Research Article GU J Sci 38(1): 131-144 (2025) DOI: 10.35378/gujs.1446469

Gazi University

Journal of Science

http://dergipark.gov.tr/gujs

Prioritization of Regression Test Cases Based on Machine Learning Methods

Selcuk KIRAN, Ilkim Ecem EMRE*, Selen TASDELEN

Marmara University, Faculty of Business Administration, Department of Management Information Systems, İstanbul, Türkiye

Highlights

- This study confirms ML techniques can prioritize regression test scenarios effectively.
- RF model emerged as the most effective in classifying scenarios based on their importance.
- NLP integration with ML for test case prioritization improves the efficiency of regression testing.

Article Info

Received: 05 Mar 2024
Accepted: 23 Nov 2024

Keywords

Machine learning
Natural language processing,
Software testing
Regression testing
Test case prioritization

Abstract

Due to resource and time constraints involved in the software testing process, it is not possible to implement all test scenarios for each release. Test scenarios can be prioritized according to certain criteria defined by the developers to ensure effective execution of the testing process and detection of errors. This study investigated whether machine learning based models could be used to prioritize test scenarios created in regression testing. It is attempted to determine which tests can be prioritized for execution based on different independent variables. In total, each of the 964 test scenarios in the dataset was labelled as minor (482) and major (482) by two experts. In the models, the number of related requirements, the number of related errors, and the age of the scenario were used as independent variables, and the scenario classes labelled as minor - major were taken as the target variable. The scenarios were pre-processed using natural language processing techniques and different machine learning algorithms were used for model development. In the classification based on test scenarios, the random forest algorithm showed the best performance with a F1-score of 81%. In the classification based on the number of related requirements, the number of interrelated errors, and the age of the test scenarios, the random forest model once again demonstrated the highest success rate at 79%. This study demonstrates that machine learning techniques offer a variety of models for test case prioritization.

1. INTRODUCTION

The software development life cycle encompasses the entire process of developing a software project, from the initial planning stages to final deployment. Stages including planning, analysis, design, development, testing, and deployment are part of the software life cycle. The testing phase is a critical phase in the software development life cycle. Software testing activities have become primordial in the software development life cycle [1]. Software testing evaluates and verifies the functionality, accuracy, reliability, performance, usability, compatibility, and security of the software being developed. Software testing can be performed using several alternative methods and techniques. Software testing encompasses a variety of activities along the software development process [2]. Common testing methods include unit testing, integration testing, system testing, acceptance testing, and performance and regression testing [3].

Testers may occasionally encounter difficulties in identifying defects in software products and can uncover error-prone test cases by examining the requirements. The reason is that in complex software systems, several modules in a single component are developed by different stakeholders or teams. In the case of integrated systems, access authorization may be restricted to modules. Therefore, it may not be possible to examine the source code. Software testing checks that user requirements are fully met.

For imperative factors such as funding and time constraints, it may not always be possible to perform the comprehensive testing process. As a result, testing activities may be terminated before they are completed, affecting the quality of the software and causing problems before the system is accepted by the customer

*Corresponding author, e-mail: ecem.emre@marmara.edu.tr

[4]. Therefore, costs can be reduced by prioritizing more critical test cases for efficient execution and acceleration of the testing process.

Test Case Prioritization (TCP) methods organize and execute test cases in a test suite to achieve prioritization goals. Different test cases have different abilities to locate faults and satisfy user needs. When test cases are aligned according to certain criteria, the fault detection rate can be optimized [5]. This study aims to create models for prioritizing regression test cases as major and minor. For this purpose, case descriptions, number of linked requirements, number of linked faults, and case age variables were used as independent variables and tested with different machine learning algorithms.

1.1. Regression Testing

Software applications are complex to develop due to high user requirements and multifunctionality. The analysis phase at the beginning of the software lifecycle determines the users's needs and expectations for the software. When the system developed according to these requirements is then launched, the faults that occur are detected, verified and tested. The larger and more complex the software system, the more difficult it is to test [6]. In a software project, existing modules may be affected by bug fixes, new feature enhancements, or code changes that may break previously working features or introduce new bugs. Regression testing is considered a key quality assurance test for modern software systems [7].

A regression testing is a type of test performed to verify that improvements made to a software component or system do not affect the functionality of other components or the system [8]. Updates made during the software development process are intended to correct errors in the previous version or add new features. However, these changes may affect other features or functions of the software and introduce new bugs. A regression testing is a type of test that checks for these bugs. A regression testing, which is also part of the software maintenance process, focuses on what kind of errors the updated versions cause in other components or systems, and reports the detected faults to the developers.

The benefits of regression testing include detecting the potential implications of bugs in the software, verifying that previously tested features of the software work properly, and enhancing the reliability of the software. This test also provides early detection of the consequences of potential faults in the software, verifying the proper operation of the software at an early stage and improving its performance [9].

The cases created for regression testing assay the existing components of the existing software. The prioritized regression test cases are executed sequentially with the required test data. These test cases cover different parts of the software and their linked functions. For each test case, the results are compared with the expected results. Failed test results are reported to the developers as errors.

Regression tests can be classified as one of the most expensive testing methods as it requires all tests to be repeated multiple times, which is time-consuming [10, 11]. As it is difficult to allocate enough time and resources to test all possible contingencies, comprehensive testing may not be feasible. Test teams therefore manage the testing process by prioritizing important and critical functionality. With each update, it can become a challenge to completely execute all existing test cases. For this reason, techniques such as risk analysis are used to focus on components that are likely to be affected by the update [12].

