Machine Learning Driven Software Test Case Selection

by

Victor Cheruiyot

A project report submitted in conformity with the requirements
for the degree of Master of Science in Information Technology

Department of Mathematical and Physical Sciences
Faculty of Graduate Studies
Concordia University of Edmonton

MACHINE LEARNING DRIVEN SOFTWARE TEST CASE SELECTION

VICTOR CHERUIYOT

Approved:

Supervisor: Saha Baidya Date

Committee Member: Date

Dean of Graduate Studies: Rorritza Marinova, Ph.D. Date

Machine Learning Driven Software Test Case Selection

Victor Cheruiyot
Master of Science in Information Technology
Department of Mathematical and Physical Sciences
Concordia University of Edmonton
2021

Abstract

After each software is developed, tests are carried out to identify defects that are subsequently deleted. But testing a non-trivial software completely is a really complex endeavor. Therefore, testing the software using critical test cases is important. Thus test case selection aims to minimize unnecessary test data, which is important for determining testing methods. In this study, we have created an approach for regression testing based on machine learning software tests. We initially clean the data and preprocess it in order to construct the approach. The category data is then converted to its numerical value. To generate a bag of characteristics for text features such as test case title, we apply natural language processing. For test cases selection we evaluate various machine learning models. The results of the experiments show that machine learning-based models can eliminate the need for domain experts to select test cases manually.

Keywords

Machine learning, classifiers, confusion matrix, test case selection.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Introduction

Software testing is a quality control activity that concentrates on detecting defects and then they are removed. During software or web development or at its completion different test cases are conducted. These tests are essential to assess the effectiveness of the software. However, it is impossible to completely test any nontrivial module or system because it suffers from both theoretical and practical perspectives. Theoretically, it suffers from a halting problem: it's impossible to write a program that tests whether every program halts in a finite amount of time. Practically, executing all test cases involves enormous time and cost. Hence, it's crucial to create a test suite that is a subset of test cases for testing. To facilitate a smooth test case selection process, test case prioritization is employed, in this technique, each test case is assigned a priority. Priority is set according to some criterion and test cases with the highest priority are scheduled first which are important from the user's perspective based on the frequency of usage, criticality, and probability of failure. This research work is limited to develop a machine learning-based test case selection strategy for regression testing. Regression testing retests software that has been changed or extended by new features during software development.

1.2 Background

Machine learning has paved the way for great inventions, among those areas that have benefited from machine learning agility are test case selection and prioritization. The ability to train and test data using machine learning has provided the ability to train and test, test case data makes it achieve good results over time. Apart from that their many classifiers to implement each with different performance accuracy.Retecs

by Spieker, Gotlieb, and Mossige, a new approach for automatically learning test case selection and prioritizing in Continuous Integration (CI) that leverages reinforcement learning, is one of the ways that use machine learning. [1]. Based on supervised machine learning, Lachmann and Schulze proposed a novel technique for test case prioritizing for human system-level regression testing.[2]. Tonella, Avesani, and Susi presented case-based ranking, a test case prioritizing technique that leverages user expertise through a machine learning algorithm. (CBR) [3]. Among other studies that have been conducted. Thus it is evident that machine learning is the game-changer in test case selection and prioritization.

1.3 Problem Statement

To explore the efficacy of software test case selection using machine learning and deep learning algorithms. Thus shedding light to the fact that test case selection can be achieved using machine learning, not only through manual selction of cases but also through automation of test case selection.

1.3.1 Previous works

Several algorithms based on different techniques have been proposed for test case selection and prioritization. These techniques are placed into different categorize based on the method applied. The different methods are highlighted. Ant colony optimization is a technique for solving computational problems which can be reduced to finding good paths through graphs. Solanki, Singh, and Sandeep applied M-ACO (Modified Ant Colony Optimization) for test case prioritization [4]. Chen and Zhang adopted ant colony optimization (ACO) to build this prioritized pairwise interaction test suite (PITS) [5]. Panwar, Pradeep, and Singh suggested a Cuckoo Search (CS) algorithm followed by Modified Ant Colony Optimization (M-ACO) algorithm to conclude the test cases in an optimized order in a time-constrained environment [6]. A genetic algorithm is a technique used in optimization and search problems by relying on biologically inspired operators such as mutation, crossover, and selection. Nucci, Panichella, and Zaidman proposed a Hypervolume-based Genetic Algorithm, namely HGA, to solve the Test Case Prioritization problem when using multiple test coverage criteria [7]. Kumar and Sandip in their paper presented a technique that is based on Genetic algorithms (GA) for test case prioritization [8]. Machine learning approaches are methods that employ different machine learning techniques for test case selection and prioritization. Tonella, Avesani, and Susi In their paper proposed a test case prioritization technique that takes advantage of user knowledge

through a machine learning algorithm, case-based ranking (CBR) [3]. Moghadam used model-free reinforcement learning to build a self-adaptive autonomous stress testing framework [9]. Test suite minimization aims to reduce the size of test suites by removing redundant test cases. Jeffrey and Gupta proposed a new approach for test suite reduction. They said that as test suite minimization reduces fault detection effectiveness increases [10]. The regression testing technique is rerunning functional and non-functional tests to ensure that previously developed and tested software still performs after a change. Zengkai and Jianjun implemented Apros, a test case prioritization tool, and perform an empirical study on two real, non-trivial Java programs [11].

1.4 Contribution of this thesis

This study is based on a machine learning technique on test case selection.

- We conducted an extensive study for machine learning-based text case selection which allows choosing the best classifier for test case selection that offers state-of-the-art performance.
- The approach of this study of using natural language processing combining with classifier models to fit trained and tested, test case data paves the way to further study on this approach. With the results discussed the approach could help ease the process of test case selection.

1.5 Organization of this thesis

This thesis is organized into five chapters. Chapter 1 introduces the study topic. Chapter 2 reviews past studies through paper citations related to the topic in discussion. While chapter 3 explains the proposed architecture and expounds more on data, data cleaning, and textual classification. In chapter 4 results of classifier models considered are discussed along with model evaluation. The final chapter discusses the conclusion and future. And the last part of the paper highlights a bibliography of the cited papers.

Chapter 2

Literature Review

Numerous studies have been performed by researchers on test case prioritization and selection methods. These methods were categorized into several domains according to the nature of the study conducted by the researchers. In this paper, a couple of those research findings have been highlighted to shed light on the whole idea on the topic. The main aim of this is to gear towards a minimal or error-free process of prioritization and selection.

2.1 Ant colony optimization

M-ACO (Modified Ant Colony Optimization) determines the best test suite prioritization solution by altering the phenomenon utilized by natural ants to find and pick food sources. In their study, Solanki, Singh, and Sandeep proposed to compare and contrast the suggested m-ACO technique for test case prioritization with several current meta-heuristic strategies [4].

Suri et al. collaborated on the development of a hybrid test case selection technique. They offered a new strategy to lower the cost of regression testing by reducing the test case suite in their article. Their proposed method is based on BCO and GA ideas. The technique chooses a collection of test cases from the given test suite that will cover all of the previously found errors in the shortest amount of time[12].

To address the problem of test case prioritizing, Chen and Zhang designed and developed the Weighted Density Algorithm, a biassed covering array (WDA). To come up with a better method, they used ant colony optimization (ACO) to create this prioritized pairwise interaction test suite in their article (PITS). They presented four concrete test generation algorithms based on Ant System [5].

In their paper, Panwar, Pradeep, and Singh developed a Cuckoo Search (CS) algorithm followed by a Modified Ant Colony Optimization (M-ACO) algorithm to

finish the test cases in an enhanced order in a time-constrained environment. The CS algorithm was inspired by some species of cuckoos that exhibit constrained brood lethargy and lay their eggs in the nests of other host birds[6].

2.2 Genetic Algorithm

In their paper, Nucci, Panichella, and Zaidman discovered that AUC metrics are a bi-dimensional (simplified) version of the hypervolume metric, which is widely used in many-objective optimization. In order to solve the Test Case Prioritization problem while applying multiple test coverage criteria, they proposed a Hypervolume-based Genetic Algorithm or HGA. Analysis of techniques revealed that HGA is more cost-effective, improves Test Case Prioritization efficiency, and has a higher selective pressure when dealing with more than three criteria[7].

In their paper, Kumar and Sandip presented a technique for the priority of test cases based on genetic algorithms (GA). A genetic algorithm is a generative technique that takes natural evolution to find an optimal solution. In their study, a unique Genetic algorithm was employed for regression testing, that leverages the statement coverage technique to rank test cases. The results improve the effectiveness of algorithms by using the Average Percentage of Statement Coverage (APSC) metric. This prioritization technique produces the best results for prioritizing the test case[8].

In their paper Zhang, Wei and Huisen combined genetic algorithm with the coverage with test points to accomplish specific contributions in the field of prioritization of test cases, in particular for functional tests. First of all, APTC and its APRC C improvement presented two new test case priorities appraisals. Later, proposed a test case priority method based on the genetic algorithm which is designed for black-box test, representation, selection, crossover, and mutation. The experimental proposed method suggested can achieve results.[13].

In their paper, Yiling, Hao, and Zhang proposed a novel test-case prioritization approach for software evolution that first simulates real faults in software evolution using mutation faults on the difference between the early version and the later version, and then schedules the execution order of the test cases based on their fault-detection capability. They presented two models that are based upon statistics and likelihood models for the computation of fault detection. Experimental results show statistical model surpasses the probability one[14].

2.3 Machine learning approach

Alexandre, Aurora, and Vergilio emphasised that structural and fault-based criteria provide general measures for assessing test sets. Thus, in their work, they present an approach based on machine learning techniques for connecting test results from different testing techniques. The method divides test data into functional clusters. Following that, it generates classifiers (rules) relying on the tester's goals, that can be used for a range of functions, including test case selection and prioritisation. The paper also presents experimental evaluation results and illustrates such applications[16].

Susi, Avesani, and Tonella In their work developed a test case prioritizing method called case-based ranking, which uses a machine-learning algorithm to take advantage of human expertise (CBR). In the form of pairwise test case comparisons, CBR obtains only relative priority information from the user. In an iterative process, user input is combined with many prioritization indexes to enhance the test case ordering. As per preliminary results from such a case study, CBR outperforms earlier approaches[3].

In his research, Moghadam used model-free reinforcement learning to create a self-adaptive autonomous stress testing framework capable of learning the optimal policy for stress test case generation without the need for a model of the system under test. The experiment results indicated that the proposed smart framework can efficiently and adaptively generate stress test conditions for varied software systems without access to performance models.[9].

In their work, Spieker, Gotlieb, and Mossige describe Retecs, a new method for automatically learning test case selection and prioritizing in Continuous Integration (CI) to shorten the length between code changes and developer feedback on failed test cases. The Retecs technique employs reinforcement learning. It learns to prioritize error-prone test cases higher in a constantly changing environment where new test cases are created and obsolete test cases are deleted. Results demonstrate that reinforcement learning enables automatic adaptive test case selection and prioritization[1].

Schulze and Lachmann presented a unique supervised machine learning-based technique for test case prioritization for manual system-level regression testing in their research. They investigated their approach using two subject systems and measured the quality of prioritizing using the machine learning algorithm SVM Rank. In contrast to random order, the findings showed that the technique significantly improves the failure detection rate. Additionally, it can outperform a test expert's test case order. Also, adopting natural language descriptions improves the rate of finding defects[2].

2.4 Test suite minimization

Jeffrey and Gupta developed a new test suite reduction strategy. They argue that as test suite minimization decreases, fault detection efficacy increases. As a result, their approach employs additional coverage information about test cases, resulting in the retention of a few more test cases in the reduced test suite. The findings demonstrate that defect identification enhances reduced test suites without placing the size of the test suite at stake [10].

To enhance the test suite's defect detection rate, Kumar and Vivek introduced an integrated test case prioritizing approach. To rank test cases, three major parameters are considered: program change level (PCL), test suite change level (TCL), and test suite size (TS). The proposed method is tested on a variety of internal programs to ensure its accuracy. When contrasted to optimal prioritizing strategies, which always result in an upper bound of APFD values, the model findings are found to be very effective [17].

2.5 Regression testing technique

In their study, Salehie, Li, and Moore attempt to prioritize requirements-based regression test cases. To that aim, system-level testing in industrial environments focuses on two practical issues, addressing numerous goals in terms of quality, cost, and effort in a project, and employing non-code metrics in some instances due to the lack of precise code metrics. This study described Research In Motion's (RIM) goal-driven approach to prioritizing requirements-based test cases for these challenges. In order to identify metrics for prioritization, the Goal-Question-Metric (GQM) approach is used. Afterward, they reviewed the skills learned from using the goal-driven approach and conducting experiments, as well as a few research directions for the future[18].

