University of Stavanger

Faculty of Science and Technology

MASTER'S THESIS

Study program/ Specialization: Reliable and Secure Systems	Autumn semester, 2023 Open access
Writer: Xiaoyan Sun	........Xiaoyan Sun........................ (Writer's signature)
Faculty supervisor: Morten Mossige	
Thesis title: Machine Learning-Based Analysis of Test Results	
Credits (ECTS): 30	
Key words: Ranking, AHP, WSM, LambdaMART, Agglomerative Clustering, Jaccard Similarity, Cosine Similariy	Pages: 52 + enclosure: 56 Stavanger, 15 December, 2023

University of Stavanger

Faculty of Science and Technology
Department of Electrical Engineering and Computer Science

Machine Learning-Based Analysis of Test Results

Master's Thesis in Computer Science
by

Xiaoyan Sun

Internal Supervisors

Morten Mossige

December 12, 2023

Abstract

A comprehensive understanding of the overall performance of the tests is critical in software testing to make necessary adjustments to the test schedule or conduct targeted investigations. In ABB Bryne, the software testing is processed by a test system named NAST for the IPS software for paint robots. This thesis addresses the critical challenges by investigating two key aspects: ranking frequently failing test cases and clustering test cases based on error messages.

Two models were employed for the ranking task: the conventional Analytic Hierarchy Process-Weighted Sum Model (AHP-WSM) and the Learning-to-rank model LambdaMART. The results highlight the superiority of the AHP-WSM model over LambdaMART, attributed to the influence of dataset quality and size in the machine learning technique.

Agglomerative clustering with Jaccard and cosine similarity metrics was applied for the clustering. The findings reveal that cosine similarity yielded superior outcomes due to its calculation characteristics to capture semantic relationships between texts based on overall content rather than relying on exact term matches.

This research provides valuable insights into effective methodologies for addressing the complexities associated with test case prioritisation and clustering in software testing.

Acknowledgements

I sincerely thank my supervisor, Morten Mossige, whose continuous guidance has been invaluable throughout this research journey. Special thanks to Nina Svense, Lasse Ånestad, and Donjing Liu for their insightful and constructive feedback, which significantly enhanced the quality of this work.

I also deeply appreciate my friend, Ronbing Li and my boyfriend, Njål Brocker Næss, for their reliable support during the stressful periods of my master's thesis. Their encouragement and understanding were invaluable throughout my academic journey.

Contents

Abbreviations

ABB	**A**SEA **B**rown **B**overiist
NAST	**N**orway **A**utomatic **S**ystem **T**est
IPS	**I**ntegrated **P**rocess **S**ystem
AHP	**A**nalytic **H**ierarchy **P**rocess
WSM	**W**eighted **S**um **M**odel
MCDM	**M**ulti-**C**riteria **D**ecision **M**aking
MADM	**M**ulti-**A**ttribute **D**ecision **M**aking
CV	**C**onsistebcy **V**ector
CI	**C**onsistency **I**ndex
CR	**C**onsistency **R**atio
RI	**R**andom **I**ndex
MART	**M**ultiple **A**dditive **R**egression **T**rees
LTR	**L**earning **t**o **R**ank
DB	**D**ata**b**ase
NDCG	**N**ormalized **D**iscounted **C**umulative **G**ain
MAP	**M**ean **A**verage **P**recision
MRR	**M**ost **R**ecent run's **R**esult
RFP	**R**ecent **F**ailures **P**ercentage
AP	**A**bort **P**ercentage
WP	**W**arning **P**ercentage
EP	**E**rror **P**ercentage
ACJ	**A**gglomerative **C**lustering with **J**accard Similarity
ACC	**A**gglomerative **C**lustering with **C**osine Similarity

Chapter 1

Introduction

This chapter will begin by discussing the motivation behind this study, followed by introducing the use cases that will be applied. Subsequently, the objectives of this research, which incorporate the research questions, will be presented. The thesis outline will be provided last.

1.1 Motivation

In the context of ABB Bryne, where hundreds of tests run continuously every day, gaining a clear view of test run performance is crucial for optimizing development time. Two key analysis strategies can be implemented to provide valuable insights into the test run performance: identifying and prioritizing frequently failing test cases and clustering test cases based on error messages. Effective implementation of these strategies enhances the developer's understanding of test run dynamics and contributes to streamlined development processes.

This research undertakes a comprehensive investigation, employing advanced methodologies toward each challenge. The Analytic Hierarchy Process-Weighted Sum Model (AHP-WSM) and LambdaMART are considered for ranking, while for clustering, agglomerative techniques with Jaccard and cosine similarity metrics are explored. The following chapters will present the performance of the methodologies, discuss the results and provide a detailed performance analysis for each.

1.2 Use Cases

Analysing test results is a critical aspect of software testing and quality assurance, playing a pivotal role in ensuring the reliability and robustness of software systems. This section introduces the two strategies that can be applied to present the analysis for test run performance efficiently.

1.2.1 Ranking Frequently Failed Tests

A practical approach involves using a ranking model to identify the ten most often failing test cases. From the test framework runs in ABB Bryne, a test case is a defined file that outlines the requirements for testing a particular function or segment of the software. Evaluating test case performance is crucial in guaranteeing the testing process's comprehensiveness.

This analysis provides an understanding of the under-performance of test cases by assigning a rating to the test cases that have continuously failed recently. The term "failed" denotes that the outcome of a test case falls into one of the following categories: Abort, Error, or Warning. This strategy enables developers to prioritise and concentrate on resolving these concerns.

1.2.2 Clustering Test Cases Based on Error Messages

Another beneficial approach entails grouping test cases according to their error messages. The primary cause of test run failure is usually discovered by extracting an error message from the raw log generated during the test run. A test clustering method can effectively categorise error messages into distinct groups based on similar debugging difficulties, streamlining the root cause analysis process. This is particularly advantageous given that most tests operate on comparable software. By classifying test cases in this manner, the debugging process is enhanced, facilitating the development team's ability to tackle various difficulties originating from known sources using a unified solution. The primary objective is to expedite the development cycle, minimise the duration spent on debugging, and augment the overall quality of the product.

1.3 Objectives

The objective of this study is to identify an optimized ranking and clustering algorithm that can yield a satisfactory outcome within the context of our specific circumstances. In order to do this, the following objectives have been identified:

- Organize and process the data obtained from the test runs.
- To provide an initial ranking result for frequently failing test cases, it is proposed to implement the AHP-WSM model as the non-machine learning ranking method.
- Deploy a machine learning ranking model using LambdaMART and evaluate its outcomes and efficacy compared to the AHP-WSM model.
- Two machine learning clustering methods will be implemented to compare their performance and results for text clustering of error messages: Agglomerative Clustering with Jaccard similarity (ACJ) and Agglomerative Clustering with Cosine similarity (ACC).
- Analyze the performance from the models and select the optimal solution for utilization.

Based on the above objectives, several research questions can be listed:

- RQ1: What data preprocessing techniques are most suitable for preparing the test data for ranking and clustering algorithms?
- RQ2: By comparing the conventional ranking model and the learning-to-rank model, which one performed better and why?
- RQ3: How do the two machine learning clustering models compare their performance and results when applied to error message text clustering?
- RQ4: Based on the analysis of the results, which model is the most optimal for clustering the test cases?

These objectives and research questions will guide this study to develop an optimized ranking and clustering algorithm.

1.4 Outline

The thesis is organized into the following chapters:

Chapter 1: Introduction

Chapter 2: Background

- This chapter provides an overview of the technique employed in this study. It introduces the methodology and discusses the prior work done at ABB. Additionally, it explores the relevant literature in the field.

Chapter 3: Approaches for Ranking Model

- This chapter primarily focuses on the conventional ranking algorithm utilized in the current study and introduces a learning-to-rank model. Furthermore, an evaluation of the learning-to-rank model is presented.

Chapter 4: Approaches for Clustering Model

- This chapter will provide an overview of the two clustering methods utilized in this study and an evaluation of both of them.

Chapter 5: Results

- The results of the models implemented in this study are provided in this chapter, including both ranking and clustering implementations.

Chapter 5: Discussion

- This chapter compares the outcomes generated by different models, along with an analysis of their respective performance.

Chapter 7: Conclusion

- This chapter serves to summarize the findings of the study and provide answers to the research questions. And a discussion about potential future work.

Chapter 2

Background

This chapter will provide an overview of the research approaches taken for this study and the previous work done at ABB. To begin with, it will expound upon the ranking and Learning to Rank algorithms used in creating the platform's frequently failed test cases feature. Then, the clustering algorithms assist in dividing the test cases into groups to boost the debugging process's efficiency. And finally, ABB's already-established test system and database will be presented.

2.1 Ranking and Learning to Rank

This section presents the methodologies to develop a function to get the frequently failed test cases. This study uses two methods: a conventional ranking model and a learning-to-rank model. The conventional ranking model used in this study is the AHP-WSM model, which integrates the Weighted Sum Model (WSM) with the Analytic Hierarchy Process (AHP). For the learning-to-rank algorithm, LambdaMART is used.

2.1.1 AHP-WSM Model

Firstly, the Multi-Criteria Decision Making problem is introduced in this part. Then, the AHP-WSM model is presented by going through a step-by-step presentation of WSM and the AHP model.

Multi-Criteria Decision Making

Multi-Criteria Decision Making (MCDM) is a field of study that aims to identify the optimal alternative or establish a ranking of alternatives by considering all relevant criteria with varying levels of importance[1]. Criteria, also referred to as attributes or goals, represent the different dimensions from which the alternatives can be viewed, could be quantitative or qualitative and may relate to cost, performance, quality, sustainability, and other factors.

MCDA methods are usually divided into Multi-Attribute Decision Making(MADM) and Multi-Objective Decision Making(MODM). The significant difference is that MADM operates within a continuous decision space involving a finite number of attributes. On the other hand, MODM operates within a discrete decision space with an infinite number of choices and attributes. Nevertheless, the names MADM and MCDM refer to the same category of methodologies[2][1]. A general structure of MCDA is shown in Figure 2.1.

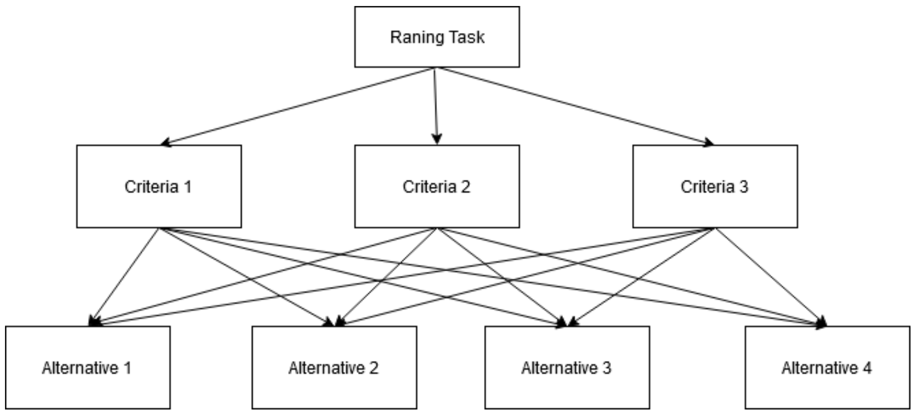

Figure 2.1: A general structure of MCDA.

The Weighted Sum Model

The Weighted Sum Model (WSM) is a widely used approach within MCDM for addressing ranking problems, it requires that criteria be assigned weights of importance and aggregate these criteria to calculate an overall score for each alternative. Alternatives can then be ranked based on their calculated scores. A MADA problem can be easily expressed in matrix format. As shown in Table 2.1, a decision matrix $(n \times m)$ is formed, where $A = \{A_i, for\ i = 1, 2, 3, ..., m\}$ be a set of alternatives, $c = \{c_j, for\ j = 1, 2, 3, ..., n\}$ be a set of criteria and $w = \{w_j, for\ j = 1, 2, 3, ..., n\}$ is the corresponding weight for criteria. Then the score for each alternative can be calculated by equation 2.1, where element a_{ij} indicates the value of the i-th alternative in terms of the j-th criteria, and w_j is the weight of j-th criteria[3]. Then, the alternatives get sorted based on their score.

$$S_i = \sum_{j=1}^{n} a_{ij} w_j,\ for\ i = 1, 2, 3, ..., m \tag{2.1}$$

	Criteria				
	c_1	c_2	c_3	$\dots$	c_n
Alts.	$(w_1$	w_2	w_3	$\dots$	$w_n)$
A_1	a_{11}	a_{12}	a_{13}	$\dots$	a_{1n}
A_2	a_{21}	a_{22}	a_{23}	$\dots$	a_{2n}
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
A_m	a_{m1}	a_{m2}	a_{m3}	$\dots$	a_{mn}

Table 2.1: A typical decision matrix for MADA problem.

Analytic Hierarchy Process

As previously mentioned, the WSM requires the allocation of weights to each criterion. This study uses the Analytic Hierarchy Process (AHP) to assign these weights. As a popular approach developed by Satty in the 1970s[4], AHP has been used in many situations related to MCDM. The subsequent procedure outlines the stages of employing AHP to compute the weight assigned to each criterion[5].

In the first step, the pairwise comparison matrix ($n \times n$), as shown in Table 2.2, is constructed based on the relative importance of each criterion. Each criterion will be compared with another criterion at a time, and the decision maker will then assign a relative importance scale based on the scales in Table 2.2.

