography of MRs is shown in Table 1. We split the dataset into 80% for training and 20% for testing, following common recommendations (*Joseph, 2022*; *Rácz, Bajusz & Héberger, 2021*).

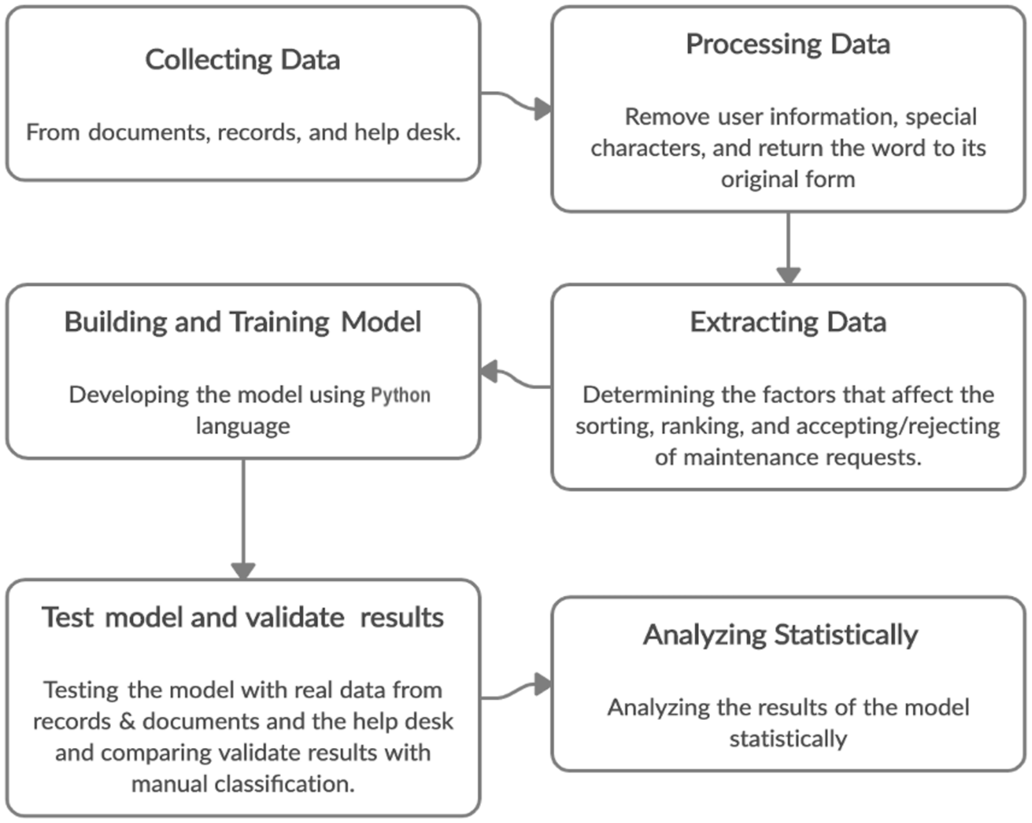
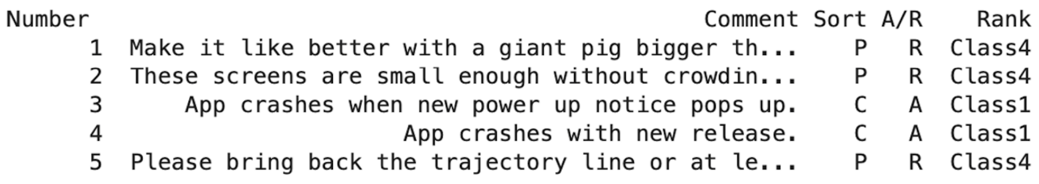

Collecting Data
From documents, records, and help desk.
Processing Data
Remove user information, special characters, and return the word to its original form
Extracting Data
Determining the factors that affect the sorting, ranking, and accepting/rejecting of maintenance requests.
Building and Training Model
Developing the model using Python language
Test model and validate results
Testing the model with real data from records & documents and the help desk and comparing validate results with manual classification.
Analyzing Statistically
Analyzing the results of the model statistically

Figure 2 **Stages of methodology.**

Full-size DOI: 10.7717/peerjcs.2228/fig-2



```
Number                                          Comment Sort A/R    Rank
      1  Make it like better with a giant pig bigger th...    P   R  Class4
      2  These screens are small enough without crowdin...    P   R  Class4
      3      App crashes when new power up notice pops up.    C   A  Class1
      4                      App crashes with new release.    C   A  Class1
      5  Please bring back the trajectory line or at le...    P   R  Class4
```

Figure 3 **Dataset Excel file.**

Full-size DOI: 10.7717/peerjcs.2228/fig-3

Processing data

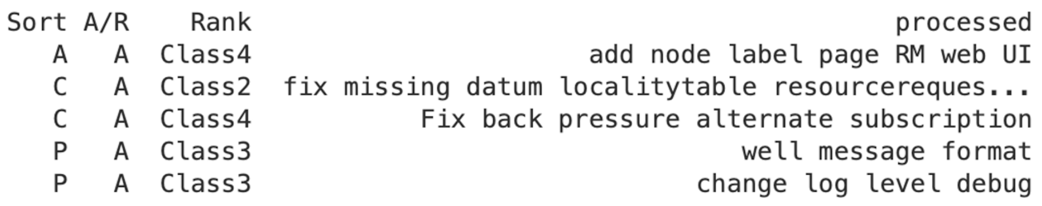
Before applying DTs, we processed the natural language to convert unstructured data into structured data. We removed links, user/application information, numbers, stop words, and punctuation. We then applied lemmatization to retrieve words in their normalized form, as shown in Fig. 4 (*Razno, 2019*; *Korenius et al., 2004*). The Natural Language Tool Kit (NLTK) library was used for lemmatization (*Bird, 2006*).