Chauhan and Gupta worked to develop a hybrid approach to priority test cases. They, therefore, proposed a new hybrid technique. In the present paper clustering, the test cases and setting clusters first on the basis of the priorities of the requirements of clusters and the series of selections and priorities of test case levels decreased the number of test cases to a level that is manageable[19].

Jianjun Zengkai In their work, they proposed a new technique to priority test priority setting based on a study of the program structure to improve the rate of severe fault detection both for regression tests and non-regression tests. The approach is to assess the importance of testing for each module covered by test cases. As a concept proof, they implement Apros, a tool for prioritizing test cases, to study two true, non-trivial Java programs. The experimental results show their approach to

improve the rates of serious failure detection as a feasible technique[11].

Kim and Porter proposed the history-based test case selection and prioritization technique. This technique selects the test cases that should be conducted for the new version of the software based on information from earlier testing cycles. This method selects a subset of a test suite, and the test cases should be chosen based on historical data; it is commonly referred to as a "regression test selection approach [21]"

Chapter 3

Proposed Methodology

This chapter talks about the proposed machine learning test case selection architecture, it describes the process from data collection, data preprocessing to classifiers evaluation.

3.1 Proposed Architecture

The figure 3.1 below gives a detailed description of the whole process from collection of data, data processing, cross validation and finally evaluation of the classifiers.

- Data preparation preceeds this study the dataset for the study contains several features that can be used to determine the test cases to be executed.
- Data preprocessing involves feature selection through removal of unwanted features, not all features contained in the data is considered in test case selection. Thus redundant features are removed. Only five categorical features and one textual feature remain afte the process.
- Preprocessed data further undergoes more processing through natural language processing(NLP). NLP encompasses two processes text preprocessing and text feature generation.
- Machine learning proceeds NLP it encompasses two main processes, text feature selection, and cross-validation which is further divided into three more processes. Model generation, hyperparameter optimization, and model evaluation.

3.2 Dataset Preparation

For this study data related to test cases and Natural Language test case description is considered as an input to classification learning models to predict the selection

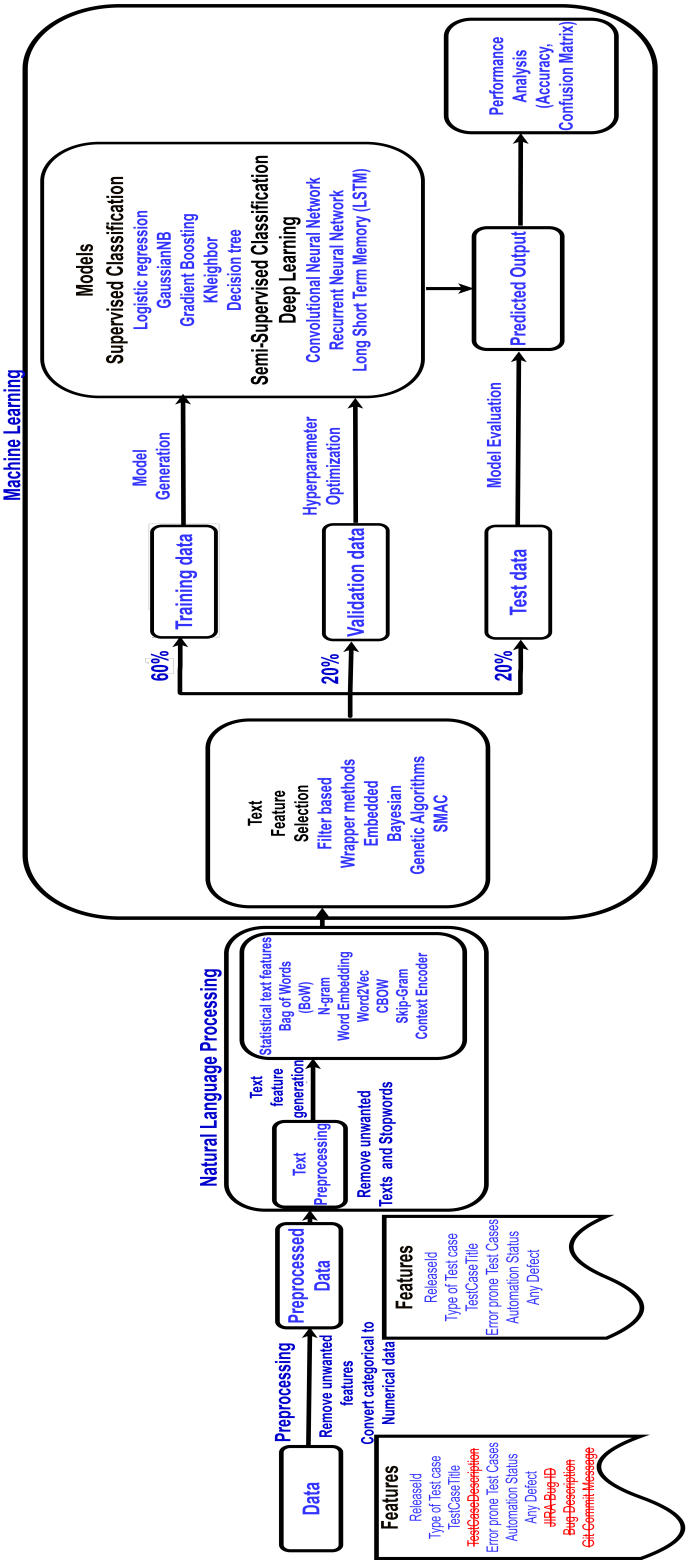

Figure 3.1: proposed architecture for machine learning test case selection

of test cases. For this proof of concept authorization, microservice test cases across four release cycles are considered as the test Data. This microservice is based on the Oauth2 standard which is widely used in the industry for authorization across systems/microservices.

3.2.1 Removal of unwanted features

The dataset contains several features to be considered in test case selection. The features include:

i Unique Identifier of Records.

ii Release Identification number.

iii Type of Test Case.

iv TestCaseTitle

v TestCaseDescription.

vi Error-Prone Test Cases.

vii Automation Status.

viii Any Defect

ix Bug ID.

x Bug Description.

xi GIT Commit Message

xii Target: binary classification of test case selection.

But not all these features are considered in the study, some of the features present in the dataset are redundant for the process of test case selection. These features are dropped during feature selection. Upon removal of unwanted features the dataset will consist of one textual feature **TestCaseTitle** and five categorical features:

i Release Identification number.

ii TestCaseDescription.

iii Error-Prone Test Cases.

iv Automation Status.

v Any Defect

All of this retained features are related to the selection of test cases i.e. 'Target' variable. This features will be utilized for training classifier models.

3.3 Natural Language Processing

Natural language processing (NLP) refers to computer systems that analyze, attempt to understand or produce one or more human languages [24]. NLP consists of two main processes, text preprocessing, and text feature generation.

3.3.1 Text Preprocessing

This process involves conversion of the textual feature (**TestCaseTitle**) into a sparse matrix of numerical features to create a corpus. The following NLP-based preprocessing tasks are carried out to create a corpus that is compatible with classifier models.

i Remove unwanted words: Remove irrelevant characters and words such as special characters and numbers to get a clean text for further processing.

ii Uppercase to lowercase transformation: Transform all uppercase letters to lowercase because upper and lower case letters have different ASCII codes.

iii Remove stopwords: Stopwords are usually the most common words in a language and are irrelevant in predicting the response variables.

iv Stemming words: Stemming is the process of reducing words to their stem, base, or root form. It is used to reduce the dimensions of Bag of Words features. Bag of Words (BoW) describes the occurrence of words within a document which involves a vocabulary of known words and a measure of the presence of known words.

3.3.2 Text Feature Generation

This process mainly involves the generation of statistical text features, which are discussed below.

3.3.2.1 Context encoder

Based on some context words, the algorithm attempts to predict the target word between them. This is achieved by first computing the sum of the embeddings of the context words by selecting the appropriate rows from W_0.

$$W_0, W_1, \in \mathbb{R}^{N \times d} \tag{3.1}$$

$$t \in \mathbb{R}^{K+1} \tag{3.2}$$

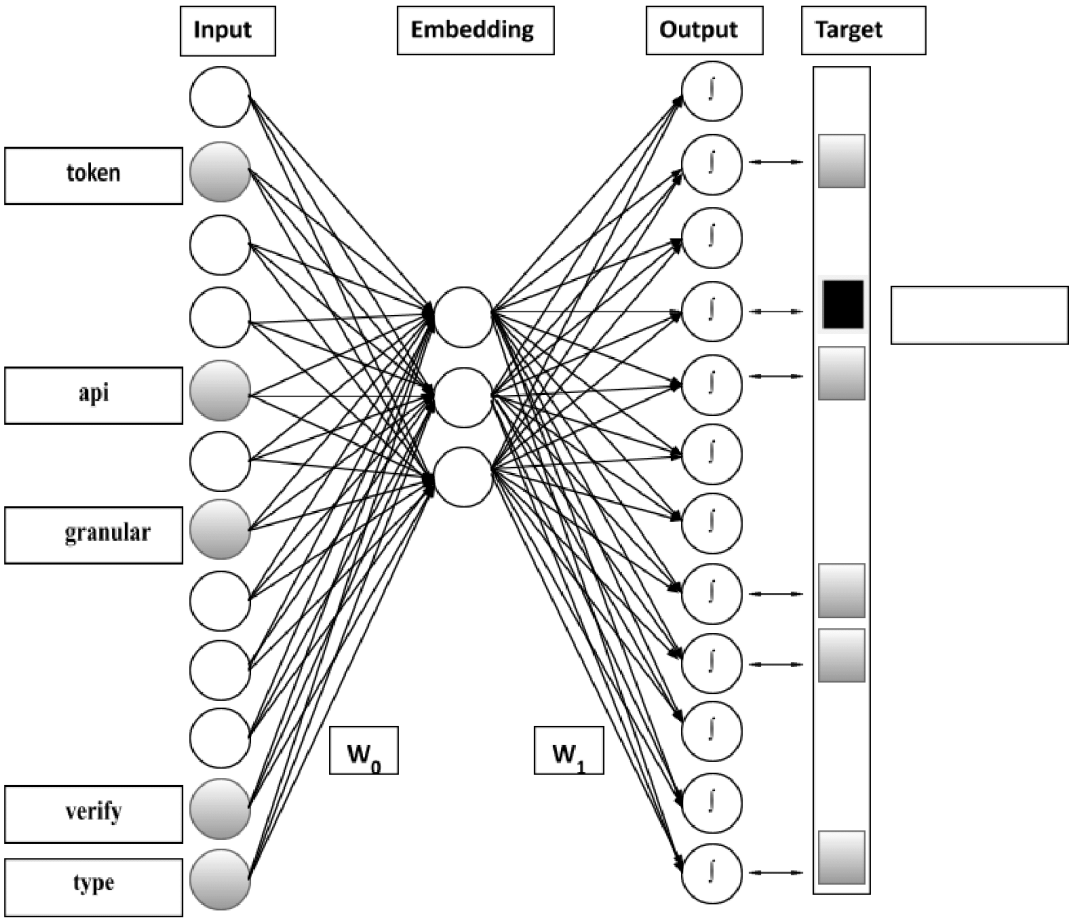

Figure 3.2: **Context encoder**

3.3.2.2 N-gram

N-gram modeling is used to identify features and analyze them, It is a contiguous sequence of items with length n. N-gram figure 3.3, shows words that are common for the test case selection process, whether appearing alone as 1-gram, 2-gram up to n-gram. This narrows down the understanding on how test cases are selected and which are the keywords in the process.

