Tuijin Jishu/Journal of Propulsion Technology
ISSN: 1001-4055
Vol. 45 No. 2 (2024)

Prioritizing Test Cases Using Supervised Machine Learning Techniques Based on Requirement Correlation and Fault Severity

Tapas Kumar Choudhury[1], Subhendu Kumar Pani[2], Jibitesh Mishra[3], Jyotirmayee Rautaray[4]

[1, 2] *Biju Patnaik University of Technology, Rourkela, Odisha, India*
[3, 4] *Odisha University of Technology and Research, Bhubaneswar, Odisha, India*

Abstract:- Test case prioritisation plays a vital role in regression testing as test case prioritisation provides significant outcomes for producing effective in regression and other testing test cases methodologies. Many factors can be used to prioritise taste cases. Prior researchers created approaches for prioritising test cases based on requirements. But they used rigid algorithms for computation, which suggests that the outcomes are imprecise and rigid. Here, a model is developed for test case prioritisation using requirement correlation and fault severity. This study was conducted using an experimental research methodology which accepts 1000 test cases that have been classified as positive and negative by specialists for the experiment. Natural Language Processing (NLP) principles were employed during the pre-processing of the datasets, which used as input to the proposed model. Following pre-processing, the Term Frequency-Inverse Document Frequency (TF-IDF) approach is used to vectorize the textual data format. The machine learning algorithms Support Vector Machine (SVM), K-Nearest Neighbor (KNN), Naive Bayes (NB), and Decision Tree (DT) were used to generate the model. Each trial's accuracy results ranged from 74% to 94%, which enabled us to find and select a model that performed well in our experiment. SVM based on requirement correlation and fault severity did well for prioritizing test cases. KNN is the sole useful technique for determining how significant a fault is. This outcome is due to the fact that SVM outperforms KNN for a large number of features while KNN outperforms SVM for a smaller number of features.

Keywords*:* NLP, software testing, Test Case Prioritization, Machine learning.

1. Introduction

Making the best and most prioritized test cases you can is essential to expediting the software testing process as a whole [10]. Software businesses created Test Case Prioritization (TCP) strategies in an attempt to lower testing expenses without sacrificing product quality. Regression testing is an ongoing process that is part of software maintenance and is done in conjunction with software product review.. The phrase "testing phase" here refers to the initial regression testing, subsequent regression testing, and so forth, as well as the modifications that are concentrated on the regression testing. Instead of using source code information, which can occasionally make it difficult for testers to uncover flaws in software products, the quality of the requirements can be utilised to find error-prone test cases for the essential criteria. In the end, software testing determines whether the system complies with the requirements [4]. Considering this, increasing test efficiency and customer satisfaction may be achieved by linking test cases to requirement attributes. Test suites are frequently saved by software engineers so they can utilise them again during regression testing. Due to time and resource constraints, it might not be possible to finish all of the test cases during each iterative testing cycle [11,12]. Because of this, testers might wish to organize the test cases in a way that prioritizes particular criteria and runs the higher-priority instances before the lower-priority ones. Methods for ranking test cases for regression testing have been created as a consequence of this research. Leveraging requirement qualities is the foundation of these strategies. This study project's performance is evaluated using f1-score indications, recall, accuracy, and precision. Most of the times, individual requirements are interdependent rather than independent. One criterion may be dependent on another, such as in the case of the

inclusion relation. The functional requirements of the modules typically involve a number of sub-functional requirements. When all of the minor requirements have been tested and found to pass, a functional requirement is considered to pass the test [4, 11].

TCP techniques set up and carry out a test suite which contains test cases to achieve their prioritizing goals [4]. To increase software testing's efficiency, TCP is made to adapt test instances according on predetermined criteria. Different test cases have varying levels of fault detection and requirement coverage. It improves the fault detection rate or requirement coverage rate by giving test cases with a high capacity to identify faults a higher priority. The effect that a flaw has on either the creation or operation of any programme determines how serious the flaw is. It is the extent to which a flaw affects the application. Consequently, determining fault severity is crucial for the next regression testing. The development of a test case prioritising method that took into consideration requirement correlations and fault severity was the main objective of this research project.

In order to determine how requirements are interdependent, the main goal of this work is to develop a method that ranks test cases according to correlations between requirements and fault severity. To prepare raw datasets, train the model, and assess whether the model's performance is satisfactory. To classify the defect severity by reviewing various literatures [1]. In this paper the label data is taken to employ supervised machine learning technique. The attributes for the study are each test case's dependence value, fault severity, number of faults, fault number, and requirement coverage. The dataset consists of two categories: positive test cases and negative test cases. Positive test cases indicate the presence of defects and are considered highly significant, while negative test cases are test cases with fewer faults found and are of lesser value. Experts fill up the label for each test case according to its significance. By consulting several literatures on fault severity. The dataset is preprocessedusing Natural Language Toolkit (NLTK) NLP techniques, then, a vectorization method known as TF-IDF is applied for feature extraction. Four machine learning classification algorithms are then trained, and after classification is completed, the classification model is ranked to produce an ordered set of test instances.

2. Related Works

In this section, Credit is given to earlier work on machine learning-based prioritization techniques and test case prioritization. Current methods differ from ours in terms of both the fundamental ideas and the artefacts used (i.e., either merely using a portion of our technique or using different data). Unlike our innovative approach, no other methods can evaluate or leverage requirement correlation in test case prioritization using machine learning. The relevant work is presented in accordance with the artifacts used in the following.

Xiaofang Zhang et al. [11] proposed prioritizing test cases based on various testing requirements, priorities, and test case costs. They factored in the requirement's unpredictability, fault-proneness, and customer-assigned priority, among other considerations. They developed a metric to quantify the proportion of "priority-met units of testing requirement per unit test case cost." Their focus was on different methods for ranking test cases according to their costs and testing specifications.

R. Krishnamoorthi et al. [13] suggested prioritising new and regression test cases for systems based on demand analysis. They used six requirements' factors—implementation difficulty, requirement changes, priority of customers, application flow, usability, and impact of faults —to prioritise the system test cases. Their prioritisation method increased the rate of severe fault identification and has been validated with numerous student and industrial projects. In comparison to [11], it adds more variables to the test case prioritising process.

According to needs and risk criteria, Praveen Ranjan Srivastva and his colleagues [14] suggested prioritising test cases. They investigated two prioritising criteria as part of their research. The first is the order in which customers, developers, and managers have ranked their requirements. The degree and importance of risk variables that are included in the requirements make up the second factor. They determined the total value after determining the value for each need based on the two considerations. Test cases connected to a need with a higher total factor value should be prioritised, according to the analysis.

There is still a potential that the priority test case will not be successful even though they were taken into consideration as a requirement risk element. In [15], Rayapureddy Kalyani recommended ranking test cases

according to requirement clustering. Prior to assigning priorities, they combined the requirements for the study and used clustering as a machine learning technique. Map the requirements to test cases after grouping them. It is simple to prioritise the test cases because they were mapped to the requirements once the test cases had been chosen. Prioritizing test cases for object-oriented applications by using a clustering strategy was advised by Dharmveer Kumar Yadav [16]. By using a k means clustering technique to a test scenario, they conducted research. A cluster of information about test case coverage may be created. For tests that look for errors, one cluster can be built. They simply arranged the test cases in a cluster using an intra-cluster prioritization technique such that the most important test case could be chosen first from the cluster.

Omdev Dahiya [32] introduced the effective Ant Colony and Particle Swarm Optimization Hybrid Technique (APHT) to rank test cases based on the importance of requirements.The proposed method considered four variables: the perceived difficulty of the code implementation by the developer, the importance of the requirements as provided by the customer, the impact of faults, and changes in the requirements. After computing each value, the required factor value (RFV) was calculated by summing the four factor values and then dividing the total by four. A requirement with a higher RFV indicates greater complexity and a higher likelihood of containing errors.

In 2015, Yiting Wang and his colleagues [33] proposed an effective strategy for prioritizing test cases based on the severity of faults. Their approach involves identifying the root cause of each error and assessing its impact on software functionality. The flaws are then categorized into four groups: fatal, serious, general, and minor. Test cases involving fatal flaws are given the highest priority.

In our survey, Khatib Syarbini et al. [34] demonstrate that the number of publications on test case prioritisation has continued to expand in recent years. However, code analysis for priority computation remains the primary emphasis of most test case prioritisation techniques. Tonella and his colleagues [35] developed a case-based ranking mechanism for test case prioritization, utilizing attributes such as statement coverage and code complexity. However, the code was not accessible. Learning for prioritising was suggested by Busjaeger and Xie et al. [36], This method also uses test age, test-failure history, code coverage, failure history, and failure history to train the model. Therefore, whitebox testing could make advantage of this model.