Extracting features

We used the term frequency-inverse document frequency (TF-IDF) technique for text feature extraction, which measures document relationships (*Zhang, Zhou & Yao, 2020*;

Table 1 **Demography of sort, rank, accept/reject MRs.**

MRs	Classes	Number of commits	Percentage
Sorting	Corrective	818	50%
	Perfective	486	29%
	Adaptive	352	21%
Ranking	Class 1 very important (Critical Severity and High Priority)	346	21%
	Class 2 (Critical Severity and Medium Priority) or (Normal Severity and High Priority)	230	14%
	Class 3 (Critical Severity and Low Priority) or (Normal Severity and Medium Priority)	421	25%
	Class 4 Not important (Normal Severity and Low Priority)	659	40%
Accepting or Rejecting	Accept	1,060	64%
	Reject	596	36%



```
Sort A/R    Rank                                     processed
   A   A  Class4                     add node label page RM web UI
   C   A  Class2  fix missing datum localitytable resourcereques...
   C   A  Class4           Fix back pressure alternate subscription
   P   A  Class3                                well message format
   P   A  Class3                               change log level debug
```

Figure 4 **The dataset after process natural language.**
Full-size DOI: 10.7717/peerjcs.2228/fig-4

Qaiser & Ali, 2018). The TF-IDF vectorizer library was used to extract features from the processed dataset.

Decision tree

We developed the DT using Python, which is suitable for machine learning and AI. We used the Scikit-learn library for supervised learning algorithms and statistical analysis (*Pedregosa et al., 2011*). The dataset was divided into 80% for training and 20% for testing. The DT was trained to sort, rank, and accept/reject MRs. We used the fit() method to train the model, taking the result of feature extraction (TF-IDF) and the desired column in the trained dataset. The workflow of the implemented DT is shown in the Fig. 5. The DT classified MRs into three classes: perfective, corrective, and adaptive. The model was trained for ranking MRs based on severity and priority and for accepting/rejecting MRs.

RESULTS

In this section, we present the results of applying DT to classification, divided into three subsections: sorting, ranking, and accepting/rejecting MRs.

Result of sorting MRs

When we applied the DT to sort MRs, as shown in Fig. 6, the accuracy was 71.08%, precision was 71.1%, and recall was 71.1%. The confusion matrix of sorting MRs is represented in Fig. 7. By using actual and predicted values, we found that 42 MRs were correctly classified