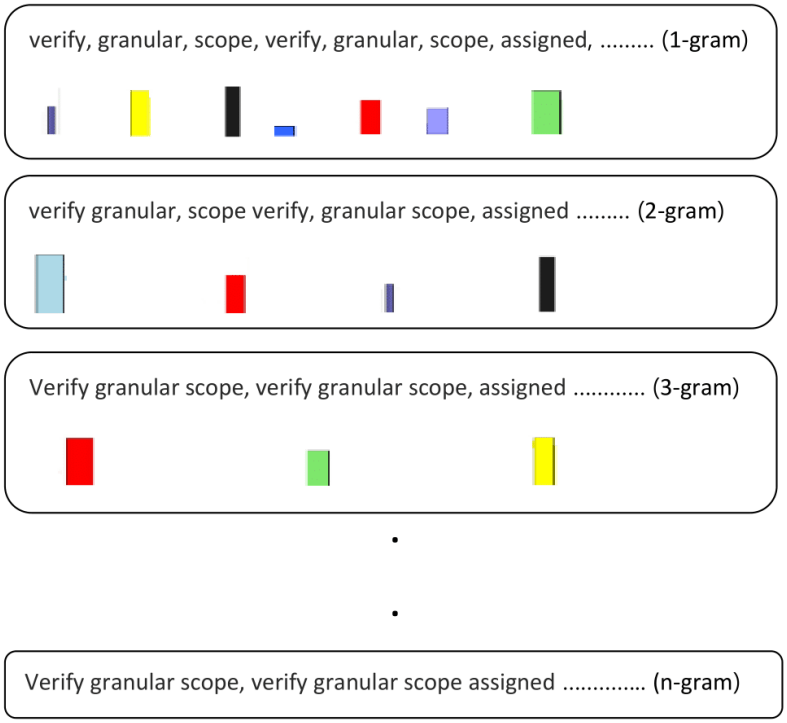

Figure 3.3: **N-gram**

3.3.2.3 Skip-gram model

Skip-gram uses word representations to predict words in a sentence, given a sequence of training words the objective of the Skip-gram model is to maximize the average log probability.

$$\frac{1}{\mathbf{T}}\sum_{\mathbf{t=1}}^{\mathbf{T}}\sum_{\mathbf{-c\geq j\geq c, j\neq 0}}\log_{\mathbf{p}}\left(\mathbf{w_{t+j}} \mid \mathbf{w_t}\right) \tag{3.3}$$

[26]. In figure 3.4 summation of weights of the collection of words in a test case dataset show how the words play a bigger role in how test are selected.

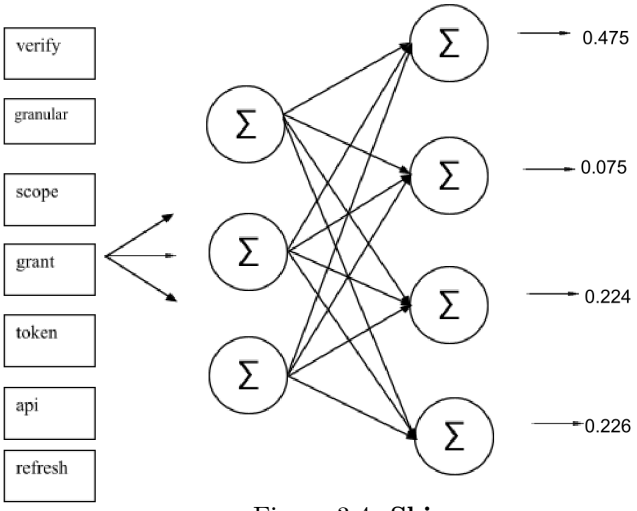

Figure 3.4: **Skip-gram**

3.3.2.4 Continous bag of words

A continuous bag of words is to use embeddings to train a neural network where the context is represented by multiple words for given target words [27]. In this approach, context words are used as inputs of a neural network and try to predict the target word. After training, the input weights are the final word embeddings in figure 3.5.

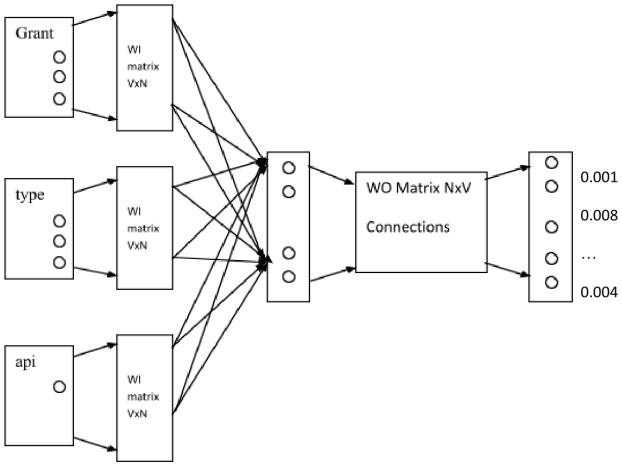

Figure 3.5: **Continous bag of words**

3.4 Machine Learning

Machine learning is an evolving branch of computational algorithms that are designed to emulate human intelligence by learning from the surrounding environment [9]. ML has to main processes text feature selection and cross-validation.

3.4.1 Clustering

Clustering is dividing data into many groups (clusters) such that data in the same clusters are more similar to other data in the same clusters than those in other clusters. The aim is to segregate clusters with similar traits and assign them into clusters.

3.4.2 Cross Validation

Cross-validation is a resampling procedure used to evaluate machine learning models on a limited data sample. Cross-validation is divided into three parts training data 60%, validation data 20%, and testing data 20%.

3.4.3 Hyperparameter optimazation

Hyperparameters are all the parameters that can be arbitrarily set before training data.Tuning of these hyperparameters through optimization can lead to finding the right combination of their values which can help find either the minimum (eg. loss) or the maximum (eg. accuracy) of a function.

3.4.4 Models(supervised collection)

3.4.4.1 Logistic Regression

This is a statistical model that uses a logistic function to model a binary dependent variable. It is used to determine if an independent variable affects a binary dependent variable. Has two potential outcomes.

3.4.4.2 Gaussian Naive Bayes

It conforms to a Gaussian normal distribution and deals with continuous data where continuous values related to each class are distributed according to Gaussian distribution.

3.4.4.3 Multinomial Naive Bayes

it is suitable for classification with discrete features. With a multinomial model, samples represent the frequencies with which certain events have been generated by

a multinomial $(p_1, \ldots, p_n)$ where p_i is the probability that event i occurs.

3.4.4.4 Gradient Boosting

Gradient boosting is a machine learning technique for regression and classification problems that produce a prediction model in the form of a set of weak prediction models.

3.4.4.5 K-Nearest Neighbors

KNN is a non-parametric classification that uses both classification and regression to solve problems where the output depends if it is used for classification or regression.

3.4.4.6 Decision Tree

Supervised Machine Learning works by splitting data according to some parameter. It uses a tree-like model where the branches to conclusions about the item's target value are represented in the leaves.

it consists of :

i Nodes: Test for the value of a certain attribute.

ii Branch: Correspond to the outcome of a test and connect to the next node or leaf.

iii Leaf nodes: Terminal nodes that predict the outcome represent class labels or class distribution.

3.4.4.7 Gaussian Process

It is a stochastic process that follows Gaussian probability distribution and can be used for non-parametric machine learning algorithms for classification and regression.

3.4.4.8 Bagging Classifier

Is a statistical and regression classification method that works through redistribution of the training set randomly.

3.4.4.9 Random forest

The random forest is a learning method for classification and regression. It employs randomness, works by combing individual decision trees to form a forest, this combination is more accurate than an individual tree.

3.4.4.10 Neural Network

A neural network works like a human brain, with its algorithms it tries to identify the relationship in the dataset using artificial neurons.

3.4.5 Semi-supervised classification

This is machine learning an approach that combines a small amount of labeled data with a large amount of unlabeled data during training.

3.4.6 Recurrent Neural Network

Type of neural networks where connections between nodes form a directed graph along a temporal sequence.

$$h_t = \sigma_h(W_h x_t + U_h h_{t-1} + b_h) \tag{3.4}$$

$$y_t = \sigma_y(W_y h_t + b_y) \tag{3.5}$$

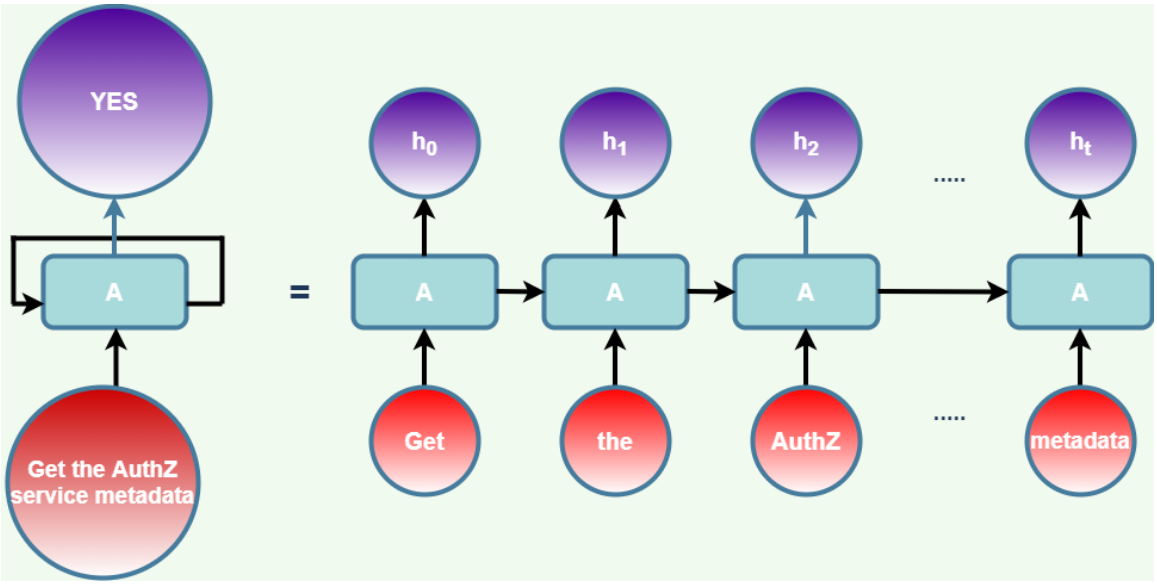

Figure 3.6: **RNN**

3.4.6.1 Long short term memory

This is a deep learning type of neural network with ability to learn the order of dependence in sequence prediction problems.The first step in constructing an LSTM network is identifying information that is redundant and will be dropped from the cell in that step. This process is decided by the sigmoid function.

$$f_t = \sigma(W_f[h_{t-1}, X_t] + b_f) \tag{3.6}$$

$$i_t = \sigma(W_i[h_{t-1}, X_t] + b_i) \tag{3.7}$$

$$N_t = \tan h(W_n[h_{t-1}, X_t] + b_n) \tag{3.8}$$

$$C_t = C_{t-1} f_t + N_t i_t \tag{3.9}$$

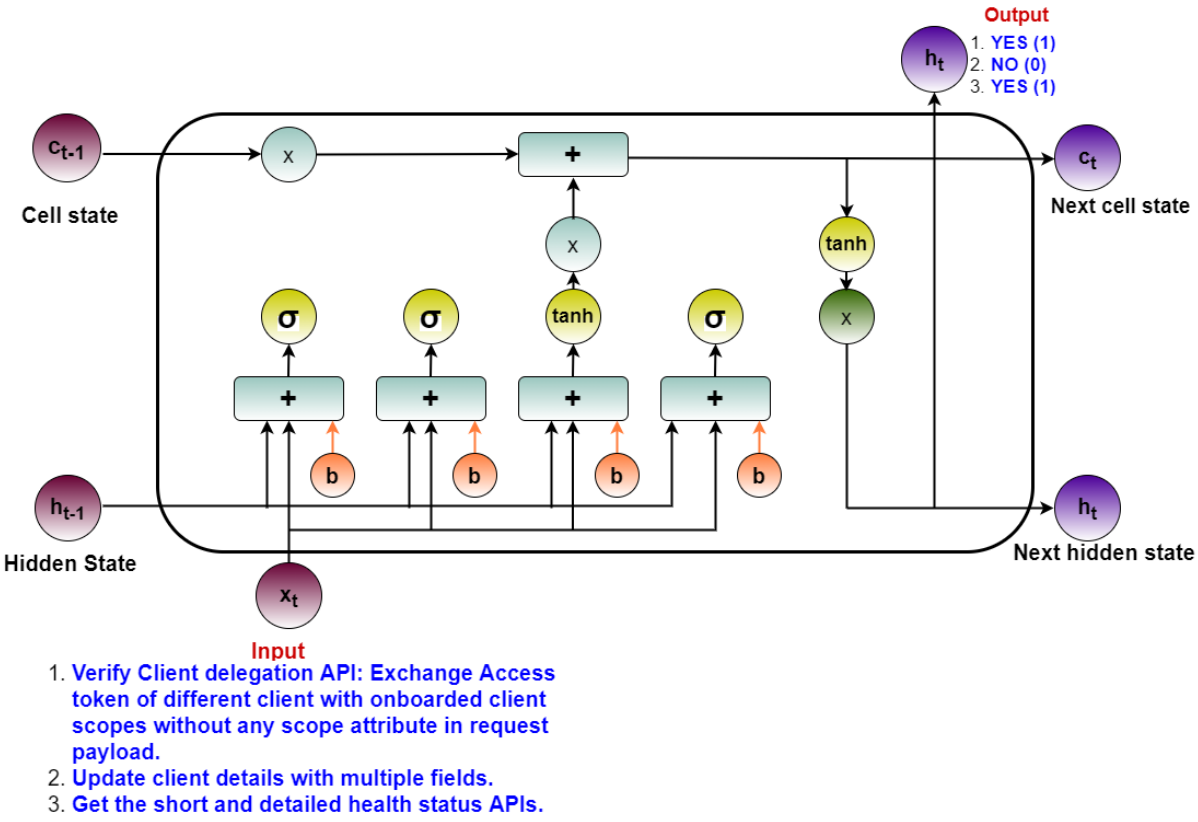

Figure 3.7: **LSTM**

3.4.6.2 Convolution Neural Network

A type of neural network that contains a layer of nodes, containing an input layer, hidden layers, and an output layer. Each node connects to another. It works through node activation, for a node to be activated there is a threshold to has to be surpassed if this achieved data is sent and anything below that threshold means no passing of data.