As noted above, numerous academics presented cost-based, requirement-based, and coverage-based test case prioritization approaches in relation to various artifacts. Most of the offered methods are computed utilising hard computing algorithms. Consequently, those methods have challenges with precision. White-box testing has already benefited from the application of machine learning. Catal has surveyed a variety of defect prediction methods that can be used after code analysis [37]. A white-box technique, on the other hand, necessitates code access, for example, to examine updated code fragments for their test relevance. Due of the absence of code access. Numerous studies have also suggested prioritising black-box regression test cases according to various criteria, including the quantity of requirements, the severity of defects, the length of the execution, etc. Whenever a study suggested TCP, the traceability between artefacts— employing machine learning approaches, for example, the relationship between test failures and test cases and the reliance between requirements —was not taken into consideration. The machine learning-based TCP technique that is being offered is unique in that it considers correlations between the requirements, the traceability of errors in terms of their severity, and test cases.

3. Proposed Model

A test management system stores data for test case prioritization. Each test case in hypotheses is associated with at least one requirement, such as the functionality of a specific feature. Failures detected by a test case should be linked similarly. According to our method, a test expert must select an initial set for the ML computation. The expert to select several "positive test cases," or test cases that are highly significant, frequently used, or significant for another reason for the current iteration being tested. The expert provides a set of unsuccessful test cases as an add-on to this step (Fig. 1). As the specific utility has not altered for some varieties or is typically safe, for example, these are of little consequence.

The methodologies prioritized test cases based on requirement correlation and fault severity. Features for each test case are identified using the completed requirement correlation matrix, which serves as input for the learning algorithms. To do this, several tasks were carried out, including data pre-processing, Feature selection, numerical vectorization of data and assessing the model utilising various machine learning classification techniques.

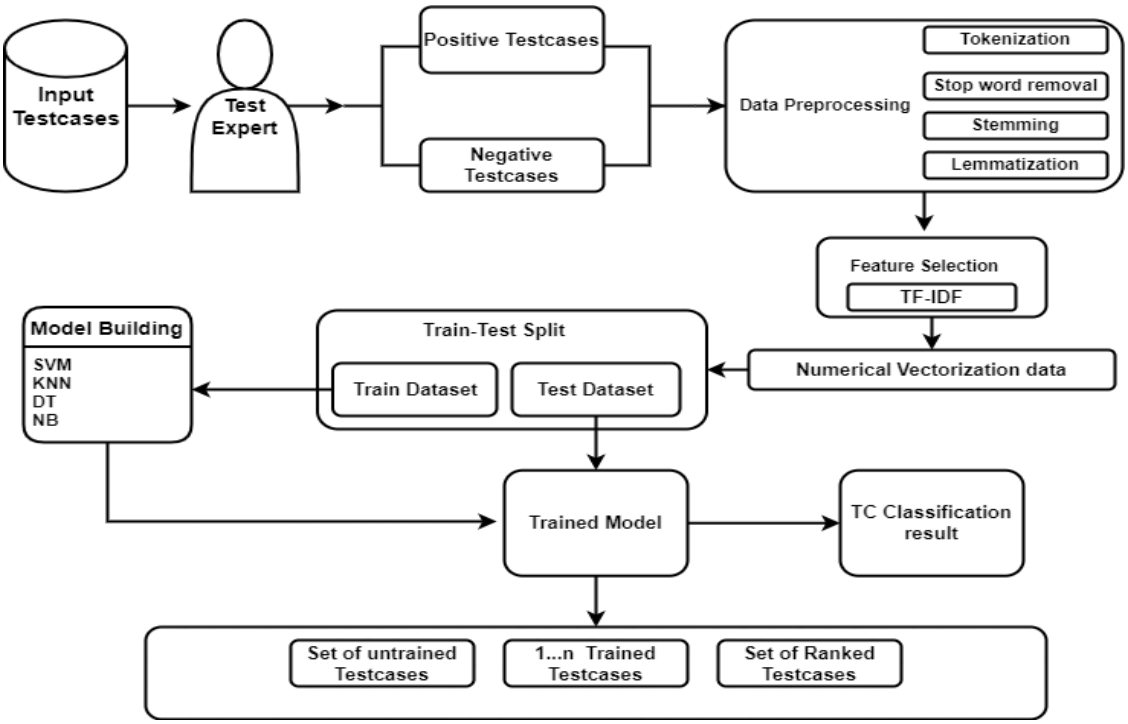

Figure 1: Proposed framework for Test Case Prioritization

Several preprocessing techniques include converting text to lowercase, breaking it into tokens, removing punctuation and stop words, applying stemming and lemmatization. In this study, the feature selection technique employed is Term Frequency-Inverse Document Frequency (TF-IDF). Following preprocessing, the cleaned data undergoes feature selection, where text representation is transformed into numerical format using TF-IDF. Once features are selected, creating a model to classify test cases becomes straightforward. Evaluation of the model aligns with the algorithm utilized for model generation. The primary goal of this study is to prioritize test cases.

4. Methods

To achieve the specific objectives, experimental research is adopted to investigate the causal link between one or more independent factors and the dependent variables. The requirement correlations and fault severity are the two independent variables because our research selected the test cases according to these two factors. When aresearcher wants to identify relationships between variables that indicate cause and effect, they should do experimental study. Nevertheless, causal inference in experimental studies faces notable constraints, and the specific experimental design employed greatly influences the interpretations feasible regarding trial results [38].

The objectives were achieved by several continuous tasks. In the context of textual data processing, establishing the environment and development setup, preparing the dataset, performing dataset preprocessing, and converting the preprocessed data into a numerical or vector space format are essential steps to ensure the data is interpretable by the model.To produce ordered test cases, the data is collected using machine learning methods and sent them to a ranked classification model.

4.1 Data Collection

One of the key duties in a variety of research projects is the collection and preparation of data. The first stage of doing the experiment is to gather the relevant information from different sources. Supervised machine learning

methods is used to prioritise a test case, to arrange and categorise the data since the model chooses test cases determined by the correlation of requirements and the severity of faults., the number of requirements with exposed defects is obtained from the organization. To make it straightforward to determine the relationship between criteria, the research data was gathered from an E-College system, a single subject system (*http://ir.bdu.edu.et/handle/123456789/14444)*. This system was chosen because it included a variety of different subsystems, including a student information system, a system for tracking income and expenses, a system for managing libraries and inventories, a system for managing human resources, a system for managing exams, and a system for tracking attendance. The E-College system contained fifty-four (54) errors and one hundred seventy-four (174) prerequisites. The system testers compile the discovered flaws, with 174 requirements and 1000 test cases.

4.2 Data Analysis

In order to achieve the desired outcome, it's essential to analyze the acquired data. A team of specialists discussed and completed the interdependencies among the requirements. Utilize a correlation matrix to identify the connections between these requirements. A training set needs to be chosen by a testing expert for the machine learning algorithm. These experts will select a set of test cases that are notable, commonly used, or pertinent for another purpose in the current version under testing [4]. To aid this process, the experts provided a range of unsuccessful test scenarios. While their relevance may be limited, it's left to the tester to determine the importance of specific test cases, particularly in cases where a feature hasn't significantly changed across multiple versions or the risk is low. Test cases are developed to assess each functional requirement.

From the correlation matrix, the features for each test instance are extrapolated, which then fed into the classification algorithms. The test case description, how many parents have high severe faults (NPWHSF), how many parents have medium severe faults (NPWMSF), how many parents have less severe faults (NPWLSF), how many siblings have high severe faults (NSWHSF), how many siblings have medium severe faults (NSWMSF), how many children have high severe faults (NCWHSF), and how many children have medium severe faults (NSWLSF) Based on the provided dependence. There could be numerous test cases for a single criterion. The percentage of requirements that correlate to test cases is known as "requirement coverage" (RC). How many problems are discovered or not discovered in the test cases is referred to as "fault coverage" (FC). The average severity rating of the defects identified in the test case is determined by the fault priority (FP) feature. By analysing several literatures,a consistent numerical value for each fault's severityis determined. Assign six (6) when a high severity flaw must be rectified in full before publication, four (4) when a medium severity flaw must be of typical relevance, and two (2) when a less severe flaw must be cosmetic [39, 33]. For instance, a test case might contain two faults of high severity, one fault of medium severity, and zero faults of lower severity. Fault priority (FP) = ((2*6) + (1*4) + (0*2))/3, which is 5.333.

A labelled dataset is necessary for the supervised machine learning algorithm to train. The experts filled in the label for each test case as positive or negative depending on their significance. The results of prioritising are significantly influenced by the quality of the training data. When choosing training data, certain points of view should be taken into consideration [40]. Each test case should, in theory, be related to at least one need, such as the operation of a certain feature. A test case should be used to identify any flaws, and those flaws should then be connected appropriately. Since our research is experimental, the experiment needs reliable data for training and testing our findings. Nevertheless, the classifier cannot acquire raw data on its own, it needs to be pre-processed. The pre-processing jobs are ones that get the classifier ready. NLP techniques are used like lower case, tokenization, stemming, lemmatization, etc. to lessen repetition and ambiguity between words.