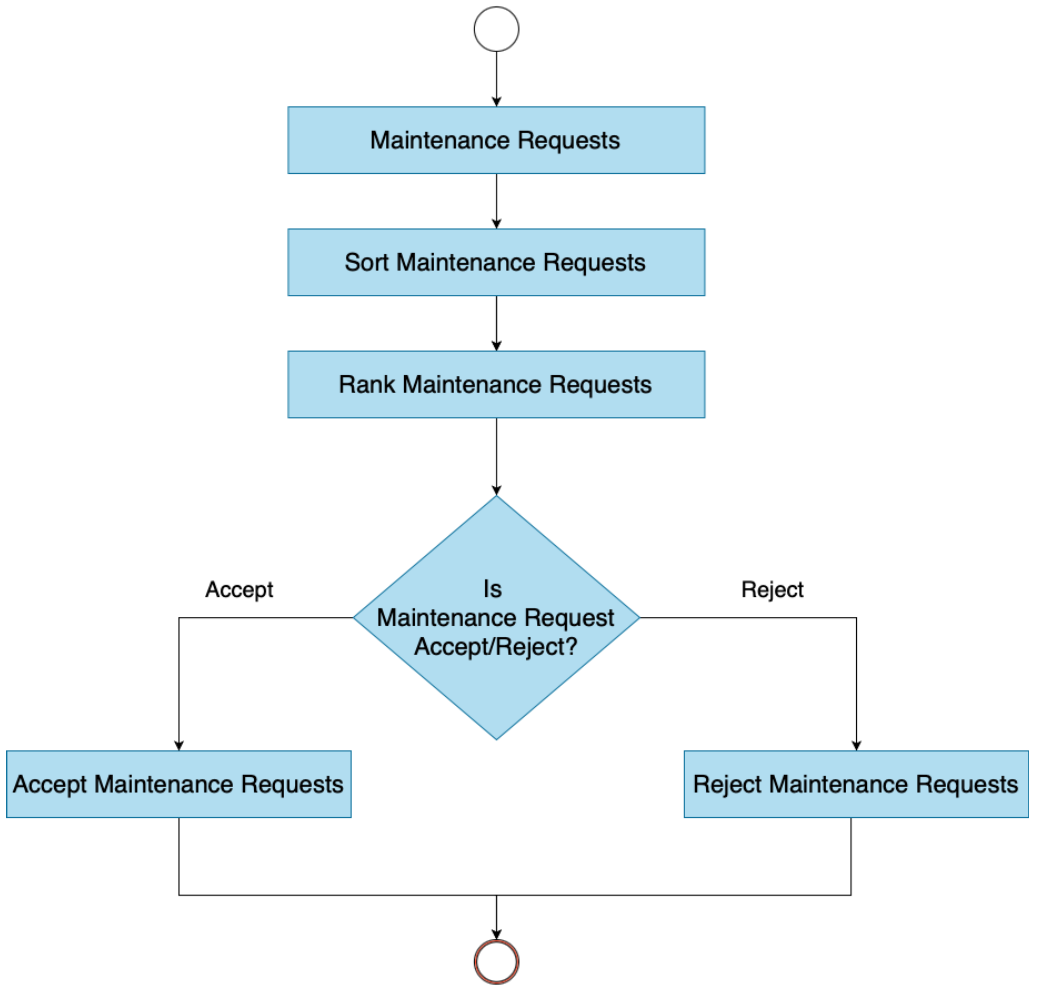

Maintenance Requests
Sort Maintenance Requests
Rank Maintenance Requests
Is Maintenance Request Accept/Reject?
Accept
Reject
Accept Maintenance Requests
Reject Maintenance Requests

Figure 5 **Workflow of decision tree.**

Full-size DOI: 10.7717/peerjcs.2228/fig-5

as adaptive, 136 as corrective, and 58 as perfective. The adaptive class represented 20.5% of the actual and 19.9% of the predicted classes, the perfective class represented 28.6% of the actual and 30.1% of the predicted classes, and the corrective class represented 50.9% of the actual and 50% of the predicted classes, as shown in Fig. 8.

Result of ranking MRs

When we applied the DT to rank MRs, as shown in Fig. 9, the accuracy was 50%, precision was 50%, and recall was 50%. The confusion matrix of ranking MRs is shown in Fig. 10. Using actual and predicted values, we obtained 35 MRs correctly classified as class 1, 11 as class 2, 29 as class 3, and 91 as class 4. Class 1 represented 22.6% of actual and 20.5% of predicted classes, class 2 represented 13.9% of actual and 10.8% of predicted classes, class 3 represented 23.8% of actual and 19.9% of predicted classes, and class 4 represented 39.8% of actual and 48.8% of predicted classes, as shown in Fig. 11.

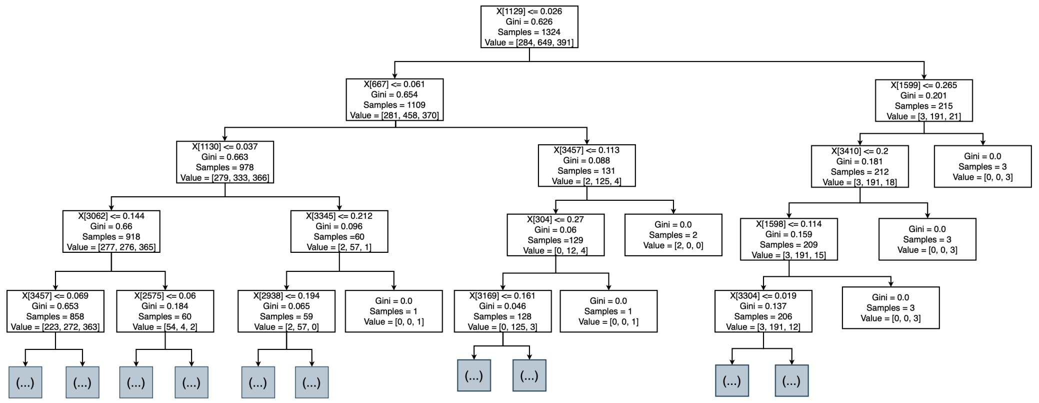


Figure 6 **Decision tree of sort MR.**

Full-size DOI: 10.7717/peerjcs.2228/fig-6

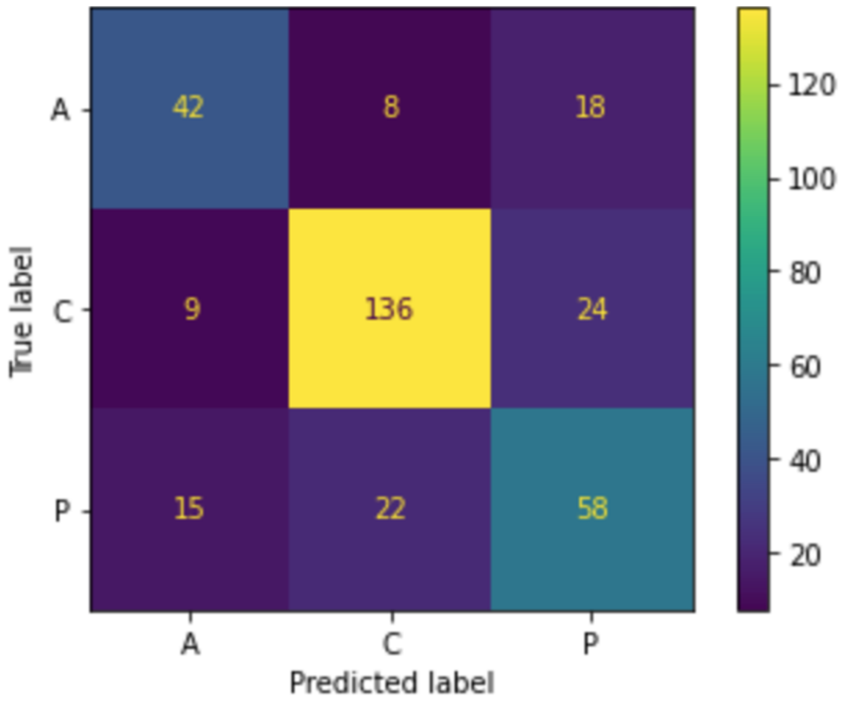


Figure 7 **Confusion matrix of sorting MR.**

Full-size DOI: 10.7717/peerjcs.2228/fig-7

Result of accepting/rejecting MRs

When we applied the DT to accept or reject MRs, as shown in Fig. 12, the accuracy was 64.15%, precision was 64.2%, and recall was 64.2%. The confusion matrix of accepting or rejecting MRs is shown in Fig. 13. By using actual and predicted values, we obtained 149 accepted MRs and 64 rejected MRs. The accepted MRs represented 66.6% of the actual and 59% of the predicted classes, while the rejected MRs represented 33.4% of the actual and 41% of the predicted classes, as shown in Fig. 14.