Scale	**Definition**
1	Equal importance
2	Weak or slight
3	Moderate importance
4	Moderate plus
5	Strong importance
6	Strong plus
7	Very strong or demonstrated importance
8	Very, very strong
9	Extreme importance
1/9,1/8,...,1	Reciprocals of above

Table 2.2: Fundamental scale of AHP [6].

$$C = \begin{bmatrix} c_{11} & c_{12} & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & c_{2n} \\ \vdots & \ddots & & \vdots \\ c_{n1} & c_{n2} & \dots & c_{nn} \end{bmatrix} \tag{2.2}$$

Then, the sum for each column is calculated. After that, each element c_{ij} in the matrix is divided by the sum of its respective column to get the normalised pairwise matrix C' as in Equation 2.3.

$$C' = \begin{bmatrix} \frac{c_{11}}{\sum_{j=1}^{n} c_{j1}} & \frac{c_{12}}{\sum_{j=1}^{n} c_{j2}} & \cdots & \frac{c_{1n}}{\sum_{j=1}^{n} c_{jn}} \\ \frac{c_{21}}{\sum_{j=1}^{n} c_{j1}} & \frac{c_{22}}{\sum_{j=1}^{n} c_{j2}} & & \frac{c_{2n}}{\sum_{j=1}^{n} c_{jn}} \\ \vdots & \ddots & & \vdots \\ \frac{c_{n1}}{\sum_{j=1}^{n} c_{j1}} & \frac{c_{n2}}{\sum_{j=1}^{n} c_{j2}} & \cdots & \frac{c_{nn}}{\sum_{j=1}^{n} c_{jn}} \end{bmatrix} = \begin{bmatrix} c'_{11} & c'_{12} & \cdots & c'_{1n} \\ c'_{21} & c'_{22} & \cdots & c'_{2n} \\ \vdots & \ddots & & \vdots \\ c'_{n1} & c'_{n2} & \cdots & c'_{nn} \end{bmatrix} \tag{2.3}$$

Following this, the weights w_i for each criterion will be calculated by adding each row in the normalised pairwise matrix and dividing by the number of criteria.

$$W = \begin{bmatrix} \frac{\sum_{j=1}^{n} c'_{1j}}{n} \\ \frac{\sum_{j=1}^{n} c'_{2j}}{n} \\ \vdots \\ \frac{\sum_{j=1}^{n} c'_{nj}}{n} \end{bmatrix} = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{bmatrix} \tag{2.4}$$

Before using the calculated weights in WSM, a validation is needed. First, a weighted sum matrix (C_{sum}) is calculated by multiplying the pairwise comparison matrix and the weights.

$$C_{sum} = \begin{bmatrix} w_1 \times c_{11} & w_2 \times c_{12} & \cdots & w_n \times c_{1n} \\ w_1 \times c_{21} & w_2 \times c_{22} & \cdots & w_n \times c_{2n} \\ \vdots & \ddots & & \vdots \\ w_1 \times c_{n1} & w_2 \times c_{n2} & \cdots & w_n \times c_{nn} \end{bmatrix} \tag{2.5}$$

Then, the Consistency Vector (CV) is generated by dividing the row sum of C_{sum} by weights, as shown in Equation 2.6. The Consistency Index (CI) is calculated by

$CI = \frac{\lambda_{\max}-n}{n-1}$, where $\lambda_{\max}$ is the average of CV, and n is the number of criteria[7].

$$CV = \begin{bmatrix} \frac{\sum_{j=1}^{n} c_{1j} \times w_j}{w_1} \\ \frac{\sum_{j=1}^{n} c_{2j} \times w_j}{w_2} \\ \vdots \\ \frac{\sum_{j=1}^{n} c_{nj} \times w_j}{w_n} \end{bmatrix} \tag{2.6}$$

Finally, the Consistency Ratio can be obtained by $CR = \frac{CI}{RI}$, where RI is the Random Index value from Table 2.3. The acceptable value of CR is less than 0.1[8][9]. If CR is larger than 0.1, the pairwise comparison matrix's judgement is inconsistent and needs to be re-determined.

N	1	2	3	4	5	6	7	8	9	10
Random Index	0	0	0.52	0.89	1.11	1.25	1.35	1.40	1.45	1.49

Table 2.3: Random Index[10].

2.1.2 LambdaMART

To compare with the AHP-WSM model, a Learning to Rank method, LambdaMART, is used. This section will start by introducing the idea of Learning to Rank, followed by an explanation of the LambdaMART methodology.

Learning to Rank

Learning to Rank (LTR) is a supervised machine learning technique that aims to address the ranking tasks, such as search results, documents or recommendations, in the order that aligns users' preferences.

A typical LTR flow is shown in Figure 2.2. The process contains training and test phases since it is a supervised learning task. A training set is needed in the training phase, which contains several queries $q_i (i = 1, 2, ...m)$ and the associated documents $D_i = \{d_1^i, d_2^i, ..., d_n^i\}$. Also, a list of relevance labels represents the degree of relevance of the document concerning the query. Moreover, a set of features, such as the length, date or other attributes from the documents, are also fed to the learning model, which can help the ranking model learn how to rank the list. Then, the training set is fed to the learning system that employs a specific learning algorithm and produces a ranking model h. This ranking model will assign a weight to the features that can be used in the test phase.

In the subsequent test phase, a test set will be given to the training model, which is a query q with the related documents $D = \{d_1, d_2, ..., d_n\}$. The trained ranking model is then used to assign a score to each document based on the weight of their features, and it subsequently ranks the list of documents in descending order based on their assigned scores.

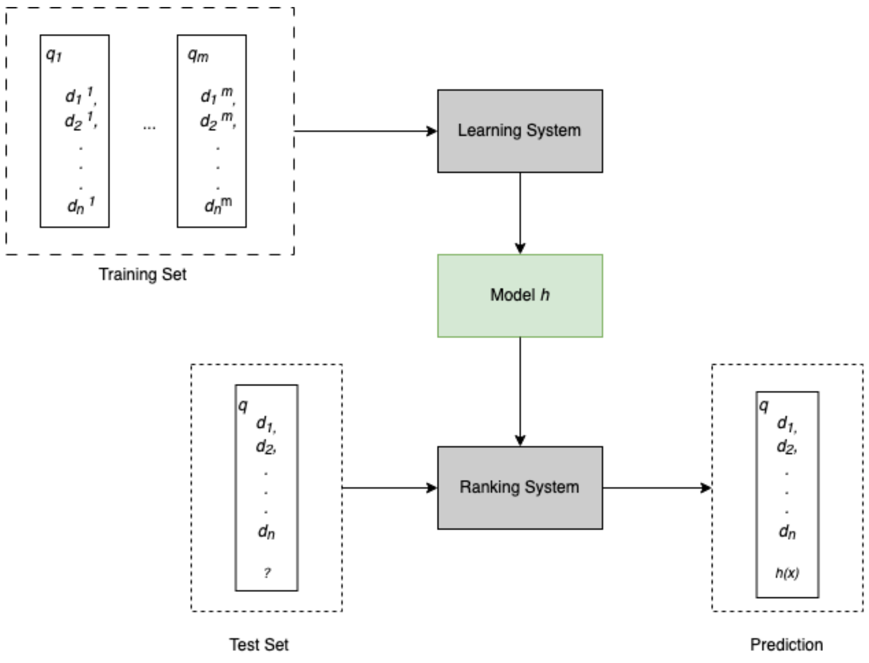
q_1
$d_1{}^1$, $d_2{}^1$, . . . $d_n{}^1$
...
q_m
$d_1{}^m$, $d_2{}^m$, . . . $d_n{}^m$
Training Set
Learning System
Model h
q
d_1, d_2, . . . d_n
?
Test Set
Ranking System
q
d_1, d_2, . . . d_n
$h(x)$
Prediction

Figure 2.2: A typical learning-to-rank flow.

LambdaMART

This study used the LambdaMART algorithm to implement the LTR model for comparison with the AHP-WSM model. LmabdaMART can be divided into two distinct components, Lambda and MART. Lambda refers to lambda gradients and denotes a weighting factor that adjusts the gradient employed to optimise the loss function during the training process. The loss function is defined based on the differences between anticipated and true ranking. The term MART, short for Multiple Additive Regression Trees, refers to a model design that integrates decision trees and gradient boosting. [11][12].

The general algorithm is presented in 2.1, where the boosting iterations begin at line 4. Within each iteration, the relevance label y_i is assigned, followed by the computation of the gradient w_i for y_i using the prediction F_{k-1} from the previous model. Next, a regression tree $\{R_{lk}\}_{l=1}^{L}$ is constructed with L leaves. The values for each leaf in the tree are determined by dividing the predicted relevance label of the data point given to the leaf by the sum of the importance weights of the data points. Subsequently, the model will go through an update process by utilising the recently generated decision tree. To account for each data point x_i, the model $F_k(x_i)$ is modified by taking a step toward the leaf value γ_{lk} if x_i belongs to leaf R_{lk} in the decision tree. The step size taken during each iteration is determined by the learning rate, denoted as η. The final model will be produced after the boosting iterations.

Algorithm 2.1 LambdaMART

```
Input: Training Data:$\{x_m, y_m\}$,$m = 1, 2 \ldots, m$;
    Number of Trees: $N$;
    Number of Leaves(per tree): $L$;
    Learning Rate: $\eta$
Output: Model: $f(x, N)$
 1: for $i = 0$ to $m$ do
 2:     $f(x, 0) = BaseModel(x)$ //If BaseModel is empty, set $f(x, 0) = 0$.
 3: end for
 4: for $k = 1$ to $N$ do
 5:     for $i = 0$ to $m$ do
 6:         $y_i = \lambda_i$
 7:         $w_i = \frac{\partial y_i}{\partial F_{k-1}(x_i)}$
 8:     end for
 9:     $\{R_{lk}\}_{l=1}^{L}$ //Create L leaf tree on $\{x_i, y_i\}_{i=1}^{m}$.
10:     $\gamma_{lk} = \frac{\sum_{x_i \in R_{lk}} y_i}{\sum_{x_i \in R_{lk}} w_i}$ //Assign leaf values.
11:     $F_k = F_{k-1}(x_i) + \eta \sum_l \gamma_{lk} I(x_i \in R_{lk})$ // Take step with learning rate $\eta$.
12: end for
```

2.2 Clustering

This section presents the clustering algorithms that divide the test cases into groups. The clustering methodology can be split into two main components: the similarity metrics and the clustering algorithm. First, this paper presents a comprehensive overview of the clustering method. Subsequently, the similarity metrics are explained: Jaccard Similarity and Cosine Similarity. Next, a thorough examination of the Agglomerative Clustering algorithm is offered.

2.2.1 Clustering Method

Clustering is an unsupervised machine-learning technique that categorises items into clusters according to their similarity. Typically, the grouping process will start by calculating the similarity between each data point and generating the similarity matrix[13][14]. Then, the clustering algorithm will employ the similarity matrix to group data points into clusters based on the characteristic grouping strategy with the restriction that each data point should be assigned to just one cluster. The outcome clusters can be described as follows:

Given a set of cluster $C = C_1, C_2, \ldots, C_k$ that generated from dataset X:

1. $C_i \neq \phi, i = 1, 2, \ldots, k$;
2. $\bigcup_{i=1}^{k} C_i = X$;
3. $C_i \cap C_j = \phi, i, j = 1, 2, \ldots, k$ and $i \neq j$.

2.2.2 Similarity Matrix

Before introducing the details about the clustering algorithms, it is essential to understand the concept of measuring the similarity between data points. A similarity matrix is a tool used to compare two vectors, x_i and x_j, by assigning a numerical value that quantifies the level of similarity between them, denoted as $s(x_i, x_j)$ and $s(x_i, x_j) = s(x_j, x_i)$. Higher values indicate a higher degree of similarity, while lower values indicate dissimilarity[13].

Jaccard Similarity

Jaccard similarity, denoted as $Jaccard(A, B)$, measures the similarity between two sets, A and B. As defined in Equation 2.7, where $|A \cap B|$ represents the size of the intersection of sets A and B, capturing the number of common elements between them, and $|A \cup B|$ denotes the union's size of A and B, encompassing all unique elements from both sets.

The Jaccard similarity ranges from 0 to 1, where 1 implies complete set identity, indicating that the sets A and B are identical. Conversely, a Jaccard similarity of 0 suggests absolute dissimilarity, signifying that there are no common elements between the sets[15].

$$JaccardSimilarity = \frac{|A \cap B|}{|A \cup B|} \tag{2.7}$$

Cosine Similarity

Cosine Similarity measures the similarity by computing the cosine of the angle between the two vectors, as expressed by Equation 2.8. In this equation, A and B represent the vectors, and $A \cdot B$ denotes the dot product of vectors A and B, the $||A||$ and $||B||$ calculates the magnitudes of vectors A and B, respectively[15].

As in Jaccard Similarity, the resulting Cosine Similarity range ranges from 0 to 1, and 1 denotes the two vectors are perfectly aligned, signifying maximum similarity—conversely, a similarity score of 0 suggests no similarity between the vectors.

$$CosineSimilarity = \frac{A \cdot B}{||A|| \cdot ||B||} \tag{2.8}$$

2.2.3 Agglomerative Clustering

Agglomerative clustering is a hierarchical clustering technique that produces a dendrogram, a tree-like structure representing the relationships between data points. A typical dendrogram structure is illustrated in Figure 2.3. In the dendrogram, the bottom nodes correspond to individual items in the dataset, forming the initial clusters. Subsequent levels of the dendrogram are constructed through a series of merge operations, ending at the root node, representing the entire dataset as a single group. The number of clusters can be determined by selecting an appropriate level to cut the dendrogram.

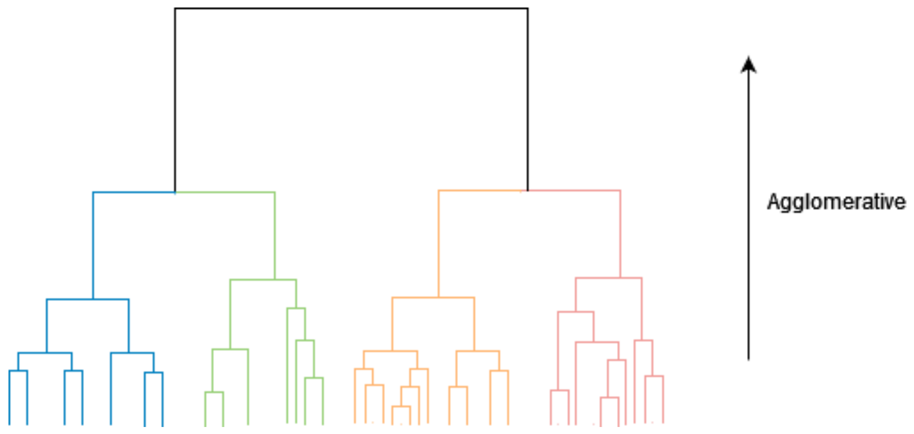

Figure 2.3: The typical structure for Agglomerative Clustering.

2.3 Previous Work

This section begins with a brief introduction to the software test system in ABB Bryne. Then, a detailed description of the current database structure used for this test system is presented. The database is separated into two components: Cosmos DB, responsible for storing detailed data of each execution, and Blob Storage, which stores logs and attachments.

2.3.1 ABB's Test System

The tested software is the Integrated Process System (IPS), an embedded and distributed software suite specifically developed to control the paint robots manufactured by ABB. The test is conducted using the Norway Automatic System Test (NAST), and the data analysed in this study are the outcomes obtained from executing the test cases with NAST.

The test case run by NAST is a file that specifies how a particular function or section of the IPS should be tested, detailing the test parameters, input data, and anticipated results for a specific function or component. After a test case has been run, NAST creates a JSON file with detailed information about that particular test run, including details such as the test case's name, unique identifier, the script used to execute it, and more. Moreover, the raw log and additional attachments, such as PNG images or CSV files, will also be generated.