Machine learning algorithms are chosen after preprocessed data. Support vector machines, K Nearest- Neighbor (KNN), Nave Bayes (NB), and decision trees. Since these four machine learning methods are the most widely used classifiers and perform well with minimal datasets. As a result, the learning outcome is fed into a ranked classification model to produce test cases that are either ordered or prioritised according to their priority value. The setting up of sample datasets is shown in the table 1. Where P stands for Positive and N for Negative in the label.

Table 1: Sample datasets

TC_ID	TC_Description	NPWHSF	NPWMSF	NPWLSF	NSWHSF	NSWMSF	NSWLSF	NCWHSF	NCWMSF	NCWLSF	RC	FC	FP	Label
TC1	valid username and password	0	0	0	0	0	0	1	2	1	1	1	2	P
TC2	invalid username and password	0	0	0	0	0	0	1	1	0	1	0	2	N
TC3	valid username and invalid password	0	0	0	0	0	0	0	1	1	1	0	4	N
TC4	Invalid username and valid password	0	0	0	0	0	0	0	0	0	1	0	0	N
TC5	Valid username, new password, old password and confirm password	0	0	1	1	0	0	0	2	0	1	1	6	P

4.3 Model Building

4.3.1 Pre-processing of prepared dataset phase

Pre-processing is the procedure used to transform the unprocessed dataset into one that can be predicted and analysed. Pre-processing plays a big part in creating better models.To have clean data for creating our model, pre-processing entails several procedures, including tokenization, lower casing, stop words removal, stemming, and lemmatization [28]. The procedures are used to reduce dimensionality. These phases are utilised to optimise vectorization since each word or term in the word representation phase functions as an axis or dimension. For this work, thousand test cases are created that corresponded to their relevant features. Test case descriptions were then developed in text from using those features.

4.3.2 Tokenization

Tokenization is the process of breaking up raw data into sentence or word segments, with each segment being referred to as a token. The NLTK tool words tokenize was utilised in this experiment to do word tokenization (Figure-2).

	TC_ID	TC_Description	Label
0	TC1	['check', 'the', 'results', 'by', 'entering', ...	negative
1	TC2	['check', 'the', 'results', 'by', 'entering', ...	negative
2	TC3	['check', 'the', 'results', 'by', 'entering', ...	negative
3	TC4	['check', 'the', 'results', 'by', 'entering', ...	negative
4	TC5	['check', 'the', 'responses', 'when', 'usernam...	negative

Figure 2: Sample dataset after word tokenization

4.3.3 Lowercasing

At this step of the pre-processing process, all letters in the data are changed to lowercase letters. Every identical word will have the same value once the data has been vectorized via lowercasing (Figure 3). For instance, while having various writing styles, the words "RESULT," "Result," and "result" are the same; nonetheless, all three words will be presented as "result" when the lowercasing pre-processing technique is used.

	TC_Description	Label
0	check the results by entering valid username a...	negative
1	check the results by entering invalid usernam...	negative
2	check the results by entering invalid usernam...	negative
3	check the results by entering valid username ...	negative
4	check the responses when username and password...	negative

Figure 3: Sample dataset after lower case conversion

4.3.4 Stop word removal

Stop words are common phrases representing the most frequently used terms in a language. They are words that can be removed from a sentence without significantly altering its meaning or comprehension. Removing stop words can enhance the performance of a model as it allows the focus to be on learning the essential words rather than these common terms. Additionally, eliminating stop words can expedite the training and testing processes of the model, especially when dealing with limited datasets.

```
In [9]: stopwords.words('english')
Out[9]: ['i',
         'me',
         'my',
         'myself',
         'we',
         'our',
         'ours',
         'ourselves',
         'you',
         "you're",
         "you've",
         "you'll",
         "you'd",
         'your',
         'yours',
         'yourself',
         'yourselves',
         'he',
         'him',
```

Figure 4: English stop word examples

The dataset appears as follows once the stop words have been removed:

	TC_ID	TC_Description	Label
0	TC1	check results entering valid username password	negative
1	TC2	check results entering invalid username password	negative
2	TC3	check results entering invalid username valid ...	negative
3	TC4	check results entering valid username invalid ...	negative
4	TC5	check responses username password empty press ...	negative

	TC_ID	TC_Description	Label
0	TC1	check results entering valid username password	negative
1	TC2	check results entering invalid username password	negative
2	TC3	check results entering invalid username valid ...	negative
3	TC4	check results entering valid username invalid ...	negative
4	TC5	check responses username password empty press ...	negative

Figure 5: Sample dataset without stop words

4.3.5 Stemming

Stemming is a vital step in transforming a derivative or inflected word into its root or base form. For instance, it allows matching nouns such as "checking," "checked," and "checks" to their core word, "check." Stemmers are software tools designed for this purpose. Similar to lowercasing, stemming helps decrease the variation in vectorization values for identical words.

	TC_ID	TC_Description	Label
0	TC1	check result enter valid usernam password	negative
1	TC2	check result enter invalid usernam password	negative
2	TC3	check result enter invalid usernam valid password	negative
3	TC4	check result enter valid usernam invalid password	negative
4	TC5	check respons usernam password empti press log...	negative

Figure 6: Sample dataset after stemming

4.3.6 Lemmatization

Lemmatization and stemming are related techniques. The process of returning inflected words to their base structure is alike in both methods. However, the difference lies in ensuring that the root word, or lemma, remains a valid part of the language. It's worth noting that some stemmed words may not be spelled correctly due to the removal of prefixes and suffixes from the words.Lemmatization is necessary because when it does, it complies with a few requirements but does not confirm that the word was correctly stemmed. WordNetLemmatizer is utilised from the NLTK package, which uses a variety of lemmatizers. A large, publicly accessible lexical database called Wordnet is utilised to build organised semantic links between words in English. Also, it features the earliest and most widespread lemmatization for English nouns.

	TC_ID	TC_Description	Label
0	TC1	check result entering valid username password	negative
1	TC2	check result entering invalid username password	negative
2	TC3	check result entering invalid username valid p...	negative
3	TC4	check result entering valid username invalid p...	negative
4	TC5	check response username password empty press l...	negative

Figure 7: Sample dataset after performing lemmatization

4.4 Model Learning using word representation and Feature selection

After cleaning and lemma word preparation, the further tasks are feature selection, training, validation, and testing. Word representation and feature selection both employ TF-IDF. The vectorized data is subsequently subjected to machine learning methods to categorise it into positive or negative classifications.

Word representation, or text vectorization, refers to the conversion of preprocessed textual content into a format understandable by machines. Many different techniques can be used to vectorize data. By eliminating terms that are not as important for analysis and so lowering the input dimensions, the model building process can be made simpler.

Using CountVectorizer involves simply tallying the frequency of each word in the document, whereas Tf-idf assigns a score, and Word2vec generates a vector for individual words. As a result, the tf-idf method is more suitable for small datasets. TF-IDF calculates the importance of a word in a document by multiplying the

frequency of the word in the document by its average inverse document frequency across the entire document collection [28]. The following formula is utilized to compute the TF-IDF score for the word t which is inside the document d from the document collection D:

$$tf - idf(t, d, D) = tf(t, d).\ idf(t, D) \quad \text{(Eq1)}$$

Many machine learning classification algorithms are fed the vectorized data after each vectorization on our pre-processed dataset. Following the completion of the two-class classification, the classification model underwent evaluation to prioritize test examples and determine the most effective classification method. SVM, KNN, Naive Bayes, and Decision Tree classifiers were employed to classify the data, primarily due to their frequent utilization and effectiveness with small datasets.

4.5 Model Training

A. SVM

Support Vector Machine (SVM), a supervised machine learning technique, aids in both classification and regression tasks by identifying an optimal boundary between potential outcomes. SVM employs various kernels such as linear, polynomial, Gaussian, and Radial Basis Function (RBF) to transform data. Initially designed for binary classification, SVM now extends to multiclass classification scenarios. In multiclass classification, SVM seeks to find a line that separates the dataset's points in two-dimensional space into distinct classes. Even in cases where the data isn't linearly separable, SVM's kernel function facilitates the transformation of data into a higher-dimensional feature space, enabling classification. This involves determining a strategy for dividing the classes and adjusting the data to create a hyperplane as the separator.

B. K-Nearest-Neighbour (KNN)

KNN computes a value by considering the labels of neighboring instances and their distances from each other. The parameter "k" specifies the number of neighbors to be considered when determining the class of an unknown instance. In our approach, we utilize Euclidean distance with a k value of 5, as it yields optimal results.