1.2. Test Cases

Test cases are a collection of test cases designed to verify different features, functionalities, and usage steps of the software. Each test case models a specific situation and defines the expected results. Test cases are created to verify that the software operates smoothly and meets requirements and expectations. Test cases are written according to the functions of the software and the requirements of the user [13].

Test cases and scenarios are used to verify multiple functions and features of software systems through the software in which they are created. For instance, many test cases can be designed to test only the login

page. These cases compare the expected results with the actual results. Should an error occur, it is communicated to the software team, along with the causes and evidence.

1.3. Test Case Prioritization - TCP

Software testing is a process used to evaluate and improve the functionality, accuracy, and quality of software. Potential faults are identified early in the process to meet user needs, ensure reliability and improve the performance of the software. In this regard, Test Case Prioritization (TCP) attempts to prioritize test cases ideally to achieve objectives such as early fault detection [14]. Each new version of the software introduces the possibility of new bugs due to updates and new features. These bugs can be identified using regression tests. This testing process involves re-running the test cases tested in previous releases to maintain the functionality and features of the current release. Due to limited resources, it can be costly to run all possible test cases in regression tests [11] which makes it impracticable to carry out comprehensive testing [15].

Test case prioritization refers to the determination of which scenarios will be executed first in the test process. This prioritization process ensures that test processes are controlled effectively and efficiently and increases the probability of fault detection [16] Test cases are prioritized based on key features of the software or the severity of deficiencies. The prioritization of test cases depends on the code coverage, the requirements involved, the number of bugs, and the severity of the impact of the potential bug, so the aim is to execute the cases according to the requirements [7]. Different methods can be used for this purpose. For example, [10] attempted to determine the prioritization of test cases to be executed using an ant colony optimization (ACO) algorithm. As suggested in another study, it is possible to examine the correlation between requirements and prioritize the regression test cases accordingly [17]. When user expectations are taken into account during test case prioritization, the key features of the software, especially most frequently used ones are tested. So, customer satisfaction is enhanced while ensuring that the software performs effectively. [18] proposed to explore similarities between test suites, cluster them, and reduce redundant tests by using Word2Vec method and k-means algorithm. [19] used string distance metrics for prioritization of test cases. [20] proposed topic modelling technique to prioritize test cases. They've calculated the dissimilarity to prioritize the most dissimilar cases. [21] have used reinforcement learning methods for the test case selection and prioritization. [22] recommended using genetic algorithm to prioritize the selected test cases. [23] proposed a deep learning model for regression test case prioritization. [24] used neural networks to classify the test cases and they have achieved 96% accuracy.

[16] stated that prioritizing tests can be practiced for several purposes:

- Testers may wish to increase the rate of fault detection of a test suite–that is, the likelihood of revealing faults earlier in a run of regression tests using that test suite.
- Testers may wish to increase the coverage of coverable code in the system under test at a faster rate, thus allowing a code coverage criterion to be met earlier in the test process.
- Testers may wish to increase their confidence in the reliability of the system under test at a faster rate.
- Testers may wish to increase the rate at which high-risk faults are detected by a test suite, thus locating such faults earlier in the testing process.
- Testers may wish to increase the likelihood of revealing faults related to specific code changes earlier in the regression testing process

However, incorrect prioritization of test cases can lead to poor test performance. In this case, critical errors may not be detected and the intended quality and performance of the software may be reduced. User requirements may change over time, so test cases may also need updating. This might necessitate changes to the resources planned at the start of the project, incurring wasted effort and time.

1.4. Test Case Prioritization Using NLP Techniques

Machine learning (ML), a subfield of artificial intelligence research, offers methods that provide opportunities for use in nearly every field. ML encompasses a wide variety of algorithms designed to make

intelligent predictions based on data sets [25] and has been increasingly applied to test case prioritization (TCP) in software testing, with a focus on improving efficiency and effectiveness [26]. Natural language processing (NLP), a sub-discipline of machine learning, includes a set of methods and algorithms for processing and analyzing text data. These methods enable computers to understand, interpret and transform text into structured data. NLP uses methods from computer science, computational linguistics and artificial intelligence. NLP techniques can interpret and generate new data from unstructured data such as audio, text, PDF, video and images [27].

ML is increasingly utilized in conjunction with NLP to interpret unstructured text data and also the integration of ML with NLP techniques enables researchers to extract various types of information from diverse sources [28]. ML algorithms can be used to classify test cases into specific categories or priority levels. These algorithms can perform prioritization by recognizing certain words or patterns in the script text [27]. To prioritize test cases, NLP evaluates and classifies test cases based on grammatical and semantic features provided in their descriptions. For example, attributes such as important functions, important data inputs or system interactions in the content of test cases can help to prioritize test cases. Dependencies and similarities between test cases can also be identified using NLP techniques.

In order to analyze text, the text to be analyzed is pre-processed using various NLP techniques. One such technique is tokenization, which is an NLP technique that breaks text into smaller pieces called tokens. Tokenization breaks text data into smaller pieces, making the text suitable for use in grammar analysis, text classification and machine learning algorithms. It is often used to break down sentences, words or structures (roots, affixes, etc.) into smaller grammatical units. For example, the phrase “incorrect username or password” can be split into four words such as “incorrect”, “username”, “or”, “password” [29].