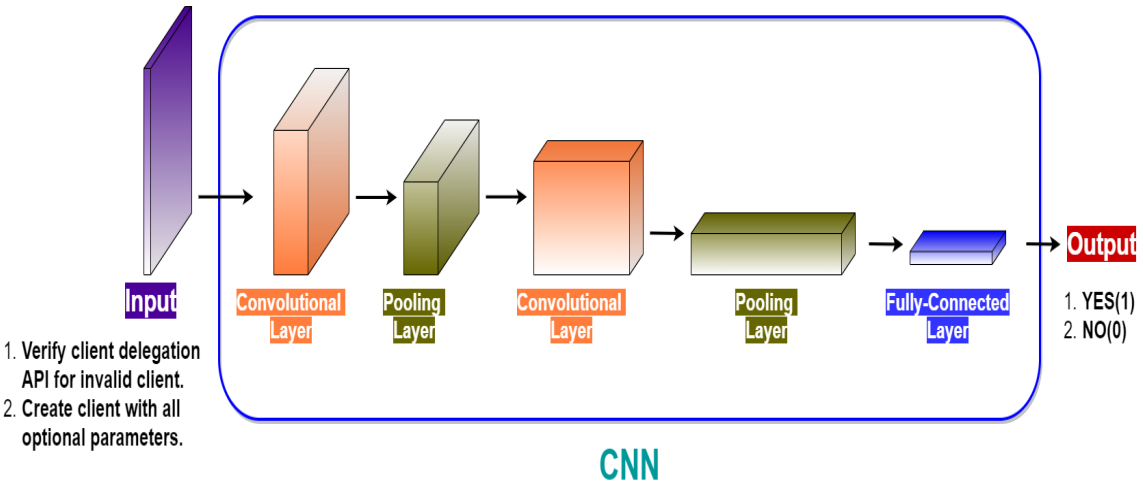

Figure 3.8: **CNN**

3.4.7 Text Feature Selection

Feature selection is the process of identifying and choosing a subset of variables that are most relevant to the target variable. This technique is subdivided into different categories

3.4.8 Filter based

These methods use statistical techniques to evaluate the relationship between each input variable and the target variable, evalution is based on scores are used as the basis to choose those input variables that will be used in the model.

3.4.8.1 Information Gain

It is a feature selection technique that deals with entropy reduction from the dataset through transformation.

3.4.8.2 ANOVA f-test

ANOVA is a statistical approach to feature selection, which deals with comparison of two or means of samples of data to deduce if they are of the same distribution or not. In this approach features nor related to target variable are omitted.

3.4.8.3 Fisher's Score

Is a statistical form of text feature selection which deals with maximum likelihood occurence of text fetures.

3.4.8.4 Chi-Square Test

This type of feature selection deals with categorical features in a dataset. Desired results are obtained through the best Chi-square scores.

3.4.8.5 Variance Threshold

In this type of technique, a threshold is set and if the variance of the features does not meet the threshold they are omitted among other removed features are those with zero variance.

3.4.9 Wrapper Methods

Deals with feature subset of features use machine learning algorithm to search for the best performing subset.

3.4.9.1 Recursive Feature Elimination

(RFE) is a feature selection method based on the strength of the features, features are selected based on their strength weakest features are omitted until results are obtained.

3.4.9.2 Forward Feature Selection

This method works exactly opposite to the Forward Feature Selection method. Here, we start with all the features available and build a model. Next, we the variable from the model which gives the best evaluation measure value. This process is continued until the preset criterion is achieved.

3.4.9.3 Backward Feature Selection

This method is the opposite of the Forward Feature Selection, the model is built and it is selected based on performance, it's a continuous process. .

3.4.10 Model Evaluation

Finally, the predicted output results of the classifiers are put on a performance scale. This is achieved using a classification report outlining the following performance measures.

Recall

This is the fraction of relevant instances that were retrieved. It is calculated by dividing the total number of true positives by the sum of true positives and false negatives.

$$Recall = \frac{TP}{TP + FN} \tag{3.10}$$

Precision

This is the fraction of relevant instances among the retrieved instances. It is calculated by dividing the total number of true positives by the sum of true positives and false positives.

$$Precision = \frac{TP}{TP + FP} \tag{3.11}$$

F-score

This is a measure of a test's accuracy, calculated from the precision and recall of the test.

$$Fscore = \frac{2 \times Precision \times Recall}{Precision + Recall} \tag{3.12}$$

Confusion matrix

Which is formed from the four outcomes produced as a result of binary classification, predicts all data instances of a test dataset as either positive or negative. This prediction) produces four outcomes which are defined as follows.

i True positive (TP): correct positive prediction

ii False positive (FP): incorrect positive prediction

iii True negative (TN): correct negative prediction

iv False negative (FN): incorrect negative prediction

A confusion matrix for binary classification as demonstrated is a two by two table constructed by counting of the number of the four outcomes of a binary classifier which are denoted as TP, FP, TN, and FN. The table 3.1 illustrates the confusion matrix.

		Predicted	
		Positive	**Negative**
Observed	**Positive**	**TP(of TPs)**	**FN(of FNs)**
	Negative	**FP(of FPs)**	**TN(of TNs)**

Table 3.1: **Confusion Matrix**

Chapter 4

Results and Discussions

This chapter describes and discusses the results obtained from the study, the results from various parts of the study are presented in this chapter through visualization and in table form. The results discussed here are those of text feature selection, confusion matrix, and clustering. The last part of the chapter highlights the performance of the classifier models.

4.1 Results

This section puts into categories and discusses the various results collected from the study. Text feature selection and the confusion matrix are discussed in different subsections of this section.

4.1.1 Text feature selection

Results of the two methods, filter-based and wrapper are visualized graphically here.

4.1.1.1 Fliter based method

Filter-based presents different categories results, the graphs used to visualize the results show how cases are selected based on filters of the most common words that are chosen in the test case selection. From the results, it can be noted that words like **token**, **verifi**, and **client api** play a vital role in the selection of test cases due to their frequency as presented in the results.

Based on *fisher's score* where more than one text is used in the selection, we see a combination of text **verifi token** shows a higher frequency of features selected compared to the phrase with more than two words thus we can conclude that these two words play a bigger role in test case selection.