C. Decision tree

The decision tree is a commonly employed supervised learning method for both classification and regression tasks. It operates on a tree structure, with leaf nodes representing outcomes and root nodes, along with other nodes lacking leaf nodes, reflecting the attributes of the dataset.

D. Naïve Bayes

A Naive Bayes classifier, based on Bayes Theorem, assesses the probability that a particular record or data point belongs to each class by computing membership probabilities for each class. The class with the highest probability is considered the most probable outcome.

This can be mathematically stated as follows:

$$\mathbf{P(A/B)} = \frac{\mathbf{[P(B/A)P(A)]}}{\mathbf{P(B)}} \quad \text{(Eq2)}$$

Where:

P(A)- likelihood, P(B)- evidence, P(A/B)- posterior, P(B/A)-prior

3.5 Prediction phase

This study aimed to prioritize test cases based on the correlation of requirements and the severity of faults. The process involved two main stages: test case classification and ranking. Initially, test examples were classified as positive or negative, followed by ranking the classification model. During the prediction phase of test case classification, the model's ability to categorize events into suitable classes (positive or negative) was assessed. To make predictions, the model was supplied with new test instances, which underwent pre-processing and feature extraction similar to the training data. This involved steps such as data cleaning, stemming, and lemmatization.

Feature vectors were then generated from the pre-processed test case, and the vectored data, along with test case labels and hypotheses, were passed to the trained model. Utilizing knowledge gained from the training data, the model classified the new test case as either positive or negative.

4.6 Evaluation metrics

To gauge the trained model, evaluation metrics are needed. The classification accuracy modelis used for evaluation metrics. By dividing the overall number of forecasts (both correctly and wrongly classified) by the total number of accurate forecasts, accuracy may be calculated.

$$\text{Accuracy} = \frac{\text{number of correct classified instances}}{\text{Total number of instances}} \quad \text{.....} \quad \text{(Eq3)}$$

Classification accuracy provides a single value that summarises the entire performance of the model but does not provide specific information. F-score and confusion metrics are used. A true positive, true negative, false positive, and false negative are all performed by confusion metrics. The resilience and accuracy of a classifier are indicated by the harmonic mean of Precision and Recall, or F-Score (F-Measure). These can be determined mathematically in a similar manner. Precision of class the percentage of cases correctly classified as yes in a classification is equal to the total percentage of occurrences the classifier correctly classed as yes.

$$\text{Precision} = \frac{TruePositive(TP)}{TruePositive(TP)+FalsePositive(FP)} \quad \text{.....} \quad \text{(Eq4)}$$

$$\text{Recall} = \frac{TruePositive\ (TP)}{TruePositive(TP)+FalseNegative(FN)} \quad \text{.....} \quad \text{(Eq5)}$$

The F-measure accounts for both measurements and is the harmonic-mean (average of rates) of precision and recall.

$$\text{F measure} = \frac{2*TP}{2*TP+FP+FN} \quad \text{.....} \quad \text{(Eq6)}$$

5. Result Analysis

To train our model, a laptop with an x64-based processor, an Intel(R) Core (TM) i5-4200M CPU clocked at 2.50GHz, and 4GB of internal RAM is used. Anaconda environment installation was carried out with Python 3.10. This environment management system, which is free and open-source, consists of many Python distributions with hundreds of open-source packages and free community support.

After establishing the working environment and installing the necessary libraries, tests on the dataset are carried out. Pre-processing the prepared data is the first step of the experiment.To transform textual data into numerical values, which are necessary for machine learning comprehension. Only textual information that has been numerically encoded or vectorized may be understood by machine learning algorithms.Features using the traditional TF-IDF method is chosen.

Table 2: Preprocessing tasks and their method of application

Preprocessing tasks	Employed methods
Tokenization	Word Tokenization (Word_Tokenize)
Lower casing	English small letters
Stop word removal	English stop word
Stemming	Porter Stemmer
Lemmatization	WordNet Lemmatize

Table 2 contains pre-processing tasks and their methods of applications are finished to produce the pre-processed and vectorized dataset. To achieve the intended goal, the most popular machine learning algorithms are used on the pre-processed and vectorized dataset, choosing 80% of the 1000 samples for training and 20% for testing [43].

Two distinct trials were conducted. Building a classification model for test cases was done in the first experiment, and ranking the classification model was done in the second.

5.1. Performance evaluations for test case classification:

Several trials using SVM, KNN, DT, and NB, four machine learning algorithms are used to classify the test instances as positive or negative. It examined several variables or traits to assess the models' efficacy.

5.1.1 Performance of proposed models by considering SF

Using TF-IDF vectorization algorithms on the cleaned dataset, the data's vectorized.Based onprecision, recall, and f1-score it produced various accuracy outcomes using the four algorithms on the vectorized data which is shown in (Table-3).

Table 3: Precision, Recall, and F1-Score result by considering severity of faults

Sl.no	Classifiers	Class	Precision	Recall	F1-Score
1	SVM	Positive	0.81	0.67	0.74
		Negatve	0.73	0.85	0.79
2	KNN	Positive	0.89	0.66	0.76
		Negatve	0.74	0.92	0.82
3	DT	Positive	0.75	0.79	0.76
		Negatve	0.78	0.75	0.76
4	NB	Positive	0.76	0.71	0.74
		Negatve	0.74	0.78	0.76

As seen in table 3 above, the SVM model's accuracy, recall, and F1-Score scores are reasonably high in percentage terms. It demonstrates that the model misclassifies test cases according to the severity of their flaws, classifying them as either positive or negative. Precision is defined as the ratio of correctly anticipated positive labelling to all predicted positive labelling. How many overall positive labels there are is a question that can be answered by precision metrics. It is possible to use (Eq4) to compute the precision value. Although certain test cases were projected to be positive for those whose real test case values are negative, the precision value for positives is 0.81**.**

The ratio of successfully completed test cases to fully anticipated test cases with a label positive is used to calculate recall or sensitivity. Even though certain test instances were classified as negative even though the true label was positive, the recall value for positive is 0.67. Negative classes operate in the opposite manner. The weighted average of recall and precision is known as the F1-Score. Therefore, both false positives and false negatives are considered while calculating this score. According to (Eq6), the positive class's F1-Score value is 0.74. The simplest logical performance metric is accuracy, which is simply the proportion of accurate forecasts to all other predictions. Our experiment had a 76% accuracy rate and incorporated the TF-IDF feature selector and SVM classifier while taking the severity of faults into consideration. The results demonstrate that test case prioritisation based on fault severity is a challenging task for the SVM classifier.

On the cleaned dataset, the vectorized value of the data is extracted using TF-IDF vectorization algorithms. Based on a variety of criteria, accuracy result is achieved using the vectorized data and the KNN algorithm. As seen in the above table, the KNN model produces results for precision, recall, and F1-Score with a reasonable percentage value. The model's accuracy while utilising the KNN classifier to determine fault severity is 80%. The experiment's findings show that, when compared to other classifiers, the KNN classifier performs well for prioritising test cases based on the severity of defects. The precision, recall, and F1-Score outcomes for the DT model are moderately high percentage values, as indicated in the table above. The model's accuracy while employing the DT classifier based on fault severity is 77%. The experiment's findings show that the DT classifier performs poorly when test cases are prioritised according to the severity of defects. The precision, recall, and F1-

Score outcomes for the NB model are moderately high percentage values, as indicated in the table above. The model's accuracy while utilising the NB classifier based on fault severity is 75%. The experiment's findings show that the NB classifier performs poorly when test cases are prioritised according to the severity of defects.

