Kingdom of Saudi Arabia
Ministry of Education
Princess Nourah bint Abdulrahman University
College of Computer and Information Sciences
Graduation Project 2

المملكة العربية السعودية
وزارة التعليم
جـــــامعة الأميرة نورة بنت عبد الرحمـــن
كلية علوم الحـــــاسب والمعلومات
مشروع تخرج 2

SENTIMENT ANALYSIS OF PUBLIC PERCEPTIONS REGARDING ATTRACTING FOOTBALL SUPERSTAR PLAYERS IN THE SAUDI LEAGUE

By

Name	ID
Danyah Alobaid	442005141
Deema Alotaibi	442000406
Remas Alsaeed	442002447
Ghadah Alsalamah	442004739
Nafla Alotaibi	442004474

Supervised by

Dr. Hessa Alfraihi

Graduation Project Report Submitted to

College of Computer Sciences and Information at PNU

In Partial Fulfillment of the Requirements for the Degree of

Bachelor of Science in Computer Sciences

PNU CCIS

Riyadh, KSA

1445

Table of Contents

List of Tables

List of Figures

List of Symbols and Abbreviations

PNU	Princess Nourah bint Abdul Rahman University
NLP	Natural Language Processing
ML	Machine Learning
DL	Deep Learning
SM	Sentiment Analysis
CB	Corpus-Based
DB	Dictionary-Based
RN	Reinforcement Learning
DT	Decision Tree classifier
LC	Linear Classifier
SVM	Support Vector Machine
LR	Logistic Regression
K-mean	K-mean clustering
NN	Neural Network
RB	Rule-Based classifier
PC	Probabilistic classifier
NB	Naive Bayes
ME	Maximum Entropy
BN	Bayesian Network
ABSA	Aspect-Based Sentiment Analysis
RF	Random Forest
KNN	K nearest neighbors
MNB	Multinominal Naïve Bayes
TL	Transfer Learning
AI	Artificial Intelligence
API	Application Programming Interface
CSV	Comma-Separated Values
TF-IDF	Term Frequency-Inverse Document Frequency
DFD	Data Flow Diagram
ERD	Entity Relationship Diagram
LGBM	Light gradient boosting machine
XGBoost	Extreme Gradient Boosting
RNN	Recurrent Neural Network
BERT	Bidirectional Encoder Representations from Transformers
RoBERTa	Robustly optimized BERT approach

DistilBERT	Distilled BERT
DNN	Deep Neural Network
RvNN	Recursive Neural Network
CNN	Convolutional Neural Network
LSTM	Long short-term memory
NSP	Next Sentence Prediction
MLM	Masked Language Modeling
MLP	Multi-Layer Perceptrons
TP	True positive
TN	True negative
FP	False positive
FN	False negative
ROC	Receiver Operating Characteristic curve
AUC	Area Under the Curve

Table 0–1 List of Symbols and Abbreviations

Acknowledgment

We would like to thank Princess Nourah Bint Abdul Rahman University for providing us with invaluable resources and for its supportive environment. We would like to thank the faculty of The College of Computer Science and Information since they taught us the necessary information and techniques to complete this project from the very beginning. Lastly, we would like to express our sincere gratitude to our supervisor, Dr. Hessa Alfraihi, who guided us through this project, for her encouragement, support, and understanding.

Abstract

Social media is crucial for community engagement, enabling discussions and event sharing. This research investigates public perception regarding the attraction of level-A football players to the Saudi league, leveraging English datasets from the X platform (Twitter). Following data cleaning, the experiment comprises 118,298 samples. Employing a variety of methodologies including deep learning techniques like Bidirectional Encoder Representations from Transformers (BERT), Recurrent Neural Networks (RNN), and machine learning algorithms such as Extreme Gradient Boosting (XGBoost), Support Vector Machine (SVM), and Light Gradient-Boosting Machine (LGBM), the data was classified into positive, negative, and neutral sentiments. Visualization through a dashboard was essential to provide comprehensive insights, facilitating intuitive interpretation of the data patterns, trends, and sentiment distributions, thus aiding decision-making processes and enhancing the comprehension of this analysis.

The results showed that all algorithms achieved high accuracy but it was dominated by Bert where it achieved the highest among all with 97.27% following RNN with 95.60%. XGBoost achieved 90.26%, followed by SVM at 88.90% and LGBM at 88.81%.

Keywords: sentiment analysis, social media, deep learning, machine learning, vision 2030, Saudi league, football superstar players, sport, tourism.

Chapter 1: Introduction

1. Introduction

Football, or the "beautiful game," it transcends beyond being a mere sport. It has the ability to unite people, surpass cultural barriers, and instill a sense of national pride. This is especially true in Saudi Arabia, where football is not just a beloved hobby, but a way of life. Football in Saudi Arabia is a powerful force that is transforming the nation's social and economic landscape. As the country progresses towards its Vision 2030 initiative, football embodies the spirit of transformation, uniting people and promoting national pride. Through investment in sports infrastructure, football has become a symbol of the kingdom's rich culture and a catalyst for social engagement and economic growth.

The recent inclusion of three international football stars - Cristiano Ronaldo [1], Neymar Junior [2], and Karim Benzema [3] in Saudi Arabian teams Al Nassr, Al Hilal, and Al Ittihad has generated excitement among local supporters and garnered global attention. These players not only bring their exceptional skills to the field, but also act as ambassadors for the sport, helping to raise its profile and promote its positive impact on society.

With the incorporation of international football stars and continued investment in sports infrastructure, the future of football in Saudi Arabia looks bright, and with it, the future of the nation as a whole.

1.1 Problem Statement and Significance

As the Saudi Professional League's (SPL) recent surge in sporting prominence, a gap exists between its ambitions and the public perception surrounding its ability to attract "Level-A" players. This disparity manifests prominently on social media, where the buzz and excitement surrounding potential superstar signings in the SPL pales compared to other major leagues [4]. This lack of public perception, encompassing both domestic and global football enthusiasts, creates a significant obstacle to the league's aspirations for sports development, global recognition, economic aspects, fan engagement, and perception and image. The global football landscape is marked by intense competition among leagues vying to secure top talent, and the Saudi League's underappreciated status hampers its ability to become a prominent player in this arena. By understanding and addressing the factors contributing to the undervaluation of the league, strategic improvements can be made that will elevate the league's standing and potentially reshape the dynamics of football talent distribution globally [5].

1.2 Proposed Solution

Suggested solution: developing a dashboard that analyzes English posts on X platform (Twitter) of people’s opinions about international football stars coming to Saudi teams using Machine Learning (ML) and Deep Learning (DL) algorithms would be the most appropriate solution for the specified problem, which includes accomplishing the following objectives:

- Utilize ML and DL algorithms to classify the sentiment of collected data into positive, negative, and neutral categories.
- Gain an understanding of how people feel towards attrackting football superstar players to Saudi league.
- Analyze the sentiment across different countries and dates to address the variation of perceptions.
- Present the sentiment analysis results in a visualized dashboard.

1.3 Project Domain and Limitation

The project's main objective is to implement Machine Learning models for sentiment analysis on the X platform, focusing on analyzing the sentiment of global English posts related to the significant Saudi deals in the football field. The project concentrated on the level A football players, including Cristiano Ronaldo who's well known for his standout performances at Sporting Lisbon, Manchester United, and Real Madrid, winning numerous awards [6], Neymar at Santos FC and FC Barcelona, showcasing his dazzling skills and winning countless domestic and international title [7]. And lastly, Karim Benzema became well-known for his goal-scoring prowess at Lyon before joining Real Madrid, where he played a pivotal role in winning multiple UEFA Champions League titles. Their consistent success and contributions to top clubs made them globally recognized football stars [8]. Insights into the football transfer market in Saudi Arabia will be gained for the project through using deep learning, machine learning, and data collection. Another objective of the project is to measure the impact of Saudi deals on improving tourism, which aligns with the vision of 2030, "Investing in sport to enhance tourism" [9].

The project will implement the sentiment analysis process by utilizing advanced ML algorithms such as SVM, LGBM, and XGBoost, as well as cutting-edge DL algorithms like RNN and BERT.

As work progresses on the project, it's essential to remember that there are additional limitations to consider. The main focus will be on football and a finite number of players, and only English posts will be analyzed and collected.

The project is limited to one social media platform, X. This can pose challenges as X has no posting limits, leading to false information and errors. Posts may also be irrelevant to users and contain slang, abbreviations, and acronyms, affecting sentiment analysis accuracy. Sentiment analysis may be biased due to limited data and difficulties interpreting sarcasm and nuanced expressions. Short posts can also make it hard to understand the context accurately. Obtaining user consent and anonymizing data are crucial, and project design must address any limitations [10] [11].

1.4 Methodology

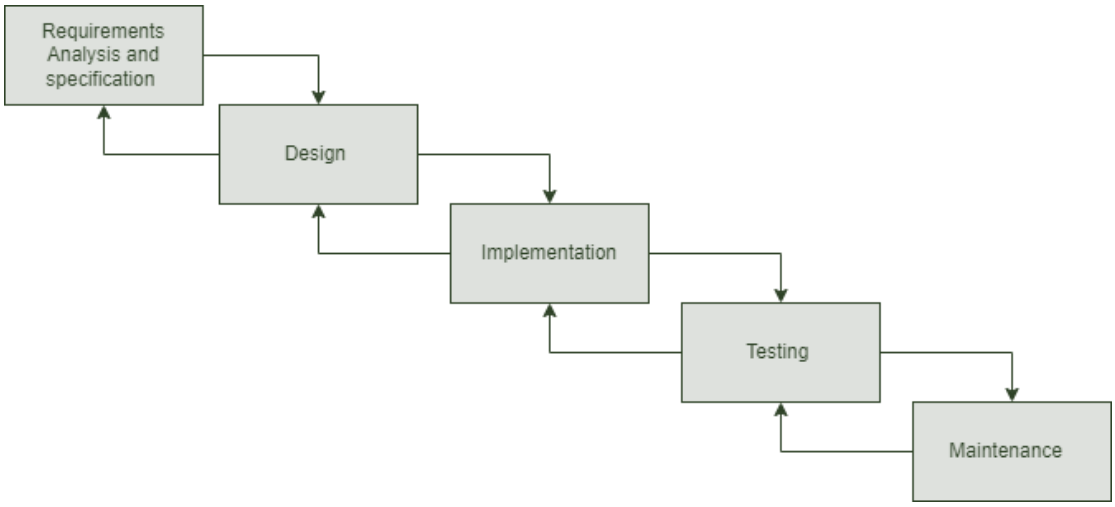


Figure 1–1 Project Development life cycle Iterative Waterfall

This project (as shown in Figure 1) will be developed using the iterative waterfall approach. It combines the principles of the sequential steps of the traditional Waterfall Model with iterative and flexible practices. In this approach, the project is divided into phases, like the Waterfall Model, but each phase includes iterative cycles. These iterations allow for feedback, adjustments, and the ability to revisit and revise previous stages of the project [12].

Some advantages of the iterative waterfall methodology:

Improve flexibility: the ability to iterate allows for adjustments based on feedback.

Provide quality assurance: the ability to test and provide feedback throughout the process.

Risk management: identifying and addressing issues early in the process [12].

Some disadvantages of the iterative waterfall methodology:

Limited user involvement.

The flexibility and continuous changes may lead to scope creep [12].

Why the iterative waterfall methodology?

This approach is ideal for the project because it allows adjustments and course corrections to be made by receiving feedback from the supervisor, resulting in a final product that meets the needs and requirements.

Chapter 2: Background Information and Related Work

2 Background Information and Related Work

This chapter aims to shed some light on the foundational concepts essential for understanding the subject matter of this dissertation. Starting with exploring the main scenes of football in Saudi Arabia, providing an overview of its history, influence, and current state in the country. The chapter then delves into the field of sentiment analysis, examining its methodologies, applications, and relevance to football discourse. Lastly, it presents a review of the most recent and related work in this area, aiming to contextualize our study within the larger academic landscape.

2.1 Background Information

This section will highlight the importance and the history of football in Saudi Arabia, and it will give a description of the sentiment analysis in details, including its methodologies and applications.

2.1.1 The importance of football in Saudi Arabia

The football scene in Saudi Arabia is long and deeply ingrained in the nation's sports culture. Football in Saudi Arabia is more than just a sport; it's a source of national unity and pride. The Saudi Arabian national football team, known as "Green Falcons," represents the country's unity. Its performance at international competitions, like the FIFA World Cup, is closely followed and celebrated by millions of passionate fans. Furthermore, football has a substantial economic impact, attracting sponsorship deals, merchandise sales, and tourism, contributing significantly to Saudi Arabia's sports industry and overall economy. Saudi football clubs like Al Nassr, Al Hilal, and Al Ittihad have attracted immense followings, especially when superstar players joined the Saudi league, Cristiano Ronaldo [1], Neymar Junior [2], and Karim Benzema [3]. Saudi Arabia's Vision 2030 is a transformative socio-economic blueprint that extends its influence on the football scene, taking advantage of its immense popularity. As part of this vision, there is a notable focus on developing and enhancing football infrastructure. Plans include constructing modern football facilities, establishing youth academies, and supporting local football clubs to achieve success locally and internationally. Additionally, Vision 2030 emphasizes the importance of attracting major international football events and tournaments to Saudi

Arabia, fostering football culture, promoting tourism, and enhancing the nation's global image as a sports hub in the Middle East [13].