Stemization is used to reduce lexical variation, to reduce the size of text data, and to represent similar words in the same lexical stem. The aim is to identify the roots of words and give them meaning. The root of a word can be thought of as a form free of various affixes. For example, the root of words inflected with different tenses such as “playing”, “plays” and “played” can be considered as “play” [30].

The removal of numerical expressions and punctuation marks is one of the text pre-processing steps and aims to convert texts into a more homogeneous format. This pre-processing step enables texts to be processed more consistently and effectively [31]. Removing numerical expressions means removing numbers and numerical expressions from the text or replacing them with a special label. This can reduce the impact of numerical information in text on NLP applications such as text analysis or text classification. For example, numbers in a text document may affect the vocabulary or features in the text, while on the other hand they may be unimportant or undesirable for the purpose of analysis [32].

To delete punctuation means to remove or discard punctuation marks from the text. Punctuation marks are marks that determine the structure of sentences or phrases in the text, such as periods, commas, exclamation marks, question marks, quotation marks, parentheses. Removing punctuation marks allows the text to have a simpler structure and provides cleaner data entry in some NLP applications [33].

Stop word cleaning is used to remove words that are frequently used in texts but often have no meaning, such as “and”, “or”, “but”. Removing these stop words makes the text more comprehensible and easier to analyze while retaining its meaning. The aim of this process is to obtain cleaner and more focused data for NLP applications such as text analysis or text classification [34].

It should be noted that there are several studies in the literature on this topic. Different researchers use NLP techniques to prioritize test cases. [35] proposed an approach to find the most likely regression test cases from the master test suite using NLP by selecting a test case based on its intent match with defects. In another study, [36] used NLP techniques to support prioritization so that defects can be detected at an early stage for testing a mobile application that uses Chinese scripts. It is recommended to prioritize scenarios by calculating the APFD criterion. [37] on the other hand, used NLP techniques and machine learning algorithms to prioritize test scenarios. Test scenarios are divided into negative and positive cases and classified using KNN, LR, SVM and ANN algorithms. This study is most similar to our study. [38] used

SVM MAP to categorize cases through a model that learns from data tagged as 'relevant' or 'irrelevant'. [39] proposed a methodology consisting of two steps, the first step is NLP. Following the application of NLP techniques, the authors propose to classify each test case into a dependent or an independent class for test optimization purposes such as parallel test execution, test automation, test case selection and prioritization, and test suite reduction.

2. MATERIAL AND METHODS

This section presents the development and implementation processes of the test case prioritization model. In order to achieve the intended results of the study, the independent variables were controlled and their causal effects on the target variables were analyzed. The purpose of the developed model is to investigate whether performance can be optimized by prioritizing the cases generated during regression testing. In line with this objective the test case descriptions, the number of related requirements, the number of linked bugs, and the age of the test cases were examined in order to construct a prediction model and conduct classification. To make the data more intelligible, the pre-processed text data was vectorized. To prioritize the test cases, sequential categorization was conducted using machine learning methods. The study's methodology is discussed in depth in the Data Collection and Data Analysis sections.

2.1. Data Collection

Collecting data and organizing s is one of the fundamental steps of the study. The dataset used in the research was provided by a software development company in Turkey. The analysis process was carried out with regression testing of the entertainment application of a company that offers promotions and campaigns in the retail sector. Regression test cases, created manually by testers in Turkish, are stored in Jira software (an agile project management tool) for reporting and analysis. The dataset of 964 test cases was obtained from the Jira test suite that the organization is working on. The main reason for selecting this dataset is that the data is from a project that has been running for three years and it demonstrates diversity in terms of test cases.

2.2. Dataset Pre-Processing

We worked with two experts (testers) to classify and label the data before advancing to pre-processing stage since supervised machine learning techniques could be utilized to prioritize the test cases. The inclusion procedure of the experts was meticulously designed to guarantee that both testers have comparable levels of expertise. The requirements that could be completed earlier and the high number of defects were helpful factors in prioritizing the test cases were defined as major test cases. On the other hand, the requirements that can be completed later and the test cases with a low number of defects were labelled as minor test cases. Based on the testers' perspectives, the test cases were labelled as major and minor.

Although it was originally planned to work with 1000 test cases (samples), 36 of them were discarded and not included in the dataset due to technical conditions. The reasons referred as the absence of comprehensive test data, the absence of a test scenario, a new version not being compatible with old systems, a component found in the old version not being present in the new version, thus making the test scenarios created based on the old version unnecessary for the new version. As a result, 964 test cases were left for the labeling process. Within 20 days, the first expert labeled 600 test cases, while the other expert labeled 364 test cases, resulting in 486 major and 478 minor test cases being classified. When dealing with datasets that are not evenly distributed, algorithms can struggle to accurately understand the characteristics of the data distribution and could affect the interpretability of the models models [40, 41]. Therefore, balancing techniques can be utilized in machine learning studies. Although the difference between major and minor test scenarios was not too much, to balance the number of samples in each lass, four minor test case scenarios were developed and four major scenarios were deleted randomly.