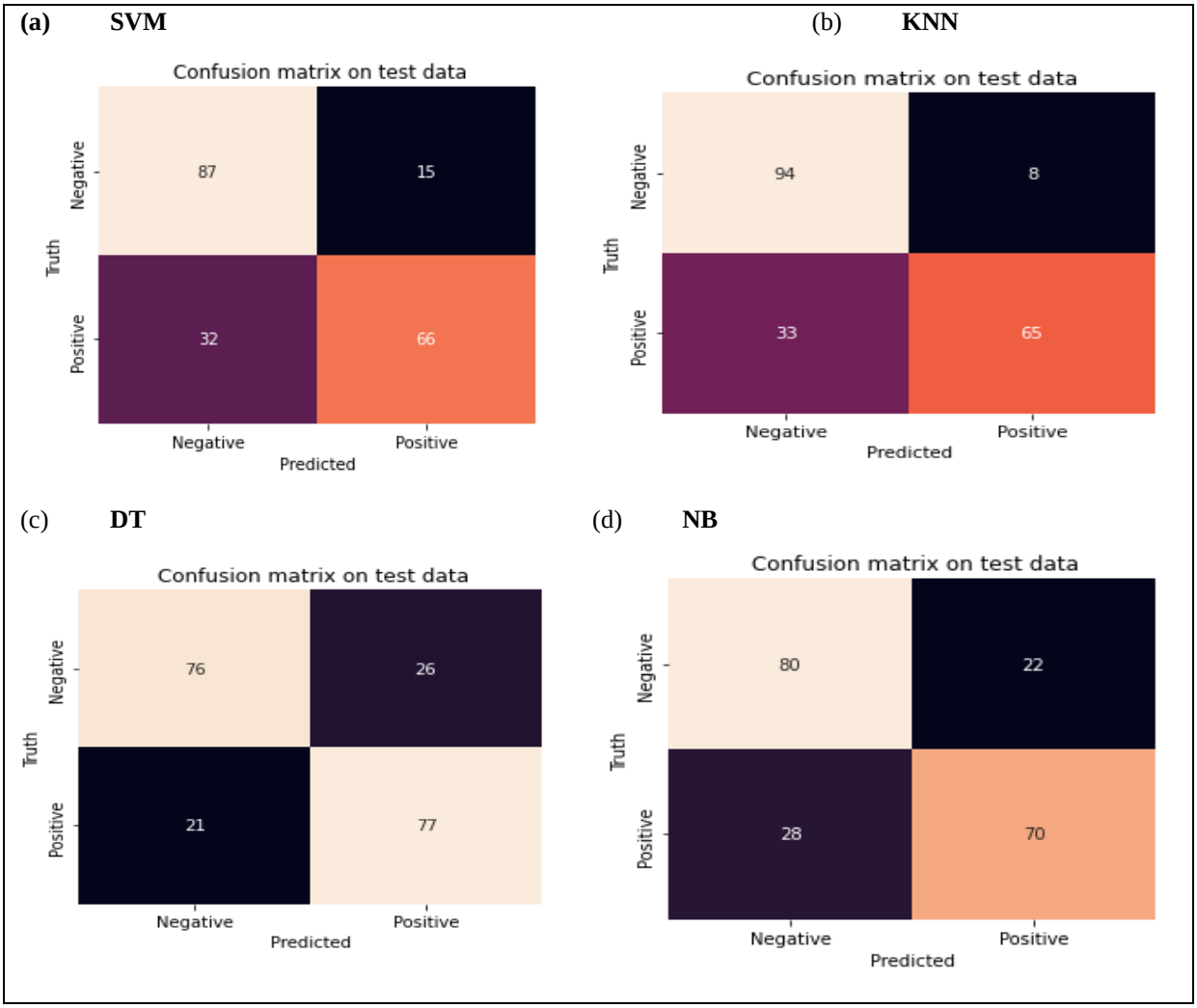

Figure-8: Confusion matrix by considering severity of faults

The confusion matrix displays the test data's TP, FP, TN, and FN values, as shown in figure 8(a). Out of 200 test data samples, the model identified 66 test cases as positive, which means that TP = 66. 15 test cases were deemed positive by the model, even though the actual 32 test cases classified as negative even though their actual class is positive, FN=32; 87 test cases classed as expectedly negative class, TN=87; and class of those test cases is negative, it suggests FP=15. Out of a total of 200 test instances, 47 are incorrectly identified as such by the model, whilst 32 test cases belong to the positive class. The actual class of 15 test cases is negative yet the model is classed as negative, and vice versa. As a result, conclusion is drawn that the SVM model performs poorly when it comes to classifying and ranking test instances according to the severity of defects.

5.1.2. Performance of proposed models by considering RCR

The SVM model, when considering necessary correlation as indicated in Table 4, achieves high precision, recall, and F1-Score outcomes. With consideration of needs correlation, the SVM classifier attains an accuracy rate of 91%. This suggests that prioritizing test cases based on the correlation between requirements and fault severity leads to strong performance by the SVM classifier. Similarly, the KNN model demonstrates high precision, recall, and F1-Score values, with the KNN classifier achieving a 93% accuracy rate when based on requirements correlation. These findings indicate that the KNN classifier performs effectively when prioritizing test cases

according to the closeness of requirements to fault severity. Moreover, the DT classifier, when utilizing requirements correlation, achieves an accuracy rate of 92%. This underscores the effectiveness of the DT classifier when prioritizing test cases based on the association between requirements and fault severity.

Table 4: Performance of the proposed models by considering RCR

Sl.no	Classifiers	Class	Precision	Recall	F1-Score
1	SVM	Positive	0.90	0.93	0.91
		Negatve	0.93	0.90	0.92
2	KNN	Positive	0.90	0.97	0.93
		Negatve	0.97	0.89	0.93
3	DT	Positive	0.90	0.94	0.92
		Negatve	0.94	0.90	0.92
4	NB	Positive	0.79	0.88	0.83
		Negatve	0.87	0.77	0.82

The model using the NB classifier based on requirements correlation has an accuracy rating of 83%. The results of the experiment demonstrate that when test cases are prioritised according to how requirements and fault severity are associated, the NB classifier performs well.

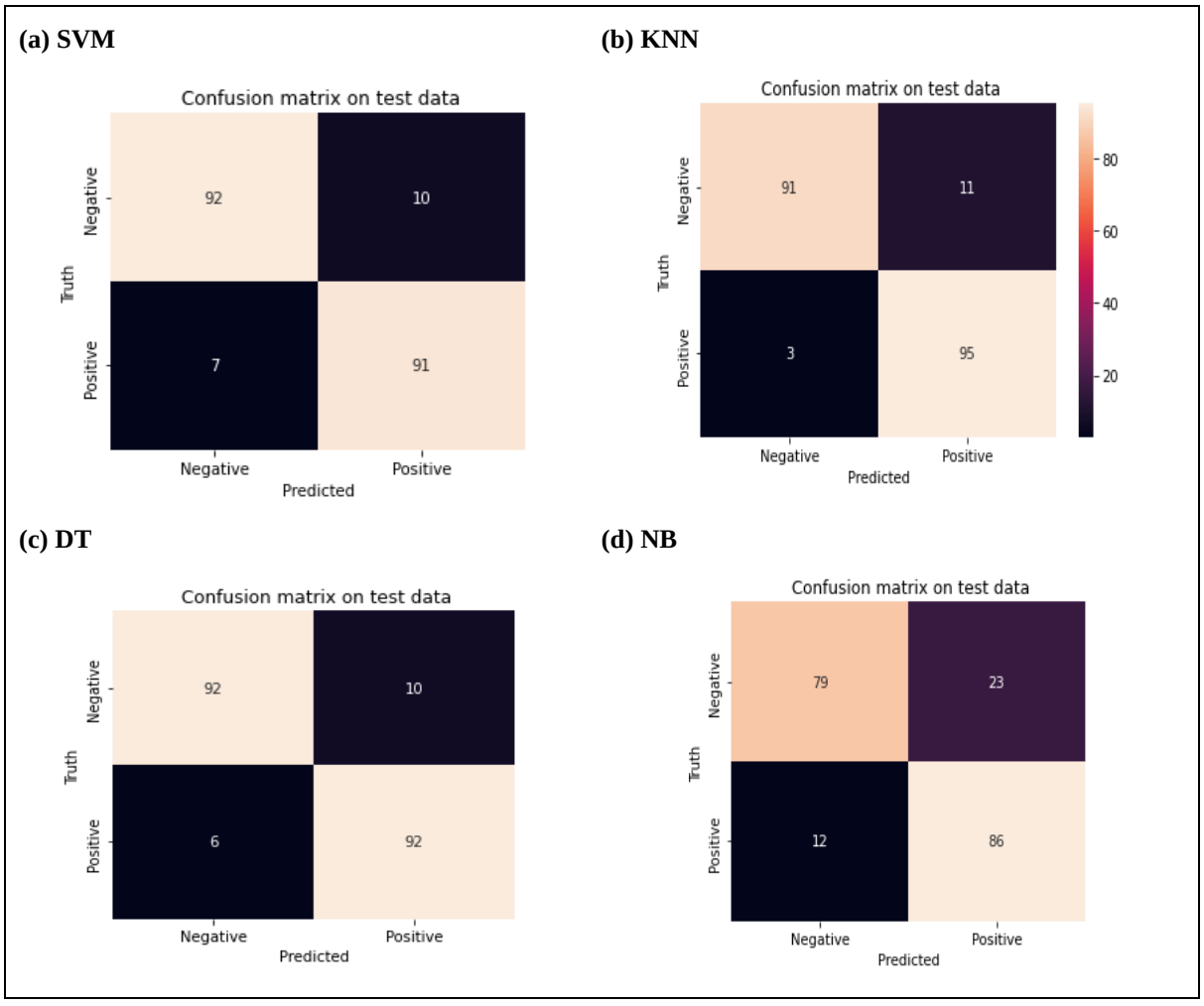

Figure 9: Confusion matrix by considering requirements correlation

Figure 9 illustrates the confusion matrix, which indicates the TP, FP, TN, and FN value of the test data. The model identified 91 test instances out of 200 test data as positive, which suggests that TP = 91. These test cases indeed belong to the positive class. FP=10 is implied by the model's classification of 10 test instances as positive even if their real class is negative. 93 test cases were classified as the expected negative class, which is TN=93; Despite having a positive class, 7 test cases were assigned to the anticipated negative class; this equals FN=7. A total of 17 test instances out of 200 are incorrectly classified by the model, meaning that 10 test cases are in a negative class despite the model classifying them as positive, and 7 test cases belong to a positive class.