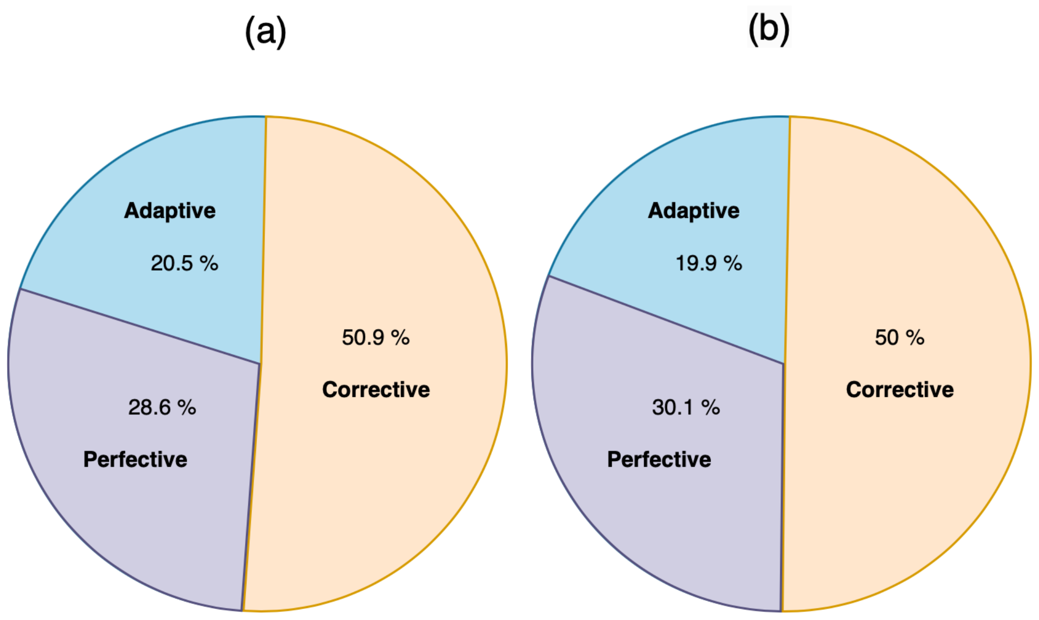


Figure 8 (A) The actual classes of sort MR. (B) The prediction classes of sort MR.
Full-size DOI: 10.7717/peerjcs.2228/fig-8

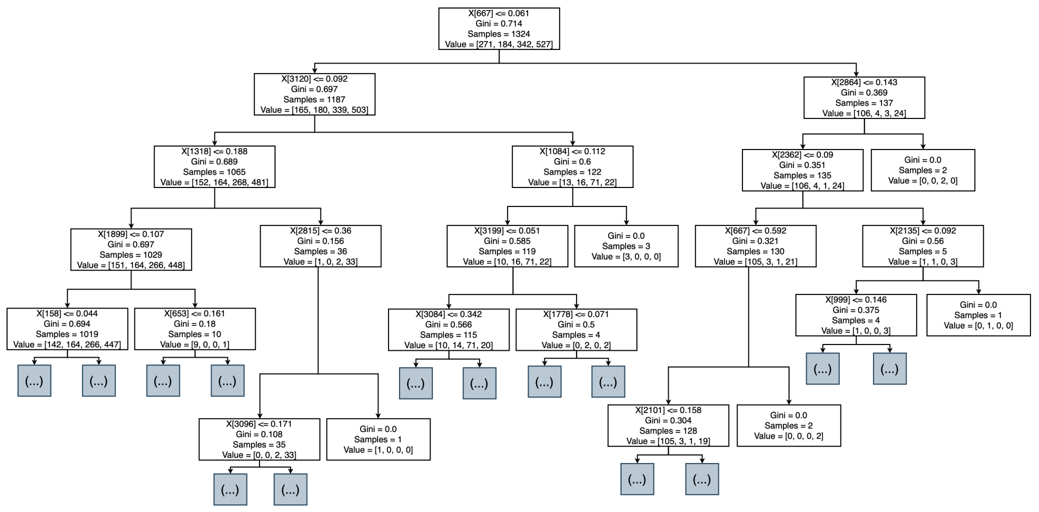


Figure 9 Decision tree of rank MR.
Full-size DOI: 10.7717/peerjcs.2228/fig-9

DISCUSSION

The purpose of this research was to evaluate the suitability and accuracy of the DT technique for automating the sorting, ranking, and acceptance/rejection of MRs. The experimental results indicate that DTs can be applied to sort and accept/reject MRs with acceptable accuracy, but their ranking accuracy is lower. The sorting accuracy was 71.08%, while the ranking accuracy was 50%, and the acceptance/rejection accuracy was 64.15%. In sorting