2.1.2 Sentiment Analysis

The process of evaluating the thoughts and perceptions of individuals concerning various topics, products, subjects, or services is commonly known as sentiment analysis. Identifying and extracting subjective information from textual data involves using Natural Language Processing (NLP), Machine learning (ML), and data analytics techniques. This process has gained widespread acceptance in recent years, with businesses, governments, organizations, and researchers all utilizing it [14].

The applications of sentiment analysis are numerous and varied, such as assessing patient mental health status based on social media posts and analyzing customer opinions. Sentiment analysis has become an increasingly valuable tool in various fields due to the expansion of applications through technological advancements like Cloud Computing, Big Data, the Internet of Things (IoT), and Blockchain. These advancements have widened the scope of sentiment analysis, making it a vital tool for almost any purpose [14]. Figure 2-1 depicts the general process of sentiment analysis.

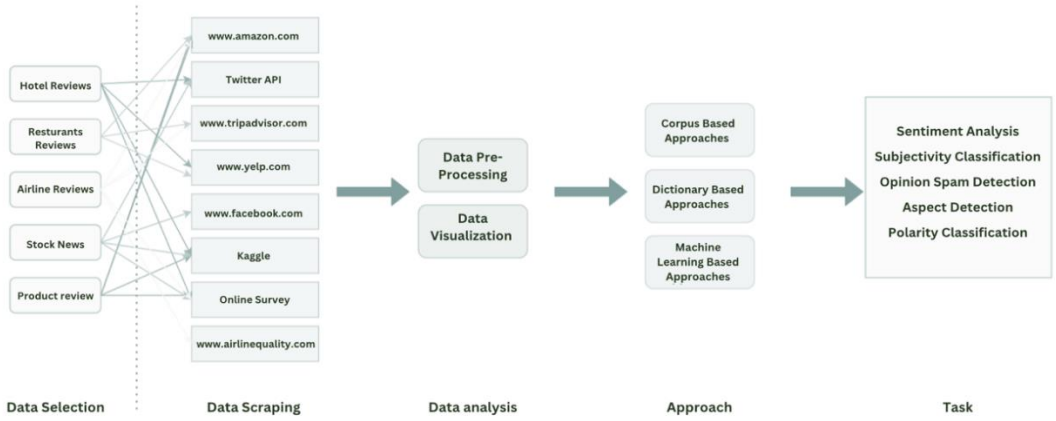


Figure 2–1 General procedure of Sentiment Analysis adapted from [14].

Sentiment analysis operates at various levels of granularity, each serving distinct purposes. At document-level gives the entire document a single polarity but is utilized rarely. However, sentence-level analysis examines each sentence and assigns its corresponding polarity, making it highly useful for text with a range of sentiments. Furthermore, phrase-level sentiment analysis is accomplished by extracting and classifying opinion words, making it particularly helpful for product reviews with multiple lines. Finally, aspect-level sentiment analysis is conducted, which assigns polarity to each aspect used in a sentence [14]. Different levels of sentiment analysis are presented in Figure 2-2.

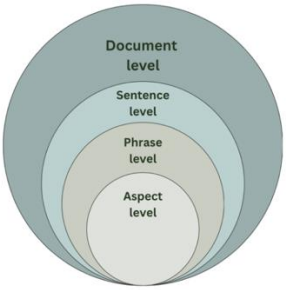


Figure 2–2 The levels of Sentiment Analysis adapted from [14]

Sentiment Analysis divided into Lexicon-based, Machine Learning, and Hybrid—also, other approaches as presented in Figure 2-3 [14].

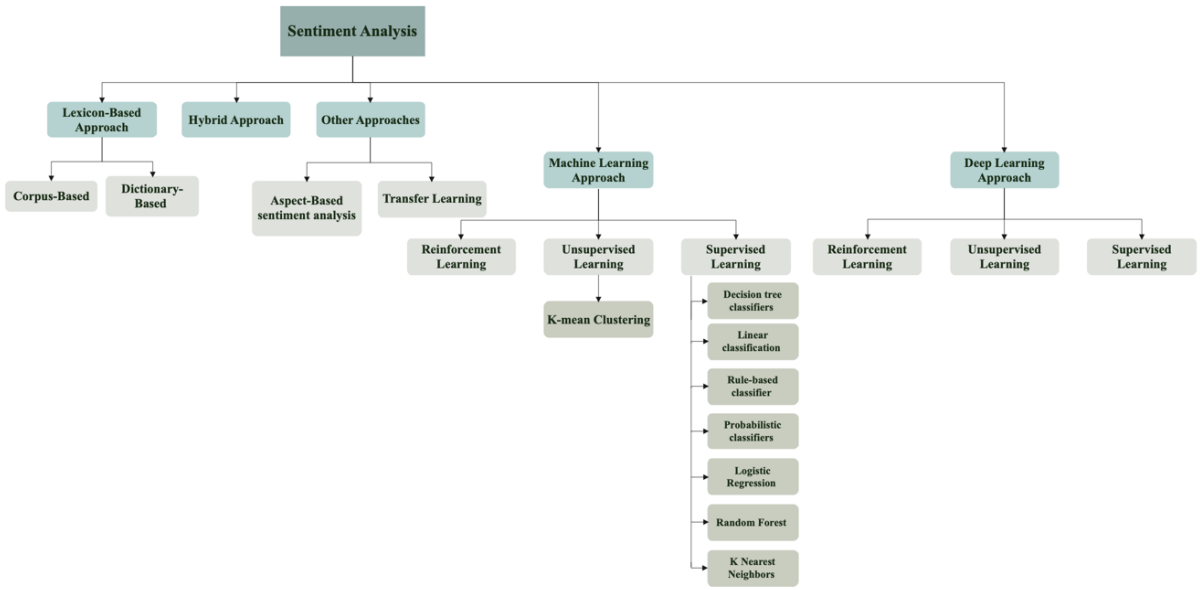


Figure 2–3 Approaches of Sentiment Analysis. adapted from [14] [15] [16]

2.1.2.1 Deep Learning Approach

Deep Learning is an advanced subfield of Machine Learning that brings ML closer to AI. It enables the modeling of complex relationships and concepts through multiple levels of representation. Supervised and unsupervised learning algorithms build progressively higher levels of abstraction, defined using the output features from lower levels [15].

DL is all about automatically learning multiple levels of representations of the underlying distribution of the data to be modeled. In other words, a DL algorithm automatically extracts the low- and high-level features needed for classification. High-level features are hierarchical and depend on different features. Deep Neural Networks (DNN) have achieved great success in Supervised Learning. Additionally, DL models have been immensely successful in unsupervised, hybrid, and reinforcement learning. DL algorithms include Recursive Neural Networks (RvNN), Recurrent Neural Networks (RNN), Convolutional Neural Networks (CNN), and BERT (Bidirectional Encoder Representations from Transformers) [15] [16].

DL has shown superior learning and classification performance. Some examples of its applications and techniques include Speech and Audio Processing, Visual Data Processing, and Natural Language Processing [15] [16].

RNN (Recurrent Neural Network)

RNN is a widely used and popular algorithm in DL, particularly in NLP and speech processing. Unlike traditional neural networks, RNN uses sequential information in the network, which is vital in many applications where the embedded structure in the data sequence holds valuable knowledge. For instance, to comprehend a particular word in a sentence, it is necessary to know the context. Therefore, RNNs can be seen as short-term memory units that include the input layer x, hidden state layer s, and output layer y.

Three RNN approaches are proposed: "Input-to-Hidden," "Hidden-to-Output," and "Hidden-to-Hidden." They create a deep RNN that simplifies learning in deep networks.

RNNs are sensitive to vanishing and exploding gradients during training. This can cause the network to forget initial inputs as new ones enter, reducing efficiency. However, the sensitivity decreases over time.

Long short-term memory (LSTM) network addresses vanishing gradients in RNNs by using memory blocks with memory cells and gated units. Residual connections can also be added to deep networks to alleviate the issue further [17].

Bidirectional Encoder Representations from Transformers (BERT)

BERT revolutionized NLP by introducing a bidirectional understanding of text context. Bert is a pre-trained model developed by researchers at Google in 2018 Based on the Transformer model. It is trained on Wikipedia (~2.5B words) and Google's BooksCorpus (~800M words) [18]. It is deeply bidirectional, from both left-to-right and right-to-left text representation, which solves other models' lack of context. Context-free models currently available typically produce a single-word embedding that applies to all words in the vocabulary, regardless of the context. For instance, this happens with the word "matches," which will have a single embedding in the sentences "Ring perfectly matches with the bracelet" and "Today, both the matches are interesting" [19]. Bert's pre-training involved two unsupervised tasks: The Masked Language Model (MLM) enforces the model to predict masked (hidden) words within a sentence given the context of the surrounding words. A random sentence sample is replaced with a [MASK] token before being fed to the model. The model task is to predict the original identities of these masked tokens based on the context provided by the surrounding words. The key objective is to allow Bert to understand the bidirectional context. Since the model has to predict masked tokens based on both left and right contexts, it learns to understand the relationships between words in both directions within a sentence. Next Sentences Prediction (NSP): Give two input sentences to Bert; for each pair, a unique classification token [CLS] is added to the beginning of the first sentence, and a separator token [SEP] is added between the two sentences. Additionally, a binary label (0 or 1) indicates whether the second sentence continues the first sentence. The model learns to predict whether the second sentence follows the first in the original text. The goal of NSP is to teach Bert to understand the relationships between pairs of sentences [20]. There are currently two ways to apply the pre-trained model to downstream tasks: The feature-based approach utilizes BERT as a feature extractor, extracting contextualized word embeddings from the input text, which are then integrated into traditional machine learning models. These embeddings capture rich semantic and contextual information, enhancing the performance of traditional models

by providing a deeper understanding of the input text's context. By fine-tuning the task at hand, BERT transfers its knowledge learned during pre-training to the new task, leading to state-of-the-art performance. Fine-tuning allows BERT to learn task-specific representations and adapt its internal representations to the nuances of the new task, making it a flexible and efficient approach for various natural language processing tasks such as sentiment analysis, named entity recognition, and question answering [19]. BERT's impact extends beyond its original formulation, with various improved versions tailored to specific requirements. For instance, RoBERTa enhances BERT's performance by training on larger datasets without the NSP task, while ALBERT reduces computational costs by sharing parameters across layers. DistilBERT offers a lightweight alternative by distilling knowledge from a pre-trained BERT model into a smaller one, making it more accessible for resource-constrained environments. Another variant, Electra, introduces a different pre-training objective focused on token replacement detection, enhancing efficiency and performance. These iterations reflect ongoing efforts to optimize BERT's capabilities for diverse applications and computational resources.

2.1.2.2 Machine Learning Approach

Machine Learning is a multi-disciplinary field that enables computers to learn without explicit programming. It allows machines to handle data more efficiently, which can be particularly useful when programmers struggle to interpret the extracted essential information from the data. As the availability of datasets continues to grow, the demand for ML is on the rise [21].

ML can be utilized in multiple computing areas to create and implement practical algorithms that produce high-quality results. Some examples of its application include email spam filtering, fraud detection on social networks, online stock trading, face and shape recognition, medical diagnosis, traffic prediction, character recognition, and product recommendations [21]. The ultimate goal of ML is to learn from data, and many studies have been conducted on how to make machines learn independently without explicit programming [22].

ML is classified into three main types, each with its own distinct characteristics and functions. Supervised Learning is a fundamental ML task characterized by learning a

function that maps input data to output based on a set of labeled examples of input-output pairs. This Supervised approach uses external guidance through labeled training data to infer the underlying function, which is constructed from the training instances [22].

In Supervised Learning, the input data is typically divided into two distinct subsets. The training dataset, which incorporates the output variable to be predicted or classified, and the test dataset, which is employed to evaluate the model's performance on unseen data [22].

Supervised Learning algorithms are classified into different classifications each suited to different problem domains and data characteristics. These classifications encompass Decision Tree Classifiers (DT) including eXtreme Gradient Boosting (XGBoost) Algorithm, Light Gradient Boosting Machine (LGBM), Linear Classifiers (LC) such as Neural Network (NN) Algorithms and Support Vector Machine (SVM) Algorithms, Rule-Based Classifiers (RB), and Probabilistic Classifiers (PC) Encompassing Naive Bayes (NB), Maximum Entropy (ME), and Bayesian Network (BN). Furthermore, Logistic Regression (LR), Random Forest (RF), and K-Nearest Neighbors (K-NN) classifiers [14] [22].

Additionally, Unsupervised Learning allows algorithms to explore and identify interesting structures in data without a designated correct answer or teacher. It learns a few features from the data for recognizing new data and is used for clustering and feature reduction, including the K-mean Clustering (K-mean) algorithm [22].