After labelling, the sentences in the test cases were pre-processed to reduce clutter and ambiguity by converting uppercase letters to lowercase letters, removing numbers and punctuation marks, tokenization, stop word cleaning, and stemization. The word_tokenise function in the nltk library in Python [42] was

used for tokenization. The corpus created by the authors [43] was used for stop word cleaning. For the stemization process, the TurkishStemmer [44] function adapted to Turkish in the snowball stemmer [45] library was used to find the roots of words in Turkish. After data pre-processing, the statistical method called frequency-inverse document frequency (TF-IDF) was used to convert the textual data into the numerical format necessary for the model to understand and process the textual data. The TfidfVectorizer function from the scikit-learn library in Python [46] was used for vectorization. In a text, some words occur more frequently but are less related to the context; however, there may be words that occur less frequently but are more meaningful/relevant in terms of content [47]. In order to prevent the more frequent words from overshadowing the rare words, TF-IDF helps to re-weight them and allows us to determine how relevant a particular word is to the particular context or document [47].

2.3. Data Analysis

Different classification algorithms were used to classify the test cases as either major and minor. Four different independent variables were used as predictor variables: test case descriptions, number of linked requirements, number of linked bugs and faults, and age of test cases. These variables were selected based on the expert's experience and recommendations.

The test case description refers to a set of actions to be performed to verify compliance with a specific requirement and includes the expected result. The number of linked bugs and faults is the number of existing linked bugs on systems and units after updates, newly added features, and correction of previous bugs. The number of linked requirements includes the number of business and stakeholder requirements analyzed when creating the test case. Test case age is the time taken to analyze the business and stakeholder requirements of the test cases and create them in the test environment, define and create the test environment and the data required to perform the test, and perform the entire test process.

The target variable is the major and minor class labels identified for each test case. The target variable was estimated in two different ways; in the first case, only the test case descriptions were included in the models as independent variables, while in the second case, the number of linked requirements, the number of linked bugs and faults, and the age of the test cases, all of which contain numerical values, were included in the models. The analyses were performed using Python programming language, an open-source programming language, via Google Colaboratory platform which is a free-to-use cloud-based platform supporting open-source tools and libraries for machine learning and data analysis.

In the study, training and test datasets were created after labelling and vectorization processes, models were built using five different classification algorithms, and performance evaluation was conducted. It is known that the quality of the dataset used for training has a great impact on the prioritization results. In this study, a balance between the categories was considered for the reliability of the results, and an equal number of test cases from both classes were included in the dataset. The cross-validation method was used to separate the dataset into training and test datasets. By applying 12-fold cross-validation, the dataset was divided into 12 parts, 11 parts were determined as the training dataset in each iteration, while the remaining part was tested and performance metrics were calculated. The same steps were performed in 12 iterations. Models were then built using five different classification algorithms. The classification algorithms used in the modelling are Naive Bayes (NB), Logistic Regression (LR), Support Vector Machine (SVM), Random Forest (RF), and K-Nearest Neighbor (KNN). These algorithms were selected from among those used in software testing in the literature [48]. The scikit-learn library [46] was used to split the dataset into training and test datasets and to access the algorithms used. The multinomial NB algorithm was used to classify the test cases, and the Gaussian NB algorithm was used for classification where all values were numeric. Each algorithm, shown in Figure 1, illustrates the method used to prioritize the test cases, which consists of the different stages described above.

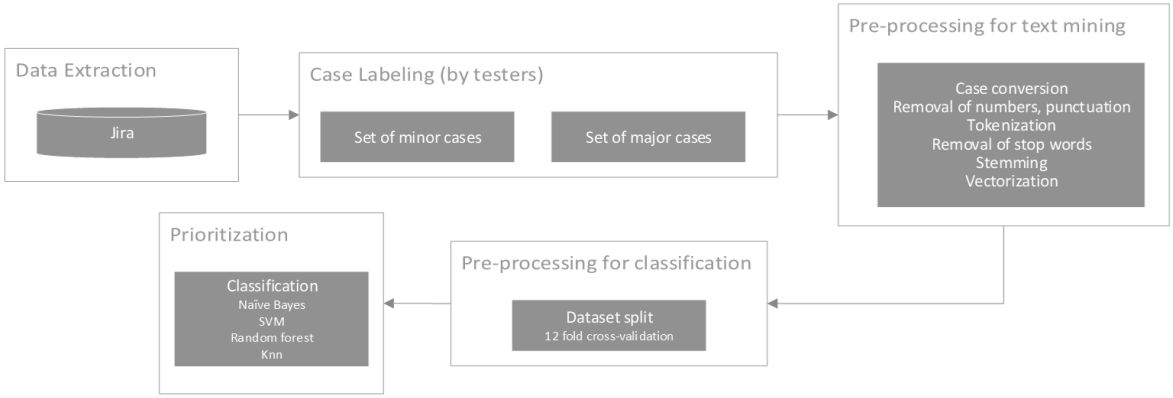

Figure 1. *Data analysis process*

2.4. Algorithms

Naïve Bayes, which is used in machine learning for tasks such as classification and probability estimation, analyses the membership probabilities for each class, i.e., the probability that a given piece of data belongs to that class. The most likely class is the one with the highest probability. The basic approach of the Naive Bayes algorithm, which is widely used in applications such as NLP, spam filtering, and sensitivity analysis, is that all attributes are independent of each other [49].