2.3.2 Database

As mentioned, the database structure combined Azure Cosmos DB and Azure Blob Storage. After the NAST processes the test cases, the JSON file generated is uploaded to Azure Cosmos DB. In contrast, the raw log file and any additional attachments are stored in Azure Blob storage. The execution flow in ABB is visually shown in Figure 2.4.

Azure Cosmos DB

The amount of data produced by test case runs has been rising significantly at ABB Bryne, and it now averages 400 MB records every week, from 200 to 400 test runs. The database currently has a vast collection of more than 60,000 test results.

Microsoft offers a cloud-native NoSQL database solution called Azure Cosmos DB[16]. Its outstanding scalability makes data distribution across several servers or nodes possible, making it ideally suited for managing large data volumes and high-velocity applications. Additionally, because of its inherent flexibility, data in various structured and semi-structured formats, including the JSON files, may be easily accommodated while allowing for quick change adaptation. Furthermore, strong authentication and role-based access control techniques are provided by Azure Cosmos DB.

In this scenario, the database is accessed through a connection string with a distinct identifier. Data is retrieved by formulating queries using the Structured Query Language (SQL) as a JSON query language.

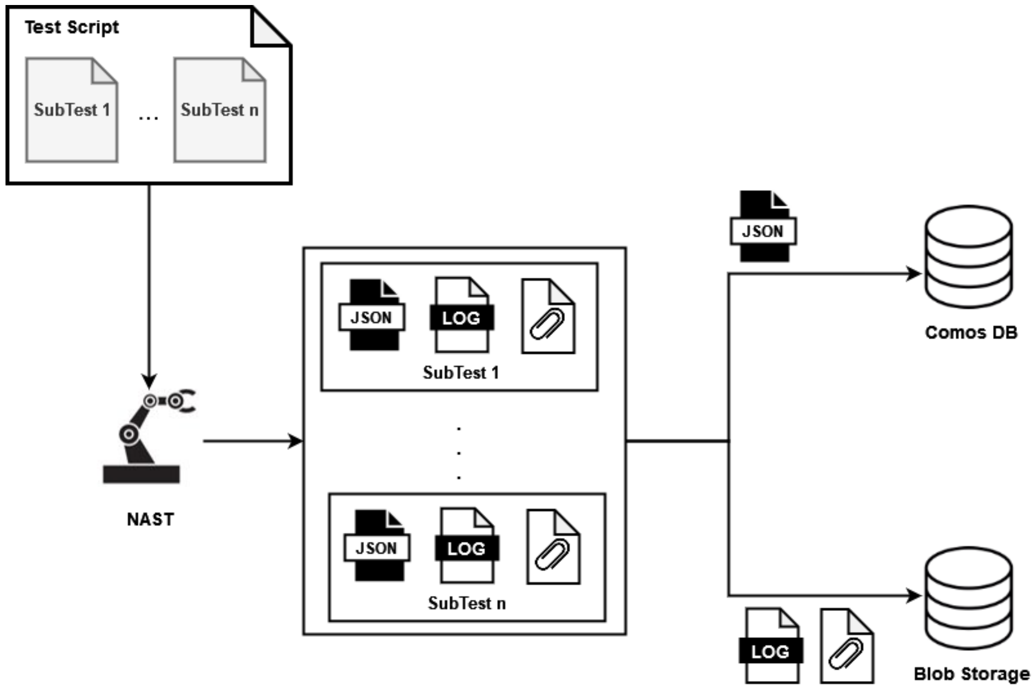

Figure 2.4: The overall execution flow in ABB includes the customary arrangement of the input, specifically the test script, and the output, comprising the detail, log, and attachment files.

Azure Blob Storage

Azure Blob Storage is a suitable storage solution for storing raw logs and diverse attached files. It serves as an optimal repository for handling large volumes of unstructured data that lack a predefined data model or description. This includes both textual and binary data. The retrieval of objects in blob storage can be easily accomplished by utilising the HTTP/HTTPS protocols. Furthermore, to increase security measures, establishing a connection to Blob Storage can be accomplished by utilising the Secure Shell File Transfer Protocol (SFTP). Additionally, the mounting of Blob Storage containers can be facilitated by employing the Network File System (NFS) 3.0 protocol[17].

In this structure, to build the connection between Cosmos DB and Blob Storage, first, the *local name* and the *remote name* for the log file and attachments are added in the JSON file that is stored in the Cosmos DB from the result of a test case run. The *local name* refers to an easily understandable name for a specific file. The *remote name* refers to the designated name assigned to Blob storage, which includes the *localname* of the file and a 16-character unique identifier for the blob file. In the case of a log file, this name typically represents the system on which the test case was executed. On the other hand, for attachment files, the name is typically provided by the user responsible for writing the test execution script. By utilising the HTTPS protocol combined with a distinct Shared Access Signature (SAS) token, it becomes possible to display the human-readable name and gain access to the blob file through the remote name. In this manner, a connection between Cosmos DB and Blob storage is established.

2.4 Related Work

Previous studies on ranking models have yielded valuable insights across several applications, serving as the basis for the approaches applied in this research. The study conducted by [9] employed the Analytic Hierarchy Process-Weighted Sum Model (AHP-WSM) to assess seismic risk. The research emphasized the effectiveness of the AHP-WSM in easing decision-making procedures.

The research conducted by [18] presented an application that utilized LambdaMART with XGBoost. The study also incorporated two assessment techniques, Normalized Discounted Cumulative Gain (NDCG) and Mean Average Precision (MAP), to assess the application's performance.

The research conducted by [19] offers significant insights into agglomerative clustering. This study primarily examines several similarity measurements to enhance our comprehensive comprehension of the strengths and limits of the agglomerative algorithm.

The research by reference [20] also presents an in-depth investigation of using Jaccard similarity and cosine similarity metrics in nearest-neighbour clustering. This work examines the performance and associated attributes of both similarity measures.

Chapter 3

Approaches for Ranking Model

The ranking models developed in this study will be presented in this chapter, first with the data preprocessing needed before the implementation, then the detailed explanation of the conventional ranking model: AHP-WSM model. After that, the LTR ranking model implemented with LambdaMART is presented.

3.1 Data Preprocessing

The data preprocessing for later use in the ranking model is divided into feature filtering and data transformation, as described separately in the following sections.

Feature Filtering

As the information provided in Chapter 2.3, following the execution of each test case, a JSON file will be generated and uploaded to the database. This file contains various details such as the date of the test run, the duration of the run, the software version used (identified by the *build_id*), the corresponding test case (identified by the *test_case_id*), the outcome of the run, the individual responsible for executing the run, and specific information about each subtest. The subtest details encompass elements such as the name of the subtest, the system on which it was executed, the duration of the subtest, the name of the associated log file, and additional relevant information. The JSON file representing a single run can range from several hundred to several thousand lines. To optimize the performance of the ranking algorithm, it is necessary to extract essential details while minimizing the inclusion of extraneous data.

Attribute Name	Description
test_case_id	The identifier for the respective test case
test_case_name	The name assigned to the test case
test_result	Indicates the outcome of this particular run
start_of_test	Indicates the date and time of this run's execution

Table 3.1: The information extracted from each run.

The needed information for each run are as follows: the test case ID and name it belongs to, the date and the result. Table 3.1 presents a comprehensive overview of each attribute's name and briefly explains each attribute.

Data Transformation

To rank the test cases based on their performance, it is necessary to group the features from each execution and restructure the data at the test case level. After combining those extracted details, the generated attributes for each test case are as follows:

- *test_case_id*: the identifier for the test case.
- *test_case_name*: the name assigned to the test case.
- *total_runs*: the number of runs from this test case.
- *most_recent_run*: the date from the latest run.
- MRR: as the most recent run's result.
- RFP: as the recent failures percentage, is calculated by the count of failed runs until a successful one is reached (also can be denoted as recent failures) divided by *total_runs*. A failed result can be denoted as abort, error, or warning.
- AP: as abort percentage, is calculated by dividing the number of runs resulting in *abort* among the recent failures by the total number of recent failures.
- EP: as error percentage, is calculated by dividing the number of runs resulting in *error* among the recent failures by the total number of recent failures.
- WP: as warning percentage, is calculated by dividing the number of runs resulting in *warning* among the recent failures by the total number of recent failures.

The reason abort, error, and warning are calculated separately instead of a single failed percentage is that although abort, error, and warning are all denoted as a failed run, they hold different priorities in ABB. Specifically, abort holds the highest priority, error follows, and warning assumes the lowest importance among all three. The percentage of each is one of the influential factors for ranking the test cases list.

The RFP counts the failed runs until it reaches the most recent successful run because when it reaches a successful run, the results before it only have minor importance since a successful run indicates the issue that caused the failure before it was resolved.

Additionally, percentages are preferred to calculate each test case's historical running state over the exact number of events. This approach ensures an unbiased perspective, as the various test cases may have distinct running schedules. Some test cases may be executed numerous times within a day, while others may only run once a week. The comparison of the percentage of the events and the exact number of them are shown in Figure 3.1.

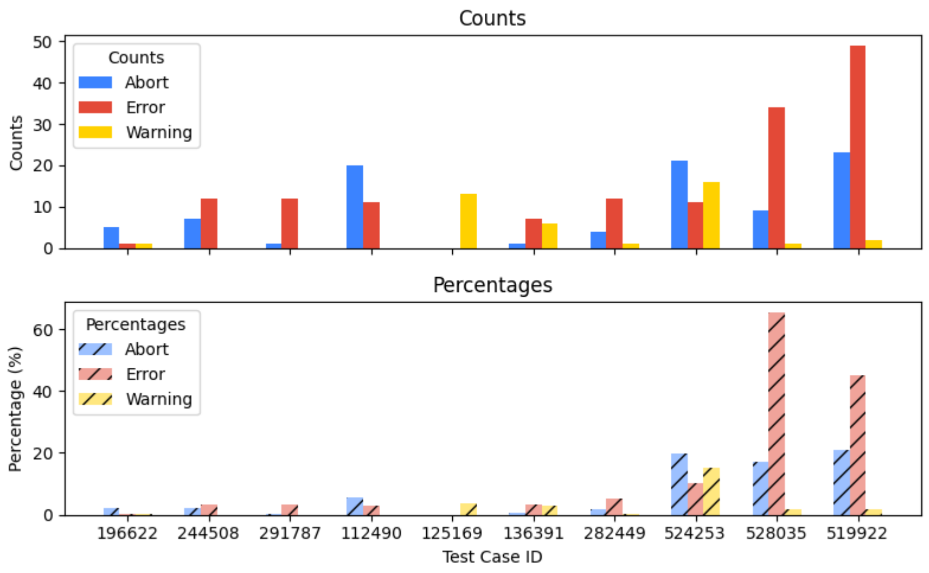

Figure 3.1: The counts for each result and the percentage of each result from the given test case ID.

After this process, the precise information of 223 test cases is generated from 59109 runs, which can be used in the following ranking models in the next section.

3.2 AHP-WSM model

As described in section 2.1, the conventional ranking model implemented in this study combines AHP and WSM. The AHP was chosen to calculate the weight for each criterion since the hierarchical structure can benefit complicated decision scenarios, as it allows for a systematic review of multiple criteria. Moreover, it takes the user's preference as a count. The WSM will then utilise the calculated weight to generate a score for each item based on the values from each criterion. The major advantage is that the straightforward calculation can be valuable for a large number of criteria or requires real-time processing, which can provide computational efficiency and clarity. The subsequent sections will provide a detailed description of the implementation process.

3.2.1 AHP

The AHP determines the weight assigned to each criterion. First of all, a pairwise comparison matrix is needed. As outlined in section 2.1, the pairwise comparison matrix is derived from the relative importance of each criterion pair regarding the Fundamental scale table[4].

Following the data processing outlined in the previous section, this situation involves the consideration of five criteria. These criteria include the percentage of runs that led to abort, error, and warning, denoted as AP, EP, and WP, respectively. Additionally, the percentage of recent failures referred to as RFP, and the outcome of the most recent execution of the test case is also considered, denoted as MRR. The determination of the priority of each criterion, as shown in Table 3.2, is established in the following manner:

The RFP attribute holds the highest priority compared to the other four attributes. It signifies the most current outcomes seen in the test scenario. Suppose a test case exhibits a high value for this property. In that case, it indicates that the test case has yet to achieve a successful execution for a considerable period when considering the test case's execution schedule.

The MRR is in the second place. The priority for different outcomes from the test run is abort> error > warning, as described before. Suppose the outcome of the test case is aborted during the most recent execution. In that case, it indicates that the script for the test case includes a critical problem that requires immediate resolution.

Next, the analysis includes evaluating the failure rate for each outcome, namely AP, EP, and WP. These three attributes can offer helpful insights where test cases have performed similarly in RFP and MRR. Among these three, the priorities are similar to the MRR: $AP > EP > WP$.

The normalized pairwise comparison matrix is presented in Table 3.3, while the corresponding weights derived from this matrix are displayed in Table 3.4. To verify the consistency of the weights, the weighted sum matrix (Table 3.5) and consistency vector (Table 3.6) are computed. The confidence interval (CI) value is

$$CI = \frac{\lambda_{\max} - n}{n-1} = \frac{(5.3896 + 6.3064 + 6.4962 + 5.7497 + 5.6384)/5 - 5}{5-1} = 0.08533$$

	WP	EP	AP	RFP	MRR
WP	1	$\frac{1}{2}$	$\frac{1}{3}$	$\frac{1}{9}$	$\frac{1}{7}$
EP	2	1	$\frac{1}{3}$	$\frac{1}{9}$	$\frac{1}{7}$
AP	3	3	1	$\frac{1}{9}$	$\frac{1}{7}$
RFP	9	9	9	1	3
MRR	7	7	7	$\frac{1}{3}$	1

Table 3.2: Pairwise comparison matrix.

	WP	EP	AP	RFP	MRR
WP	0.0454	0.0243	0.0162	0.0662	0.0316
EP	0.0909	0.0487	0.0162	0.0662	0.0316
AP	0.1363	0.1463	0.0491	0.0662	0.0316
RFP	0.4090	0.4390	0.4426	0.6024	0.6787
MRR	0.3181	0.3414	0.3443	0.1987	0.2262

Table 3.3: Normalized pairwise comparison matrix.

WP	0.0367
EP	0.0507
AP	0.0859
RFP	0.5143
MRR	0.2857

Table 3.4: Weights.

	WP	EP	AP	RFP	MRR
WP	0.0367	0.0253	0.0283	0.0565	0.0399
EP	0.0734	0.0507	0.0283	0.0565	0.0399
AP	0.1101	0.1521	0.0859	0.0565	0.0399
RFP	0.3303	0.4563	0.7731	0.5143	0.8571
MRR	0.2569	0.3549	0.6013	0.1697	0.2857

Table 3.5: Weighted sum matrix.