Lastly, Reinforcement Learning (RL) is considered an intermediate form of learning, as the algorithm receives only feedback indicating whether the output is correct. The algorithm must explore and eliminate various possibilities for the correct output. It is often referred to as learning with a critic, as the algorithm does not offer suggestions or solutions to the problem [21].

Support Vector Machine (SVM)

SVM is a supervised machine learning algorithm that uses a linear classifier to classify data into two or more classes. The SVM algorithm is a popular choice especially for text classification uses due to its ability to handle high-dimensional data and its effectiveness in dealing with complex relationships within the data [23]. The SVM algorithm implementation involves the training and testing separation to use the training set to estimate a function from a set of examples and then testing the performance of the estimated function on a separate testing set. The SVM algorithm covers key concepts such as the maximum-margin hyperplane, support vector regression, and kernel methods. SVM can be extended to handle multi-class classification problems using various strategies. One popular approach is the one-vs-one decomposition method, which decomposes the multi-class problem into multiple binary classification problems. In, a binary classifier is trained for each pair of classes, and the class with the most votes is chosen as the final prediction. Another approach is the one-vs-rest strategy, where a binary classifier is trained for each class against the rest. These strategies allow SVM to handle multi-class classification problems effectively [24].

Xtreme Gradient Boosting (XGBoost)

XGBoost is a machine learning algorithm that employs parallel tree-building techniques and built-in regularization to enhance computational efficiency and control model complexity, leading to improved predictive performance. The algorithm utilizes optimization techniques such as approximate splitting and histogram-based methods for binning continuous features, along with efficient sorting operations, which facilitate parallel processing and accelerate model training. XGBoost's robust capabilities in handling large datasets, weighted and sparse data, and support for out-of-core computing make it particularly suited for sentiment analysis, by enabling accurate identification of sentiment and extraction of valuable insights from textual data. Overfitting is a potential concern in XGboost, while it includes regularization techniques to prevent overfitting, it is still possible for the algorithm to overfit the training data, especially when the model is too complex or when hyperparameters are not properly tuned [25].

Light Gradient-Boosting Machine (LGBM)

LightGBM, developed by Microsoft, emerges as a high-performance framework well-suited for various machine learning tasks, including ranking, classification, and regression, particularly excelling in handling large and intricate datasets. Innovative techniques such as histogram-based algorithms and leaf-wise (vertical) growth facilitate its efficiency in training speed and accuracy. Through the strategic bucketing of values and the adoption of a leaf-wise growth strategy, LightGBM optimizes loss reduction, leading to expedited and more precise model generation. While this approach may increase the risk of overfitting, it can be managed through parameters like max-depth. Moreover, LightGBM's parallel and GPU learning support significantly enhances its scalability and performance. Noteworthy techniques like Gradient-Based one-sided sampling (GOSS) and Exclusive Feature Bundling (EFB) further bolster its speed and accuracy. GOSS prioritizes instances with more significant gradients in gradient-boosted decision trees, while EFB effectively reduces dimensionality without sacrificing accuracy [26]. When applied to sentiment analysis tasks, LightGBM proves advantageous for several reasons. Its rapid training speed facilitates quick iteration and experimentation with various models and features, vital in the dynamic realm of sentiment analysis, where new data and trends continuously emerge. Additionally, its adeptness at handling large and complex datasets enables efficient processing of vast amounts of text data commonly encountered in sentiment analysis, such as social media posts. The algorithm's high accuracy ensures dependability. Moreover, LightGBM's support for parallel and GPU learning expedites sentiment analysis tasks, allowing for real-time or near-real-time analysis across diverse channels and platforms. Furthermore, incorporating techniques like GOSS and EFB enhances LightGBM's ability to capture subtle nuances in sentiment expression, ultimately leading to more nuanced and accurate sentiment analysis results [27].

2.1.2.3 Lexicon-Based Approach

Lexicons contain a set of tokens that are given a predetermined score to indicate whether they are neutral, positive, or negative. The score assigned to each token is determined by polarity-based or intensity-based scoring. This technique is highly effective for sentiment analysis at both the sentence and feature levels and is considered an unsupervised approach because no training data is required. One advantage of this method is that it does not depend on training data. However, it is highly domain-specific, and words from one domain cannot be used in another. For instance, the word “huge” can be positive or negative depending on the domain. Therefore, considering the domain, it is crucial to assign polarity to words carefully [28].

The Lexicon-Based Approach in sentiment analysis is divided into two primary approaches that offer valuable tools for sentiment analysis. In the Dictionary-Based approach, sentiment analysis involves the manual compilation of a set of sentiment words, each accompanied by specific instructions for their usage. In addition, DB focuses on curated lists of words and their associated sentiments. In contrast, the Corpus-Based approach employs pattern analysis to gauge the emotional context of a sentence, achieved through the examination of sentiment tokens and their orientation within a vast corpus of text [28].

2.1.2.4 Hybrid Approach

Utilizing a blend of ML and Lexicon-based techniques has proven a valuable tool in sentiment analysis. This hybrid approach boasts effectiveness and accuracy, as it can distinguish the nuances and complexities of human language. Numerous businesses and scholars favor this method, granting them greater insight into their audience’s perspectives and sentiments. Although hybrid models exhibit superior performance compared to individual models, there is still room for more research to refine their accuracy through fine-tuning and training [29].

2.1.2.5 Other Approach

The field of sentiment analysis has experienced significant growth in recent years, with Aspect-Based sentiment analysis (ABSA) emerging as a precious and rapidly expanding study area. ABSA involves a three-phase process that includes aspect detection, sentiment categorization, and aggregation. The aspect detection phase is crucial, setting the stage for subsequent sentiment calculations. In addition, Transfer Learning (TL) is an advanced Artificial Intelligence (AI) technique that allows pre-trained models to share their knowledge with new models by leveraging similarities in data, distribution, and task. The new model can be trained without explicit training data using previously learned features [30].

2.2 Related Work

This section will discuss the most related works within the sentiment analysis and machine learning field.

Dewi and Arianto, [31] analyzed X (Twitter) users' sentiment regarding Qatar as the host of the 2022 World Cup. It uses TextBlob, an NLP tool, to determine the sentiment of posts(tweets). The study is divided into three stages: before Qatar was selected as the host, after Qatar was selected, and during the 2022 World Cup in Qatar. It found that the sentiment was predominantly positive in all three stages, with an accuracy score of 83%, indicating the model's ability to predict sentiment accurately. It has a lot of strengths; it can predict new data without having to be labeled first. It mentions data visualization through word clouds, which can help readers understand the most frequent positive and negative words associated with Qatar as the host of the World Cup. It collects 244,832 posts for analysis. This large data set provides a substantial amount of data for sentiment analysis, potentially enhancing the reliability and generalizability of the findings. The weakness is that it briefly states web scraping as the method for data collection from the X platform. Still, it does not detail the specific keywords or criteria for data selection. Additionally, it does not mention any potential biases or limitations in the collected data. While the percentages of positive and negative sentiment are mentioned in each stage, it does not provide a thorough analysis or interpretation of these results. There is limited discussion on the implications or significance of the findings.

Alhadlaq and Alnuaim [32] conducted a study using NLP, a mixed methods approach, to analyze the emotions expressed by Arabic and Hispanic viewers during a football match (Argentina vs. Saudi Arabia FIFA 2022 World Cup).

The study successfully compared and investigated the emotional and sentimental traits displayed by both groups of viewers based on their posts. Using the EmoRoberta model, which uses the cutting-edge Roberta approach, researchers analyzed and quantified emotions and sentiments.

The study revealed that Arabic speakers displayed higher levels of emotion and arousal. The study looked at how well machine translation preserves emotional context, but only for posts related to a football match. The study found differences in how dynamic content is translated across languages but did not explore why emotional expression varies across cultures. It offers valuable insights into the emotional dynamics of Arabic and Hispanic X platform users, but more research is needed on cross-cultural emotional and sentimental traits.

Almateg et al. [33] analyzed the public's sentiments towards Saudi women's sports. The study aimed to compare opinions before and after the permission of Saudi women's sports on the X platform. To collect data, the researchers used four hashtags to gather 12,000 Posts related to Saudi women's sports. The Posts were then processed to eliminate irrelevant content and noise. Sentiment analysis was performed using a lexicon-based approach and an SVM classifier. This method effectively captured the overall sentiment of Posts and provided insights into societal attitudes. Social media data provided a platform for individuals to express their opinions freely. However, the method's main limitation was the inability to analyze complex sentiments and sarcasm. Additionally, focusing only on the X platform may only partially represent part of the spectrum of public opinion. The study provided a valuable understanding of evolving attitudes towards Saudi women's sports. Future research could address the limitations of using more diverse data sources and improving sentiment analysis techniques.

Nuno Pombo et al. [34] analyzed sentiments in social networks using case studies of the FIFA World Cup 2018 and Ronaldo's transfer to Juventus. It explores the correlation between sentiment polarity (positive or negative) and fans' sentiments on X platform. Regarding strengths, the paper analyzed social media sentiment from various unstructured sources. It proposes a comprehensive experiment to evaluate different machine learning models' performance recognizing emotions. The study focuses on real-world situations, making the findings more practical. The paper does have some weaknesses. It lacks implementation details for the sentiment analysis system, such as specific methods used for sentiment analysis, feature extraction, and training of machine learning models. The paper focuses more on the overall design and evaluation of the system rather than providing in-depth technical details about the specific methods employed. Regarding the approaches used, the paper mentions employing machine learning models like SVM, Naive Bayes, ANNs, K-NN, and Logistic Regression for sentiment analysis.

Ardianto et al. [35] analyzed sentiment analysis on e-sports for education curricula using Naive Bayes and Support Vector Machine algorithms. It compares the accuracy of the two algorithms and concludes that Naive Bayes performs better in predicting the achievement of e-sports for students' learning curriculum. The strengths are the paper provides a comprehensive evaluation of the topic and, in the end, it provides suggestions for the continuation of the study. The weakness is that the result accuracy is less than 80%, which is considered low accuracy. Also, the percentage of negative and positive tweets resulting from the sentiment analysis is not provided.

Patel and Passi [36] performed an analysis on X platform’s data for World Cup soccer 2014 that aimed to detect the sentiments of people worldwide. The results were analyzed using SVM, K-NN, RF, and NB that gave the best accuracy of 88.17%, which considered the strength of the study, and the information on this study could be valuable for decision-making and event promotion for football. The study relied only on X platform data; therefore, it might not represent the sentiment expressed on other platforms. The study was based on World Cup data from 2014, which may not reflect the current sentiment. In addition, it didn’t show how the data were sampled.

Aloufi and El Saddik [37] have worked on this study to clarify the importance of people's opinions on X platform about football events such as the FIFA World Cup, which attracted millions of fans to generate massive posts. Also, it admitted the challenges of football-related posts analysis, such as using informal language, abbreviations, and expletives. The strength of this study is that it provided an insight into applying different classifiers of sentiment analysis. In contrast, the weakness of this study is that it didn't discuss any potential ethical basis that may occur in the sentiment analysis of football-related posts on the X platform.

2.3 Proposed & Similar System Comparison

This section will show a comparison between the discussed related work and the proposed system, as shown in Table 2-1.

Name	Year	Algorithm	Programming language library	Country and language Domaine	Dataset's source	Field	Result accuracy	Main result
Our system	2024	DL: Bert and RNN ML: XGBoost, SVM, and LightGBM	Python	English posts	X platform	Sentimental analysis on social network and sport	Bert (97%) RNN (95%) XGBoost (90%) SVM (89%) LGBM (88%)	Bert 47.13% Positive posts 18.53%Negative Posts 34.33%Neutral Posts RNN 47.37% Positive posts 18.11%Negative Posts 34.52%Neutral Posts XGBoost 44.87% Positive posts 14.80%Negative Posts 40.33%Neutral Posts SVM 44.72% Positive posts 17.37%Negative Posts 37.89%Neutral Posts LGBM 43.87% Positive posts 15.66%Negative Posts 40.47%Neutral Posts
[31]	2023	NLP	Python, Text Blob	English posts, Indonesia	X platform	Sentimental analysis on social network and sport	NLP (83%)	34.4% Positive posts 16.1% Negative Posts 49.6% Neutral Posts
[32]	2023	NLP (EmoRoberta model)	Python script	Spanish, Arabic posts About world cup football match	X platform	Sentimental analysis on social network and sport	Not mentioned	Arabic 26% Positive posts 50% Neutral posts 25% Negative posts Spanish 8% Positive posts 72% Neutral posts 21% Negative posts
[33]	2023	SVM	Python (specifically, Jupyter notebook in Anaconda navigator)	English and Arabic posts related to women sport	X platform	Sentimental analysis on social network and sport	SVM Pre-Hashtag (91%) Pre and Post Hashtag (85%) Post Hashtag (72%)	Pre-hashtag 90% positive posts 10% negative or neutral posts Pre and post hashtags 85% positive posts Post-hashtag 72% positive posts
[34]	2021	SVM, K-NN, NB, ANN, and LR	Python	Not mentioned	X platform	Sentimental analysis on social network and sport	FIFA World Cup 2018: SVM (0.882 - 0.938) LR (0.759 - 0.884) NB (0.707 - 0.892) K-NN (0.773 - 0.898) ANN (0.093 - 0.338) Cristiano Ronaldo's Transfer: Juventus | Real Madrid SVM: 0.453 | 0.485 LR: 0.772 | 0.705 NB: 0.759 | 0.707 K-NN: 0.773 | 0.700	SVM is most effective for sentiment polarity. France's national team had the highest positive polarity in FIFA 2018.
[35]	2020	SVM and NB	Not mentioned	Indonesian posts, Indonesia	X platform	Sentimental analysis on social network and e-sport	NB with SMOTE (70.32%) SVM with SMOTE (66.92%)	Not mentioned
[36]	2020	SVM, K-NN, RF, and NB	Python and NL toolkit (NLTK) library	English posts about the world cup from all around the world	X platform	Sentimental analysis on social network and sport	NB (88.17%) RF (85.32%) SVM (41.77%) K-NN (87.49%)	7,254 positive posts 12,993 neutral posts 4,088 negative posts
[37]	2018	SVM, MNB, and RF	Not mentioned	Portuguese posts about the World Cup from all around the world	X platform	Sentimental analysis on social network and sport	RF (79%) MNB (61%) SVM (54%)	Not mentioned

Table 2–1Similar System Comparison

2.4 Conclusion

In this chapter, the importance of football in Saudi Arabia and the field of sentiment analysis was highlighted, and sentiment analysis was examined. Powered by natural language processing and machine learning, enables organizations to tap into the sentiments and emotions of customers, which helps in getting insights for decision-making. The different levels of sentiment analysis were discussed, and highlighted three primary approaches: lexicon-based, machine learning-based, and hybrid, each with its strengths and applications.