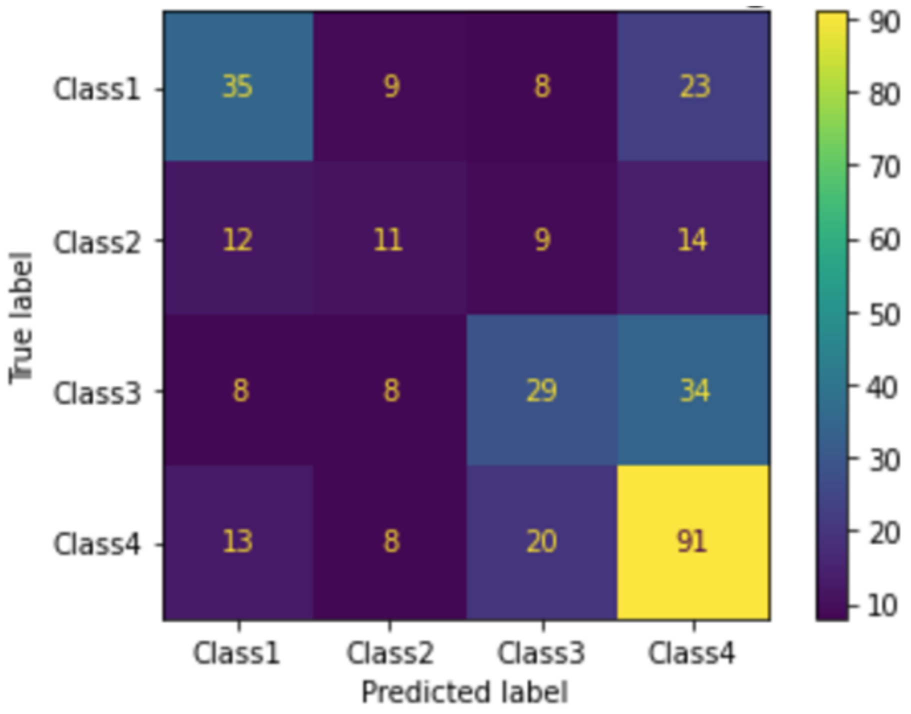


Figure 10 **Confusion matrix of ranking MR.**
Full-size DOI: 10.7717/peerjcs.2228/fig-10

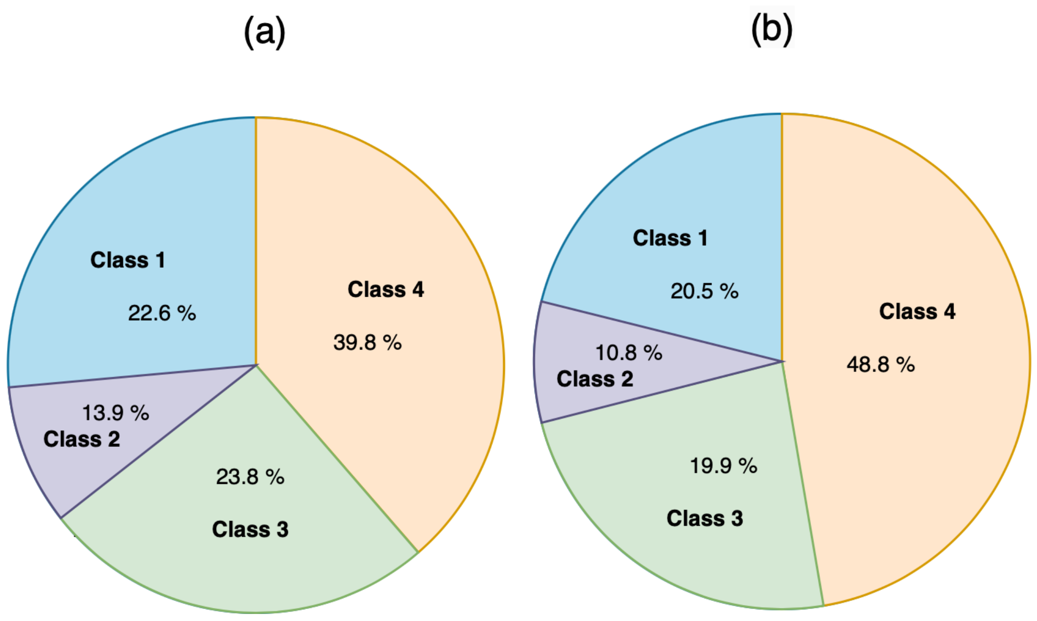


Figure 11 **(A) The actual classes of rank MR. (B) The predicted classes of rank MR.**
Full-size DOI: 10.7717/peerjcs.2228/fig-11

MRs using DT, 236 out of 332 MRs were correctly classified, while the rest were incorrectly classified. In ranking MRs using DT, 166 out of 332 MRs were correctly classified, while

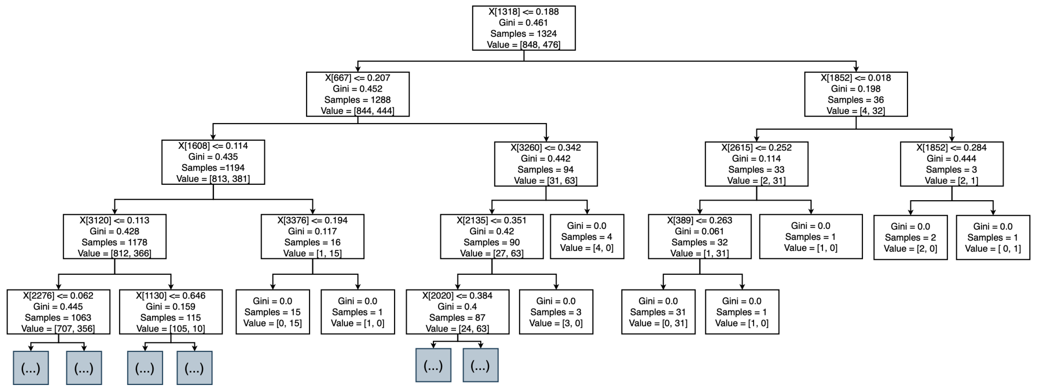


Figure 12 Decision tree of accept or reject MR.