WP	5.0857
EP	4.9072
AP	5.1746
RFP	5.6992
MRR	5.8400

Table 3.6: Consistency vector.

	Criteria				
	WP	EP	AP	RFP	MRR
Alts.	(0.036	0.0507	0.085	0.514	0.285)
tc_115370	0.454	0.454	2.272	0	0
tc_528035	1.923	65.384	17.307	75.0	3
tc_519922	1.834	44.954	21.100	6.422	3
tc_354400	0	3.846	96.153	100.0	5
tc_507580	57.758	6.896	0	54.741	3

Table 3.7: Decision matrix.

While the correlation coefficient (CR) is

$$CR = \frac{CI}{RI} = \frac{0.0862}{1.11} = 0.0768$$

It is less than 0.1, which means the judgement is consistent, and the weights for criteria are available to use in the next step.

3.2.2 WSM

In order to construct a decision matrix suitable for utilization in the WSM, it is necessary to transform all linguistic variables into numerical values. As previously mentioned, the data preprocessing stage generates five criteria: AP, EP, WP, RFP, and MRR. MRR is the only one with linguistic value, indicating the outcome from the most recent run. In this study, the propriety for outcomes is established, and the corresponding number values assigned to linguistic outcomes are as follows: $abort = 5$, $error = 3$, $warning = 2$, $pass = 0$. An example of a decision matrix is shown in Table 3.7, which contains five test cases.

The score can be calculated by following the formula 2.1 of the given test cases, which are:

$$\begin{aligned} S_{tc\_115370} &= 0.454 \times 0.036 + 0.454 \times 0.0507 + 2.272 \times 0.085 + 0 \times 0.514 + 0 \times 0.285 \\ &= 0.232 \end{aligned} \tag{3.1}$$

Similarly, the others are: $S_{tc\_528035} = 14.606$, $S_{tc\_519922} = 8.294$, $S_{tc\_354400} = 61.19$, $S_{tc\_507580} = 31.420$.

Therefore, among the five test cases provided as examples, the following ranking is derived: $tc\_354400 > tc\_507580 > tc\_528035 > tc\_519922 > tc\_115370$. The overall result will be presented in Chapter 5.

3.3 LambdaMART

LTR has three main types of approaches: pointwise, pairwise and listwise. The main difference between these three types of approach is how many documents they will consider each time and what/when the loss function is used. The pointwise approach only considers one document at a time and uses standard regression and classification algorithms as loss functions. The pairwise approach considers pairs of documents within a query, and the loss functions used here target to minimize the number of inversions to increase the correct ranking of pairs. The listwise approach considers all the documents in a query simultaneously and uses particular loss functions capable of assessing the overall quality of the entire list[18].

The reason pairwise LambdaMART is chosen is the robustness in handling label noise, which often happens in real-world datasets. Instead of relying on absolute relevance scores, the model learns from the relative ordering of alternatives, making it more adjustable to noisy or uncertain labels.

3.3.1 Additional Data Preprocessing

Inside the data preprocessing phase of the AHP-WSM model, the dataset undergoes two essential procedures: feature selection and transformation. It is important to note that LambdaMART, a supervised machine learning technique, requires a relevant label for each item inside the dataset. In addition, it is necessary to partition the dataset into two sets, namely the training set and the test set, to employ them in their respective phases.

Relevance Label Assigning

The primary approach for assigning relevance labels to items is by human judgment. However, in the context of this study, the implemented feature might require daily execution. Therefore, a more practical approach involves automating the process of assigning labels.

As discussed in the preceding section, the AHP-WSM model involves the computation of scores for each test case. These scores are then utilized to give relevance labels in this section. The assignment of labels is accomplished by categorizing the scores into five distinct groups, each associated with a label ranging from 4 to 0, representing different levels of relevance, with higher values indicating higher relevance.

The reason for not using the original score derives from its wide range of 60 to 0, which creates significant relevance levels. This wide range of levels can complicate the training

Label	0	1	2	3	4
Number	207	9	1	3	13
Pct(%)	88.84%	5.57%	3.86%	1.28%	0.43%

Table 3.8: The number and percentage of the values that are labelled in the range of 0 to 4.

process and demand a larger dataset. However, as previously discussed in the data preprocessing section in the AHP-WSM model, the dataset size becomes significantly reduced after performing the transformation procedure. A comparison of the range of calculated labels and the original score is shown in Figure 3.2.

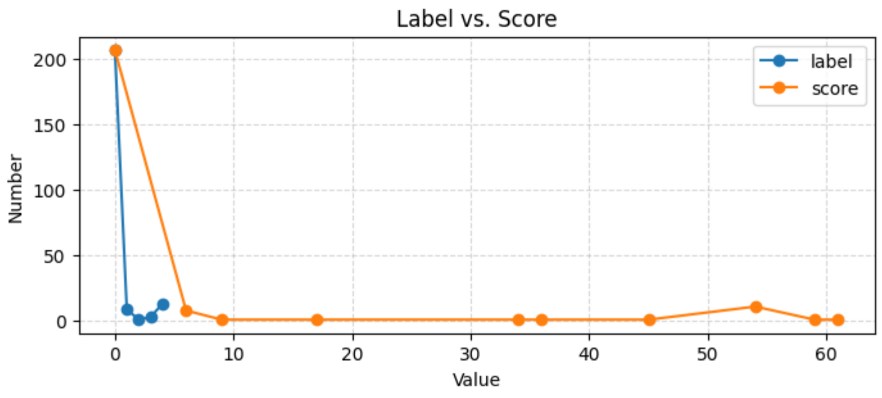

Figure 3.2: The comparison of the range of calculated labels and the original scores.

Dataset Splitting

Once the relevance labels have been assigned, the data is prepared for split into a training set and a test set. The technique employed for this purpose is known as Stratified Sampling[21]. The rationale for selecting this approach stems from the presence of an imbalanced dataset in the current investigation. According to the statistics presented in Table 3.8, most of the data, precisely over 88%, was labeled as 0. Conversely, a relatively minor portion, approximately 11.15%, was categorized into groups ranging from 1 to 4. Furthermore, the levels of relevance can be regarded as strata, and each test case can be treated as an independent unit that fits the requirements for Stratified Sampling[22].

Stratified sampling can be implemented using the pandas, an open-source Python library, with a division of 70% for training and 30% for testing. Figure 3.3 displays the range covered within each set, and the corresponding number and percentage are as shown in Table 3.9.

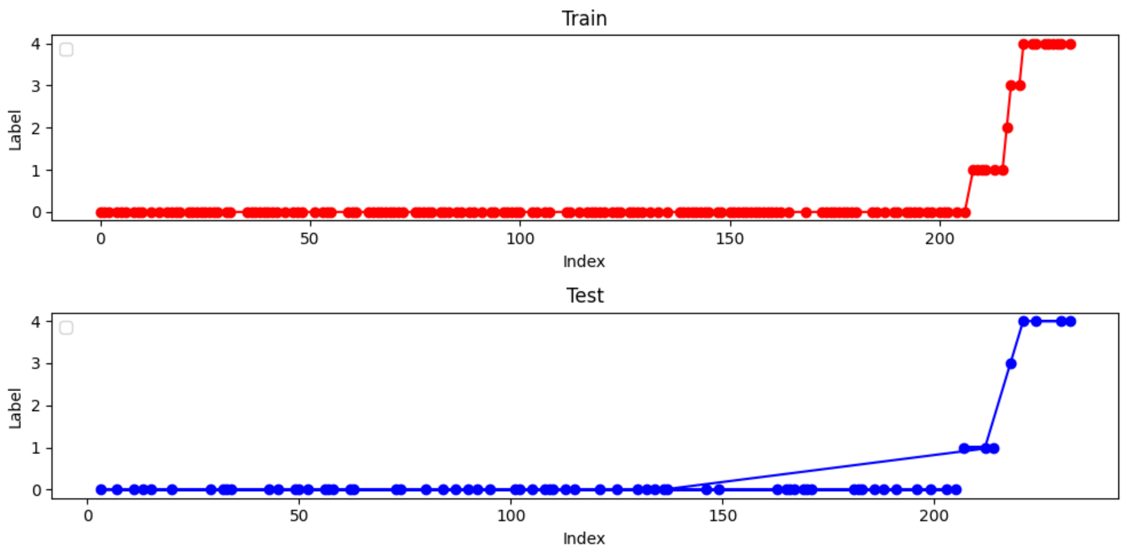

Figure 3.3: The label covered by the training set and the test set.

	Label	0	1	2	3	4
Training	Number	145	6	1	2	9
	Pct(%)	88.95%	3.68%	0.61%	1.22%	5.52%
Test	Number	62	3	0	1	4
	Pct(%)	88.57%	4.28%	0	1.43%	5.71%

Table 3.9: The number and percentage of the values labelled in a range of 0 to 4 from the training and test sets.

Transformation

An additional transformation is needed to calculate the decision matrix that is going to be fed into the LambdaMART algorithm. An example dataset that may be used for this purpose is MSLR-WEB30k[23], a dataset released by Microsoft specifically for research on learning to rank. The required format needs to be the following structure:

- The first column represents the relevance label.
- The query ID and corresponding value will be in the second column.
- The remaining columns will consist of the features and their respective values, presented in the format of $featurenumber : value$.

One row from the training set is shown in Figure 3.4. In this scenario, all query IDs are equal to 1 because there is only one query, which is "find the often failed test cases.". The feature number to the corresponding feature is in the order of RFP, WP, EP, AP, and MRR.

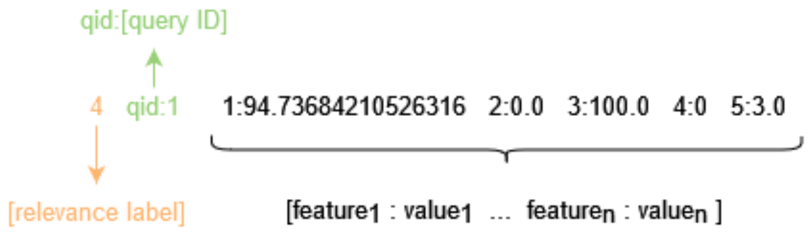
qid:[query ID]
4 qid:1 1:94.73684210526316 2:0.0 3:100.0 4:0 5:3.0
[relevance label]
[feature1 : value1 ... featuren : valuen]

Figure 3.4: A example for a line in the dataset after transformation.

3.3.2 Modelling

XGBoost is used to implement the LambdaMART ranking model. XGBoost, short for Extreme Gradient Boosting, is an open-source library that provides an optimised gradient boosting framework in which the machine learning algorithms are implemented[24]. Gradient Boosting is a machine learning technique that combines the predictions of multiple weak learners, such as the decision trees in LambdaMART, to create a strong predictive model. The fundamental idea behind gradient boosting is to iteratively improve the model's performance by optimising a loss function. XGBoost's exceptional speed and efficiency have made it a popular choice among Kaggle competition winners compared to other implementations.

By specifying the *objective* parameter as *rank* : *pairwise*, setting the learning rate to 0.01, and running 1000 iterations, a LambdaMART ranking model can be effectively trained. Once the training set is fed into the model, the resulting decision tree and feature importance are visualised, as depicted in Figure 3.5 and 3.6.

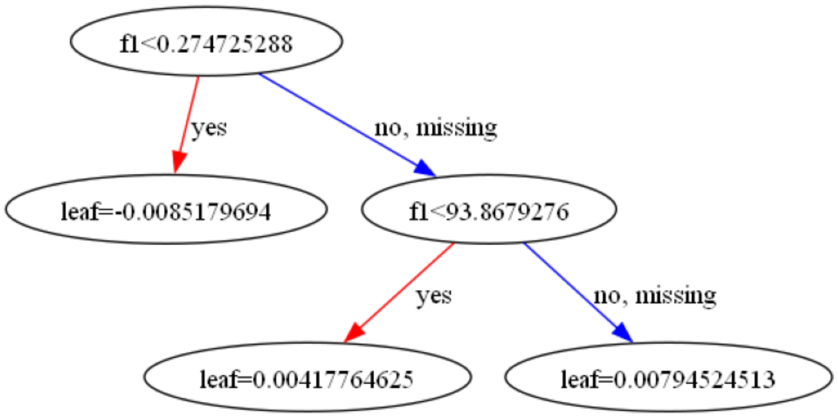

Figure 3.5: The visualized decision tree from the trained LambdaMART ranking model, where f1 represented RFP as feature 1, and each leaf represented a determined value from the dataset.

Subsequently, the model is used to make predictions by applying the test dataset. These predictions are numerical values representing the relevance of each document for a specific

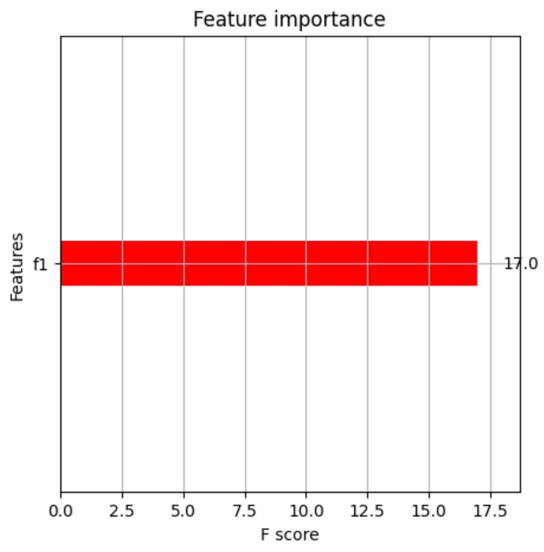

Figure 3.6: The visualized feature importance from the trained LambdaMART ranking model, where f1 represented RFP as feature 1, and the feature score (F score) for importance is 17.

query derived from the leaves of the decision tree. The ranked list is created by sorting these predictions and presented in Chapter 5.

3.4 Evaluation

The Normalised Discounted Cumulative Gain (NDCG) evaluation metric is used to evaluate the LambdaMART model's performance. NDCG is a widely used evaluation metric in learning-to-rank scenarios. This metric evaluates the accuracy of the ranking produced by the model, considering both the relevance labels and the ranking position assigned by the model. A higher NDCG value indicates a more satisfying ranking quality, representing that the items at the top of the ranked list have higher relevance labels compared to others. The NDCG calculation comprises two components: Discounted Cumulative Gain (DCG) and ideal DCG (iDCG). DCG can be computed using the following formula (3.2), where $relevance_i$ represents the relevance score of items, and i denotes the position of an item in the ranked list. iDCG, on the other hand, represents the ideal DCG score when the top-k items are perfectly ranked in descending order of relevance labels. NDCG is calculated by the formula 3.3.

$$DCG_k = \sum_{i=1}^{k} \frac{relevance_i}{log_2(i+1)} \tag{3.2}$$

$$NDCG = \frac{DCG}{iDCG} \tag{3.3}$$

As illustrated in Figure 3.7, multiple k levels are used to evaluate the model.