To provide a broader perspective within the larger academic landscape, a review was conducted for the related works, the algorithms used, and the results. These works offer valuable insights and methodologies for understanding sentiment in various contexts.

Chapter 3: System Analysis

3 System Analysis

This chapter provides a detailed picture of the requirements specifications, including information gathering techniques, functional requirements, non-functional requirements. Furthermore, it delves into the requirement analysis where diagrams are used to describe the project.

3.1 Requirements Specification

This section will show the information gathering techniques used in this project, the functional requirements, the non-functional requirements.

3.1.1 Gathering Information

Requirements gathering constitutes a pivotal element in any project and its management. Achieving a comprehensive understanding of the project's intended outcomes is paramount for its successful execution. While the importance of requirements gathering might seem self-evident, somewhat surprisingly, often receives inadequate consideration.

The main techniques that were used for gathering information were:

- **Literature Review**

  The literature review is a systematic exploration of published work to understand the existing knowledge regarding the proposed research topic [38].

  The previous chapter, where research about the field of machine learning and sentiment analysis has been conducted, and related work has been discussed is going to serve as a valuable asset for the system analysis in several keyways.

  First, it provides a contextual understanding of prior research efforts within our field, offering a roadmap of the methodologies and findings that have come before us. This knowledge is essential for framing our own research, ensuring that it builds upon existing insights and addresses gaps in the current literature.

  Additionally, the background section offers a glimpse into the diverse range of methods and tools employed in sentiment analysis.

  This exposure to various approaches equips us with a broader perspective on how we can approach our analysis and select the most suitable techniques for our specific context.

Furthermore, the background studies offer insights into the complexities of data collection, interpretation, and potential biases, which are critical considerations for our analysis. By learning from the strengths and weaknesses of prior research, we can make informed decisions and navigate potential challenges more effectively.

- **Brainstorming**

  Brainstorming is a method in which a group seeks to generate ideas or solve a particular problem by spontaneously and non-judgmentally gathering a multitude of ideas [39].

  In a brainstorming session, the objective was to generate a wide range of insights for this project. As a result, our brainstorming session generated many ideas that are instrumental in developing our system. The overarching vision centers around creating a sophisticated dashboard meticulously designed to analyze English posts on the X platform. To achieve this vision, six pivotal phases were discussed. They formed our roadmap: data collection, data preprocessing with a strong focus on data cleaning, sentiment analysis, secure data storage, the development of an intuitive dashboard, and the incorporation of advanced filtering functionalities. These phases are instrumental in developing our system, enabling us to achieve our goals effectively.

3.1.2 Functional requirements

This section will delve into the essential aspects of functional requirements, Functional requirements define the specific functions or features a system or software must provide its users. These requirements describe the system's behavior and capabilities in response to user inputs or other stimuli [12].

System functional requirements:

- The system should be able to connect to the comma-separated values (CSV) file that store data.
- The system should be able to visualize the data using dashboards and sheets in a comprehensible manner using charts or graphs.
- The system should be able to be customized to present data in different graphs and formats.
- The system should be able to store the results as an image, pdf, or an powerpoint file.

Admin functional requirement:

- The admin should be able to add dashboard components, filters, and widgets.
- The admin should be able to delete dashboard components, filters, and widgets.
- The admin should be able to customize the dashboard components and widgets.
- The admin should be able to navigate between different dashboards and sheets.
- The admin should be able to download the reports from the dashboard.
- The admin should be able to add comments to the dashboard.
- The admin should be able to add data source to the dashboard.

User functional requirement:

- The user should be able to view the dashboard.
- The user should be able to navigate between different dashboards using buttons.
- The user should be able to filter the results.
- The user should be able to view the data details.
- The user should be able to add comments to the dashboard.
- The user should be able to download the reports from the dashboard.

3.1.3 The Non-functional requirements

This section will delve into the essential aspects of Non-functional requirements. Non-functional requirements, referred to as quality attributes or performance characteristics, describe how the system should perform rather than what it should do [12].

- Performance: minimize the maximum response time to data queries. And

maximize the minimum response time to data queries.

- Security: protected data against unauthorized access.
- ccuracy: predefined level of accuracy in sentiment classification.
- Usability: presentable interface.
- Maintenance: periodic updates and patches for the system components and support for future enhancements.

3.2 Requirements Analysis

The structured approach is a problem-solving and decision-making methodology. The structured paradigm focuses primarily on decomposing behaviors. The analyst defines what the system should do before deciding how it should do it [40] [41].

This section will show the different diagrams that were used during the requirements analysis, which include Use Case Diagram, Data Flow Diagram (DFD), and Entity Relationship Diagram (ERD), to depict the proposed system visually.

A structured approach was chosen for the sentiment analysis, as it enhances the analysis process's accuracy, dependability, and efficiency. Which leads to ultimately results in more refined and actionable insights.

3.2.1 Use case diagram

The use case diagram is a specialized structural modeling of the system functionality. It's usually applied during requirements gathering to capture requirements that define what the system should do. It typically starts early in a project and continues throughout the system development process [42]. Figure 3-1 shows the use case diagram of the sentiment analysis dashboard.

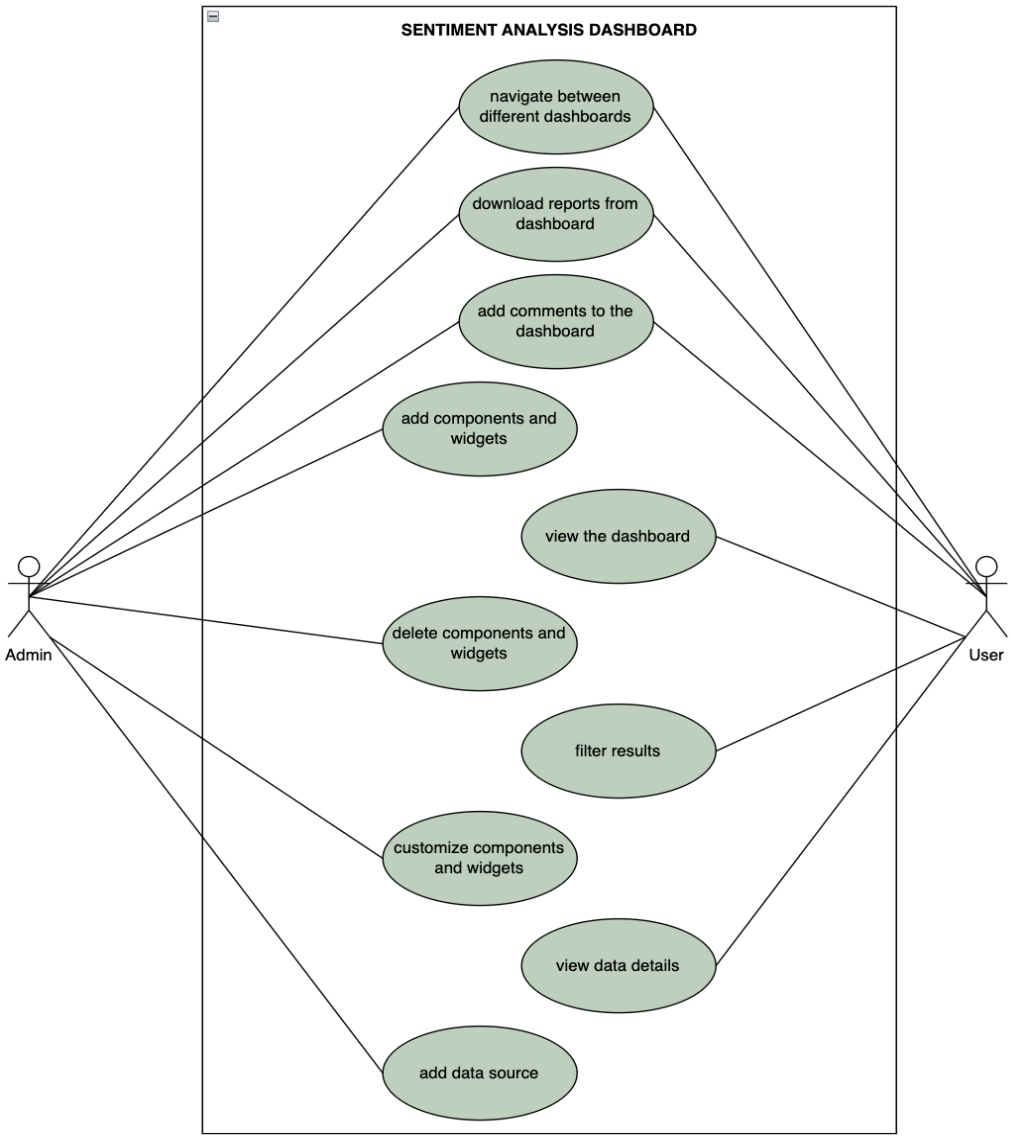


Figure 3–1Use Case Diagram

The high-level description of each use case is in Appendix A.

3.2.2 Data Flow Diagram (DFD)

A Data Flow Diagram is a powerful diagramming tool that visually illustrates how information flows within a process or system. The DFDs shows how data moves between components, including inputs, outputs, data stores, and the data flows between them.

Using standardized symbols and notations, DFDs make understanding and analyzing complex systems more straightforward [43]. The DFD of the proposed system is presented in Figure 3-2, Figure 3-3, and Figure 3-4.