Full-size DOI: 10.7717/peerjcs.2228/fig-12

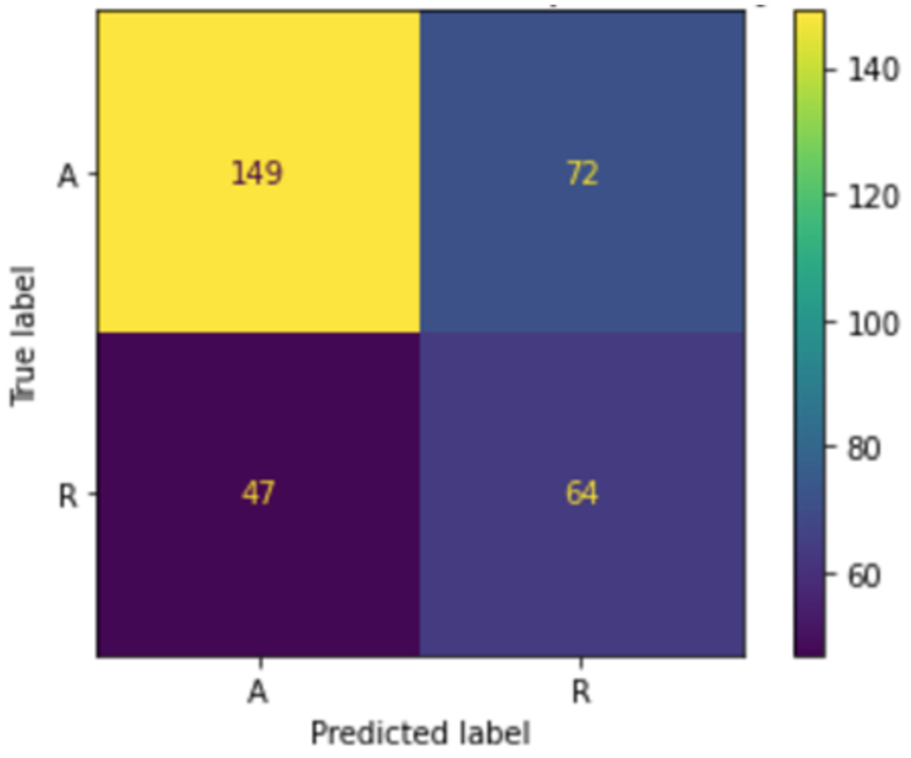


Figure 13 Confusion matrix of accepting or rejecting MR.

Full-size DOI: 10.7717/peerjcs.2228/fig-13

the rest were incorrectly classified. In accepting/rejecting MRs using DT, 213 out of 332 MRs were correctly classified, while the rest were incorrectly classified.

Our research aligns with other studies, such as *Srewuttanapitikul & Muengchaisri (2016)*, who found that DT accuracy is acceptable for sorting MRs in the classification phase. Previous studies have reported sorting accuracy ranging from 63% to 88.78% (*Ahmed, Bawany & Shamsi, 2021*). Comparing our sorting accuracy of 71.08% with *Levin & Yehudai*

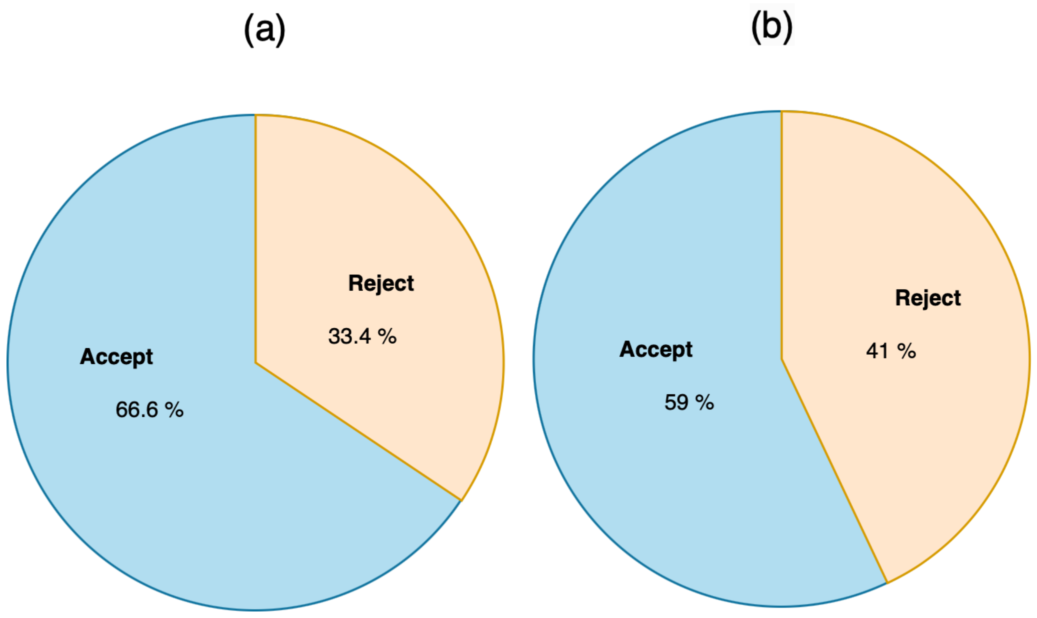


Figure 14 **(A) The actual classes of accept reject MR. (B) The predicted classes of accept or reject MR.**
Full-size DOI: 10.7717/peerjcs.2228/fig-14