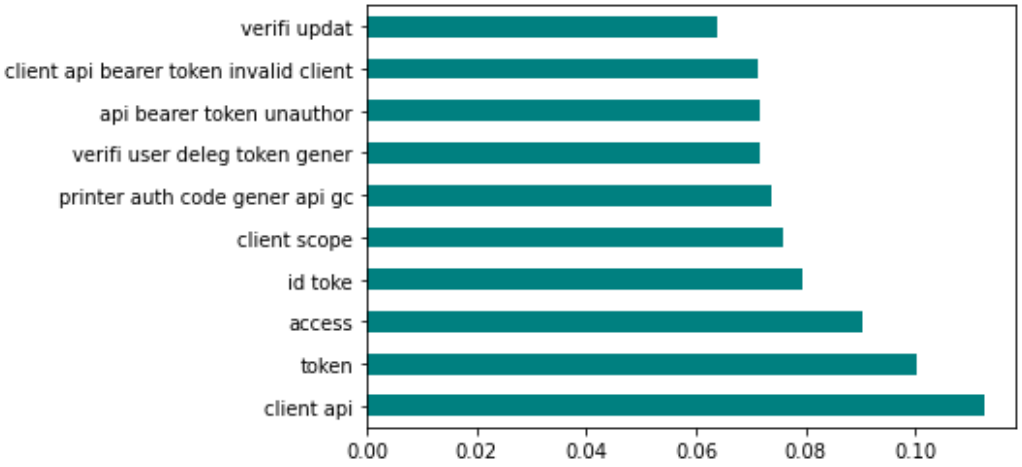

Figure 4.1: **Information gain**

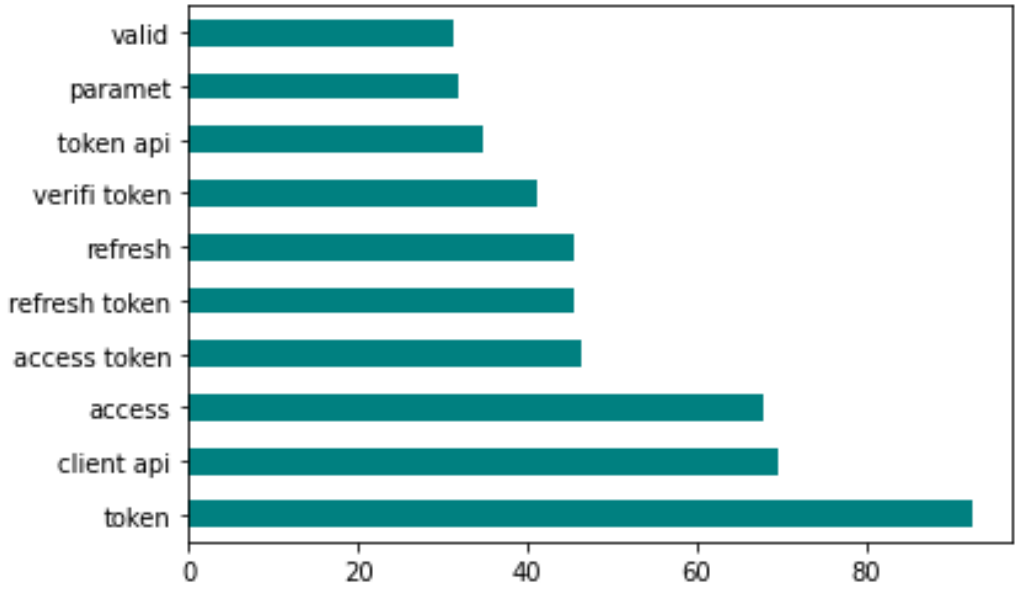

Figure 4.2: **Anova F-test**

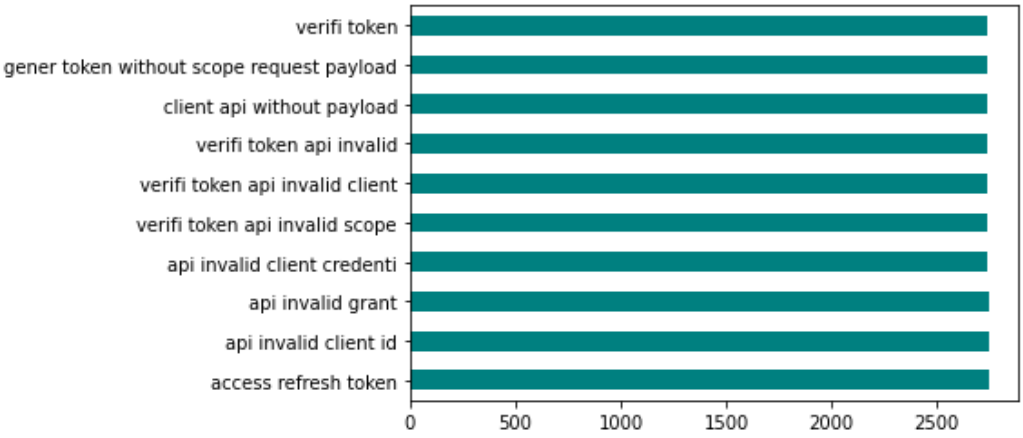

Figure 4.3: **Fisher‘s score**

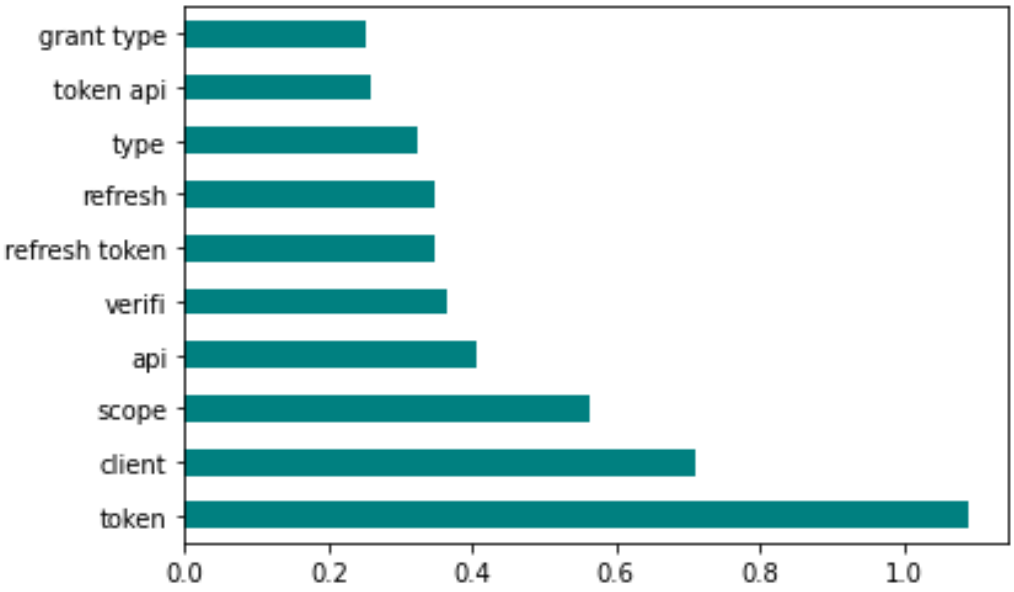

Figure 4.4: **Variance threshold**

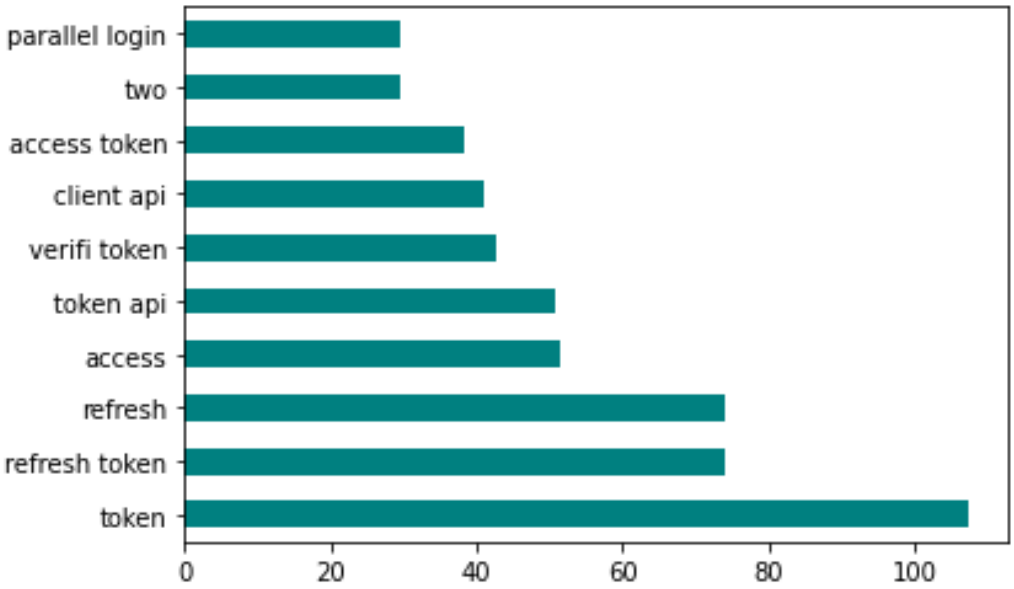

Figure 4.5: **Chi-square test**

4.1.1.2 Wrapper methods

On the other hand, wrapper methods present a different approach in text selection features, it concentrates on the number of features and performance. Where the best performance is close to 0.9 and the lowest is around 0.75. **Recursive feature elimination** unlike other wrapper methods presents a graph of cross-validation score of the number of selected features against the number of selected features and exhibits a stationary time series-like distribution, with the highest cross-validation score of just above 0.8 and lowest at 0.77.

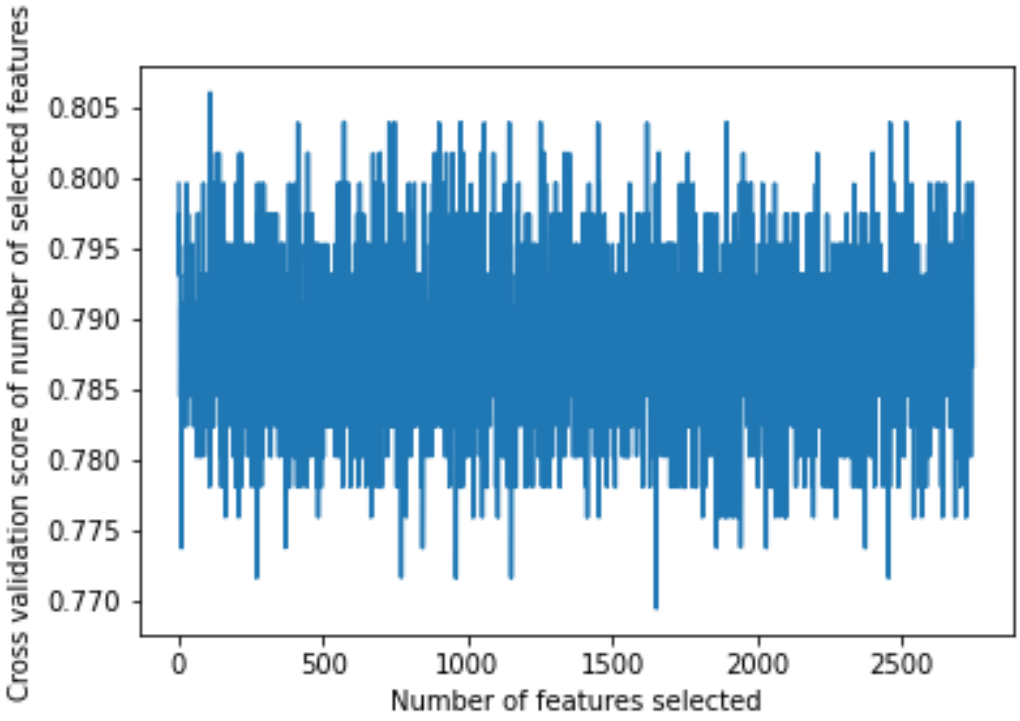

Figure 4.6: **Recursive feature elimination**

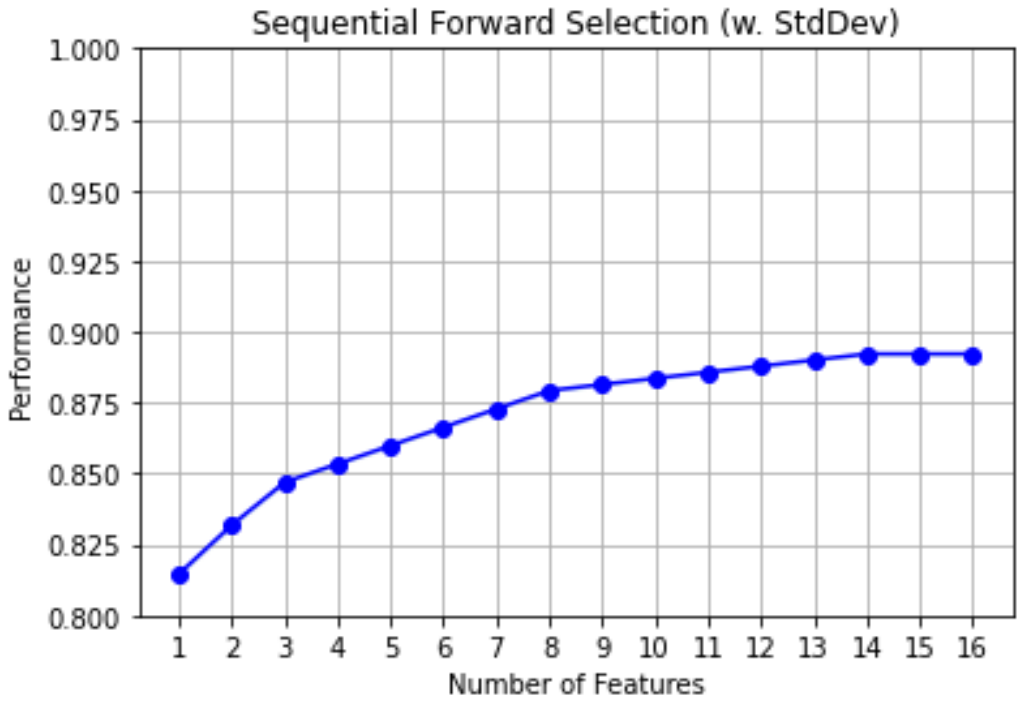

Figure 4.7: **Forward feature selection**

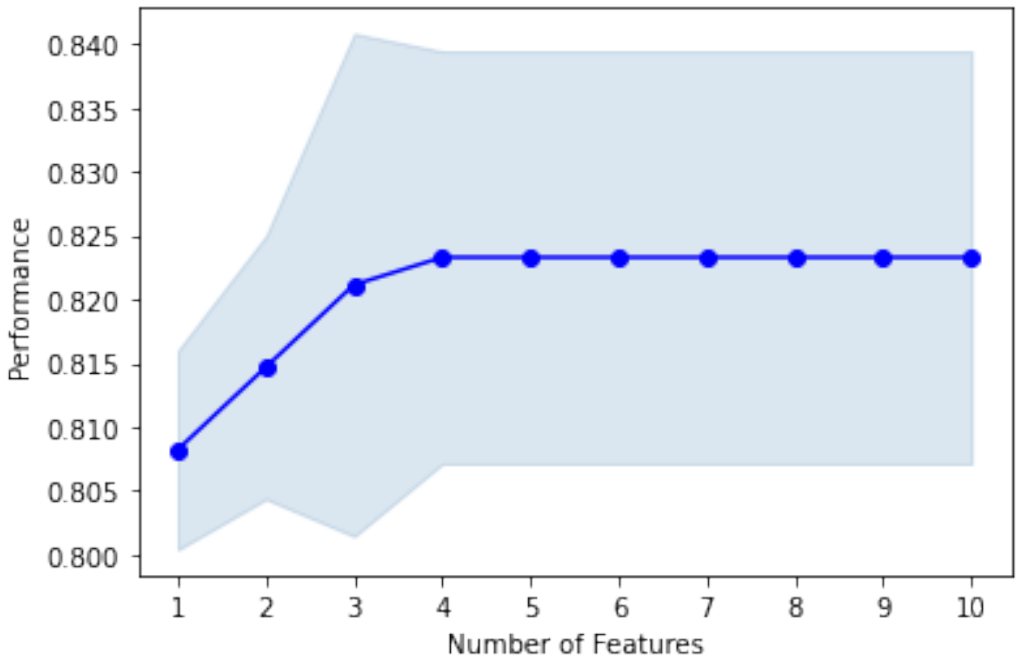

Figure 4.8: **Backward feature selection**

4.1.2 Clustering

Creating clusters improves the performance metrics of the test data, this means it helps improve accuracy from the figure 4.9 below, we can see the changes in the metrics, an increase in all metrics for every new cluster.