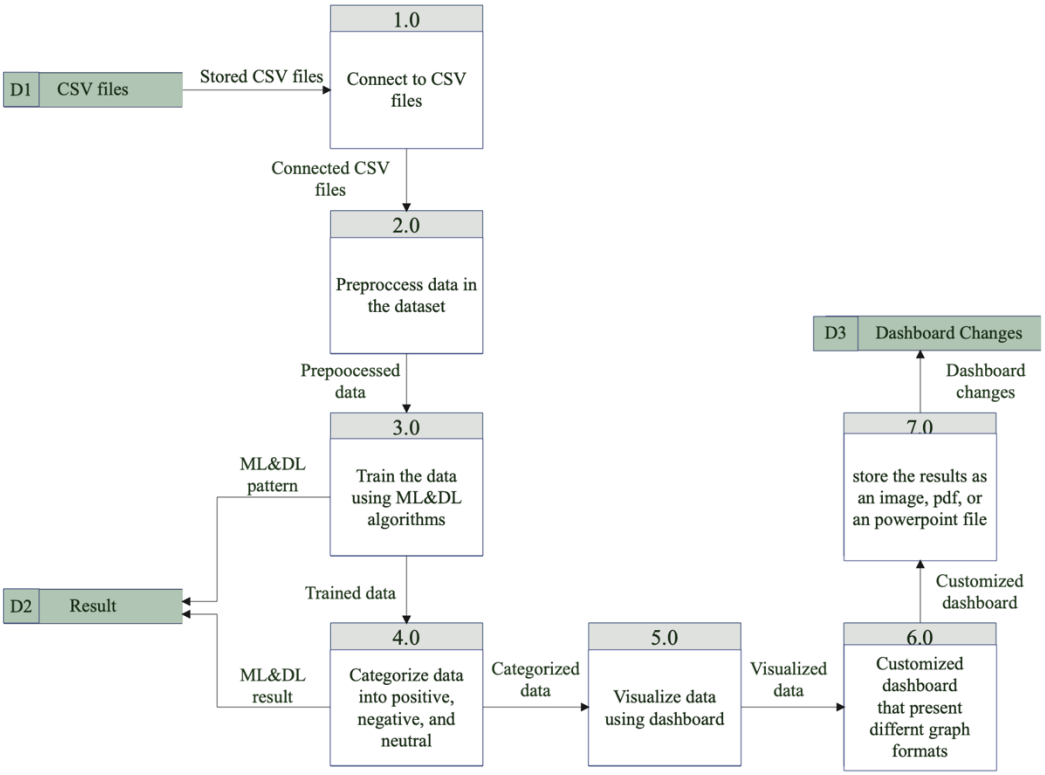


Figure 3–2 DFD System functionality

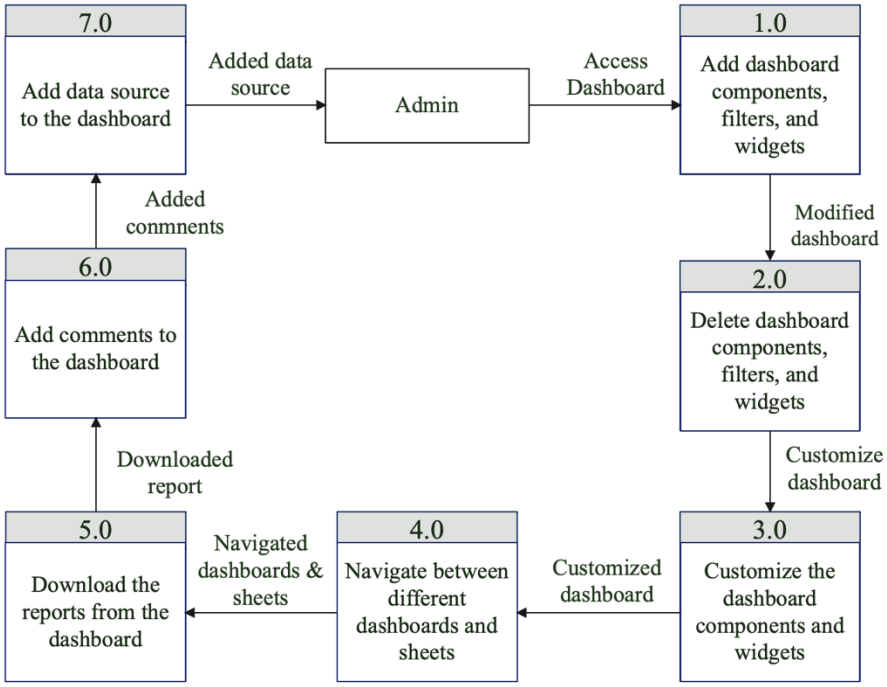


Figure 3–3 DFD Admin functionality

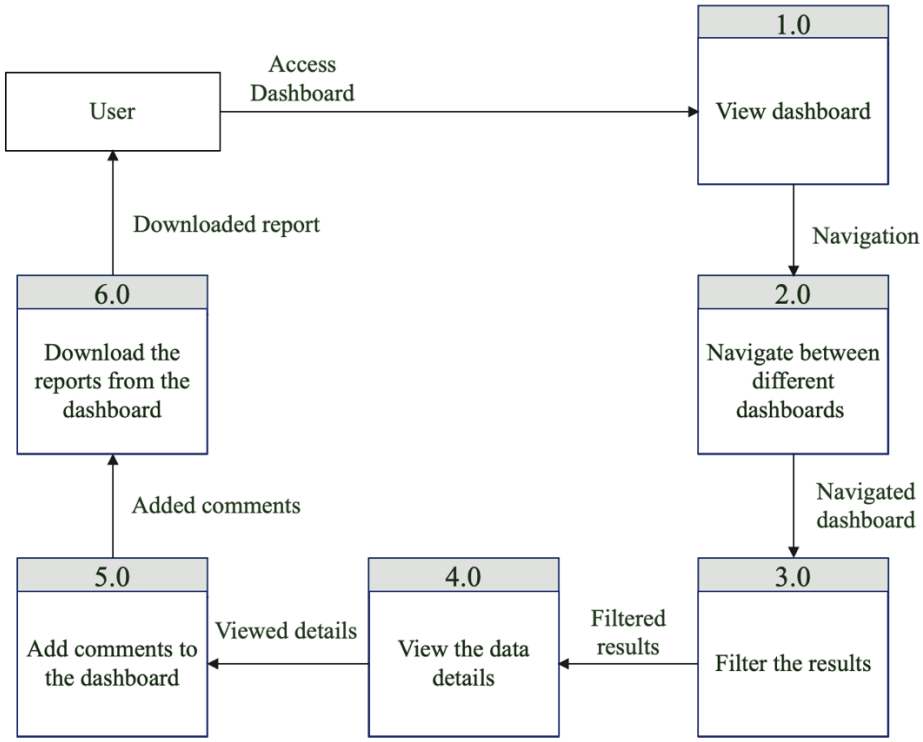


Figure 3–4 DFD User functionality

3.2.3 Entity Relationship Diagram (ERD)

The Entity Relationship Diagram is a type of structural diagram used in database design to illustrate the logical structure of a database, showing how different entities or data objects relate to each other [44]. The system's ERD is illustrated in Figure 3-5.

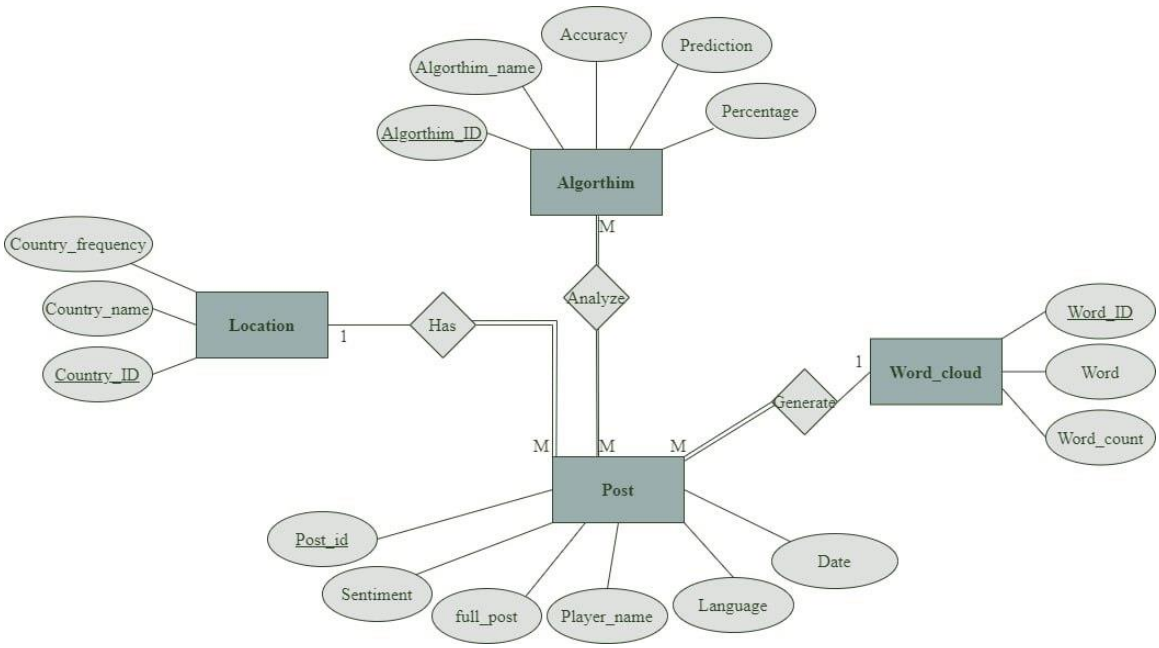


Figure 3–5 Entity Relationship Diagram

Relational Schema

Algorthim (Algorthim_ID, Algorthim_name, Accuracy, Prediction, Percentage)

Location (Country_Id, Country_frequancy, Country_name)

Post (Post_ID, Sentiment, Full_post, Player_name, Language, Date, Country_Id, Word_ID)

FK: Country_Id references Location (Country_Id)

FK: Word_ID references Word_Cloud (Word_ID)

Word_Cloud (Word_ID, Word, Word_count)

Analyze_Post (Algorthim_ID, Post_ID)

FK: Algorthim_ID references Algorthim (Algorthim_ID)

FK: Post_ID references Post (Post_ID)

3.3 Conclusion

This chapter analyzed the system by specifying its requirements and categorizing them into functional, non-functional requirements. Some information-gathering techniques were used, which are literature review and brainstorming. After requirements were specified, they then have been analyzed through a structured approach that includes a use case diagram, data flow diagram (DFD), and an entity relationship diagram (ERD), which in turn facilitate the understanding of the system processes, relationships, interactions, and data flows.

Chapter 4: System Design

4 System Designs

In this chapter, the focus is on the design of system architecture, and user interface. It will demonstrate how these components are meticulously designed to ensure the system is both well-structured and user-friendly.

4.1 System Architecture

This section will show a detailed architecture for the system. System architecture defines the structure, behavior, and multiple system views. It's a formal description and representation of the system [45].

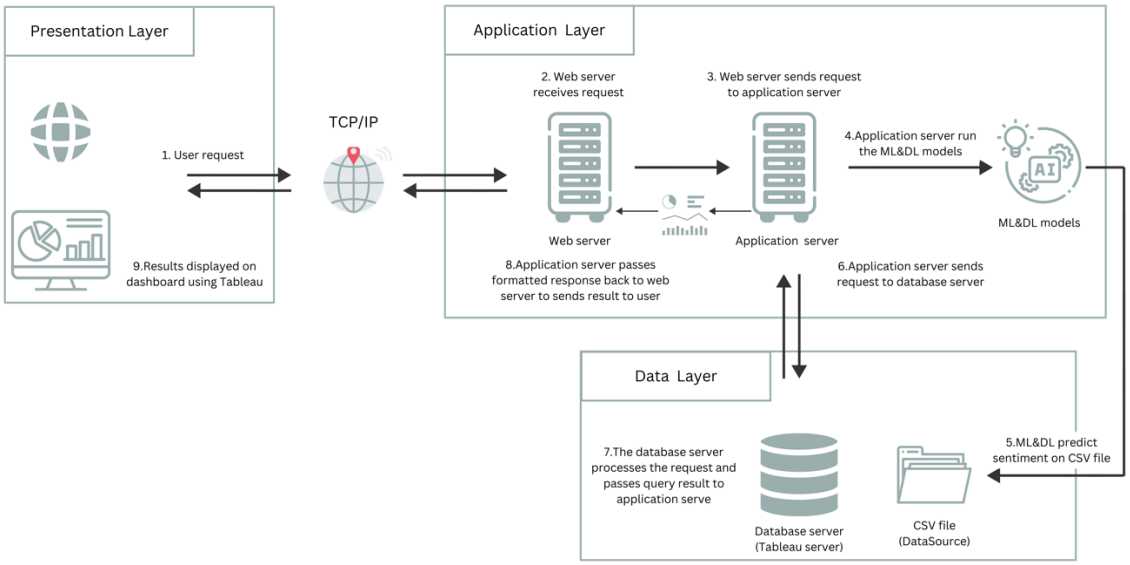


Figure 4–1 System architecture

As shown in Figure 4-1, the Sentiment analysis of X platform data is illustrated in the context of a 3-tier architecture, which consists of three fundamental layers. The Presentation Layer, the Application Layer, and the Data Layer are as follows:

- **Presentation Layer**

The presentation layer, a crucial part of the system, empowers users with control over the user interface. This includes the dashboard and system interaction. In the context of sentiment analysis of X platform data, the presentation layer allows users to customize the Dashboard of the X post they want to analyze and view the results. This layer can include web-based interfaces, mobile applications, or other forms of user interaction.

- **Application Layer**

Where the sentiment analysis happens, the Sentiment analysis process is conducted by Machine Learning and Deep Learning models at the application layer. In this layer, the web server acts as an intermediary between the application server and the web browser. The ML&DL models in the application layer deal with Comma-Separated Values (CSV) files that reside in the data layer to predict the sentiment expressed in the extracted posts of the X platform using the X Application Programming Interface (API) (SocialData) stored on the CSV files. It involves operating an ML&DL model to classify the sentiment as positive, negative, or neutral. The application layer also includes pre-processing steps such as tokenization and lemmatization to prepare the data for analysis. A request is sent to the database server by the application server whenever a user or an administrator logs in. After receiving the response from the database server, the application server formats it and sends it back to the web server. Finally, the web server uses Tableau to display the formatted results on the dashboard.

- **Data Layer**

The X platform data is stored and accessed using CSV files in the data layer. This layer includes storing and retrieving posts that were held in CSV files. The database server processes the request from the application server when the admin or user signs in and then passes the query result as a response to the application server. The data layer can utilize databases, data warehouses, or other storage systems to store and retrieve the X platform data efficiently.

The 3-tier architecture for sentiment analysis of X platform data involves.

- The Presentation layer for user interaction.
- The Application layer for sentiment analysis algorithms.
- The Data layer for storing and accessing the X platform and users' data.

This architecture allows for a modular and scalable approach to analyzing sentiment in large volumes of X platform data.

4.2 User Interface Design

This section will show the expected interfaces design of the system, including the dashboard interface appearance with its functionalities, the admin interfaces including signup and login pages, and the user signup and login pages.