Logistic regression is generally used for two-class classification problems but has been adapted over time for multi-class problems. Logistic regression analyses the probability of the dependent variable being a member of one of the two classes by estimating the probability distribution of the dependent variable [50]. Logistic regression uses the logit function to model the relationship between the dependent variable and the independent variables. This function computes probability using a weight vector along with the function of the independent variables. The logistic regression model is built by computing the values of the dependent variable and the values of the independent variable in the training dataset. Once the training phase is complete, new instances in the test dataset can be classified using the generated model [51].

Support Vector Machines (SVMs) perform classification and regression by finding an optimal boundary between possible outcomes. SVM can be used for developing linear and non-linear datasets [52]. The main purpose of SVM is to create an n-dimensional hyperplane called support vectors that perform binary classification. These support vectors play an important role in determining the decision boundary during the classification process. SVM aims to find the hyperplane with the greatest distance between classes. In this study, when SVM is applied to the data, an attempt is made to create a plane with the classifier that separates the dataset consisting of two-dimensional spatial points belonging to two classes [53].

The Random Forest (RF) algorithm focuses on achieving more accurate and stable results by combining many decision trees. Each decision tree is based on a different instance and set of variables, and a model is built based on a majority decision. In building the decision tree, a splitting criterion is used at each node to select the feature providing the best split. Each tree has a different structure when the sampling and feature selection process is repeated. Combining multiple trees avoids over learning and creates a more general model [54]. Because RF can handle complexity and noise in datasets, it can be effective in classifying test cases into main and sub-test cases. Combining multiple decision trees provides more reliable and stable results [55].

The K-Nearest Neighbor (KNN) algorithm groups or predicts a new data point based on the labels or values of neighboring data around it for classification or prediction. KNN is used to find the points closest to the new data. The algorithm uses distance functions to find neighboring points that are similar to a new data point. The distance calculation can use distance calculation methods such as Euclidean distance and Manhattan distance, which are used in k-means [50]. The KNN method has the benefits of performing well in classification and regression problems, being quick in training, and being resistant to inconsistent training

data. The KNN technique has the problem of requiring a lot of storage space since the dataset is retained in memory, and the processing load grows as the dataset size increases [56].

The confusion matrix is used to determine how accurately or inaccurately the models predict which classes. The dataset contains test cases of different classes and the correct prediction rate of the model for each test case is analyzed. The confusion matrix is used to calculate performance measures such as accuracy, precision, sensitivity, and F1 score.

3. FINDINGS

A descriptive summary of the dataset is presented in Table 1. The 964 items downloaded from Jira, the company's project management tool, were classified by the testers to have 482 major and 482 minor cases.

Table 1. *Descriptives of the dataset*

Total number of test cases	Number of major test cases	Number of minor test cases
964	482	482

Table 2 shows the accuracy and F1 values obtained from the classification algorithms. According to this, the highest accuracy value of 81% was obtained from the model obtained with the RF algorithm. According to the F1 measures, it can be seen that the RF model has the highest value.

Table 2. *Model performances based on the test case descriptions*

Algorithm	Accuracy	F1-Score
Multinomial NB	0.80	0.80
LR	0.80	0.80
SVM	0.79	0.79
RF	0.81	0.81
KNN	0.69	0.67

Table 3 shows the accuracy and F1 values obtained by the classification algorithms. As shown, the lowest accuracy value was obtained from the model built with the NB algorithm, and there was no difference between the performances of the other algorithms. While the accuracy of all models was 79%, only the F1 measure of the RF model was obtained at 79%, although all were close to each other.

Table 3. *Model performances based on the other variables*

Algorithm	Accuracy	F1-Score
Gaussian NB	0.57	0.56
LR	0.79	0.78
SVM	0.79	0.78
RF	0.79	0.79
KNN	0.79	0.78

Following the evaluation of the different algorithms to prioritize the test cases, it was found that the RF algorithm performed better once all features were taken into account. The experimental results obtained from the tables can be interpreted from two different perspectives. From the perspective of test case descriptions, it was observed that the other machine learning algorithms, except KNN, scored high (80% and above) in classification based on test case descriptions. In this case, the KNN algorithm scored the lowest. When the results of the classification analysis are analyzed according to the number of linked requirements, the number of linked faults and the age of the test cases, the LR, SVM and RF algorithms showed the highest success rate with 79%. The Gaussian NB algorithm showed the lowest performance. Although the results obtained in the performance evaluation are not bad, different algorithms, parameters or methods, such as dividing the training and test datasets with different ratios, can be tried to achieve

higher values in model performance. Among the studies covered in this article, [37] has used average percentage of faults detected (APFD) for the performance comparison between algorithms and as a result, logistic regression has outperformed other algorithms. Classifying target variables only as major and minor may limit the practical application of the model. For the further studies, as in the study of [24] test cases can be divided into more detailed groups as very high priority, high priority, medium priority, low priority, very low priority. Obtained accuracy results is lower than [24]'s results but the used algorithms are different from each other, also the number of classes are different. Therefore, another study can be conducted by using a more detailed set of test cases. Also before determining cases to prioritize, cases can be examined for being dependent or independent like in [39]'s study for more efficient results. Topic modelling like in [20]'s study can be used for clustering the cases before prioritization. In this way, the scope of the same study can be extended.