(2017), who reported 76% accuracy and 63% kappa accuracy, shows that our results are within an acceptable range. However, the ranking accuracy in our study is lower than previous studies, which reported accuracy between 70% and 90.43% (*Ahmed, Bawany & Shamsi, 2021*; *Srewuttanapitikul & Muengchaisri, 2016*). The acceptance/rejection accuracy in our study is 64.15%, lower than the reported accuracy of 77.90% to 90.56% in previous studies (*Nizamani et al., 2018*; *Umer, Liu & Sultan, 2019*; *Cheng et al., 2021*).

The lower ranking accuracy might be due to the characteristics of the training dataset, which significantly influence the classification results. The performance of DTs decreases with an increase in the number of features and categories (*Pal & Mather, 2003*). Our study's weaknesses include the broad scope of the dataset, which spans different applications. Applying the model to a specific application might yield more accurate results. Future studies should consider additional factors influencing the acceptance/rejection of MRs, such as resources, costs, and time.

CONCLUSIONS

This study aimed to evaluate the suitability and accuracy of DTs for automating the classification phases of software maintenance. Our results indicate that DTs can effectively automate sorting and acceptance/rejection of MRs, though the ranking accuracy is lower. In our research, we processed and classified 1,656 MRs using the TF-IDF feature extraction technique and applied DTs for sorting, ranking, and acceptance/rejection. The sorting and acceptance/rejection accuracies were acceptable, but the ranking accuracy was inadequate. Future research should extend the study to include preventive maintenance, explore different techniques for classification, and consider additional factors such as budget, resources, and time for acceptance/rejection. Moreover, applying the research to other languages, such as Arabic, could be beneficial.

ADDITIONAL INFORMATION AND DECLARATIONS

Funding
The authors received no funding for this work.

Competing Interests
The authors declare there are no competing interests.

Author Contributions
- Sahar Alturki conceived and designed the experiments, performed the experiments, analyzed the data, performed the computation work, prepared figures and/or tables, authored or reviewed drafts of the article, and approved the final draft.
- Sarah Almoaiqel analyzed the data, authored or reviewed drafts of the article, and approved the final draft.

Data Availability
The following information was supplied regarding data availability:

The code and raw data is available in the Supplementary File.

Supplemental Information
Supplemental information for this article can be found online at http://dx.doi.org/10.7717/peerj-cs.2228#supplemental-information.

REFERENCES

Ahmed HA, Bawany NZ, Shamsi JA. 2021. Capbug-a framework for automatic bug categorization and prioritization using NLP and machine learning algorithms. *IEEE Access* **9**:50496–50512 DOI 10.1109/ACCESS.2021.3069248.

Al-Hawari A, Najadat H, Shatnawi R. 2021. Classification of application reviews into software maintenance tasks using data mining techniques. *Software Quality Journal* **29(3)**:667–703 DOI 10.1007/s11219-020-09529-8.

Alenezi M, Banitaan S. 2013. Bug reports prioritization: which features and classifier to use? In: *Proceedings of the 12th International Conference on Machine Learning and Applications, vol. 2*. DOI 10.1109/ICMLA.2013.114.

Arshad MA, Huang Z, Riaz A, Hussain Y. 2021. Deep learning-based resolution prediction of software enhancement reports. In: *2021 IEEE 11th Annual Computing and Communication Workshop and Conference (CCWC)*. Piscataway: IEEE, 492–499 DOI 10.1109/CCWC51732.2021.9375841.

Baqais AAB, Alshayeb M, Baig ZA. 2013. Hybrid intelligent model for software maintenance prediction. In: *Proceedings of the World Congress on Engineering, London, U.K.* 358–362.

Bird S. 2006. NLTK: the natural language toolkit. In: *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*.

Cheng J, Sadiq M, Kalugina OA, Nafees SA, Umer Q. 2021. Convolutional neural network based approval prediction of enhancement reports. *IEEE Access* **9**:122412–122424 DOI 10.1109/ACCESS.2021.3108624.

Ciurumelea A, Panichella S, Gall HC. 2018. Automated user reviews analyser. In: *Proceedings of the 40th International Conference on Software Engineering: Companion Proce, Gothenburg, Sweden*. DOI 10.1145/3183440.3194988.

Ekanata Y, Budi I. 2018. Mobile application review classification for the Indonesian language using machine learning approach. In: *Proceedings of the 4th International Conference on Computer and Technology Applications, Istanbul, Turkey*. DOI 10.1109/CATA.2018.8398667.

Ekanayake JB. 2021. Predicting bug priority using topic modelling in imbalanced learning environments. *International Journal of System and Service-Oriented Engineering* **11(1)**:31–42 DOI 10.4018/ijssoe.2021010103.

Guzman E, Ibrahim M, Glinz M. 2017. Prioritizing user feedback from twitter: a survey report. In: *Proceedings of the 4th International Workshop on CrowdSourcing in Software Engineering*. DOI 10.1145/3127005.3127016.