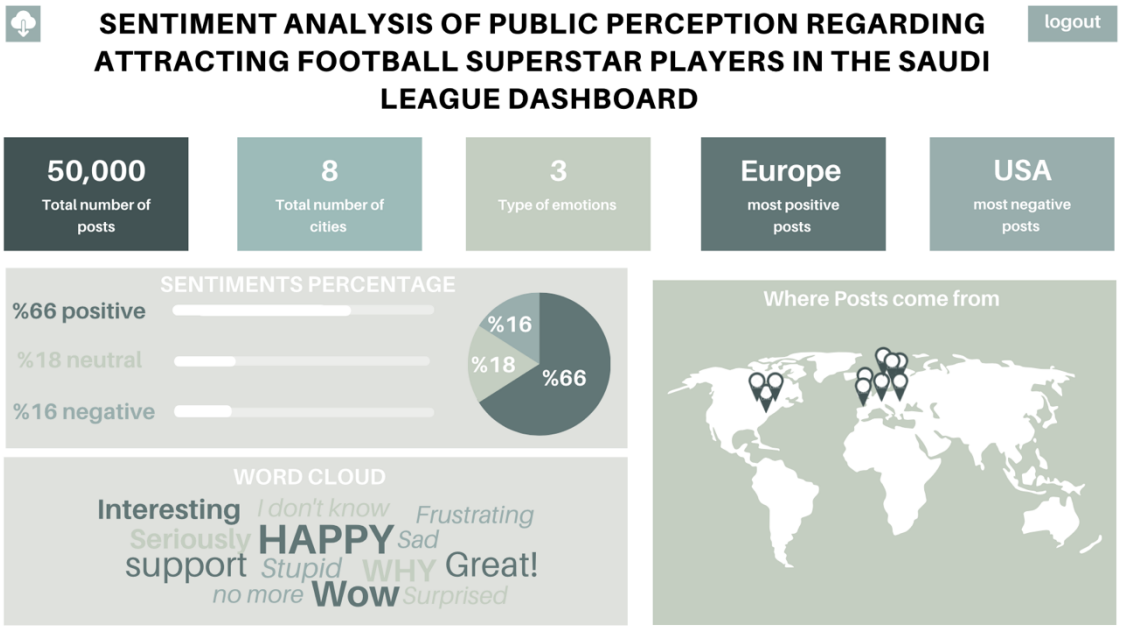


Figure 4–2 Dashboard Design

Figure 4-2 above shows the expected dashboard design with its main functionality that will appear to the user. The main widget is the sentiment percentage widget, which shows the positive, natural, and negative percentages for the posts. Word cloud widget shows the most common words and whether they are positive, natural, or negative. Where posts come from widget shows the geographical location of the cities selected to perform the sentiment analysis. Other widgets show the number of posts analyzed, the number of cities, the type of emotions, and the cities with the most positive or negative posts. The download button enables the user to download the dashboard as image. Lastly, the log-out button is used when the user wants to log out from the dashboard.

4.3 Conclusion

This chapter provided a detailed description of the system architecture which consists of three essential layers presentation layer, application layer, and data layer. Following that it displayed the design of the dashboard and the significant widgets.

Chapter 5: Implementation

5 Implementation

This chapter demonstrates the details of how the implementation occurred. It starts with outlining the hardware and software requirements. Following the implementation details of the data collection, analysis, preprocessing, and the structures of deep and machine learning algorithms provide insight into the utilization of two distinct datasets, with one dedicated to training and testing deep learning algorithms and the other for machine learning algorithms. Lastly, the dashboard I/O screens are displayed.

5.1 Implementation Requirements

This section outline the necessary hardware, software, and Python libraries requirements for our project.

5.1.1 Hardware requirements

The hardware requirements for our project, as outlined in Table 5-1.

Requirement	Recommended	Minimum
Processor	Intel Core i7 or AMD Ryzen 7	Intel Core i3 or AMD Ryzen 3
Memory	8GB or more	4GB
Storage	256GB or larger SSD for faster data access	128GB hard drive or solid-state drive (SSD)
Network Connectivity	Fast and stable internet connection	Stable internet connection

Table 5–1 Hardware requirements

5.1.2 Software requirements

The software requirements for our project, as outlined in Table 5-2.

Requirement	Description	Icon
Python	High-level programming language used to code ML algorithms.	
Google Collab	Cloud-based platform used to run Python code.	
X Platform	Social media platform used to collect the dataset.	
SocialData	SocialData API provide access to X platform data without the need for scaping.	
Draw IO	A web-based diagramming tool used to create use case diagram, entity relationship diagram, and data flow diagram.	
Microsoft word	Word processing application used to create the report.	
Microsoft PowerPoint	Presentation software used to make the presentation and the user interface design.	
Tableau	Tableau is a powerful data visualization software used to create interactive dashboards.	
Google Form	To create a form that will help with the acceptance testing.	

Table 5–2 Software requirements

5.1.3 Python Libraires

Python libraries are sets of pre-written code modules that increase the capabilities of the Python language. By utilizing pre-existing solutions, these libraries offer ready-to-use functions and classes for a variety of tasks, which allows saving time and effort [46]. As outlined in Table 5-3 and Table 5-4.

Library	Definition
Numpy	A fundamental library for numerical computing that provides support for large, multi-dimensional arrays and matrices [46].
Pandas	A library designed for data manipulation, analysis, aligning and merging [47].
CSV	A library for reading and writing CSV (Comma Separated Values) files in Python [48].
Requests	A library is used for making HTTP requests in Python. It allows you to send HTTP/1.1 requests extremely easily [49].
Datetime	A library provides classes for manipulating dates and times in both simple and complex ways [50].
Nltk	Natural Language Toolkit is a library that handles human language data. It's equipped with various tools for tasks like breaking text into words, reducing words to their root form, labeling words, and analyzing sentence structure. With access to over 50 corpora and language resources [51].
Re	The Regular Expressions library is a powerful tool for pattern matching and text manipulation, commonly used in tasks involving text processing, data validation, and information extraction [52].
Pickle	A library used for serializing and deserializing Python objects. Serialization converts a Python object into a byte stream, which can then be stored in a file or sent over a network. Deserialization is the reverse process, where the byte stream is converted back into a Python object [53].
Dateutil	A library simplifies the parsing, manipulation, and formatting of date and time information in various formats [54].
Sklearn	A library that provides a unified interface for implementing various machine learning, preprocessing, cross-validation, and visualization algorithms [55].

Table 5–3 Python Libraires

Python Libraires (cont.)

Library	Definition
TextBlob	A library used in text data processing, offering a streamlined API to facilitate natural language processing (NLP) tasks [56].
XGboost	A library that is utilized in machine learning for implementing gradient boosting algorithms [25].
LightGBM	A library used for implementing gradient-boosting algorithms in machine learning utilizes a unique leaf-wise tree growth strategy, enabling faster training and better accuracy than traditional methods [57].
Tensorflow	An open-source library for numerical computation that presents a high-level interface (Keras) for building and training models and low-level control for numerical operations [58].
Keras	A Python library that develops deep-learning models. It's part of TensorFlow and offers a high-level interface for neural networks [58].
Torch	PyTorch library, primarily used for deep learning applications. It provides tensors and dynamic neural networks in Python with strong GPU acceleration support [59].
Transformers	A library is part of the Hugging Face Transformers library, which provides state-of-the-art pre-trained models for natural language processing (NLP) tasks. It includes various transformer architectures, such as BERT and GPT, along with tools for fine-tuning and using these models [60].
Seaborn	A library for making statistical graphics in python such as confusion matrix. It also helps in exploring and understanding data, its plotting function operate on data frames containing the whole dataset to produce informative plots [61].
Matplotlib	A library in python that is primarily used for visualization, where it allows the visual access to huge amounts of data in easily designed visuals [62].

Table 5–4 Paython Libraires

5.2 Implementation details

The Figure 5-1 below illustrates the sequential stages of the implementation, providing a comprehensive overview of the sentiment analysis implementation journey. Starting with data collection where, we collected datasets from the X platform (Previously known as Twitter), using the socialdata application programming interface API. Following that the dataset pre-processing which is cleaning and preparing the dataset. Labelling the training dataset using the *Textblob* library which assesses the sentiment expressed in the text by analyzing the words and phrases used and assigning a polarity score to indicate whether the sentiment is positive, negative, or neutral. After that, the dataset was split into a training set (80% of the dataset) to train the models and a testing set (20% of the dataset) to evaluate the performance of the trained model. Eventually, extract the sentiment results percentage from the testing set.

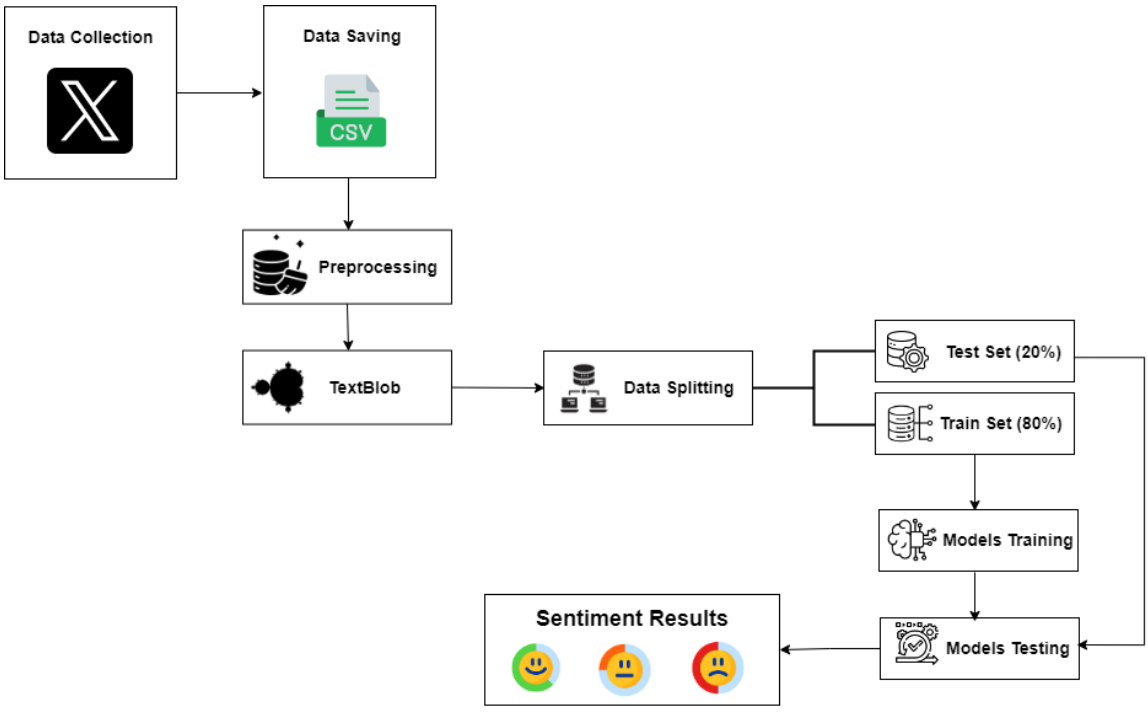


Figure 5–1Implementation details

5.2.1 Data Collection

The dataset was collected from the X platform using the third-party service SocialData API (Application Programming Interface), which offers a scalable and easy way to fetch data from the X platform [63]. The data was collected from 14 January 2024 to 31 December 2022. Posts are obtained from an X platform search by making repeated calls and adding an *until_time* parameter to each subsequent request. The timestamp of the oldest post among the ones retrieved is taken, and another call is made to the same endpoint, adding *until_time*, which is the time stamp of the oldest post in the previous response. This process is repeated until the post older than the timestamp of the relevant news announcement is found.

- Parameters and queries

```
start_with_timestamp = 1682608686

# Ronaldo + Al Nassr
query = "Ronaldo AND (\"Al Nassr\" OR Alnassr OR Alnasser OR \"Al Nasser\" OR KSA OR Saudi) -filter:retweets -filter:links lang:en"
end_at_timestamp = 1672491661

# Neymar Junior + All Hilal
query = "Neymar OR \"Neymar Junior\" AND (\"Al Hilal\" OR AlHilal OR Hilal OR KSA OR Saudi) -filter:retweets -filter:links lang:en"
end_at_timestamp = 1692061261

# Karim Benzema + AlIttihad
query = "Benzema OR \"Karim Benzema\" AND (\"Al Ittihad\" OR AlIttihad OR Ittihad OR KSA OR Saudi) -filter:retweets -filter:links lang:en"
end_at_timestamp = 1686013261
```

The above script shows the *start_with_timestamp* parameter, which represents the time in Unix format when the collection started, which is 12.01.2024. The query parameter indicates the search query for each player-club combination: Ronaldo with Al Nassr, Neymar Junior with Al Hilal, and Karim Benzema with Al Ittihad. Queries are constructed with conditions to filter out retweets and links and focus on English posts. For each player, the *end_at_timestamp* parameter marks the termination point, which is each player's joint date.