5.1.3 Performance of the proposed models by considering RCR and SF

The SVM model may generate results with a high percentage of precision, recall, and F1-Score thanks to the necessary correlation and fault severity parameters, as shown in table 5. When requirements correlation and fault severity are considered, the model's SVM classifier's accuracy is 94%. According to the experiment's findings, the SVM classifier works better when test cases are prioritised according to how closely requirements match up with fault severity. Based on requirements correlation and defect severity, the KNN classification model's accuracy is 93%. The results of the experiment demonstrate that when test cases are prioritised according to the correlation between requirements and fault severity, the KNN classifier works well. 90% of errors are successfully predicted by the model utilising the DT classifier based on requirements correlation and fault severity. The results of the experiment demonstrate that when test cases are prioritised according to the correlation between requirements and fault severity, the DT classifier performs well.

Table 5: Performance of the proposed models by considering RCR and SF

Sl.no	Classifiers	Class	Precision	Recall	F1-Score
1	SVM	Positive	0.92	0.95	0.94
		Negatve	0.96	0.91	0.94
2	KNN	Positive	0.91	0.95	0.93
		Negatve	0.95	0.91	0.93
3	DT	Positive	0.89	0.91	0.90
		Negatve	0.91	0.89	0.90
4	NB	Positive	0.78	0.72	0.75
		Negatve	0.75	0.80	0.78

The NB classifier, based on the correlation between requirements and fault severity, achieves a 77% accuracy rate according to the experiment results. These findings indicate that when test cases are prioritized based on this correlation, the NB classifier performs poorly. The confusion matrix depicted in Figure 10 reveals the true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN) values of the test data. The model correctly identifies 94 test instances out of 200 as positive (TP=94), accurately classifying them as belonging to the positive class. It incorrectly classifies 8 test instances as positive (FP=8) when their actual class is negative. Furthermore, it correctly identifies 94 test cases as belonging to the negative class (TN=94) and misclassifies 4 test cases as negative (FN=4) when they actually belong to the positive class. Overall, the model misclassifies 12 test cases out of 200, 8 as negative when they should be positive and 4 as positive when they should be negative. Consequently, the SVM model outperforms other classification and ranking methods based on the correlation between requirements and defect severity.

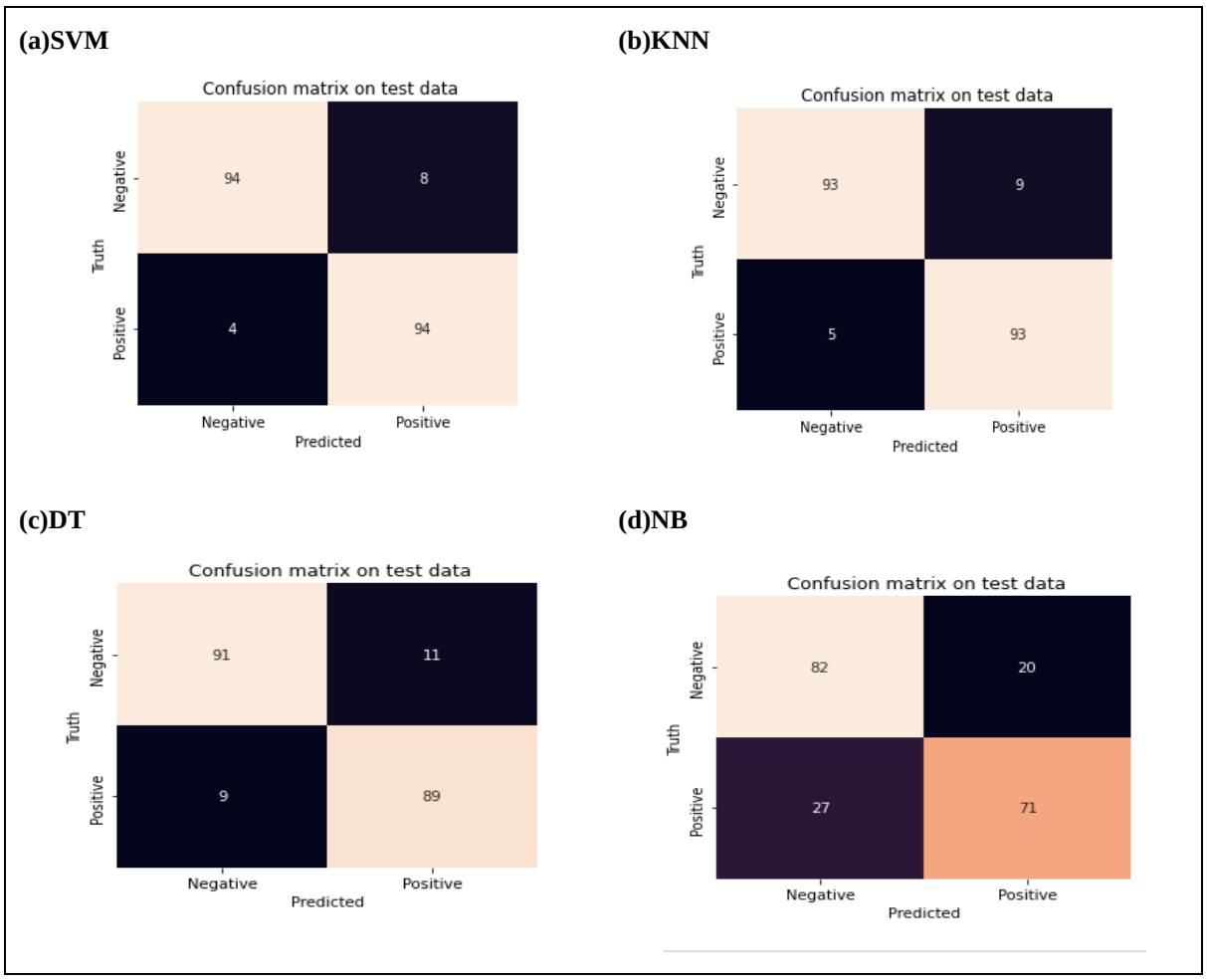

Figure 10: Confusion matrix by considering RC and SF

5.2 Comparison of different classifier results

To determine the best performing technique, four machine learning methods, namely SVM, KNN, DT, and NB, as well as the TF- IDF feature extractors are used.

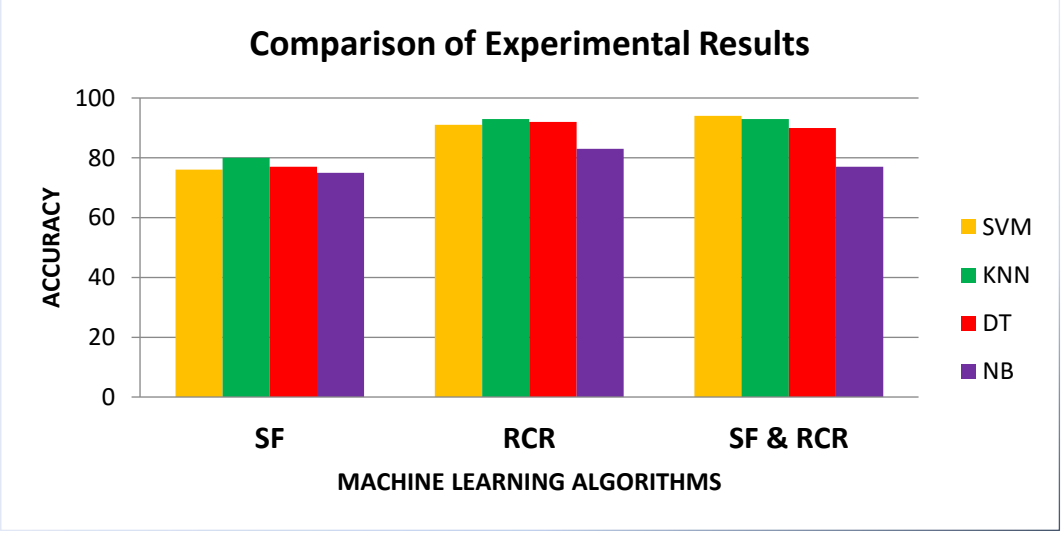

Figure-11: Comparison of ML Classifier accuracy results

The experimental results depicted in Fig. 11 offer insights from two perspectives. Firstly, they relate to the characteristics utilized for grouping test cases. When trials were conducted solely based on the severity of errors, all machine learning algorithms, with the exception of KNN, exhibited poor performance in categorizing test cases. Conversely, when considering requirements correlation, all machine learning algorithms outperformed the model's performance based on defect severity. However, they showed subpar performance when compared to the model's performance based on the correlation between requirements and defect severity. With the exception of NB, all machine learning approaches accurately identified test cases based on the correlation between requirements complexity and the severity of issues. Test examples were organized based on their likelihood of being assigned to a specific class. If a model exhibits low classification accuracy, it may not provide the correct probability value for each test scenario. According to this interpretation, all machine learning algorithms demonstrated poor performance when prioritizing test cases based solely on fault severity, moderate performance when prioritizing based on requirements correlation, and excellent performance when prioritizing based on both requirements correlation and fault severity. Secondly, the results highlight a set of highly effective machine learning algorithms for categorizing test results as positive or negative. The graph illustrates that SVM classifiers outperform other methods, while KNN performs better for small datasets with a high number of features, and SVM excels with a limited number of features.