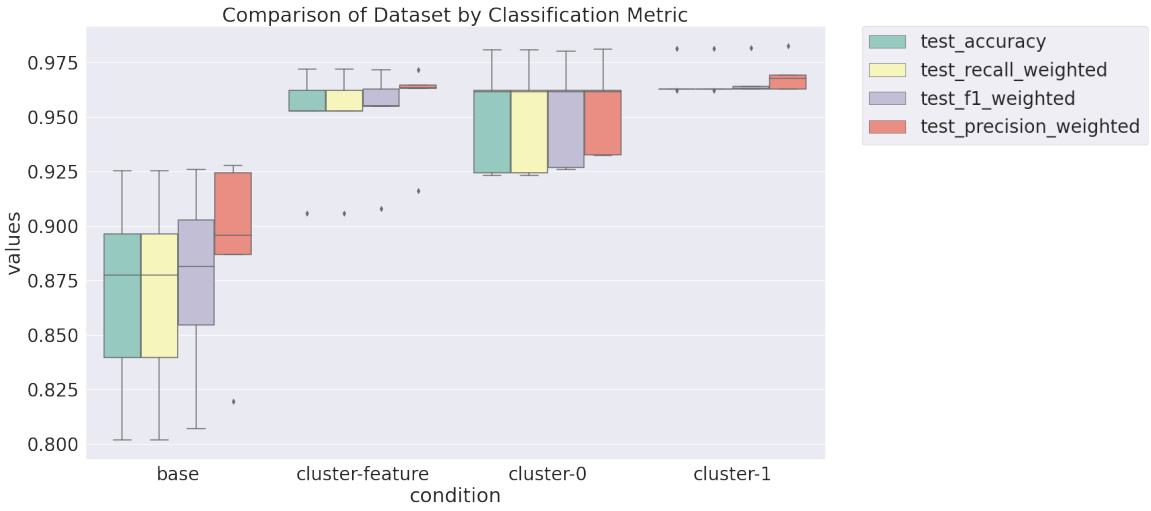

Figure 4.9: **Clustering**

4.1.3 Confusion matrix results for the classifiers

The confusion matrix of 10 classifier models results is present in this section as seaborn heatmaps. The heatmaps show a two-by-two table of binary classification. There are two possible predictions **YES** and **NO**. since we are predicting test selection. YES

means the test was selected, and NO means the test was not selected. The classifier made a total of 133 predictions. Meaning 133 cases were being considered. Different classifiers have different prediction results. The prediction is interpreted as below:

- **TP**, are selected cases which were predicted YES and it's true they were
- **TN**, are cases that were not selected and was predicted NO, which is a true prediction because they were not selected.
- **FP**, are cases that were predicted YES but in reality they were not selected.
- **FN**, are cases that were predicted NO but in reality they were selected cases

With the information above, the best-performing classifier model is the *Decision tree*. Out of 133 predictions, 98 were true positives, which means it predicted that these cases were selected and it's true they were selected. 34 were true negatives which means the classifier predicted the cases were not selected and it's true the cases were not selected. 0 false positives and only 1 false negative, a situation where it predicted case not to be selected but it was selected.

On the other hand poor-performing classifier is *Gaussian Naïve Bayes*. Out of 133 predictions, 70 (the lowest of all the classifiers considered for the study) were true positives, which means it predicted that these cases were selected and it's true they were selected. 34 were true negatives which means the classifier predicted the cases were not selected and it's true the cases were not selected. 0 false positives and 29 false negatives (the highest of all the classifiers considered for the study), which means it predicted that the cases were not selected but it was selected.