- API connection parameters

```
api_key = '54|gNr9GhykCQp2Tva2FOjV5goEAzOT1C9w7Mmt5Yqrfce543fe'
api_endpoint = 'https://api.socialdata.tools/twitter/search'
```

The script above defines two crucial components for interacting with the SocialData API to retrieve posts from the X platform: *api_key* and *api_endpoint*. The *api_key* variable stores the authentication token or API key required to authenticate requests made to the SocialData API. The *api_endpoint* variable stores the URL of the API endpoint where the script will send requests to fetch posts. An API endpoint is the specific URL the API provider provides for accessing its services and retrieving data.

- Get_timestamp() function

```
def get_timestamp(tweet):
    datetime_obj=datetime.fromisoformat(tweet['tweet_created_at'][:-1])
    return int(datetime_obj.timestamp())
```

The script above shows a function called *get_timestamp* that takes the tweet dictionary as input and then extracts the value of the *tweet_created_at* from it. This value is in ISO 8601 format [64] and represents the timestamp when the tweet was created. *datetime.fromisoformat* is used to convert this timestamp into a datatime object, which will then be in Unix format using the *datetime_obj.timestamp()* function.

- Prepare_tweet() function

```
def prepare_tweet(tweet):

    # Remove entities
    if 'entities' in tweet:
        del tweet['entities']

    # Remove quoted_statsus
    if 'quoted_status' in tweet:
        del tweet['quoted_status']

    # Remove retweeted_status
    if 'retweeted_status' in tweet:
        del tweet['retweeted_status']

    # Flatten user object
    if 'user' in tweet:
```

```
    user = {}
    for key, value in tweet['user'].items():
        new_key = "user_" + key
        user[new_key] = value
    tweet.update(user)
    del tweet['user']

return tweet
```

The script above illustrates a function called *prepare_tweet* that processes each tweet retrieved by the API to delete unnecessary fields. It inputs the tweet dictionary and checks if it includes entities key. If it does, then it deletes them. Entities contain information about hashtags, URLs, and user mentions. Delete the *retweeted_status* and *quoted_status* keys if they exited. And flatten the nested user object within the dictionary. It checks if the tweet contains a user key, representing information about its user. If it does, the function iterates over the key-value pairs in the user object and creates new key-value pairs in the tweet dictionary with keys prefixed by *'user_'*.

- Get_search_results() function

```
def get_search_results(query):

    # Set up the headers with the API key
    headers = {
        'Authorization': f'Bearer {api_key}',
    }

    # Prepare the request payload
    params = {
        'query': query,
    }
    print("Processing query:", query)

    # Execute the request
    # If something fails - don't stop execution of this script
    try:
        response = requests.get(api_endpoint, params=params, headers=headers)
    except:
        print('ERROR: API request failed. Will retry')
        return []

    # Return the tweets provided by the API
    if response.status_code == 200:
```

```
        response_data = response.json()
        return response_data['tweets']
    else:
        print('ERROR: Failed to send POST request. Status code:', response.status_code)
        return []
```

The script above shows a function called *get_search_results*, which is responsible for sending requests to the SocialData API to retrieve tweets based on a given search query. It sets up the HTTP request header, which includes the authorization header with the API key, and prepares the request payload, which includes the search query. The sends an HTTP GET request to the API endpoint (*api_endpoint*) specified, with the search query parameters (*params*) and headers (*headers*) included. The *requests.get ()* function is part of the Requests library, which makes HTTP requests in Python.

- Requesting tweets from the API

```
# Recursively request tweets from the API
    should_continue = True
    while should_continue:
        if start_with_timestamp is None:
            search_query = query
        else:
            search_query = query + " until_time:" + str(start_with_timestamp)

        # Execute search query
        tweets = get_search_results(search_query)
```

The above script fetches tweets from the API. If the *start_with_timestamp* is set to None, it means that it is an initial request, and the search query doesn't need to be modified with the *until_time* parameter. Otherwise, it is not the initial request, and the script updates the search query by appending the *until_time* parameter with the value of *start_with_timestamp*. Then, it executes the search query by sending it to the *get_search_results* function.

- Writing data into csv file

```
for tweet in tweets:

        # Remove unnecessary fields from tweet
        processed_tweet = prepare_tweet(tweet)

        # Write row to csv
        writer.writerow([
            processed_tweet['tweet_created_at'],
            processed_tweet['id_str'],
            processed_tweet['full_text'],
            processed_tweet['in_reply_to_status_id_str'],
            processed_tweet['in_reply_to_user_id_str'],
            processed_tweet['in_reply_to_screen_name'],
            processed_tweet['user_id_str'],
            processed_tweet['user_name'],
            processed_tweet['user_screen_name'],
            processed_tweet['user_location'],
            processed_tweet['user_followers_count'],
            processed_tweet['user_friends_count'],
            processed_tweet['user_listed_count'],
            processed_tweet['user_favourites_count'],
            processed_tweet['user_statuses_count'],
            processed_tweet['user_created_at'],
            processed_tweet['quoted_status_id_str'],
            processed_tweet['is_quote_status'],
            processed_tweet['quote_count'],
            processed_tweet['reply_count'],
            processed_tweet['retweet_count'],
            processed_tweet['favorite_count'],
            processed_tweet['lang'],
            processed_tweet['views_count'],
            processed_tweet['bookmark_count'],
        ])
        tweets_count += 1

    print("Total tweets saved:", str(tweets_count), "  Current start_with_timestamp =", str(start_with_timestamp))
```

This script snippet iterates through each tweet retrieved from the SocialData API *tweets*, processes each tweet to remove unnecessary fields using the *prepare_tweet()* function, and then writes the relevant data to a CSV file using the *writer.writerow()* method. After that it updates the *tweets_count* and print the progress.

- Timestamp comparison

```
if len(tweets) > 0:
        earliest_tweet = min(tweets, key=lambda x: x['id'])
        start_with_timestamp = get_timestamp(earliest_tweet) - 1
    else:
        start_with_timestamp = start_with_timestamp - 60
```

This script identifies the oldest tweet in the provided dataset and extracts its timestamp to request more tweets posted before the current one recursively.

5.2.2 Dataset Exploring

Data exploring before preprocessing is a crucial step. In this segment, analysis will be performed to get insights into the raw dataset collected before and gain a comprehensive understanding of the different features of it.

Before exploring the dataset, it is necessary to load the previously collected data from a CSV file named "*3_player_dataset*" into a DataFrame called "*data.*" This can be achieved by using the *read_csv* function from the Pandas library, which is specifically designed for reading CSV files. A DataFrame is a data structure provided by the Pandas library that is widely used for data analysis and manipulation. It organizes data in a tabular format with rows and columns. Additionally, the *low_memory=False* argument is used to disable memory optimization during the parsing process.

```
# Load the preprocessed data
file_path = '/content/3_players_dataset.csv'
data = pd.read_csv(file_path, low_memory=False)
```

- **Shape function**

Figure 5-2 shows the *Shape* function from the *Numpy* library that returns the number of rows and columns of the created DataFrame for the dataset, named *data.*

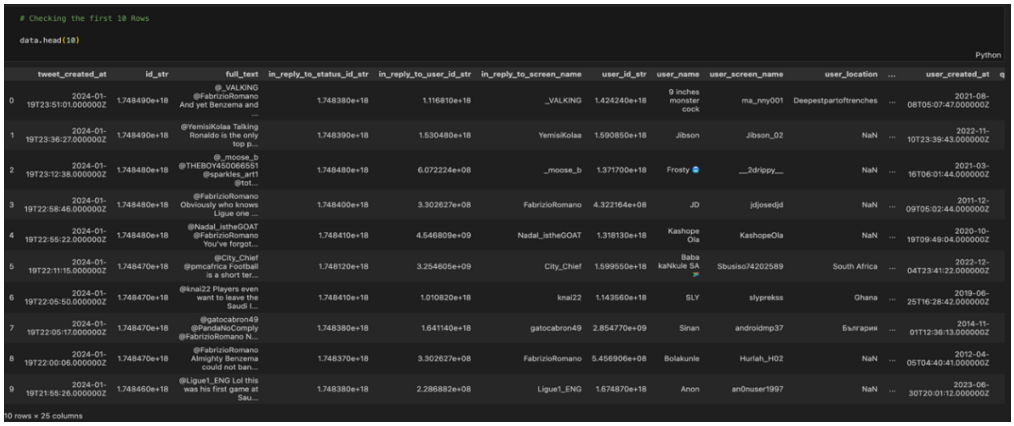
```
data.shape
```

```
(132016, 25)
```

Figure 5–2 Shape function

- **Head function**

Figure 5-3 shows The *Head* function from the *Pandas* library that is used to display the first few rows of the DataFrame.



```
# Checking the first 10 Rows

data.head(10)
```

Python

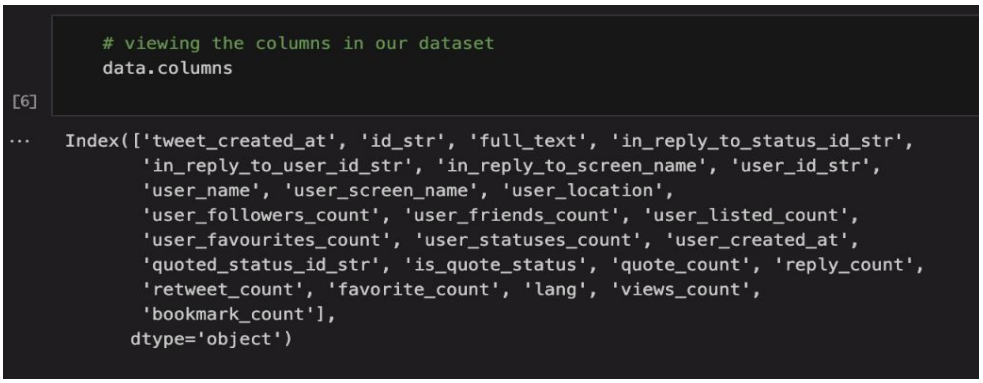
	tweet_created_at	id_str	full_text	in_reply_to_status_id_str	in_reply_to_user_id_str	in_reply_to_screen_name	user_id_str	user_name	user_screen_name	user_location	...	user_created_at	q
0	2024-01-19T23:51:01.000000Z	1.748490e+18	@_VALKING @FabrizioRomano And yet Benzema and ...	1.748380e+18	1.116810e+18	_VALKING	1.424240e+18	9 inches monster cock	ma_nny001	Deepestpartoftrenches	...	2021-08-08T05:07:47.000000Z	
1	2024-01-19T23:36:27.000000Z	1.748490e+18	@YemisiKolaa Talking Ronaldo is the only top p...	1.748390e+18	1.530480e+18	YemisiKolaa	1.590850e+18	Jibson	Jibson_02	NaN	...	2022-11-10T23:39:43.000000Z	
2	2024-01-19T23:12:38.000000Z	1.748480e+18	@_moose_b @THEBOY450066551 @sparkles_art1 @tot...	1.748480e+18	6.072224e+08	_moose_b	1.371700e+18	Frosty ❄️	__2drippy__	NaN	...	2021-03-16T06:01:44.000000Z	
3	2024-01-19T22:58:46.000000Z	1.748480e+18	@FabrizioRomano Obviously who knows Ligue one ...	1.748400e+18	3.302627e+08	FabrizioRomano	4.322164e+08	JD	jdjosedjd	NaN	...	2011-12-09T05:02:44.000000Z	
4	2024-01-19T22:55:22.000000Z	1.748480e+18	@Nadal_istheGOAT @FabrizioRomano You've forgot...	1.748410e+18	4.546809e+09	Nadal_istheGOAT	1.318130e+18	Kashope Ola	KashopeOla	NaN	...	2020-10-19T09:49:04.000000Z	
5	2024-01-19T22:11:15.000000Z	1.748470e+18	@City_Chief @pmcafrica Football is a short ter...	1.748120e+18	3.254605e+09	City_Chief	1.599550e+18	Baba kaNkule SA 🇿🇦	Sbusiso74202589	South Africa	...	2022-12-04T23:41:22.000000Z	
6	2024-01-19T22:05:50.000000Z	1.748470e+18	@knai22 Players even want to leave the Saudi l...	1.748410e+18	1.010820e+18	knai22	1.143560e+18	SLY	slyprekss	Ghana	...	2019-06-25T16:28:42.000000Z	
7	2024-01-19T22:05:17.000000Z	1.748470e+18	@gatocabron49 @PandaNoComply @FabrizioRomano N...	1.748380e+18	1.641140e+18	gatocabron49	2.854770e+09	Sinan	androidmp37	България	...	2014-11-01T12:36:13.000000Z	
8	2024-01-19T22:00:06.000000Z	1.748470e+18	@FabrizioRomano Almighty Benzema could not ban...	1.748370e+18	3.302627e+08	FabrizioRomano	5.456906e+08	Bolakunle	Hurlah_H02	NaN	...	2012-04-05T04:40:41.000000Z	
9	2024-01-19T21:55:26.000000Z	1.748460e+18	@Ligue1_ENG Lol this was his first game at Sau...	1.748380e+18	2.286882e+08	Ligue1_ENG	1.674870e+18	Anon	an0nuser1997	NaN	...	2023-06-30T20:01:12.000000Z	

10 rows × 25 columns

Figure 5–3 Head function

- **Columns function**

Figure 5-4 shows The *Columns* function from the *Pandas* library that is used to retrieve the column names of the DataFrame. It provides a list of the column names.