Using a built-in test case classification model, evaluate the priority of test cases.

After evaluating the categorization of test examples using the TFIDF vectorizer and various machine learning methods, the performance with the SVM classifier has been enhanced. The model assesses the likelihood of new test cases to prioritize them accordingly. Test cases with a high probability value of the positive class are considered more important and are arranged at the top of the list, while those with a high probability value of the negative class are placed lower. Conversely, if the probability value of the negative class for each test case is evaluated, the test cases are ranked in ascending order. As previously mentioned, the model prioritizes new test cases based on the probabilities of either positive or negative categorization. Figure 13 displays the projected class and associated probability value for the negative class for each test case. For example, the probability of TC17 being categorized as a negative class is extremely low (0.02922), indicating its crucial importance compared to other test cases. Conversely, TC16 has a high (0.99919) likelihood of being categorized as a negative class, suggesting its relatively lower significance compared to other test cases.

TC_ID	TC_Description	NPWHSF	NPWMSF	NPWLSF	NSWHSF	NSWMSF	NSWLSF	NCWHSF	NCWMSF	NCWLSF	RC	NHSF	NMSF	NLSF	FC	FP
TC1	Check the results by entering valid username a...	0	1	0	1	2	1	1	1	0	1	0	1	1	0	2.000000
TC2	Check the results by entering invalid usernam...	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0.000000
TC3	Check the results by entering invalid usernam...	0	0	0	0	0	0	0	1	0	1	0	2	1	1	3.333333
TC4	Check the results by entering valid username ...	0	3	2	0	1	0	4	1	1	1	2	1	0	2	5.333333
TC5	Check the responses when username and password...	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0.000000
TC6	Check result by entering invalid Date of Birth...	1	2	0	1	0	0	4	0	0	1	1	0	0	1	2.000000

TC7	Check result by entering invalid Program and v...	2	0	0	1	3	0	2	0	1	1	0	0	0	0	0.000000
TC8	Check result by entering invalid Religion and ...	0	2	1	1	1	3	4	0	1	1	1	0	1	2	2.666667
TC9	Check result by entering invalid Mobile Number...	1	0	0	0	0	0	0	2	0	0	0	0	0	0	0.000000
TC10	Check result by entering invalid Email and val...	0	0	0	1	0	0	2	0	1	2	0	2	0	2	2.666667
TC11	Check result by entering invalid Admission Dat...	1	1	2	1	0	2	1	2	2	0	1	2	2	0	6.000000
TC12	Check result by entering invalid Class Departm...	1	0	0	1	0	0	0	0	1	0	0	0	0	2	0.000000
TC13	Check the results by entering valid username, ...	0	0	0	0	0	1	0	0	0	1	0	1	1	0	2.000000
TC14	Check the results by entering invalid username...	0	0	0	0	0	0	0	0	0	1	0	0	0	1	0.000000
TC15	check the results by entering different value ...	0	3	0	2	0	0	1	0	0	1	0	0	1	1	0.666667
TC16	Check result by entering valid value for all a...	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0.000000
TC17	Check result by entering invalid Full Name and...	1	3	1	1	2	2	0	2	1	2	1	1	1	2	4.000000
TC18	Check result by entering invalid Gender and va...	1	0	0	0	0	0	1	0	1	1	0	0	0	2	0.000000
TC19	Check result by missing Student Photo and vali...	1	1	0	0	0	0	0	0	0	0	1	1	3	1	5.333333
TC20	Check result by entering invalid Place of Birt...	2	0	0	1	0	0	0	0	1	0	0	0	0	1	0.000000

Figure 12: Sample unordered test cases

```
[('positive', 'TC1', 0.22268129993527258), ('negative ', 'TC2', 0.9963771370725718), ('negative ', 'TC3', 0.9214178380878394),
('positive', 'TC4', 0.40574706517330217), ('negative ', 'TC5', 0.9991928501244268), ('positive', 'TC6', 0.05200220735671364),
('positive', 'TC7', 0.10665772212860762), ('positive', 'TC8', 0.08160020518967845), ('negative ', 'TC9', 0.8922769484158938),
('negative ', 'TC10', 0.6899590289089887), ('positive', 'TC11', 0.3036179365801241), ('negative ', 'TC12', 0.9645155904590877),
('negative ', 'TC13', 0.9949872993137512), ('negative ', 'TC14', 0.999067148849595), ('positive', 'TC15', 0.42554237474867984),
('negative ', 'TC16', 0.9991928501244268), ('positive', 'TC17', 0.029226028318003833), ('negative ', 'TC18', 0.981044682011946
7), ('negative ', 'TC19', 0.6046589717387056), ('negative ', 'TC20', 0.9239905895393881)]
```

Figure 13: New test cases' prediction classes and their negative class probabilities

The model arranges the new unordered test cases after calculating the negative and positive class probability values for each test case. The test case is run earlier the lower the determined negative class probability value. The model's positive class probability value and negative class probability values were used, respectively, to order the items. Figure 14 shows the test cases arranged based on their negative class probability scores.

```
[('TC17', 0.029226028318003833),
 ('TC6', 0.05200220735671364),
 ('TC8', 0.08160020518967845),
 ('TC7', 0.10665772212860762),
 ('TC1', 0.22268129993527258),
 ('TC11', 0.3036179365801241),
 ('TC4', 0.40574706517330217),
 ('TC15', 0.42554237474867984),
 ('TC19', 0.6046589717387056),
 ('TC10', 0.6899590289089887),
 ('TC9', 0.8922769484158938),
 ('TC3', 0.9214178380878394),
 ('TC20', 0.9239905895393881),
 ('TC12', 0.9645155904590877),
 ('TC18', 0.9810446820119467),
 ('TC13', 0.9949872993137512),
 ('TC2', 0.9963771370725718),
 ('TC14', 0.999067148849595),
 ('TC5', 0.9991928501244268),
 ('TC16', 0.9991928501244268)]
```

Figure 14: Ordered test cases with their negative class probability value

6. Discussion

Software testing is a critical stage in the software development life cycle, aimed at assessing the effectiveness, accuracy, and completeness of software products. Regression testing, a subset of software testing, involves retesting a software system following modifications, such as updates or new versions. Due to resource constraints, only a subset of test cases is typically executed for each release, making it challenging to identify scenarios likely to uncover most issues. Thus, prioritizing test cases becomes crucial for regression and other testing methodologies. Test case prioritization can be based on various factors, including severity of errors and the correlation between requirements. In this study, NLP techniques were employed to preprocess the dataset due to its textual nature, including steps like lowercasing, stemming, lemmatization, stop word removal, and punctuation removal. Subsequently, the TF-IDF feature extraction technique was utilized to vectorize the cleaned datasets. Four machine learning algorithms (SVM, KNN, NB, and DT) were employed to develop and evaluate the model. Various combinations of factors were analyzed to determine the algorithm with the best performance, such as SVM based on fault severity, SVM based on requirement correlation, etc. SVM exhibited superior performance in classifying test cases across these combinations, achieving an accuracy rate of 94%. The model can effectively prioritize test cases even when they are not initially ordered. KNN generally performs well in terms of fault severity, while the other three classifiers exhibit varying performance levels depending on whether they assess fault severity alone, requirement correlation alone, or both factors together.

7. Future Scopes

Due to the large quantity of datasets, both positive and negative labels are employed. However, by expanding more datasets and labelling them with more than two labels. Performance on a range of datasets can be assessed using machine learning techniques. To get excellent performances on various machine learning approaches, large datasets of test cases are prepared for classification into negative and positive classesCreating a substantial dataset enables comparison of the results between deep learning and machine learning. Several literary works categorize the severity of faults. The severity level of fault is categorised as high, medium, and low in certain publications. Degree of the error can be categorised as essential, significant, moderate, minor, or cosmetic by some persons. The three severity classifications of high, medium, and low are utilised in this work. The researchers can prioritise test cases using the five severity levels and compare the performance of the three severity levels.