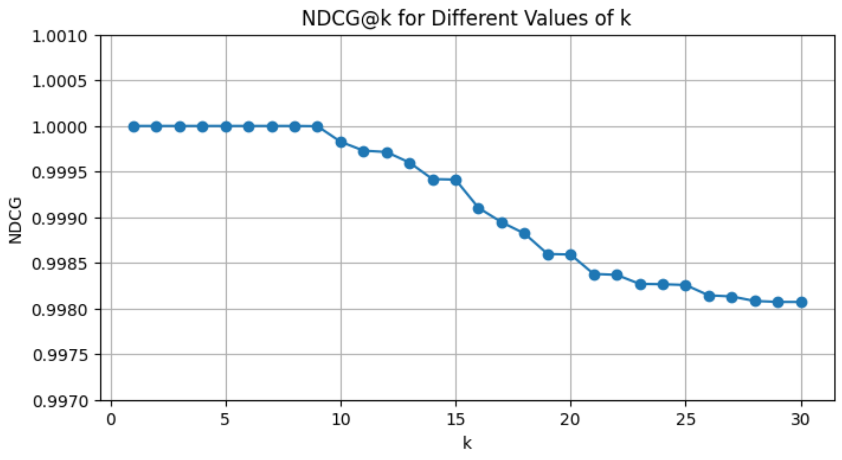

Figure 3.7: NDCG result at various level K.

Chapter 4

Approaches for Clustering Model

The primary topic of this chapter is the implementation of clustering. The initial data preprocessing stage will be discussed, containing its fundamental characteristics. Subsequently, the rationale behind the choice of similarity metrics, the clustering method, and the implementation specifics are presented within the modelling section.

4.1 Data Preprocessing

In order to cluster test cases based on error messages, it is required to consider solely the most recent execution details for each test case. Furthermore, this latest execution should yield either an Error, Abort, or Warning result. Though the clustering algorithms only need the error messages, the essential information is also needed to present the result to the user: the test case ID, the unique identifier for the most recent execution, the date of execution, and the corresponding outcome.

After extracting the needed information, the dataset contains 228 items. The basic properties of the dataset are shown in Table 4.1.

Texts	Vocabulary	Avg. Words	Max. Words	Min. Words
228	347	19.54	316	25

Table 4.1: The basic properties of the dataset, include the total number of texts, the vocabulary size, the average, maximum, and minimum number of words in messages.

4.2 Modelling

This section provides further information regarding the similarity measures employed in the present study, namely Jaccard Similarity and Cosine Similarity. The motivation behind selecting these two similarity metrics will be clarified, followed by their implementation specifics. Subsequently, a comprehensive explanation of the Agglomerative Clustering algorithm will be provided, along with a discussion of its implementation details.

4.2.1 Jaccard Similarity and Cosine Similarity

In the context of text data, both Jaccard and cosine similarities are popular choices due to their effectiveness in capturing different aspects of textual similarity.

As mentioned in Section 2.2.2, Jaccard similarity is a measure that quantifies the similarity between two sets by comparing the intersection and union of their elements. On the other hand, the cosine similarity measures the cosine of the angle between two vectors, representing the direction of similarity between them.

Jaccard similarity leans more keenly towards capturing topical overlaps, but not considering terms' frequency can overlook the importance of terms that appear multiple times. In contrast, cosine similarity considers both the presence and frequency of terms, which can benefit the capturing semantic similarity and make it less sensitive to the text length but also can be vulnerable to noise[25].

Given the distinctive characteristics of these two measures, a reasonable choice can be made for the dataset at hand by conducting a close analysis. This assessment efficiently determines which measure aligns more closely with the requirements of the dataset.

Implementation of Jaccard Similarity

The Jaccard Similarity for a pair of messages is computed using Algorithm 4.1. Each pair of messages is processed through the algorithm to construct the overall similarity matrix for the dataset. The resulting similarity matrix is depicted on the left side of Figure 4.1.

Algorithm 4.1 Jaccard Similarity

```
Input: Text Data: $m_1, m_2$
Output: Jaccard Similarity: $similarity\_score$
1: $set_1 = \text{tokenize}(m_1)$
2: $set_2 = \text{tokenize}(m_2)$
3: $intersection = set_1 \cap set_2$
4: $union = set_1 \cup set_2$
5: $similarity\_score = \frac{|intersection|}{|union|}$
```

A step-by-step calculation containing two sample messages is given to make the algorithm more straightforward to understand.

step 1. Given two messages m_1 and m_2:

$m_1 =$ *"interrupt count expected 1 got 0"*

$m_2 =$ *"0 is not Equal to the expected value: 3"*

step 2. Tokenize m_1 and m_2:

$set_1 =$ *{interrupt, count, expected, 1, got, 0}*

$set_2 =$ *{0, is, not, Equal, to, the, expected, value:, 3}*

step 3. Calculate the intersection between set_1 and set_2:

$$\begin{aligned} intersection &= set_1 \cap set_2 \\ &= \{expected,\ 0\} \\ &= 2 \end{aligned}$$

step 4. Calculate the union between set_1 and set_2:

$$\begin{aligned} union &= set_1 \cup set_2 \\ &= \{interrupt,\ count,\ expected,\ 1,\ got,\ 0,\ is,\ not,\ Equal,\ to,\ the,\ value{:},\ 3\ \} \\ &= 13 \end{aligned}$$

step 5. Calculate the similarity score:

$$similarity\_score = \frac{|intersection|}{|union|} = \frac{2}{13} \approx 0.154$$

Implementation of Cosine Similarity

Algorithm 4.2 outlines the process for calculating the Cosine Similarity between pairs of messages. The resulting Cosine Similarity matrix is presented on the right side of Figure 4.1.

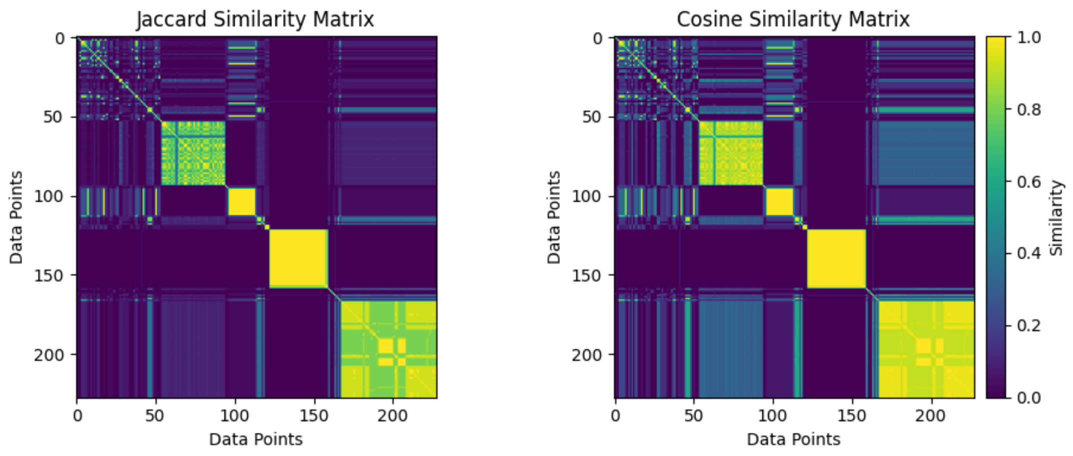

Figure 4.1: The Jaccard Similarity matrix (left) and the Cosine Similarity matrix (right). The similarity scale next to the cosine similarity matrix applies to both figures.

Algorithm 4.2 Cosine Similarity

```
Input: Text Data: $m_1, m_2$
Output: Cosine Similarity: $similarity\_score$
1: $vector_1 = \text{vectorize}(m_1)$
2: $vector_2 = \text{vectorize}(m_2)$
3: $dot\_product = \sum_{i=1}^{n}(vector_1[i] \times vector_2[i])$
4: $magnitude\_vector_1 = \sqrt{\sum_{i=1}^{n}(vector_1[i]^2)}$
5: $magnitude\_vector_2 = \sqrt{\sum_{i=1}^{n}(vector_2[i]^2)}$
6: $similarity\_score = \frac{dot\_product}{magnitude\_vector_1 \times magnitude\_vector_2}$
```

As the step-by-step calculation for jaccard similarity, an example of cosine similarity is also given.

step 1. & step 2. Same with the jaccard similarity.

step 3. Identify all the unique words from both sets and create the vocabulary:

vocabulary = {interrupt, count, expected, 1, got, 0, is, not, Equal, to, the, value:, 3 }

step 4. Vectorize two sets based on the vocabulary. If a word is present in the set, then the binary vector for this word is 1, otherwise is 0:

$vector_1 =< 1, 1, 1, 1, 1, 1 >$

$vector_2 =< 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1 >$

step 5. Calculate the dot product for $vector_1$ and $vector_2$:

$$\begin{aligned}
dot\_product &= \sum_{i=1}^{n}(vector_1[i] \times vector_2[i]) \\
&=< 1, 1, 1, 1, 1, 1 > \cdot < 0, 0, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1 > \\
&= (1 \cdot 0) + (1 \cdot 0) + (1 \cdot 1) + (1 \cdot 0) + (1 \cdot 0) + (1 \cdot 1) + (0 \cdot 1) + (0 \cdot 1) + (0 \cdot 1)... \\
&= 2
\end{aligned}$$

step 6. Calculate the magnitude for $vector_1$ and $vector_2$:

$$\begin{aligned} magnitude\_vector_1 &= \sqrt{\sum_{i=1}^{n}(vector_1[i]^2)} \\ &= \sqrt{1^2+1^2+1^2+1^2+1^2+1^2} \\ &= \sqrt{6} \end{aligned}$$

$$\begin{aligned} magnitude\_vector_2 &= \sqrt{\sum_{i=1}^{n}(vector_2[i]^2)} \\ &= \sqrt{0^2+0^2+1^2+0^2+0^2+1^2+1^2+1^2+1^2+1^2+1^2+1^2} \\ &= \sqrt{8} \end{aligned}$$

step 7. Calculate the similarity score:

$$\begin{aligned} similarity\_score &= \frac{dot\_product}{magnitude\_vector_1 \times magnitude\_vector_2} \\ &= \frac{2}{\sqrt{6}\times\sqrt{8}} \\ &\approx 0.452 \end{aligned}$$

4.2.2 Agglomerative Clustering

Before explaining why the agglomerative clustering algorithm is used, a more detailed discussion about the clustering algorithm is needed.

Clustering algorithms can be broadly categorized into two main types: hierarchical and partitioning methods. The key distinction lies in their approach to clustering. Hierarchical algorithms generate a dendrogram, a tree-like structure, and the user selects the number of clusters by choosing a specific level to cut the dendrogram. In contrast, partitioning methods require the user to specify the desired number of clusters before running the algorithm.

As in this study, a flexible cluster number can be more beneficial to the user, as the different aspect of the overall error messages appears in the recent run test cases, so the Agglomerative Clustering, a hierarchical method, is selected. Furthermore, based on the calculation of the distance between two clusters, the agglomerative clustering can be further classified into three types:

- Single-link clustering: consider the distance between two clusters equal to the closest objects from each cluster.
- Complete-link clustering: consider the distance between two clusters equal to the longest distance between two objects from each cluster.
- Average-link clustering: consider the distance between two clusters equal to the average distance between any object from the two clusters.

Among these three types, complete-link clustering usually creates more compact clusters and more usable hierarchies than the other two because single-link clustering might lead two nearby clusters to merge. In contrast, average-link clustering may cause elongated clusters to split. Overall, this study uses the complete-link agglomerative clustering algorithm.

Agglomerative Clustering with Jaccard Similarity

To implement Agglomerative Clustering, the scikit-learn library (sklearn) is utilized. Sklearn, a widely-used open-source machine learning library for Python[26], provides a comprehensive set of tools for various machine learning tasks. This study specifies the $linkage =' complete'$ parameter to enable complete-link agglomerative clustering. The Jaccard similarity matrix, generated in the previous section, is then used as input to create an agglomerative clustering model based on Jaccard similarity.

The resulting dendrogram illustrates the hierarchical merging of clusters at each step, offering insights into the underlying structure of the data is presented in Figure 4.2.

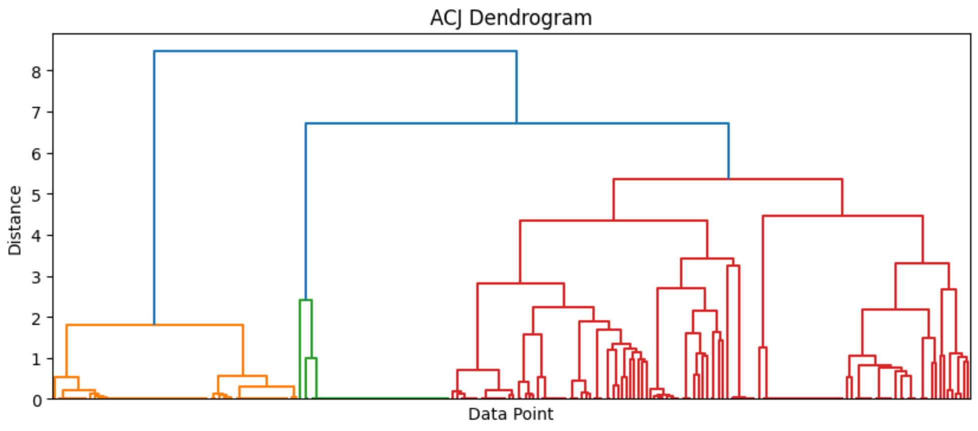

Figure 4.2: The clustering dendrogram from ACJ with data points as the x-axis and the distance between each data point as the y-axis.

Agglomerative Clustering with Cosine Similarity

Similarly, the agglomerative clustering model with cosine similarity can be generated by feeding the cosine similarity matrix to the complete-link clustering function. Moreover, the corresponding dendrogram is shown in Figure 4.3

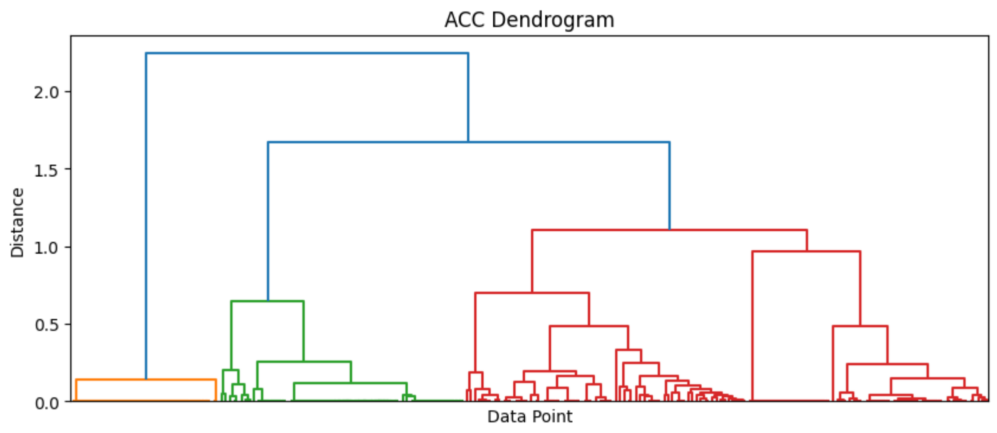

Figure 4.3: The clustering dendrogram from ACC with data points as the x-axis and the distance between each data point as the y-axis.