4. DISCUSSION AND CONCLUSION

Today, complex and large software is being developed to meet the increasing needs of users. The complexity of the software also means the complexity of the tests to be performed. As mentioned at the beginning of this paper, software testing is one of the most important phases of the software development life cycle and is used to evaluate the accuracy, completeness and quality of the software created. Optimal execution of these tests will benefit the developers. In this context, this study seeks to answer the question of which supervised machine learning algorithms perform better in prioritizing regression test scenarios.

More advanced methods, such as deep learning and reinforcement learning algorithms, can also be applied in projects where the data is constantly updated and different parameters are included, due to their high ability to make inferences from complex structures. These various learning techniques like in different studies [21-23], can be applied to the same dataset for the classification. In addition, different algorithms and approaches like ensemble techniques, such as stacking or voting ensembles can be applied in similar datasets for the enhancement of the results. In light of the findings and the developed models, a user-friendly tool or framework designed to assist software testers and developers in the prioritization of test cases can be developed. Such a tool that would integrate the trained models, can provide intuitive interfaces for the input of test case information and generating prioritized test case lists or recommendations. In this regard collaboration with industry partners or software developers can be taken into consideration in the development phase in order to gather feedback and to refine the tool's usability and enhance its practical applicability.

Test case selection can also be applied in similar studies to improve the efficiency of the testing process. In addition, test case reduction can be evaluated together with test case prioritization for the test phase. Consequently, it may be possible to eliminate and prioritize redundant and inefficient tests. As proposed in a separate study [17], different studies can be conducted to prioritize the requirements and prioritize the regression test scenarios respectively. Test case reduction (TCR), which is one of the steps that can be applied with TCP, can improve test quality and save resources provided that it is considered along with prioritization. Redundant tests can be eliminated by looking for commonalities in the suites used to test [18].

According to the results of this study, it is showed that for test case prioritization, machine learning techniques can provide a wide range of models. In addition, NLP methods help to analyze case description which is a text data. Despite the potential of machine learning in TCP, it is stated that by there is a need for further research to optimize its performance and address its limitations [4]. Moreover, machine learning techniques have been proposed to improve regression testing by selecting and prioritizing test cases to provide early feedback to developers [57]. A variety of tests can be used to ensure the functional, effective and appropriate application of the models in the real world. Furthermore, in order to assess the robustness and applicability of the approach in various real-world scenarios the dataset can be expanded for future studies. Also, in order to optimize the testing process, reduce testing effort, and accelerate the delivery of high-quality software releases, test case prioritization approach could be integrated with continuous integration and continuous delivery (CI/CD) pipelines like in the study of [58].

CONFLICTS OF INTEREST

No conflict of interest was declared by the authors.

REFERENCES

[1] Atifi, M., Mamouni, A., and Marzak, A., "A comparative study of software testing techniques", Lecture Notes in Computer Science, 10299: 373–390, (2017). DOI: 10.1007/978-3-319-59647-1_27

[2] Lonetti, F., and Marchetti, E., "Emerging software testing technologies", Advances in Computers, 108: 91–143, (2018).

[3] Shah, U.S., Jinwala, D.C., and Patel, S.J., "An excursion to software development life cycle models", ACM SIGSOFT Software Engineering Notes, 41(1): 1–6, (2016). DOI: 10.1145/2853073.2853080

[4] Khatibsyarbini, M., Isa, M.A., Jawawi, D.N.A., and Tumeng, R., "Trend application of machine learning in test case prioritization: a review on techniques", IEEE Access, 9: 166262–166282, (2021). DOI: 10.1109/ACCESS.2021.3135508

[5] Khatibsyarbini, M., Isa, M.A., Jawawi, D.N.A., and Tumeng, R., "Test case prioritization approaches in regression testing: A systematic literature review", Information and Software Technology, 93: 74–93, (2018). DOI: 10.1016/j.infsof.2017.08.014

[6] Harold, M.J., "Testing: A roadmap", Proceedings of the Conference on the Future of Software Engineering, ICSE 2000, 61–72, (2000). DOI: 10.1145/336512.336532

[7] Lou, Y., Chen, J., Zhang, L., and Hao, D., "A survey on regression test-case prioritization", Advances in Computers, 113: 1–46, (2019).

[8] Minhas, N.M., Petersen, K., Börstler, J., and Wnuk, K., "Regression testing for large-scale embedded software development – Exploring the state of practice", Information and Software Technology, 120: 106254, (2020). DOI: 10.1016/j.infsof.2019.106254

[9] Rosero, R.H., Gomez, O.S., and Rodriguez, G., "Regression testing of database applications under an incremental software development setting", IEEE Access, 5: 18419–18428, (2017). DOI: 10.1109/ACCESS.2017.2749502

[10] Ahmad, S.F., Singh, D.K., and Suman, P., "Prioritization for regression testing using ant colony optimization based on test factors", Advances in Intelligent Systems and Computing, 624: 1353–1360, (2018). DOI: 10.1007/978-981-10-5903-2_142

[11] Boyar, T., Oz, M., Oncu, E., and Aktas, M.S., "A novel approach to test case prioritization for software regression tests", Lecture Notes in Computer Science, 12955: 201–216, (2021). DOI: 10.1007/978-3-030-87007-2_15

[12] Li, Z., Harman, M., and Hierons, R.M., "Search algorithms for regression test case prioritization", IEEE Transactions on Software Engineering, 33(4): 225–237, (2007). DOI: 10.1109/TSE.2007.38

[13] Desikan, S., and Ramesh, G., "Developing and baselining test cases", Software Testing: Principles and Practices, Pearson Education, 376, (2007).