Ikram A, Jalil MA, Ngah AB, Khan AS. 2020. Towards offshore software maintenance outsourcing process model. *International Journal of Computer Science and Network Security* **20(4)**:6–14.

Jo T. 2021. Decision tree. In: *Machine learning foundations: supervised, unsupervised, and advanced learning*. Cham: Springer International Publishing, 141–165 DOI 10.1007/978-3-030-65900-4_7.

Joseph VR. 2022. Optimal ratio for data splitting. *Statistical Analysis and Data Mining: The ASA Data Science Journal* **15(4)**:531–538 DOI 10.1002/sam.11583.

Korenius T, Laurikkala J, Järvelin K, Juhola M. 2004. Stemming and lemmatization in the clustering of finnish text documents. In: *Proceedings of the 13th ACM International Conference on Information and Knowledge Management, New York, NY, USA*. DOI 10.1145/1031171.1031285.

Levin S, Yehudai A. 2017. Boosting automatic commit classification into maintenance activities by utilizing source code changes. In: *Proceedings of the 13th International Conference on Predictive Models and Data Analytics in Software Engineering, New York, NY, USA*. DOI 10.1145/3127005.3127016.

Nafees SA, Rehman FA. 2021. Machine learning based approval prediction for enhancement reports. In: *2021 International Bhurban Conference on Applied Sciences and Technologies (IBCAST)*. 377–382 DOI 10.1109/IBCAST51254.2021.9393180.

Nizamani ZA, Liu H, Chen DM, Niu Z. 2018. Automatic approval prediction for software enhancement requests. *Automated Software Engineering* **25**:347–381 DOI 10.1007/s10515-017-0229-y.

Nyamawe AS, Liu H, Niu N, Umer Q, Niu Z. 2020. Feature requests-based recommendation of software refactorings. *Empirical Software Engineering* **25**:4315–4347 DOI 10.1007/s10664-020-09871-2.

Otoom AF, Al-jdaeh S, Hammad M. 2019. Automated classification of software bug reports. In: *Proceedings of the 9th international conference on information communication and management*. 17–21 DOI 10.1145/3357419.3357424.

Pal M, Mather PM. 2003. An assessment of the effectiveness of decision tree methods for land cover classification. *Remote Sensing of Environment* **86(4)**:554–565 DOI 10.1016/S0034-4257(03)00132-9.

Pandey N, Sanyal DK, Hudait A, Sen A. 2017. Automated classification of software issue reports using machine learning techniques: an empirical study. *Innovations in Systems and Software Engineering* **13**:279–297 DOI 10.1007/s11334-017-0294-1.

Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, Vanderplas J. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research* **12**:2825–2830.

Phetrungnapha K, Senivongse T. 2019. Classification of mobile application user reviews for generating tickets on issue tracking system. In: *Proceedings of the 12th International Conference on Information & Communication Technology and System, Surabaya, Indonesia*. DOI 10.1109/ICTS.2019.8850962.

Qaiser S, Ali R. 2018. Text mining: use of TF-IDF to examine the relevance of words to documents. *International Journal of Computer Applications* **181(1)**:25–29 DOI 10.5120/ijca2018917395.

Ramay WY, Umer Q, Yin XC, Zhu C, Illahi I. 2019. Deep neural network-based severity prediction of bug reports. *IEEE Access* **7**:46846–46857 DOI 10.1109/ACCESS.2019.2909746.

Razno M. 2019. Machine learning text classification model with NLP approach. *Computer Linguistics and Intelligent Systems* **2**:71–73.

Rácz A, Bajusz D, Héberger K. 2021. Effect of dataset size and train/test split ratios in QSAR/QSPR multiclass classification. *Molecules* **26(4)**:1111 DOI 10.3390/molecules26041111.

Ren Y, Liu Z, Xing T, Chen X. 2011. Software maintenance process model and contrastive analysis. In: *Proceedings of the 2011 International Conference on Information Management, Innovation Management and Industrial Engineering, Shenzhen, China*. DOI 10.1109/ICIII.2011.324.

Sharawat S. 2012. Software maintainability prediction using neural networks. *International Journal of Engineering Research and Applications* **2(5)**:750–755.

Stojanov Z, Stojanov J. 2016. Exploring software maintenance process characteristics by using inductive thematic analysis. In: *International conference on Applied Internet and Information Technologies*. 9–17.

Srewuttanapitikul K, Muengchaisri P. 2016. Prioritizing software maintenance plan by analyzing user feedback. In: *2016 International Conference on Information Science and Security (ICISS)*. 1–5 DOI 10.1109/ICISSEC.2016.7885865.

Tian Y, Lo D, Xia X, Sun C. 2015. Automated prediction of bug report priority using multifactor analysis. *Empirical Software Engineering* **20**:1354–1383 DOI 10.1007/s10664-014-9331-y.

Umer Q, Liu H, Sultan Y. 2018. Emotion based automated priority prediction for bug reports. *IEEE Access* **6**:35743–35752 DOI 10.1109/ACCESS.2018.2850910.

Umer Q, Liu H, Sultan Y. 2019. Sentiment based approval prediction for enhancement reports. *Journal of Systems and Software* **155**:57–69 DOI 10.1016/j.jss.2019.05.026.

Zhang Y, Zhou Y, Yao J. 2020. Feature extraction with TF-IDF and game-theoretic shadowed sets. In: *Information processing and management of uncertainty in knowledge-based systems*. DOI 10.1007/978-3-030-50146-4_53.