Cluster Cut

The cluster cut operation extracts a certain number of clusters from the dendrogram produced by the clustering algorithm. Specifically, a preferred number of clusters is selected, leading to a distance on the dendrogram, which indicates the dissimilarity between the data points, and then makes the cut at that level. An example containing ten error messages selected from the dataset, labelled from 0 to 9, provides a straightforward understanding. To clarify, while the labels presented here provide understanding, they do not correspond to the actual labels in the dataset.

The dendrograms generated by ACJ and ACC for the above example messages are shown in Figures 4.4 and 4.5, respectively. The required number of clusters for this case is 5. This implies that the distance for performing cluster cuts is 0.93 from ACJ and 0.14 from ACC, which is also shown in the figures. The resulting clusters obtained from the cluster cuts and their respective members are displayed in Table 4.3.

The full results of the ACJ and ACC for the entire dataset are outlined in Chapter 5.

Index	Error Message
0	1 configfiles failed to load
1	Unable to write signal; ICI Error: Failed to write signal number '3' on: b'V1'
2	EasyICI: Unable to query signals on device: Board under client: b'\x01 \x04'; Error: Failed to retrieve signal list: b'unexpected response (internal error 2)'
3	Actual elog sequence is not as expected
4	interrupt count expected 1 got 0
5	$0 was not received when running command ips tic 2 Spi1/Fosi1 Connect
6	FPGA upgrade failed expected 20210326.2_Paint.Hardware, got 20230820.2_Paint.Hardware
7	readres/degrees: 247.637 is not Between 245.6 and 247.6
8	RunTestIteration CycleUnitTestRackPower error: [Errno Expecting value] : 0
9	PortMcobIn1/Value: 61695 is not Equal to the expected value: 37119

Table 4.2: The selected ten messages contain labels from 0 to 9.

cluster No.	**Included data points**	
	ACJ	**ACC**
cluster 1	[0, 1, 2, 8]	[1, 2]
cluster 2	[6]	[0, 4, 6]
cluster 3	[3, 5, 7]	[5]
cluster 4	[9]	[3, 7, 9]
cluster 5	[4]	[8]

Table 4.3: Clustering Results (k=5) from ACJ and ACC of the given example, shows the cluster number and the members belonging to it, where the underlined members differ between the two models.

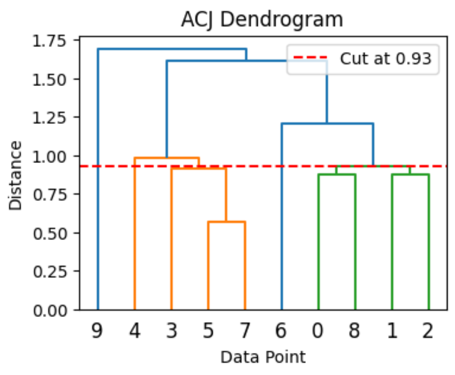

Figure 4.4: The clustering dendrogram from ACJ where data points are the x-axis, and the distance between each data point is the y-axis. The required cluster cut happens at distance=0.93.

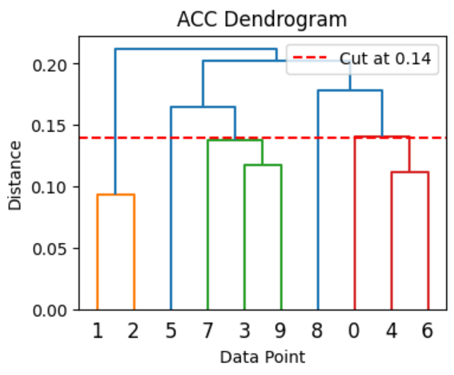

Figure 4.5: The clustering dendrogram from ACC where data points are the x-axis, and the distance between each data point is the y-axis. The required cluster cut happens at distance=0.14.

4.3 Evaluation

The evaluation measure used is the silhouette score, which computes the average silhouette coefficients of all data points. The following equation 4.1 calculates the silhouette coefficient: a is the mean distance from a data point to other data points in the same cluster, and b is the mean distance from a data point to other data points in different clusters. The range of silhouette score is from -1 to 1, where a high value indicates that the object is well-matched to its cluster and poorly matched to neighbouring clusters[27][28].

$$SilhouetteScore = \frac{b-a}{max(a,b)} \tag{4.1}$$

To measure the quality of the clustering model, a number of clusters are used, ranging from 3 to 10, as shown in Table 4.4.

No. of Clusters		3	4	5	6	7	8	9	10
Silhouette	**ACJ**	0.191	0.308	0.307	0.261	0.342	0.346	0.348	**0.508**
	ACC	0.605	0.686	**0.749**	0.728	0.713	0.730	0.708	0.707

Table 4.4: Silhouette scores for the agglomeration clustering method with Jaccard similarity and Cosine similarity.

Chapter 5

Result

This chapter presents the outcomes of the implementations covered in earlier chapters. The first outcome is ranking the 20 most commonly failed test cases produced from the AHP-WSM model and the LambdaMART model. Then, the result of clustering the test cases based on the error messages.

5.1 Ranking Results

The ranking outcomes are presented as lists, displaying the identified features alongside the scores or predictions provided by different models. Table 5.1 shows the outcomes generated from the AHP-WSM model, including the computed score. The results obtained from the LambdaMART model and corresponding predictions are offered in Table 5.2.

	ID	WP	EP	AP	RFP	MRR	Score
1	*tc*_354400	0.000	3.846	96.154	100.000	5	61.313
2	*tc*_513421	0.000	40.000	60.000	100.000	3	59.469
3	*tc*_282458	0.000	97.500	2.500	94.787	3	54.764
4	*tc*_114854	0.000	100.000	0.000	94.787	3	54.676
5	*tc*_115285	0.000	100.000	0.000	94.787	3	54.676
6	*tc*_114909	0.000	100.000	0.000	94.737	3	54.650
7	*tc*_115003	0.000	100.000	0.000	94.545	3	54.552
8	*tc*_115004	0.000	100.000	0.000	94.545	3	54.552
9	*tc*_115188	0.000	100.000	0.000	94.118	3	54.332
10	*tc*_115189	0.000	100.000	0.000	94.037	3	54.290
11	*tc*_112180	0.000	99.500	0.500	93.897	3	54.236
12	*tc*_115294	0.000	100.000	0.000	93.897	3	54.218
13	*tc*_112289	0.000	100.000	0.000	93.868	3	54.203
14	*tc*_528035	0.000	82.051	17.949	75.000	3	45.131
15	*tc*_519604	0.000	83.871	16.129	57.407	3	36.019
16	*tc*_507580	0.000	99.213	0.787	54.741	3	34.108
17	*tc*_547519	0.000	0.000	100.000	15.385	5	17.931
18	*tc*_519922	0.000	100.000	0.000	6.422	3	9.230
19	*tc*_112482	0.000	100.000	0.000	0.948	3	6.415
20	*tc*_526033	0.000	100.000	0.000	0.746	3	6.311

Table 5.1: Results from the AHP-WSM model, with the value from each attribute and the score calculated by the model.

	ID	WP	EP	AP	RFP	MRR	Preds
1	*tc*_354400	0.000	3.846	96.154	100.000	5	0.075
2	*tc*_115003	0.000	100.000	0.000	94.545	3	0.075
3	*tc*_115294	0.000	100.000	0.000	93.897	3	0.075
4	*tc*_112180	0.000	99.500	0.500	93.897	3	0.075
5	*tc*_115188	0.000	100.000	0.000	94.118	3	0.075
6	*tc*_115004	0.000	100.000	0.000	94.545	3	0.075
7	*tc*_115189	0.000	100.000	0.000	94.037	3	0.075
8	*tc*_114909	0.000	100.000	0.000	94.737	3	0.075
9	*tc*_115285	0.000	100.000	0.000	94.787	3	0.075
10	*tc*_114854	0.000	100.000	0.000	94.787	3	0.075
11	*tc*_282458	0.000	97.500	2.500	94.787	3	0.075
12	*tc*_513421	0.000	40.000	60.000	100.000	3	0.075
13	*tc*_112289	0.000	100.000	0.000	93.868	3	0.059
14	*tc*_528035	0.000	82.051	17.949	75.000	3	0.059
15	*tc*_519604	0.000	83.871	16.129	57.407	3	0.059
16	*tc*_507580	0.000	99.213	0.787	54.741	3	0.059
17	*tc*_547519	0.000	0.000	100.000	15.385	5	0.052
18	*tc*_198620	0.000	100.000	0.000	0.452	3	0.047
19	*tc*_439419	0.000	100.000	0.000	0.273	3	0.047
20	*tc*_219232	0.000	100.000	0.000	0.275	3	0.047

Table 5.2: Results from the LambdaMART model, with the value from each attribute and the prediction generated by the model.

5.2 Clustering Results

The clustering outcomes consist of two components: firstly, the dendrograms generated by two approaches, which illustrate the distance between each data point and the cluster cuts for two different required cluster numbers; secondly, the tables that display the allocation of data points to certain clusters.

The dendrogram generated from ACJ is shown in Figure 5.1, and the one produced by ACC is in Figure 5.2. As presented in Section 4.3, the evaluation results for the optimal cluster numbers, denoted as k, for Jaccard Similarity and Cosine Similarity, are $k = 5$ and $k = 10$, respectively. Therefore, the cluster cuts to meet the requirements are made and shown in the figures.

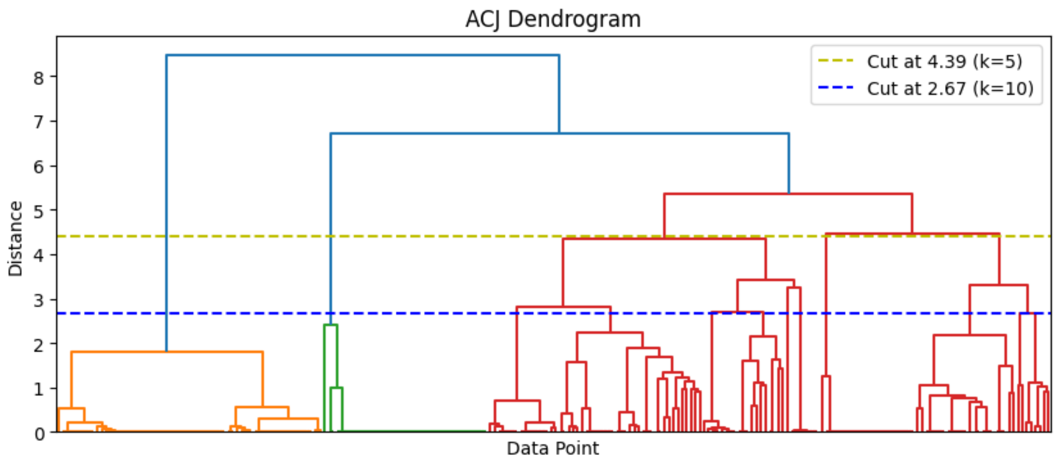

Figure 5.1: The clustering dendrogram from ACJ where data points are the x-axis, and the distance between each data point is the y-axis. The required cluster cut happens at distance=4.39(k=5) and distance=2.67(k=10).

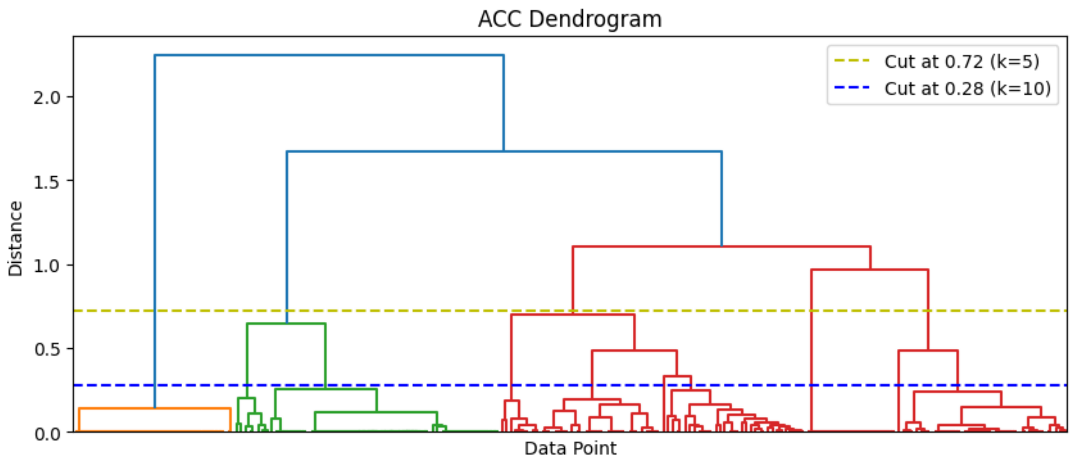

Figure 5.2: The clustering dendrogram from ACJ where data points are the x-axis, and the distance between each data point is the y-axis. The required cluster cut happens at distance=0.72(k=5) and distance=0.28(k=10)

Then, Table 5.3 shows the allocation of data points to specific clusters when the cluster number (k) is set to 5 for both ACJ and ACC. Table 5.4 displays the corresponding results when the cluster number (k) is set to 10.

cluster No.	**Included data points**	
	ACJ	**ACC**
cluster 1	[167, . . . , 227]	[167, . . . , 227]
cluster 2	[122, . . . , 158, 161]	[122, . . . , 158]
cluster 3	[3, 4, 5, 6, 9, 11, 13, 14, 20, 27, 28, 29, 30, 31, 32, 36, 37, 38, 44, 45, 46, 47, 48, 53, 54, . . . , 93, 113, 114, 115, 116, 117, 118, 159, 160, 163, 164, 165, 166]	[54, . . . , 93]
cluster 4	[7, 17, 22, 42, 50, 96,. . . , 112]	[7, 17, 42, 50, 96,. . . , 112]
cluster 5	[0, 1, 2, 8, 10, 12, 15, 16, 18, 19, 21, 23, 24, 25, 26, 33, 34, 35 39, 40, 41, 43, 49, 51, 52, 94, 95, 119, 120, 121, 162]	[0, 1, 2, 3, 4, 5, 6, 8, 9, 10, 11, 12, 13, 14, 15, 16, 18, 19 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37 38, 39, 40, 41, 43, 44, 45, 46, 47, 48, 49, 51, 52, 53, 63, 94, 95, 113, 114, 115, 116, 117, 118, 119, 120, 121, 159, 160, 161, 162, 163, 164, 165, 166]