[14] Prado Lima, J.A., and Vergilio, S.R., "Test case prioritization in continuous integration environments: A systematic mapping study", Information and Software Technology, 121: 106268, (2020). DOI: 10.1016/J.INFSOF.2020.106268

[15] Lu, Y., Lou, Y., Cheng, S., Zhang, L., Hao, D., Zhou, Y., Zhang, L., "How does regression test prioritization perform in real-world software evolution?", Proceedings of the International Conference on Software Engineering, 14-22, 535–546, (2016). DOI: 10.1145/2884781.2884874

[16] Rothermel, G., Untcn, R.H., Chu, C., and Harrold, M.J., "Prioritizing test cases for regression testing", IEEE Transactions on Software Engineering, 27(10): 929–948, (2001). DOI: 10.1109/32.962562

[17] Ma, T., Zeng, H., and Wang, X., "Test case prioritization based on requirement correlations", Proceedings of the 17th International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, SNPD 2016, 419–424, (2016). DOI: 10.1109/SNPD.2016.7515934

[18] Afshinpour, B., Groz, R., Amini, M.R., Ledru, Y., and Oriat, C., "Reducing regression test suites using the Word2Vec natural language processing tool", CEUR Workshop Proceedings, 2799: 43–53, (2020).

[19] Ledru, Y., Petrenko, A., Boroday, S., and Mandran, N., "Prioritizing test cases with string distances", Automated Software Engineering, 19(1): 65–95, (2012). DOI: 10.1007/s10515-011-0093-0

[20] Thomas, S.W., Hemmati, H., Hassan, A.E., and Blostein, D., "Static test case prioritization using topic models", Empirical Software Engineering, 19(1): 182–212, (2014). DOI: 10.1007/s10664-012-9219-7

[21] Spieker, H., Gotlieb, A., Marijan, D., and Mossige, M., "Reinforcement learning for automatic test case prioritization and selection in continuous integration", Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis, 12–22, (2017). DOI: 10.1145/3092703.3092709

[22] Musa, S., Sultan, A.B.M., Ghani, A.A.A., and Baharom, S., "Regression test case selection & prioritization using dependence graph and genetic algorithm", IOSR Journal of Computer Engineering, 16(3): 38–47, (2014). DOI: 10.9790/0661-16343847

[23] Sharif, A., Marijan, D., and Liaaen, M., "DeepOrder: deep learning for test case prioritization in continuous integration testing", Proceedings of the 2021 IEEE International Conference on Software Maintenance and Evolution, ICSME 2021, 525–534, (2021). DOI: 10.1109/ICSME52107.2021.00053

[24] Gokce, N., and Eminli, M., "Model-Based test case prioritization using neural network classification", Computer Science and Engineering: An International Journal, 4(1): 15–25, (2014). DOI: 10.5121/cseij.2014.4102

[25] Nichols, J.A., Herbert Chan, H.W., and Baker, M.A.B., "Machine learning: applications of artificial intelligence to imaging and diagnosis", Biophysical Reviews, 11(1): 111–118, (2019). DOI: 10.1007/s12551-018-0449-9

[26] Meçe, E.K., Paci, H., and Binjaku, K., "The application of machine learning in test case prioritization - a review", European Journal of Electrical and Computer Engineering, 4(1): 1-6, (2020). DOI: 10.24018/ejece.2020.4.1.128

[27] Blanck, M., "Predicting Price Residuals in Online Car Marketplaces with Natural Language Processing", Karlsruhe Institute of Technology, (2019).

[28] Sidey-Gibbons, J.A.M., and Sidey-Gibbons, C.J., "Machine learning in medicine: a practical introduction", BMC Medical Research Methodology, 19(1): 1–18, (2019). DOI: 10.1186/s12874-019-0681-4

[29] Toraman, C., Yilmaz, E.H., Sahinuc, F., and Ozcelik, O., "Impact of Tokenization on language models: an analysis for Turkish", ACM Transactions on Asian and Low-Resource Language Information Processing, 22(4): 1-10, (2023). DOI: 10.1145/3578707

[30] Korenius, T., Laurikkala, J., Järvelin, K., and Juhola, M., "Stemming and lemmatization in the clustering of finnish text documents", Proceedings of the International Conference on Information and Knowledge Management, 625–633, (2004). DOI: 10.1145/1031171.1031285

[31] Özkan, M., and Kar, G., "Multiclass classification of scientific texts written in Turkish by applying deep learning technique", Journal of Engineering Sciences and Design, 10(2): 504–519, (2022). DOI: 10.21923/jesd.973181

[32] Sun, F., Belatreche, A., Coleman, S., McGinnity, T.M., and Li, Y., "Pre-processing online financial text for sentiment classification: A natural language processing approach", Proceedings of the IEEE/IAFE Conference on Computational Intelligence for Financial Engineering, 122–129, (2014). DOI: 10.1109/CIFEr.2014.6924063

[33] Etaiwi, W., and Naymat, G., "The impact of applying different preprocessing steps on review spam detection", Procedia Computer Science, 113: 273–279, (2017). DOI: 10.1016/j.procs.2017.08.368