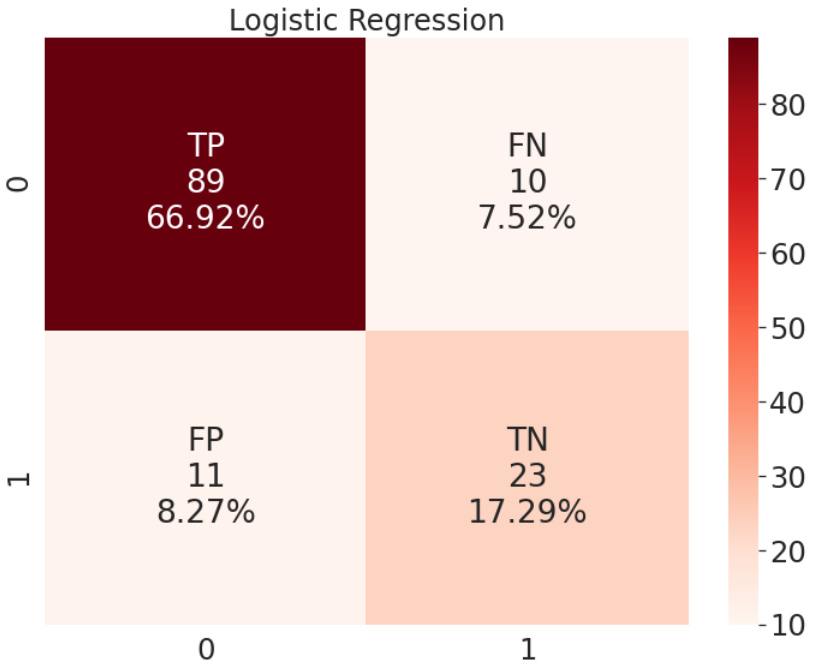

Figure 4.10: **Logistic regression**

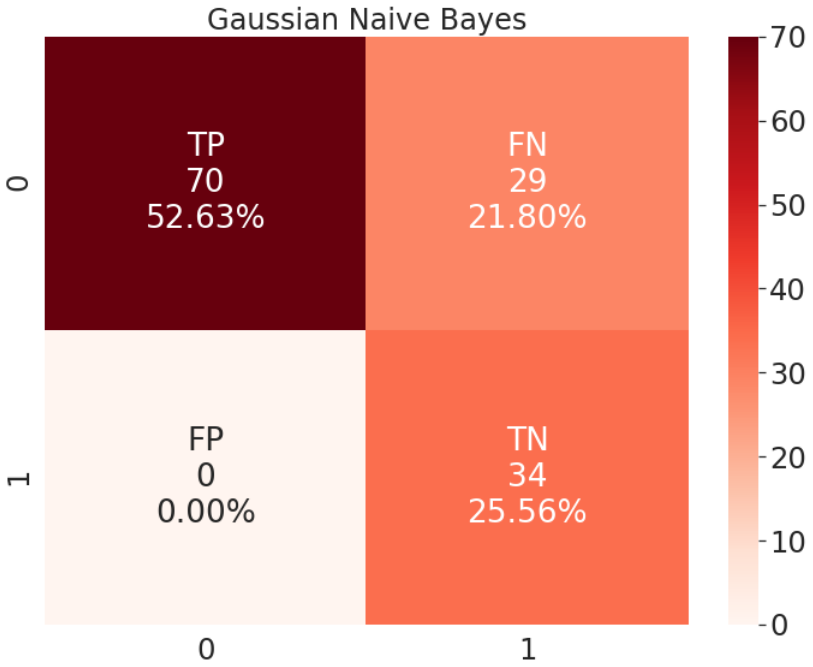

Figure 4.11: **Gaussian Naive Bayes**

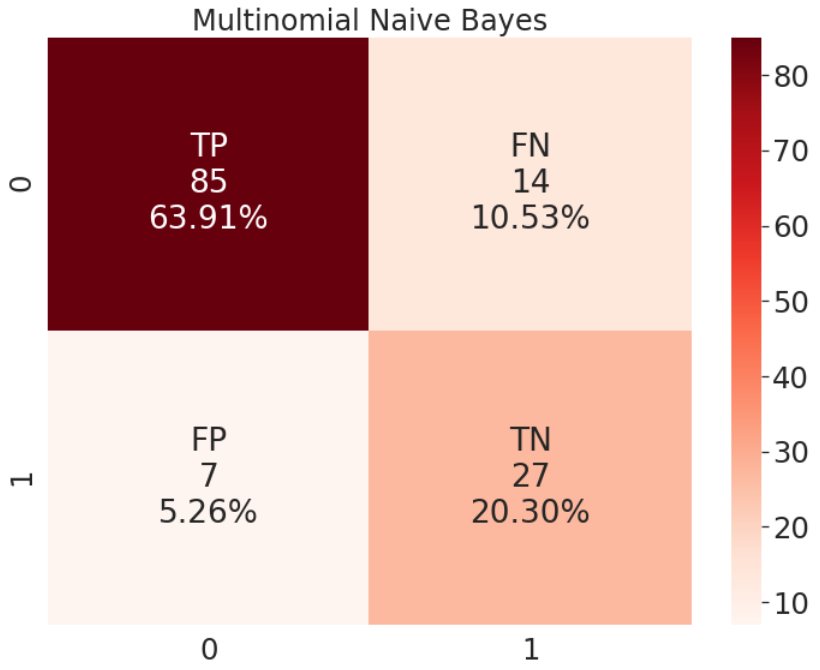

Figure 4.12: **Multinomial Naive Bayes**

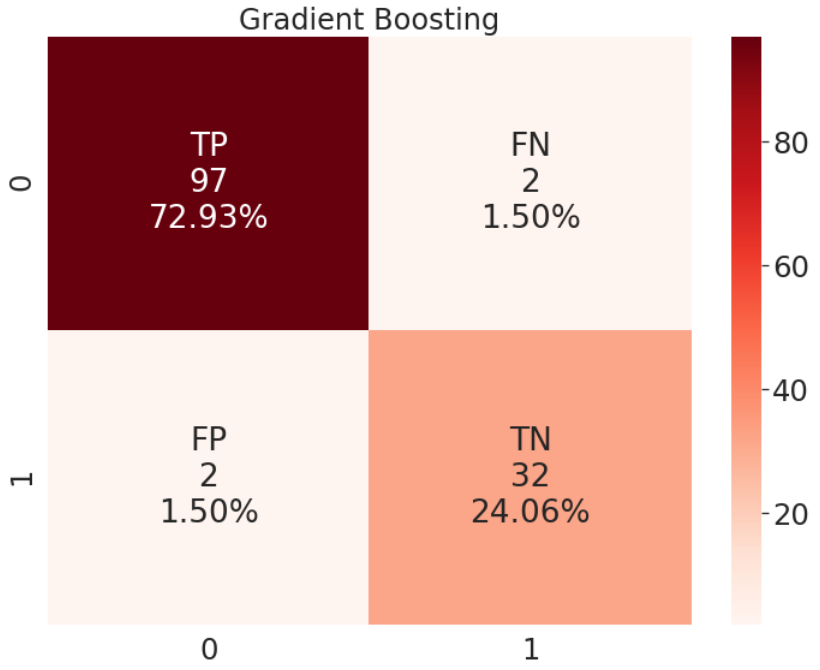

Figure 4.13: **Gradient Boosting**

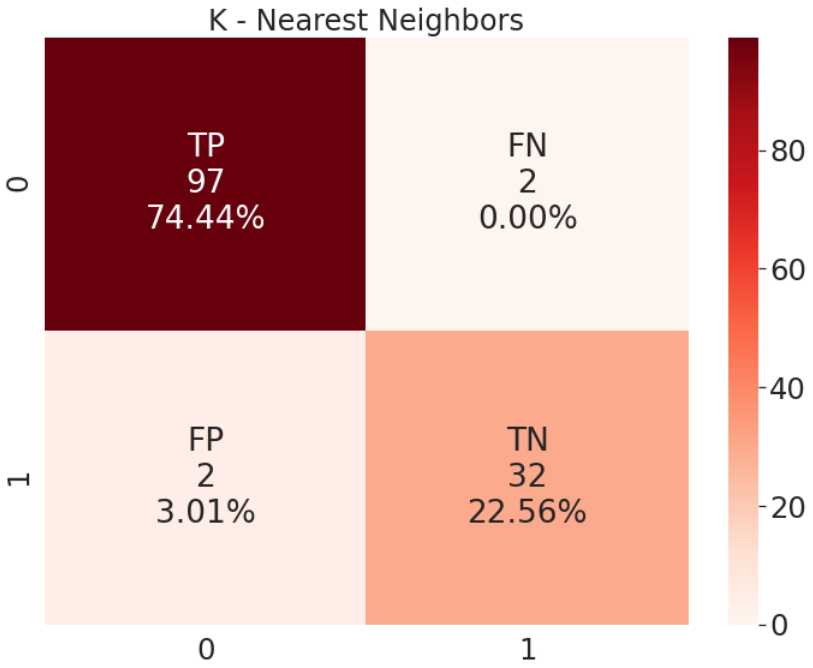

Figure 4.14: **K - Nearest Neighbors**

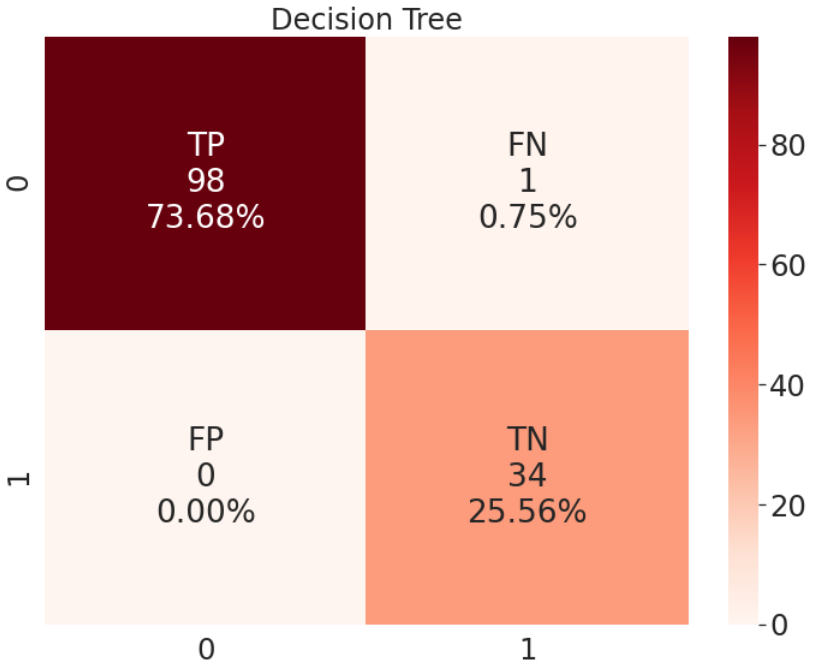

Figure 4.15: **Decision Tree**

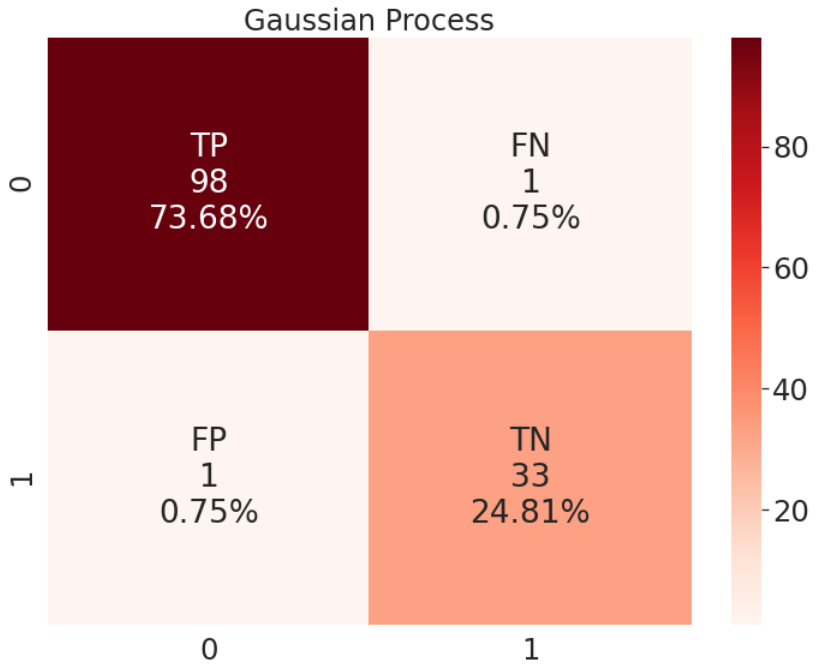

Figure 4.16: **Gaussian Process**

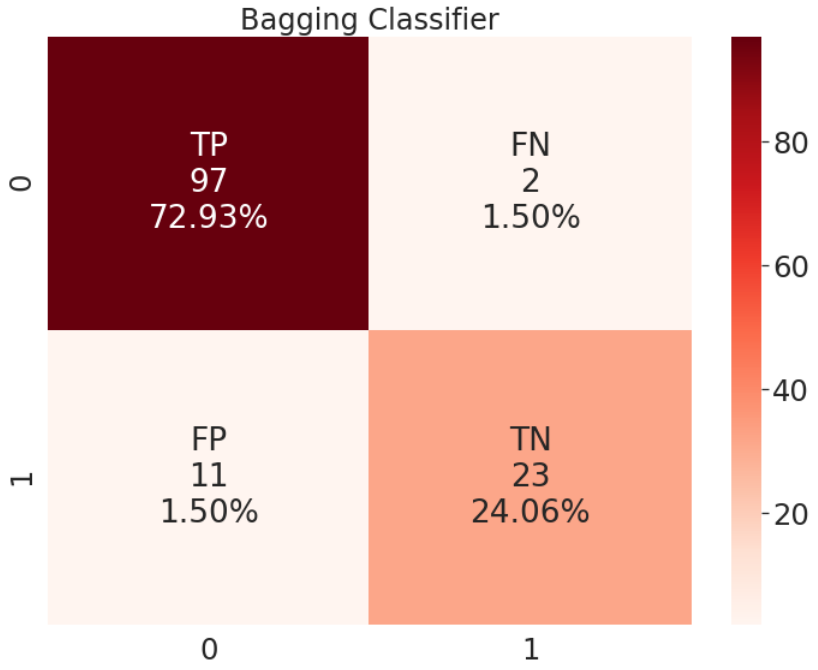

Figure 4.17: **Bagging Classifier**

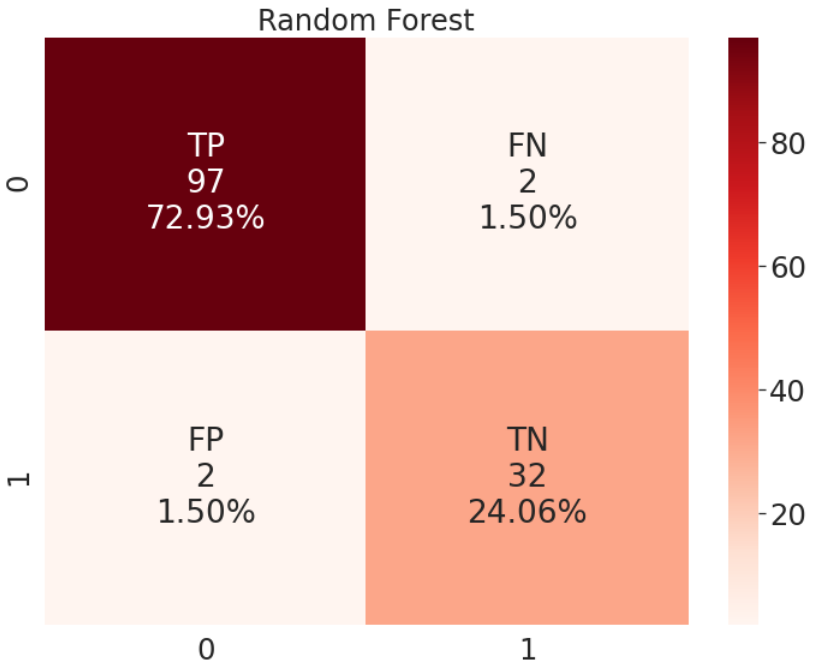

Figure 4.18: **Random Forest**

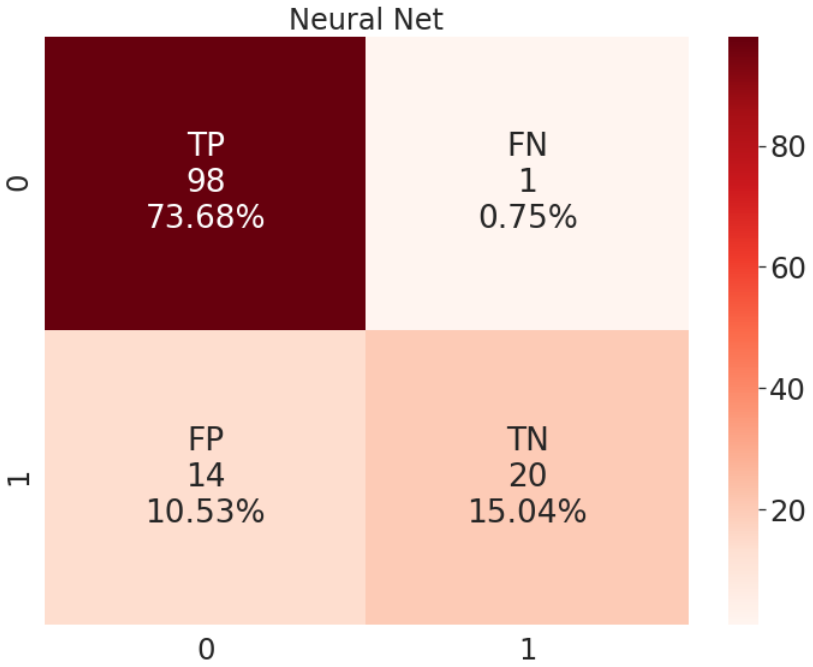

Figure 4.19: **Neural Network**

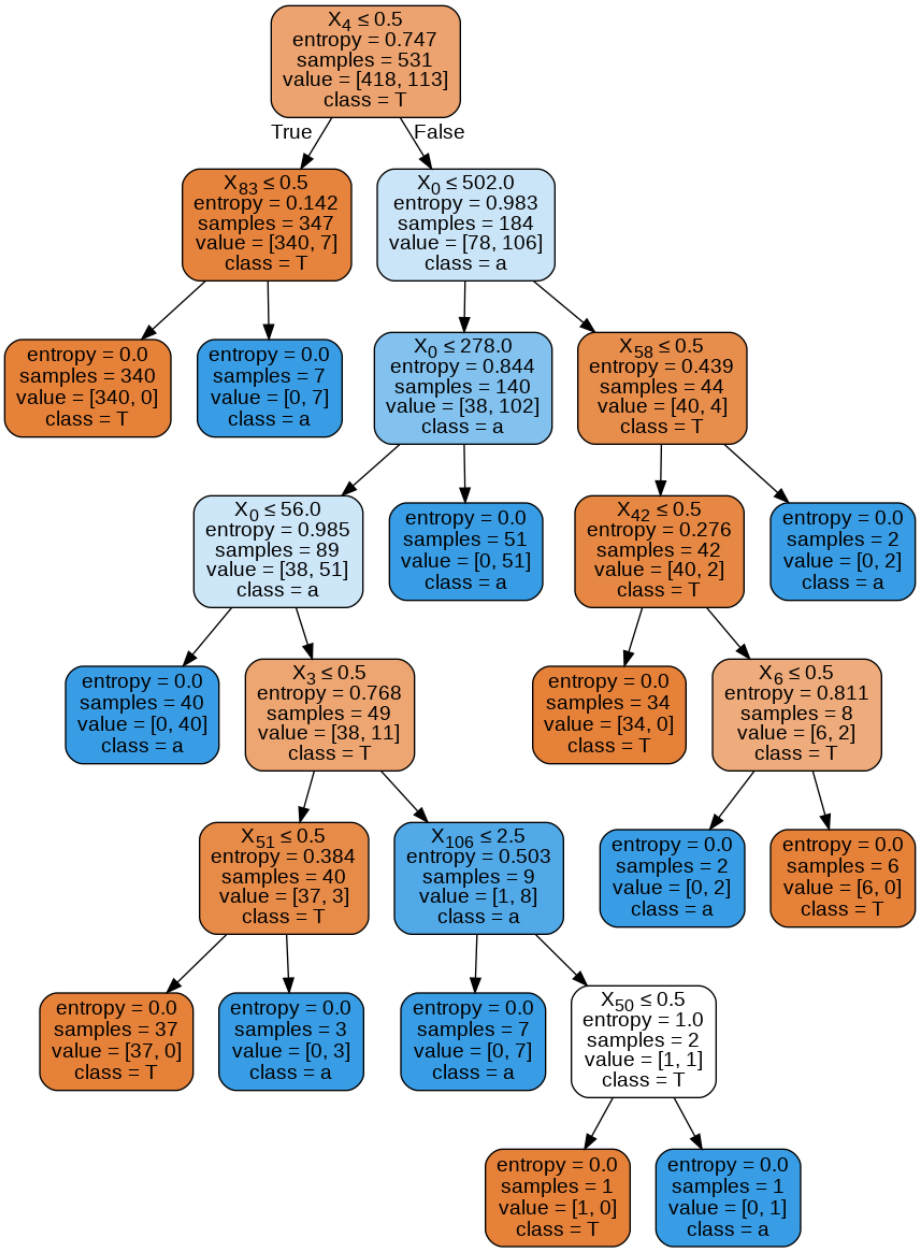

Figure 4.20: **Decision Tree**

4.2 Model performance

This section discusses the different performance metrics used to appraise the classifiers considered in this study. Among the metrics considered are classification reports and how classifiers perform in text classification.

Table 4.1 shows the performance of classifiers in terms of text classification. *Gaussian Naïve Bayes* performance across text classification metrics puts it as the best model compared to other models while *Gradient Boosting* is the underperforming model across all the metrics compared to other models.