Table 5.3: Clustering Results (k=5) from ACJ and ACC, shows the cluster number and the members belonging to it, where the underlined members differ between the two models.

cluster No.	**Included data points**	
	ACJ	**ACC**
cluster 1	[167, . . . , 184, 185, 186, 187, 188, 189, 190, 191, . . . , 227]	[167, . . . , 184, 191, . . . , 227]
cluster 2	[122, . . . , 158, 161]	[122, . . . , 158]
cluster 3	[64, 67, 68, 69, 76, 79, 80, 81, 83, 86, 87, 88, 90, 91, 92, 93]	[54, 55, 56, 57, 58, 59, 60, 61, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93]
cluster 4	[20, 27, 28, 36, 44, 48, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 65, 66, 70, 71, 72, 73, 74, 75, 77, 78, 82, 84, 85, 89, 117, 160, 164]	[62]
cluster 5	[6, 11, 29, 30, 31, 32, 38, 45, 46, 47, 53, 113, 114, 115, 116, 118, 159, 165, 166]	[159]
cluster 6	[3, 4, 5, 9, 13, 14, 37]	[2, 3, 4, 5, 6, 8, 9, 10, 12, 13, 14, 15, 16, 18, 19, 25, 26, 35, 37, 38, 39, 49, 51, 52, 113, 119, 120, 121]
cluster 7	[163]	[45, 46, 47, 114, 115, 116, 118, 163, 165]
cluster 8	[7, 17, 22, 42, 50, 96, . . . , 112]	[7, 17, 42, 50, 96, . . . , 112]
cluster 9	[2, 8, 10, 12, 15, 16, 18, 19, 23, 24, 25, 26, 34, 35, 39, 49, 51, 52, 94, 95, 119, 120, 121]	[181, 185, 186, 187, 188, 189, 190, 200]
cluster 10	[0, 1, 21, 33, 40, 41, 43, 162]	[0, 1, 11, 20, 21, 22, 23, 24, 27, 28, 29, 30, 31, 32, 33, 34, 36, 40, 41, 43, 44, 48, 53, 63, 94, 95, 117, 160, 161, 162, 164, 166]

Table 5.4: Clustering Results (k=10) from ACJ and ACC, shows the cluster number and the members belonging to it, where the underlined members differ between the two models.

Chapter 6

Discussion

A discussion of the results will be presented in this chapter, which includes comparing the results from different models and analysing the reasons behind the different performances. Start with analysing the results presented by the ranking models, followed by the results generated from the clustering methods ACJ and ACC.

6.1 Ranking Model Results Analysis

This section presents an analysis of the ranking results and the performance of the models. The initial step involves comparing the outcomes obtained from both models. Subsequently, the analysis of the dataset view is offered, taking into account the conclusion drawn from the previous comparison.

6.1.1 Results Comparison

Figure 6.1 displays the attributes statistic from the ranked list generated by the AHP-WSM model, whereas Figure 6.2 presents the attributes statistic obtained from the LmabdaMART model. The differences appear in the second and twelfth positions, where the values of the attributes are presented in Table 6.1.

Among the two items, tc_513421 has a higher value of the RFP, with a value of 100%, as opposed to tc_115003's 94.545%. As previously said, the RFP holds significant importance among the five attributes. Therefore, in the current case, the outcome produced from the AHP-WSM is accurate. Subsequently, the disagreement in the twelfth position is likewise addressed, as it relates to the comparison between tc_115294 and tc_513421, with the latter one considered suitable for the second position.

Index	Test case ID	WP	EP	AP	RFP
2	*tc*_513421	0	40.000	60.000	100.000
	*tc*_115003	0	100.000	0.000	94.545
12	*tc*_115294	0	100.000	0.000	93.897
	*tc*_513421	0	40.000	60.000	100.000

Table 6.1: The comparison of values of the second place and twelfth place in both lists.

In summary, the outcome obtained by the AHP-WSM model shows a higher level of accuracy.

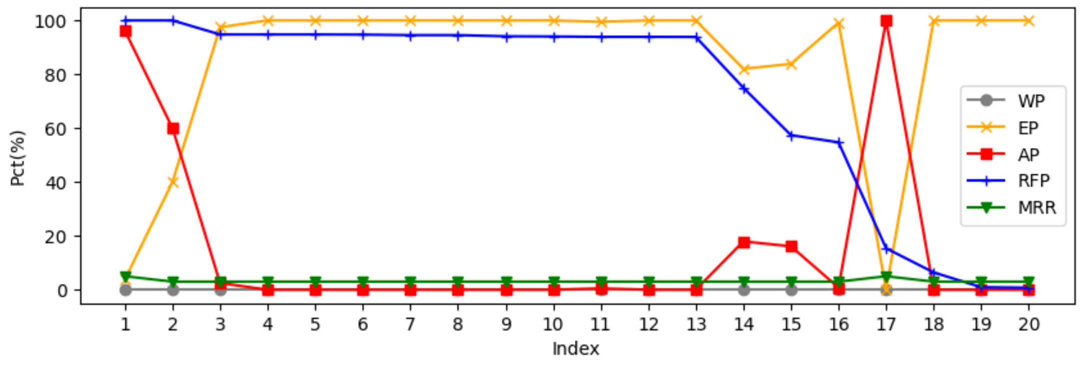

Figure 6.1: The attributes' values from the ranked results.

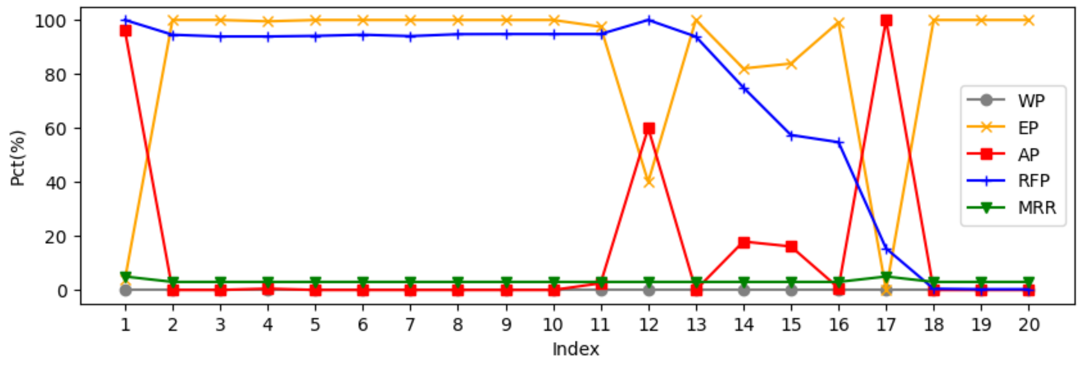

Figure 6.2: The attributes' values from the ranked results..

6.1.2 Analysis

As shown in the results from the previous section, it is evident that the conventional ranking model, employing AHP-WSM, outperforms the LambdaMART model. A detailed examination of the dataset is critical to explain the underperformance of the LambdaMART model. For that, two critical factors come to the forefront: data size and data quality.

Dataset Size

As described in data preprocessing, the dataset used in this study contains only 233 test cases after preprocessing. Furthermore, the training set covers simply 163 test cases. This limitation implies that the model might not have access to a sufficient amount of data to promote effective learning during the training phase. Moreover, as depicted in Figure 6.3, features contributing to the model's ability to learn the importance of each feature are non-zero values. However, these values represent an exceedingly minor fraction of the dataset, all falling below the 20% threshold.

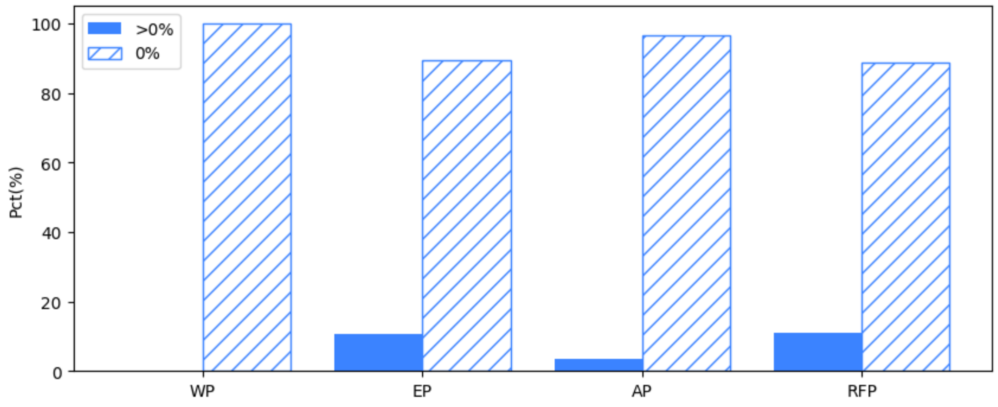

Figure 6.3: The zero and non-zero values are contained in each feature.

Data Quality

Regarding data quality, according to [29], one dimension that can affect the performance of a machine-learning model is class balance. A class represent the items containing the same label in the dataset used in this study. An imbalanced dataset can cause the algorithm to fail at identifying the structure or miss the smaller classes. The class balance can be measured by the following equations:

$$ImBalance(d) = \frac{1}{2} \times \sum_{i,j \in 1,\ldots,m} |n_{cl_i} - n_{cl_j}| \tag{6.1}$$

$$Balance(d) = 1 - \frac{ImBalance(d)}{\lceil \frac{m}{2} \rceil \times \lfloor \frac{m}{2} \rfloor \times n_{cmax}} \tag{6.2}$$

In these equations, the variables $|n_{cl_i} - n_{cl_j}|$ represent the pairwise difference between two classes: cl_i and cl_j within a set containing m classes. Meanwhile, n_{cl_i} and n_{cl_j} denote the number of items within their respective classes, and n_{cmax} signifies the maximum number of items a class contained.

Label	0	1	2	3	4
Number	207	9	1	3	13

Table 6.2: The number of items in each class(label group).

According to the data presented in Table 6.2, the calculated $Balance(d)$ for the dataset used in this study can be computed as the following calculation, which indicates the dataset is highly imbalanced.

$$\begin{aligned} ImBalance(d) =& \frac{1}{2}(|cl_0 - cl_1| + |cl_0 - cl_2| + |cl_0 - cl_3| + |cl_0 - cl_4| + |cl_1 - cl_2| \\ &+ |cl_1 - cl_3| + |cl_1 - cl_4| + |cl_2 - cl_3| + |cl_2 - cl_4| + |cl_3 - cl_4|) \\ &= 422 \end{aligned}$$

$$\begin{aligned} Balance(d) &= 1 - \frac{422}{\lceil \frac{5}{2} \rceil \times \lfloor \frac{5}{2} \rfloor \times 207} \\ &= 0.59 \end{aligned}$$

In general, the LambdaMART model's performance was unsatisfactory due to limited data supply and imbalanced class distributions. However, the conditions mentioned above are based on the test runs conducted in ABB, which are carefully planned and scheduled. Therefore, to implement the feature of ranking the frequently failed test cases, the AHP-WSM has been used.

6.2 Clustering Model Results Analysis

The comparison of results and the corresponding analysis of ACJ and ACC are presented in this section. The result is visualised in section 5.2, and a specific example is given to give more insight into the different characteristics of the two methods.

6.2.1 Result Comparison

The following figures show the visualized results from Section 5.2. Figures 6.4 and 6.5 depict the outcomes obtained from the ACJ and ACC model when the number of clusters is set to 5. Conversely, Figures 6.6 and 6.7 illustrate the results obtained when the number of clusters equals 10.

The visualized results provided insight into ACJ and ACC performance in text clustering: whether a data point is assigned in a suitable cluster or not.

The clusters produced from ACJ contain overlapped clusters, where data points are closely positioned but assigned to different clusters. This phenomenon especially appears when the number of clusters increases, indicating a limitation in the ability of Jaccard similarity to capture the relationships contained in the text data.

Conversely, the clusters obtained with cosine similarity demonstrate clear and well-defined edges and the visualizations reveal a more accurate grouping of similar documents. Even as the number of clusters varies, the distinctiveness of the clusters remains clear.

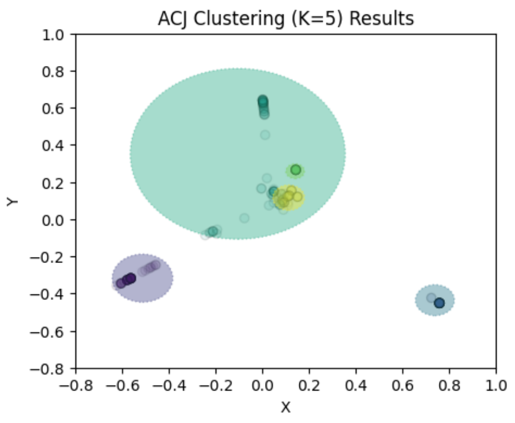

Figure 6.4: ACJ Clustering Results (k=5), where x-axis and y-axis represents similarities.

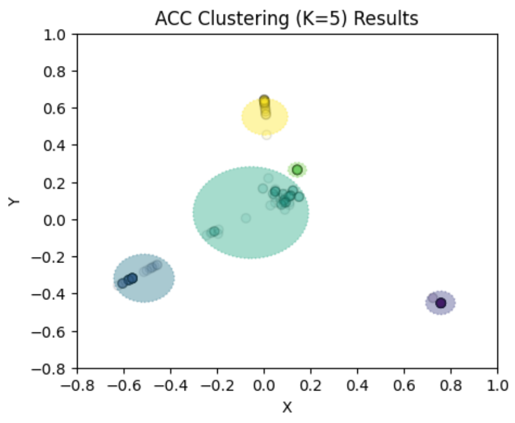

Figure 6.5: ACC Clustering Results (k=5), where x-axis and y-axis represents similarities.

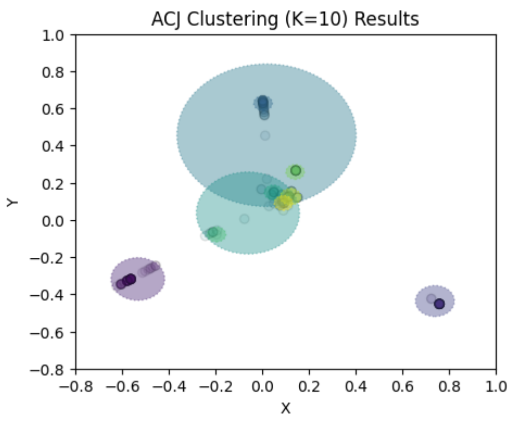

Figure 6.6: ACJ Clustering Results (k=10), where x-axis and y-axis represents similarities.