[34] Gharatkar, S., Ingle, A., Naik, T., and Save, A., "Review preprocessing using data cleaning and stemming technique", Proceedings of the 2017 International Conference on Innovations in Information, Embedded and Communication Systems, ICIIECS 2017, 1–4, (2017). DOI: 10.1109/ICIIECS.2017.8276011

[35] Sutar, S., Kumar, R., Pai, S., and Br, S., "Regression Test cases selection using Natural Language Processing", Proceedings of the International Conference on Intelligent Engineering and Management, ICIEM 2020, 301–305, (2020). DOI: 10.1109/ICIEM48762.2020.9160225

[36] Yang, Y., Huang, X., Hao, X., Liu, Z., and Chen, Z., "An industrial study of natural language processing-based test case prioritization", Proceedings of the 10th IEEE International Conference on Software Testing, Verification and Validation, ICST 2017, 548–549, (2017). DOI: 10.1109/ICST.2017.66

[37] Lachmann, R., "Machine learning-driven test case prioritization approaches for black-box software testing", Proceedings of the European Test and Telemetry Conference 2018, 300–309, (2018). DOI: 10.5162/ettc2018/12.4

[38] Busjaeger, B., and Xie, T., "Learning for test prioritization: An industrial case study", Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering, 975–980, (2016). DOI: 10.1145/2950290.2983954

[39] Tahvili, S., Hatvani, L., Ramentol, E., Pimentel, R., Afzal, W., and Herrera, F., "A novel methodology to classify test cases using natural language processing and imbalanced learning", Engineering Applications of Artificial Intelligence, 95: 103878, (2020). DOI: 10.1016/j.engappai.2020.103878

[40] Gao, Y., Zhu, Y., and Zhao, Y., "Dealing with imbalanced data for interpretable defect prediction", Information and Software Technology, 151: 107016, (2022). DOI: 10.1016/j.infsof.2022.107016

[41] Maraş, A., and Selçukcan Erol, Ç., "FuzzyCSampling: A Hybrid fuzzy c-means clustering sampling strategy for imbalanced datasets", Turkish Journal of Electrical Engineering & Computer Sciences, 31(7): 1223–1236, (2023). DOI: 10.55730/1300-0632.4044

[42] Bird, S., Klein, E., and Loper, E., Natural Language Processing with Python, O'Reilly Media Inc., (2009).

[43] https://github.com/ahmetax/trstop/tree/master. Access date: 16.08.2024

[44] https://snowballstem.org/algorithms/turkish/stemmer.html. Access date: 16.08.2024

[45] https://pypi.org/project/snowballstemmer/. Access date: 16.08.2024

[46] https://scikit-learn.org/stable/.Access date: 16.08.2024

[47] https://scikit-learn.org/stable/modules/feature_extraction.html#text-feature-extraction. Access date: 16.08.2024

[48] Ajorloo, S., Jamarani, A., Kashfi, M., Haghi Kashani, M., and Najafizadeh, A., "A systematic review of machine learning methods in software testing", Applied Soft Computing, 162: 1-15, (2024). DOI: 10.1016/j.asoc.2024.111805

[49] Troussas, C., Virvou, M., Espinosa, K.J., Llaguno, K., and Caro, J., "Sentiment analysis of Facebook statuses using Naive Bayes Classifier for language learning", Proceedings of the 4th International Conference on Information, Intelligence, Systems and Applications, 198–205, (2013). DOI: 10.1109/IISA.2013.6623713

[50] Balaban, M.E., and Kartal, E., Veri Madenciliği ve Makine Öğrenmesi Temel Algoritmaları ve R Dili ile Uygulamaları, Second Edition, İstanbul: Çağlayan Kitapevi, (2018).

[51] Cheng, Q., Varshney, P.K., and Arora, M.K., "Logistic regression for feature selection and soft classification of remote sensing data", IEEE Geoscience and Remote Sensing Letters, 3(4): 491–494, (2006). DOI: 10.1109/LGRS.2006.877949.

[52] Suthaharan, S., Machine Learning Models and Algorithms for Big Data Classification, 36: 1-20, Boston, MA: Springer US, (2016).

[53] Panesar, A., Machine Learning and AI for Healthcare: Big Data for Improved Health Outcomes, Second Edition, Apress Media LLC, (2021).

[54] Belgiu, M., and Drăgu, L., "Random forest in remote sensing: A review of applications and future directions", ISPRS Journal of Photogrammetry and Remote Sensing, 114: 24–31, (2016). DOI: 10.1016/j.isprsjprs.2016.01.011

[55] Nilsson, N.J., Yapay Zekâ - Geçmişi ve Geleceği, Boğaziçi Üniversitesi Yayınevi, (2018).

[56] Imandoust, S.B., and Bolandraftar, M., "Application of K-Nearest Neighbor (KNN) Approach for Predicting Economic Events: Theoretical Background", International Journal of Engineering Research and Applications, 3(5): 605–610, (2013).

[57] Pan, R., Bagherzadeh, M., Ghaleb, T.A., and Briand, L., "Test case selection and prioritization using machine learning: a systematic literature review", Empirical Software Engineering, 27(2): 1–43, (2022). DOI: 10.1007/s10664-021-10066-6

[58] Roza, E.A.d., Prado Lima, J.A.d., and Vergilio, S.R., "On the use of contextual information for machine learning based test case prioritization in continuous integration development", Information and Software Technology, 171: 107444, (2024). DOI: 10.1016/j.infsof.2024.107444