References

[1] Ma, T., Zeng, H., & Wang, X. (2016, May). Test case prioritization based on requirement correlations. In *2016 17th IEEE/ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD)* (pp. 419-424). IEEE.

[2] Wyrich, M., & Bogner, J. (2024). Beyond Self-Promotion: How Software Engineering Research Is Discussed on LinkedIn. *arXiv preprint arXiv:2401.02268*.IEEE. (1990).*IEEE_SoftwareEngGlossary.pdf* (p. 84).

[3] Amrita, & Gupta, P. (2021). Test Case Prioritization Based on Requirement. *Lecture Notes in Networks and Systems*, *204*(61170044), 309–314

[4] Cheng, R., Zhang, L., Marinov, D., & Xu, T. (2021). Test-case prioritization for configuration testing. *ISSTA 2021 - Proceedings of the 30th ACM SIGSOFT International Symposium onSoftware Testing and Analysis*, *069*(average 821), 452–465.

[5] Dongoor, S. P. (2019). *Selecting an appropriate Requirements Based Test Case Prioritization Technique*. *March*.

[6] Vescan, A., Chisalita-Cretu, C., Serban, C., &Diosan, L. (2021). On the use of evolutionary algorithms for test case prioritization in regression testing considering requirements dependencies. In *AISTA 2021 - Proceedings of the 1st ACM International Workshop on AI and Software Testing/Analysis, co-located with ECOOP/ISSTA 2021* (Vol. 1, Issue 1).Association for Computing Machinery.

[7] Quadri, S. M. ., & Farooq, S. U. (2010). Software Testing – Goals, Principles, and Limitations. *International Journal of Computer Applications*, *6*(9), 7–10

[8] Nešić, D., Krüger, J., Stănciulescu, Ş., & Berger, T. (2019). Principles of feature modelling. *ESEC/FSE 2019 - Proceedings of the 2019 27th ACM Joint Meeting European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 62–73.

[9] Rabbi, K., Mamun, Q., & Islam, M. D. R. (2015). An efficient particle swarm intelligence based strategy to generate optimum test data in t-way testing. *Proceedings of the 2015 10th IEEE Conference on Industrial Electronics and Applications, ICIEA 2015*, 123–128.

[10] Zhang, X., Nie, C., Xu, B., & Qu, B. (2007). Test case prioritization based on varying testing requirement priorities and test case costs. *Proceedings - International Conference on Quality Software*, *Qsic*, 15–24.

[11] Kundu, D., Sarma, M., Samanta, D., & Mall, R. (2009). System testing for object-oriented systems with test case prioritization. *Software Testing Verification and Reliability*, *19*(4), 297–333.

[12] Krishnamoorthi, R., &Sahaaya Arul Mary, S. A. (2009). Requirement based system test case prioritization of new and regression test cases. *International Journal of Software Engineering and Knowledge Engineering*, *19*(3), 453–475.

[13] Srivastva, P. R., Kumar, K., &Raghurama, G. (2008). Test case prioritization based on requirements and risk factors. *ACM SIGSOFT Software Engineering Notes*, *33*(4), 1–5.

[14] Kalyani, R., Sai Mounika, P., Naveen, R., Maridu, G., & Ramya, P. (2018). Test Case Prioritization Using Requirements Clustering. *International Journal of Applied Engineering Research*, *13*(15), 11776–11780.

[15] Yadav, D. K., & Dutta, S. K. (2019). Test case prioritization using clustering approach for object oriented software. *International Journal of Information System Modeling and Design*, *10*(3), 92–109.

[16] Harrold, M. J. (2000). Testing: A roadmap. *Proceedings of the Conference on the Future of Software Engineering, ICSE 2000, July 2000*, 61–72.

[17] Tahat, L., Korel, B., Koutsogiannakis, G., & Almasri, N. (2017). State-based models in regression test suite prioritization. *Software Quality Journal*, *25*(3), 703–742.

[18] Almog, D., & Heart, T. (2009). What is a test case? revisiting the software test case concept.*Communications in Computer and Information Science*, *42*(June 2014),13–31.

[19] Craig, R. D., &Jaskiel, S. P. (2002). Systematic Software Testing, Artech House. In *Inc., Norwood, MA*. Artech House.

[20] Zhang, C., Chen, Z., Zhao, Z., Yan, S., Zhang, J., & Xu, B. (2010). An improved regression test selection technique by clustering execution profiles. *Proceedings – International Conference on Quality Software*, 171–179.

[21] Lin, C. T., Tang, K. W., Wang, J. S., & Kapfhammer, G. M. (2017). Empirically evaluating Greedy-based test suite reduction methods at different levels of test suite complexity. *Science of Computer Programming*, *150*(May), 1–25.

[22] Yoo, S., & Harman, M. (2012). *Survy_Mark_Yoo_2010.Pdf*. *March 2010*, 67–120.

[23] Rosero, R. H., Gomez, O. S., & Rodriguez, G. (2017). Regression Testing of Database Applications under an Incremental Software Development Setting. *IEEE Access*,*5*(September), 18419–18428.

[24] Lin, C. T., Tang, K. W., & Kapfhammer, G. M. (2014). Test suite reduction methods that decrease regression testing costs by identifying irreplaceable tests. *Information and Software Technology*, *56*(10), 1322–1344.

[25] Anish, P. R., Balasubramaniam, B., Cleland-Huang, J., Wieringa, R., Daneva, M., & Ghaisas, S. (2015). Identifying Architecturally Significant Functional Requirements. *Proceedings - 5th International Workshop on the Twin Peaks of Requirements and Architecture, TwinPeaks 2015*, 3–8.

[26] Henningsson, K. (2005). *A fault classification approach to software process improvement.*

[27] Sarkar, D. (2019). Text Analytics with Python - A Practitioner's Guide to Natural Language Processing. In *Text Analytics with Python*.

[28] Shahi, T. B., & Sitaula, C. (2022). Natural language processing for Nepali text: a review. *Artificial Intelligence Review*, *55*(4), 3401–3429.

[29] Dey, A. (2016). Machine Learning Algorithms: A Review. *International Journal of Computer Science and Information Technologies*, *7*(3), 1174–1179.

[30] Binkhonain, M., & Zhao, L. (2019). A review of machine learning algorithms for identification and classification of non-functional requirements. *Expert Systems with Applications: X*, *1*.

[31] Dahiya, O., & Solanki, K. (2021). An efficient APHT technique for requirement-based test case prioritization. *International Journal of Engineering Trends and Technology*, *69*(4), 215–227.

[32] Wang, Y., Zhao, X., & Ding, X. (2015). An effective test case prioritization method based on fault severity. *Proceedings of the IEEE International Conference on Software Engineering and Service Sciences, ICSESS*, *2015-Novem*, 737–741.

[33] Khatibsyarbini, M., Isa, M. A., Jawawi, D. N. A., &Tumeng, R. (2018). Test case prioritization approaches in regression testing: A systematic literature review. *Information and Software Technology*, *93*, 74–93.

[34] Singh, A., Singhrova, A., Bhatia, R., & Rattan, D. (2023). A systematic literature review on test case prioritization techniques. *Agile Software Development: Trends, Challenges and Applications*, 101-159.

[35] Busjaeger, B., & Xie, T. (2016). Learning for test prioritization: An industrial case study. *Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, *13-18-Nove*, 975–980.

[36] Catal, C. (2011). Software fault prediction: A literature review and current trends. *Expert Systems with Applications*, *38*(4), 4626–4636.

[37] Tanner, K. (2002). Experimental research designs. *Research Methods for Students, Academics and Professionals*, 125–146.

[38] Mary, S. A. S. A., & Krishnamoorthi, R. (2011). Time-aware and weighted fault severity based metrics for test case prioritization. *International Journal of Software Engineering and Knowledge Engineering*, *21*(1), 129–142.

[39] Gao, K., Khoshgoftaar, T. M., & Napolitano, A. (2012). A hybrid approach to coping with high dimensionality and class imbalance for software defect prediction. *Proceedings - 2012 11th International Conference on Machine Learning and Applications, ICMLA 2012*, *2*, 281–288.

[40] Khanna, M. (2022). A Systematic Review of Ensemble Techniques for Software Defect and Change Prediction. *E-Informatica Software Engineering Journal*, *16*(1), 220105.

[41] Portugal, I., Alencar, P., & Cowan, D. (2018). The use of machine learning algorithms in recommender systems: A systematic review. *Expert Systems with Applications*, *97*, 205–227.

[42] Mece, E. K., Binjaku, K., & Paci, H. (2020). The Application Of Machine Learning In Test Case Prioritization - A Review. *European Journal of Electrical Engineering and Computer Science*, *4*(1)..

[43] Ralph, P. (2015). Developing and evaluating software engineering process theories. *Proceedings-International Conference on Software Engineering*, *1*, 20–31.