Models	Count Vectors	WordLevel TF-IDF	N-Gram	CharLevel
Logistic Regression	0.815	0.780	0.780	0.805
Multinomial NB	**0.840**	**0.810**	**0.835**	**0.815**
Bagging classifier	0.765	0.765		
Neural network			0.765	
Gaussian NB	**0.861**	**0.849**	**0.759**	**0.855**
Gradient boosting	**0.783**	**0.746**	**0.765**	**0.765**

Table 4.1: **Text classification**

Performance metrics help bring more understanding on the performance of each classifier, a classification report was generated for analysis showing the weighted mean of precision, recall, and f-score, and the average of these three metrics gives the accuracy of the classifiers which evaluates their effectiveness. Kappa on the other hand compares an observed accuracy with an expected Accuracy. This is done for both supervised and semi_supervised classification. For supervised classification *Multinomial Naive Bayes* is by far is the best classifier with an accuracy of **0.855** and a kappa of **0.643**. on the other hand *Random Forest* shows rather low metrics compared to other classifiers, with an accuracy of **0.789** and a kappa of **0.000**.

Classifier	Accuracy	Precision	Recall	F1	Kappa
Nearest Neighbors	0.822	0.825	0.822	0.823	0.474
LogisticRegression	0.807	0.813	0.807	0.810	0.438
Decision Tree	0.825	0.868	0.825	0.836	0.558
Random Forest	**0.789**	**0.622**	**0.789**	**0.696**	**0.000**
MLP	0.807	0.815	0.807	0.810	0.444
AdaBoost	0.819	0.833	0.819	0.825	0.494
GaussianNB	0.831	0.859	0.831	0.840	0.553
LinearDiscriminant	0.807	0.820	0.807	0.812	0.454
QuadraticDiscriminant	0.668	0.725	0.668	0.690	0.162
GradientBoost	0.801	0.793	0.801	0.796	0.376
MultinomialNB	**0.855**	**0.904**	**0.855**	**0.865**	**0.643**
SGD	0.798	0.797	0.798	0.797	0.390
LGBM	0.819	0.827	0.819	0.822	0.478
Gradient Process	0.804	0.803	0.804	0.803	0.408
SVM 1	0.816	0.815	0.816	0.815	0.444
SVM2	0.795	0.777	0.795	0.783	0.320

Table 4.2: **Supervised classifier performance analysis**

For semi_supervised learning, ***Multinomial Naive Bayes*** is by far is the best

classifier with an accuracy of **0.872** and a kappa of **0.706**. on the other hand ***Random Forest*** shows rather low metrics compared to other classifiers, with an accuracy of **0.745** and a kappa of **0.000**. Compared to supervised classification there is a drop in metrics, which helps to effectiveness of supervised classification.

Classifier	Accuracy	Precision	Recall	F1	Kappa
Nearest Neighbors	0.827	0.825	0.827	0.826	0.539
LogisticRegression	0.790	0.784	0.790	0.786	0.428
Decision Tree	0.854	0.881	0.854	0.860	0.657
Random Forest	**0.745**	**0.555**	**0.745**	**0.636**	**0.000**
MLP	0.790	0.777	0.790	0.780	0.399
AdaBoost	0.836	0.840	0.836	0.838	0.578
GaussianNB	0.854	0.874	0.854	0.859	0.649
LinearDiscriminant	0.800	0.800	0.800	0.800	0.472
QuadraticDiscriminant	0.681	0.685	0.681	0.683	0.171
GradientBoost	0.790	0.775	0.790	0.776	0.383
MultinomialNB	**0.872**	**0.906**	**0.872**	**0.878**	**0.706**
SGD	0.809	0.806	0.809	0.807	0.490
LGBM	0.827	0.829	0.827	0.828	0.550
Gradient Process	0.781	0.772	0.781	0.776	0.396
SVM 1	0.800	0.791	0.800	0.794	0.446
SVM 2	0.790	0.773	0.790	0.766	0.349

Table 4.3: **Semi_Supervised classifier performance analysis**

In another performance appraisal, we take a look at the train, test scores of the classifiers. And further, look at semi_supervised metrics of both scores. This means helps narrow down classifier performance so that an excellent classifier is singled out after all parameters have been considered. In terms of train_score, some classifiers like *Logistic regression* and *Gradient boost* scored a **0.9** which was the highest score, while Random forest scored **0.768** the lowest score. In test_score *Multinomial naïve Bayes* outperformed the rest of the classifiers with a score of **0.855**. Random forest registered the lowest score of **0.785**. on the other hand, semi_supervised metrics of both scores puts *Multinomial naïve Bayes* as the best classifier with the scores of **0.853** and **0.872** for train and test scores respectively. The *Random forest* has the lowest of the same metrics, **0.785** and **0.745** for training and testing respectively which makes it the lowest overall performing classifier.

classifier	train score	test score	semi_supervised train score	semi_supervised test score
Nearest Neighbors	0.897	0.822	0.866	0.827
LogisticRegression	0.900	0.807	0.866	0.790
Decision Tree	0.864	0.825	0.842	0.854
Random Forest	**0.768**	**0.789**	**0.785**	**0.745**
MLP	0.900	0.807	0.866	0.790
AdaBoost	0.900	0.819	0.864	0.836
GaussianNB	0.882	0.831	0.857	0.854
LinearDiscriminant	0.900	0.807	0.864	0.800
QuadraticDiscriminant	0.870	0.668	0.787	0.681
GradientBoost	0.900	0.801	0.859	0.790
MultinomialNB	**0.858**	**0.855**	**0.853**	**0.872**
SGD	0.879	0.798	0.855	0.809
LGBM	0.894	0.819	0.862	0.827
Gradient Process	0.891	0.804	0.861	0.781
SVM 1	0.891	0.816	0.864	0.800
SVM 2	0.900	0.795	0.859	0.790

Table 4.4: **Comparison classifier performance analysis**

Table 4.5 is a classification report of a Deep learning showing its performance of two classifiers CNN and DNN with accuracy of **0.74** nd **0.77** respectively, which puts *Deep Neural Network* as best performing classifier.

Classifier	Precision	Recall	F1	Accuracy	Train score	Test score
Convolution Neural Network	0.69	0.74	0.66	0.74	0.79	0.76
Deep Neural Network	0.75	0.77	0.75	0.77	0.86	0.81

Table 4.5: **Deep Learning**

Chapter 5

Conclusion and Future works

Regression testing is conducted after updating any software components. This research demonstrates that machine learning-based approach can reduce the bias and manual labour of domain expert for software regression testing. Prediction performance could be improved if large amount ofd training of data can be increased by adding more releases data. In future, we would like to explore novel ensemble based feature selection strategy for text feature selection and adapted deep learning algorithms for imbalance dataset available for test case selection.

Bibliography

[1] H. Spieker, A. Gotlieb, D. Marijan, and M. Mossige, "Reinforcement learning for automatic test case prioritization and selection in continuous integration," in *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2017, pp. 12–22.

[2] R. Lachmann, S. Schulze, M. Nieke, C. Seidl, and I. Schaefer, "System-level test case prioritization using machine learning," in *2016 15th IEEE International Conference on Machine Learning and Applications (ICMLA)*, IEEE, 2016, pp. 361–368.

[3] P. Tonella, P. Avesani, and A. Susi, "Using the case-based ranking methodology for test case prioritization," in *2006 22nd IEEE International Conference on Software Maintenance*, 2006, pp. 123–133. DOI: 10.1109/ICSM.2006.74.

[4] K. Solanki, Y. Singh, and S. Dalal, "Experimental analysis of m-aco technique for regression testing," *Indian Journal of Science and Technology*, vol. 9, no. 30, pp. 1–7, 2016.

[5] X. Chen, Q. Gu, X. Zhang, and D. Chen, "Building prioritized pairwise interaction test suites with ant colony optimization," in *2009 Ninth International Conference on Quality Software*, 2009, pp. 347–352. DOI: 10.1109/QSIC.2009.52.

[6] D. Panwar, P. Tomar, and V. Singh, "Hybridization of cuckoo-aco algorithm for test case prioritization," *Journal of Statistics and Management Systems*, vol. 21, no. 4, pp. 539–546, 2018.

[7] D. Di Nucci, A. Panichella, A. Zaidman, and A. De Lucia, "A test case prioritization genetic algorithm guided by the hypervolume indicator," *IEEE Transactions on Software Engineering*, vol. 46, no. 6, pp. 674–696, 2020. DOI: 10.1109/TSE.2018.2868082.

[8] D. K. Yadav and S. Dutta, "Regression test case prioritization technique using genetic algorithm," in *Advances in computational intelligence*, Springer, 2017, pp. 133–140.

[9] M. H. Moghadam, M. Saadatmand, M. Borg, M. Bohlin, and B. Lisper, "Machine learning to guide performance testing: An autonomous test framework," in *2019 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, IEEE, 2019, pp. 164–167.

[10] D. Jeffrey and N. Gupta, "Improving fault detection capability by selectively retaining test cases during test suite reduction," *IEEE Transactions on Software Engineering*, vol. 33, no. 2, pp. 108–123, 2007. DOI: 10.1109/TSE.2007.18.

[11] Z. Ma and J. Zhao, "Test case prioritization based on analysis of program structure," in *2008 15th Asia-Pacific Software Engineering Conference*, 2008, pp. 471–478. DOI: 10.1109/APSEC.2008.63.

[12] B. Suri, I. Mangal, and V. Srivastava, "Regression test suite reduction using an hybrid technique based on bco and genetic algorithm," *Special Issue of International Journal of Computer Science & Informatics (IJCSI), ISSN (PRINT)*, pp. 2231–5292, 2011.

[13] W. Zhang, B. Wei, and H. Du, "Test case prioritization based on genetic algorithm and test-points coverage," in *International Conference on Algorithms and Architectures for Parallel Processing*, Springer, 2014, pp. 644–654.

[14] Y. Lou, D. Hao, and L. Zhang, "Mutation-based test-case prioritization in software evolution," in *2015 IEEE 26th International Symposium on Software Reliability Engineering (ISSRE)*, 2015, pp. 46–57. DOI: 10.1109/ISSRE.2015.7381798.

[15] D. Di Nucci, A. Panichella, A. Zaidman, and A. De Lucia, "A test case prioritization genetic algorithm guided by the hypervolume indicator," *IEEE Transactions on Software Engineering*, vol. 46, no. 6, pp. 674–696, 2018.

[16] A. R. Lenz, A. Pozo, and S. R. Vergilio, "Linking software testing results with a machine learning approach," *Engineering Applications of Artificial Intelligence*, vol. 26, no. 5-6, pp. 1631–1640, 2013.

[17] A. K. Pandey and V. Shrivastava, "Early fault detection model using integrated and cost-effective test case prioritization," *International Journal of System Assurance Engineering and Management*, vol. 2, no. 1, pp. 41–47, 2011.

[18] M. Salehie, S. Li, L. Tahvildari, R. Dara, S. Li, and M. Moore, "Prioritizing requirements-based regression test cases: A goal-driven practice," in *2011 15th European Conference on Software Maintenance and Reengineering*, 2011, pp. 329–332. DOI: 10.1109/CSMR.2011.46.

[19] V. Gupta and D. Chauhan, "Hybrid regression testing technique: A multi layered approach," in *2011 Annual IEEE India Conference*, 2011, pp. 1–5. DOI: 10.1109/INDCON.2011.6139363.

[20] G. Rothermel, R. J. Untch, and C. Chu, "Prioritizing test cases for regression testing," *IEEE Trans. Softw. Eng.*, vol. 27, no. 10, pp. 929–948, Oct. 2001. DOI: 10.1109/32.962562. [Online]. Available: https://doi.org/10.1109/32.962562.

[21] J.-M. Kim and A. Porter, "A history-based test prioritization technique for regression testing in resource constrained environments," in *Proceedings of the 24th international conference on software engineering*, 2002, pp. 119–129.

[22] C. Indumathi and K. Selvamani, "Test cases prioritization using open dependency structure algorithm," *Procedia Computer Science*, vol. 48, pp. 250–255, 2015.

[23] H. Srikanth, L. Williams, and J. Osborne, "System test case prioritization of new and regression test cases," in *2005 International Symposium on Empirical Software Engineering, 2005.*, 2005, 10 pp.-. DOI: 10.1109/ISESE.2005.1541815.

[24] J. F. Allen, "Natural language processing," in *Encyclopedia of computer science*, 2003, pp. 1218–1222.

[25] O. Melamud, J. Goldberger, and I. Dagan, "Context2vec: Learning generic context embedding with bidirectional lstm," in *Proceedings of the 20th SIGNLL conference on computational natural language learning*, 2016, pp. 51–61.

[26] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, 2013, pp. 3111–3119.

[27] D. Qiu, H. Jiang, and S. Chen, "Fuzzy information retrieval based on continuous bag-of-words model," *Symmetry*, vol. 12, no. 2, 2020. DOI: 10.3390/sym12020225. [Online]. Available: https://www.mdpi.com/2073-8994/12/2/225.