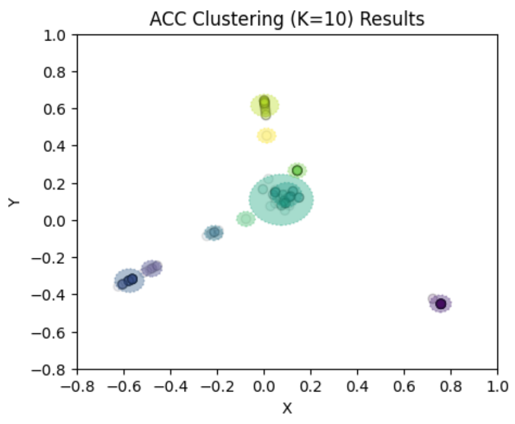

Figure 6.7: ACC Clustering Results (k=10), where x-axis and y-axis represents similarities.

6.2.2 Analysis

Given that the performance of ACJ and ACC depends on the choice of the similarity matrix, this investigation will primarily examine the utilisation of Jaccard similarity and cosine similarity.

The cluster that was discovered exhibits overlap in terms of Jaccard similarity. This can be explained by its set-based calculation, where the metric primarily considers the existence or absence of terms rather than the words' frequency. This sensitivity to term occurrences may lead to imprecise clustering outcomes, especially when texts share similar content but differ in specific terms or frequencies.

The clarity and accuracy of the cluster boundaries obtained with cosine similarity align with its ability to consider the angle between vectors. This property makes cosine similarity more robust to variations in text lengths. It enables it to capture the semantic relationships between texts based on the overall content rather than relying on exact term matches.

A specific example is provided, along with two error messages below, labelled as **10** and **113** in the dataset.

$$m_{10} = \textit{"idi1/value: 1 is not Equal to the expected value: 0"}$$

$$m_{113} = \textit{"0 is not Equal to the expected value: 3"}$$

Before calculating the similarities, the messages need to be converted into two sets of words and the vocabulary for them.

$$Set_{10} = \textit{\{idi1/value:, 1, is, not, Equal, to, the, expected, value:, 0\}}$$

$$Set_{113} = \textit{\{0, is, not, Equal, to, the, expected, value:, 3\}}$$

$$Vocabulary = \{id1/value :, 1, is, not, Equal, to, the, expected, value :, 0, 3\}$$

The Jaccard and Cosine similarity between them can be calculated by following the algorithm from Section 4.2.1, as shown below:

$$\begin{aligned} J(m_{10}, m_{113}) &= \frac{|\{is, not, Equal, to, the, expected, value :\}|}{|\{id1/value :, 1, is, not, Equal, to, the, expected, value :, 0, 3\}|} \\ &= \frac{7}{11} \\ &\approx 0.636 \end{aligned}$$

$$\begin{aligned} C(m_{10}, m_{113}) &= \frac{<1,1,1,1,1,1,1,1,1,1> \cdot <0,0,1,1,1,1,1,1,1,1>}{\sqrt{10} \times \sqrt{9}} \\ &= \frac{8}{\sqrt{90}} \\ &\approx 0.843 \end{aligned}$$

Based on this, as demonstrated in the ACC result, two variables m_{10} and m_{113} are assigned to a single cluster, cluster 6, as shown in Figure 5.4, which is recommended. However, from the results generated by the ACJ, m_{10} is assigned to cluster 6, but m_{113} is in cluster 5.

Moreover, based on the evaluation results, namely the silhouette scores obtained in Section 4.3, it can be shown that the scores derived from ACC are significantly higher than those obtained from ACJ, denoting that ACC produced more accurate results than ACJ.

Chapter 7

Conclusions

In conclusion, this study has analysed and developed an optimal solution based on the selected approaches for ranking the frequently failing test cases and clustering test cases based on error messages. The examination of ranking approaches employing the AHP-WSM model and LambdaMART highlights the importance of considering dataset features. The higher performance of the AHP-WSM model can be attributed to its efficient and straightforward calculation method, which is particularly advantageous when dealing with small datasets. Similarly, comparing agglomerative clustering with Jaccard and cosine similarity metrics reveals the latter's higher effectiveness in grouping texts containing high-frequency terms like error messages.

Following are the answers to the research questions provided in Chapter 1:

RQ1: What data preprocessing techniques are most suitable for preparing the test data for ranking and clustering algorithms?
Regarding data preprocessing, for the AHP-WSM model and both clustering models, the suitable preprocessing can be straightforward, which relates to removing redundant information and transforming the data structure provided in Sections 3.1 and 4.1. However, LambdaMART needs an additional process: dataset splitting for the training and test phases. The method used in this study is Stratified Sampling as presented in Section 3.3.1.

RQ2: By comparing the conventional ranking and learning-to-rank models, which performed better and why?
In this study, the conventional ranking model, which is the AHP-WSM model, performed better. The reason is that the dataset used in this study contains unusual characteristics unsuitable for machine learning models, especially in learning-to-rank. The detailed analysis is presented in Section 6.1.2.

RQ3: How do the two machine learning clustering models compare their performance and results when applied to error message text clustering?
The evaluation matrix named silhouette score and the visualisation of results are used to compare the clustering methods' performance, which is presented in Section 4.3 and Section 6.2.2. The silhouette scores provided an overall performance from both models, but the visualised results gave more details to analyse when selecting an optimal solution.

RQ4: Based on the analysis of the results, which model is the most optimal for clustering the test cases?
The two methods used in this study are ACJ and ACC. Based on the analysis in Section 6.2.2, the ACC performed better than the ACJ, showing that the error messages' high-frequency terms need to be considered when calculating the similarity between two texts.

7.1 Future Directions

The future direction for this study is mainly focused on the ranking task, with a specific emphasis on addressing the limitations associated with the LambdaMART model. Among the approaches in this study, the AHP-WSM model is chosen because the limitations of the current dataset make it unable to be used in the LambdaMART model. It is possible to solve them in continuous development.

The first limitation arises from the size of the current dataset. LambdaMART's two-stage process requires a sufficiently large dataset for training and testing. Strategies to overcome this limitation include a focus on increasing the dataset size through the continuous accumulation of test runs at ABB. Additionally, applying data augmentation techniques, such as introducing noise or transformations to existing data, can enhance the dataset's richness and utility.

The second limitation is data quality, a critical factor influencing machine learning model performance. The current dataset exhibits a highly imbalanced distribution of items across label groups. This imbalance can be addressed by implementing re-sampling techniques, which involve duplicating or generating synthetic examples to increase the number of items in the minority group.

As the study progresses into future phases, implementing these strategies promises to expand the analytical capabilities to encompass LambdaMART. As the dataset grows, employing a machine-learning model instead of a conventional one is essential.

Appendix A

Full results from ACJ and ACC

cluster No.	**Included data points**	
	ACJ	**ACC**
cluster 1	[167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227]	[167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227]
cluster 2	[122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 161]	[122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158]
cluster 3	[3 4 5 6 9 11 13 14 20 27 28 29 30 31 32 36 37 38 44 45 46 47 48 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 113 114 115 116 117 118 159 160 163 164 165 166]	[54 55 56 57 58 59 60 61 62 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93]
cluster 4	[7 17 22 42 50 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112]	[7 17 42 50 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112]
cluster 5	[0 1 2 8 10 12 15 16 18 19 21 23 24 25 26 33 34 35 39 40 41 43 49 51 52 94 95 119 120 121 162]	[0 1 2 3 4 5 6 8 9 10 11 12 13 14 15 16 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 43 44 45 46 47 48 49 51 52 53 63 94 95 113 114 115 116 117 118 119 120 121 159 160 161 162 163 164 165 166]

Table A.1: Clustering Results (k=5) - Cluster Number(No.) and Included Data Points

cluster No.	**Included data points**	
	ACJ	**ACC**
cluster 1	[167 168 169 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227]	[167 168 169 170 171 172 173 174 175 176 177 178 179 180 182 183 184 191 192 193 194 195 196 197 198 199 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227]
cluster 2	[122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 161]	[122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158]
cluster 3	[64 67 68 69 76 79 80 81 83 86 87 88 90 91 92 93]	[54 55 56 57 58 59 60 61 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93]
cluster 4	[20 27 28 36 44 48 54 55 56 57 58 59 60 61 62 63 65 66 70 71 72 73 74 75 77 78 82 84 85 89 117 160 164]	[62]
cluster 5	[6 11 29 30 31 32 38 45 46 47 53 113 114 115 116 118 159 165 166]	[159]
cluster 6	[3 4 5 9 13 14 37]	[2 3 4 5 6 8 9 10 12 13 14 15 16 18 19 25 26 35 37 38 39 49 51 52 113 119 120 121]
cluster 7	[163]	[45 46 47 114 115 116 118 163 165]
cluster 8	[7 17 22 42 50 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112]	[7 17 42 50 96 97 98 99 100 101 102 103 104 105 106 107 108 109 110 111 112]
cluster 9	[2 8 10 12 15 16 18 19 23 24 25 26 34 35 39 49 51 52 94 95 119 120 121]	[181 185 186 187 188 189 190 200]
cluster 10	[0 1 21 33 40 41 43 162]	[0 1 11 20 21 22 23 24 27 28 29 30 31 32 33 34 36 40 41 43 44 48 53 63 94 95 117 160 161 162 164 166]

Table A.2: Clustering Results (k=10) - Cluster Number(No.) and Included Data Points

Appendix B

Instructions to Compile and Run System

The source code for this project can be accessed by requirement.

Bibliography

[1] Evangelos Triantaphyllou. *Multi-criteria decision making methods: A comparative study*. Springer, 2011.

[2] Fatma Eltarabishi and Omar Hasan Omar. Multi-criteria decision making methods and their applications– a literature review. 2020. URL https://api.semanticscholar.org/CorpusID:226962104.

[3] Peter C Fishburn. Additive utilities with incomplete product sets: Application to priorities and assignments. *Operations Research*, 15(3):537–542, 1967.

[4] Thomas L Saaty. How to make a decision: the analytic hierarchy process. *European journal of operational research*, 48(1):9–26, 1990.

[5] Zouhour Chourabi, Faouzi Khedher, Amel Babay, and Morched Cheikhrouhou. Multi-criteria decision making in workforce choice using ahp, wsm and wpm. *The Journal of The Textile Institute*, 110(7):1092–1101, 2019.

[6] Thomas L Saaty. Decision making with the analytic hierarchy process. *International journal of services sciences*, 1(1):83–98, 2008.

[7] Hamed Taherdoost. Decision making using the analytic hierarchy process (ahp); a step by step approach. *International Journal of Economics and Management Systems*, 2, 2017.

[8] Roseanna W Saaty. The analytic hierarchy process—what it is and how it is used. *Mathematical modelling*, 9(3-5):161–176, 1987.

[9] Sukanta Malakar and Abhishek K Rai. Estimating seismic vulnerability in west bengal by ahp-wsm and ahp-vikor. *Natural Hazards Research*, 2023.

[10] Thomas L Saaty. *Theory and applications of the analytic network process: decision making with benefits, opportunities, costs, and risks*. RWS publications, 2005.

[11] Christopher JC Burges. From ranknet to lambdarank to lambdamart: An overview. *Learning*, 11(23-581):81, 2010.

[12] Krysta M Svore and CJ Burges. Large-scale learning to rank using boosted decision trees. *Scaling Up Machine Learning: Parallel and Distributed Approaches*, 2:2011, 2011.

[13] Lior Rokach and Oded Maimon. Clustering methods. *Data mining and knowledge discovery handbook*, pages 321–352, 2005.

[14] Rui Xu and Donald Wunsch. Survey of clustering algorithms. *IEEE Transactions on neural networks*, 16(3):645–678, 2005.

[15] Stephen A Collins-Elliott. Agglomerative clustering using cosine and jaccard distances: A computational approach to roman vessel taxonomy. *Archeologia e Calcolatori*, (27), 2016.

[16] Microsoft. Azure cosmos db – unified ai database, . URL https://learn.microsoft.com/en-us/azure/cosmos-db/introduction.

[17] Microsoft. Introduction to blob (object) storage - azure storage, . URL https://learn.microsoft.com/en-us/azure/storage/blobs/storage-blobs-introduction.

[18] Nunung Nurul Qomariyah, Dimitar Kazakov, and Ahmad Nurul Fajar. Predicting user preferences with xgboost learning to rank method. In *2020 3rd International Seminar on Research of Information Technology and Intelligent Systems (ISRITI)*, pages 123–128. IEEE, 2020.

[19] Hassan I Abdalla. A brief comparison of k-means and agglomerative hierarchical clustering algorithms on small datasets. In *International Conference on Wireless Communications, Networking and Applications*, pages 623–632. Springer, 2021.

[20] Lisna Zahrotun. Comparison jaccard similarity, cosine similarity and combined both of the data clustering with shared nearest neighbor method. *Computer Engineering and Applications Journal*, 5(1):11, 2016.

[21] Van L Parsons. Stratified sampling. *Wiley StatsRef: Statistics Reference Online*, pages 1–11, 2014.

[22] Siegfried Gabler. Sampling designs in surveys. *Wiley StatsRef: Statistics Reference Online*, 2014.

[23] Tao Qin and Tie-Yan Liu. Introducing LETOR 4.0 datasets. *CoRR*, abs/1306.2597, 2013. URL http://arxiv.org/abs/1306.2597.

[24] Tianqi Chen and Carlos Guestrin. XGBoost. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*.

ACM, aug 2016. doi: 10.1145/2939672.2939785. URL https://doi.org/10.1145%2F2939672.2939785.

[25] Anna Huang et al. Similarity measures for text document clustering. In *Proceedings of the sixth new zealand computer science research student conference (NZCSRSC2008), Christchurch, New Zealand*, volume 4, pages 9–56, 2008.

[26] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[27] Peter J Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. *Journal of computational and applied mathematics*, 20:53–65, 1987.

[28] Ketan Rajshekhar Shahapure and Charles Nicholas. Cluster quality analysis using silhouette score. In *2020 IEEE 7th international conference on data science and advanced analytics (DSAA)*, pages 747–748. IEEE, 2020.

[29] Lukas Budach, Moritz Feuerpfeil, Nina Ihde, Andrea Nathansen, Nele Noack, Hendrik Patzlaff, Felix Naumann, and Hazar Harmouch. The effects of data quality on machine learning performance. *arXiv preprint arXiv:2207.14529*, 2022.