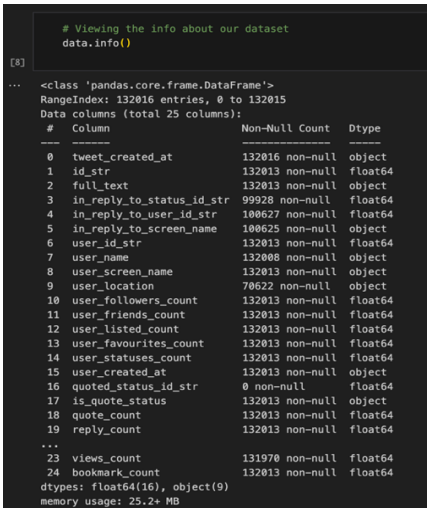
```
[6]
    # viewing the columns in our dataset
    data.columns
```

```
Index(['tweet_created_at', 'id_str', 'full_text', 'in_reply_to_status_id_str',
       'in_reply_to_user_id_str', 'in_reply_to_screen_name', 'user_id_str',
       'user_name', 'user_screen_name', 'user_location',
       'user_followers_count', 'user_friends_count', 'user_listed_count',
       'user_favourites_count', 'user_statuses_count', 'user_created_at',
       'quoted_status_id_str', 'is_quote_status', 'quote_count', 'reply_count',
       'retweet_count', 'favorite_count', 'lang', 'views_count',
       'bookmark_count'],
      dtype='object')
```

Figure 5–4 Columns function

- **Info function**

Figure 5-5 shows The *Info* function from the *Pandas* library that provides a concise summary of the DataFrame, including the column names, data types, and the number of non-null values in each column.



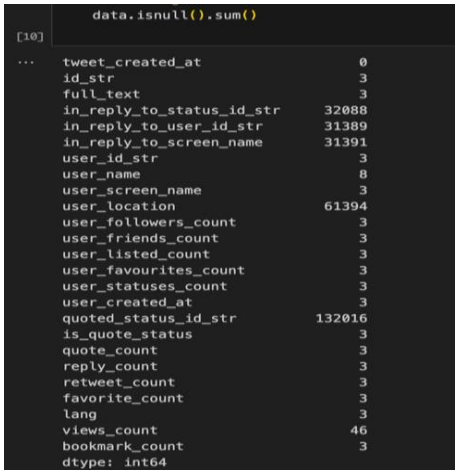
```
[8]
    # Viewing the info about our dataset
    data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 132016 entries, 0 to 132015
Data columns (total 25 columns):
 #   Column                    Non-Null Count   Dtype
---  ------                    --------------   -----
 0   tweet_created_at          132016 non-null  object
 1   id_str                    132013 non-null  float64
 2   full_text                 132013 non-null  object
 3   in_reply_to_status_id_str  99928 non-null   float64
 4   in_reply_to_user_id_str   100627 non-null  float64
 5   in_reply_to_screen_name   100625 non-null  object
 6   user_id_str               132013 non-null  float64
 7   user_name                 132008 non-null  object
 8   user_screen_name          132013 non-null  object
 9   user_location             70622 non-null   object
 10  user_followers_count      132013 non-null  float64
 11  user_friends_count        132013 non-null  float64
 12  user_listed_count         132013 non-null  float64
 13  user_favourites_count     132013 non-null  float64
 14  user_statuses_count       132013 non-null  float64
 15  user_created_at           132013 non-null  object
 16  quoted_status_id_str      0 non-null       float64
 17  is_quote_status           132013 non-null  object
 18  quote_count               132013 non-null  float64
 19  reply_count               132013 non-null  float64
...
 23  views_count               131970 non-null  float64
 24  bookmark_count            132013 non-null  float64
dtypes: float64(16), object(9)
memory usage: 25.2+ MB
```

Figure 5–5 Info function

- **Isnull function**

Figure 5-6 shows The *Isnull* function from the *Pandas* library that is used to identify missing or null values in the DataFrame, with *sum* function it returns the total of the null values in each column which help to understand how many values need to be treated in the preprocessing.



```
[10]
    data.isnull().sum()

tweet_created_at                0
id_str                          3
full_text                       3
in_reply_to_status_id_str   32088
in_reply_to_user_id_str     31389
in_reply_to_screen_name     31391
user_id_str                     3
user_name                       8
user_screen_name                3
user_location               61394
user_followers_count            3
user_friends_count              3
user_listed_count               3
user_favourites_count           3
user_statuses_count             3
user_created_at                 3
quoted_status_id_str       132016
is_quote_status                 3
quote_count                     3
reply_count                     3
retweet_count                   3
favorite_count                  3
lang                            3
views_count                    46
bookmark_count                  3
dtype: int64
```

Figure 5–6 Isnull function

- **Min function**

Figure 5-7 shows The *Min* function from the *Numpy* library that is used to find the minimum value the column *tweet_created_at*. It returns the first date where the posts where collected, which is 31/12/2022.

```
[19]  data['tweet_created_at'].min()

...   Timestamp('2022-12-31 00:00:00')
```

Figure 5–7 Min function

5.2.3 Dataset preprocessing

Dataset *preprocessing* is essential in preparing raw data for subsequent analytical or modeling endeavors. This section focuses on various preprocessing techniques applied to a dataset to clean and prepare text data for analysis.

- **Date String Parsing**

```
# Parsing date strings in a DataFrame, converts them to ISO 8601 format, and handles exceptions.
from dateutil import parser
for i, date_str in enumerate(data['tweet_created_at']):
    try:
        datetime_obj = parser.parse(date_str)
        iso8601_str = datetime_obj.isoformat()
        data.at[i, 'tweet_created_at'] = iso8601_str
    except ValueError:
        data.at[i, 'tweet_created_at'] = None
```

This code segment iterates through each tweet's creation date string, attempting to convert it into the *standardized ISO 8601* datetime format. In case if an unexpected date format occurred a "*None*" value is assigned indicating missing data.

- **Conversion of tweets creation date to Human-Readable Date Format**

```
# Converting 'tweet_created_at' to human-readable date-time format and extracting date
data['tweet_created_at'] = pd.to_datetime(data['tweet_created_at']).dt.date
```

This code segment converts the *'tweet_created_at'* column in a DataFrame called *'data'* to a human-readable date format using the *"to_datetime()"* function by extracting only the date portion of each *'datetime'* object by utilizing the *'.dt.date'* attribute.

- **Conversion of tweets creation date to Standard Datetime Format**

```
# Converting 'tweet_created_at' to standard datetime format
data['tweet_created_at'] = pd.to_datetime(data['tweet_created_at'])
```

This code segment converts the *'tweet_created_at'* column in the DataFrame *'data'* to a standard datetime format using the *"to_datetime()"* function.

- **Drop the Unneeded Columns**

```
del data['quoted_status_id_str']
del data['user_created_at']
del data['user_location']
del data['in_reply_to_status_id_str']
del data['in_reply_to_screen_name']
del data['in_reply_to_screen_name']
del data['user_screen_name']
del data['user_followers_count']
del data['user_friends_count']
del data['user_listed_count']
del data['user_favourites_count']
del data['user_statuses_count']
del data['is_quote_status']
del data['quote_count']
del data['retweet_count']
del data['favorite_count']
del data['bookmark_count']
del data['views_count']
del data['user_id_str']
del data['lang']
del data['reply_count']
del data['in_reply_to_user_id_str']
```

- **Convert 'full_text' from object datatype to string to perform preprocessing on**

```
# Convert the 'full_text' column in the DataFrame 'data' to string data type

data['full_text'] = data['full_text'].astype("string")
```

- **Text Data Cleaning and Standardization**

```
def preprocess(textdata):
    processedText = []
    # Defining regex patterns.
    urlPattern = r"((http://)[^ ]*|(https://)[^ ]*|( www\.)[^ ]*)"
    userPattern = r'@[^\s]+'
    alphaPattern = r"[^a-zA-Z0-9]"
    sequencePattern = r"(.)\1\1+"
    seqReplacePattern = r"\1\1"
```

```
    for tweet in textdata:
        if pd.notnull(tweet):  # Check for missing values
            tweet = tweet.lower()
            # Replace all URLs with 'URL'
            tweet = re.sub(urlPattern, 'URL', tweet)
            # Replace @USERNAME with 'USER'
            tweet = re.sub(userPattern, 'USER', tweet)
            # Remove emojis
            for emoji in emojis.keys():
                tweet = tweet.replace(emoji, '')
            # Remove all non-alphabetic characters
            tweet = re.sub(alphaPattern, " ", tweet)
            # Replace 3 or more consecutive letters by 2 letters
            tweet = re.sub(sequencePattern, seqReplacePattern, tweet)
            processedText.append(tweet)
        else:
            processedText.append('')
    return processedText
processed_tweets = preprocess(data['full_text'])
```

This code function, preprocess, cleans and standardizes a list of text data, likely tweets. It converts text to lowercase, replaces URLs and X usernames with placeholders, removes emojis and non-alphabetic characters, and shortens consecutive sequences of letters because it does not indicate any sentiment if there is any in the review. The cleaned texts are stored in a list, returned by the function, and assigned to the variable *"processed_tweets"* after being applied to the *"full_text"* column of the DataFrame *"data"* [65].

- **Text Data Preprocessing and Quality Assurance**

```
# Assign the preprocessed tweets stored in the 'processed_tweets' list to a new column named 'preprocessed_text' in the DataFrame 'data'.
data['preprocessed_text'] = processed_tweets
# Handle missing values
data.dropna(inplace=True)
# Handle Duplicate values
data.drop_duplicates(inplace=True)
# Remove < 3 words in full text rows
data= data[data['preprocessed_text'].apply(lambda x: len(x.split()) > 2)]
```

This code segment conducts preprocessing on the *"data"* DataFrame, focusing on the *"preprocessed_text"* column, which likely contains processed post data. It begins by assigning preprocessed tweets from the *"processed_tweets"* list to a new column,

"preprocessed_text." Missing values are removed to ensure data completeness and eliminate duplicates to maintain integrity. Additionally, rows with *"preprocessed_text"* containing fewer than three words are excluded to refine the dataset [65].

- **Tokenization, Lemmatization, and Remove stop words**

```
# Function to tokenize, remove stop words, and lemmatize text data
def preprocess_text(df, text_column):
    tokenizer = TweetTokenizer()
    stop_words = set(stopwords.words('english'))
    lemmatizer = WordNetLemmatizer()
    df[text_column + '_tokenized'] = df[text_column].apply(tokenizer.tokenize)
    df[text_column + '_tokenized'] = df[text_column + '_tokenized'].apply(lambda tokens: [lemmatizer.lemmatize(token, get_wordnet_pos(token)) for token in tokens if token.lower() not in stop_words])
    # Remove instances of the word 'user' and numerical token
    df[text_column + '_tokenized'] = df[text_column + '_tokenized'].apply(lambda tokens: [token for token in tokens if token.lower() != 'user'])
    df[text_column + '_tokenized'] = df[text_column + '_tokenized'].apply(lambda tokens: [token for token in tokens if not token.isdigit()])
    return df
```

This code segment focuses on preprocessing text data, particularly for posts.

- Tokenization is breaking down sentences into individual words and phrases [65].

- Removing stop words is a common preprocessing step in natural language processing tasks. Stop words like "the," "is," and "at" are often removed because they appear frequently in text but carry little meaningful information for many NLP tasks.

This process can help in identifying the most important words or features in a text, which is beneficial for tasks like sentiment classification [65].

- Lemmatization techniques aim to reduce words to their base or root form by removing suffixes to get the root form [65].

- Removing numerical tokens from the text data to focus on the textual content rather than numerical values, which may not contribute to sentiment analysis or text classification [65].

The below Table 5-5, shows how each step of the preprocessing works on the entered data, considering the following post as a sample:

"@markgoldbridge UCL ha ha ha don't make us all laugh also benzema is not the same he is old now and struggled in Saudi".

Pre-processing Technique	Processed Post
Remove URL	@markgoldbridge UCL ha ha ha don't make us all laugh also benzema is not the same he is old now and struggled in Saudi
Remove emojis	@markgoldbridge UCL ha ha ha don't make us all laugh also benzema is not the same he is old now and struggled in Saudi
Replace @USERNAME with 'USER'	USER UCL ha ha ha don't make us all laugh also benzema is not the same he is old now and struggled in Saudi
Tokenization	'USER', 'UCL', 'ha', 'ha', 'ha', "don't", 'make', 'us', 'all', 'laugh', 'also', 'benzema', 'is', 'not', 'the', 'same', 'he', 'is', 'old', 'now', 'and', 'struggled', 'in', 'Saudi'
Removing stop words	'USER', 'UCL', 'ha', 'ha', 'ha', "n't", 'make', 'laugh', 'also', 'benzema', 'same', 'old', 'struggled', 'Saudi'
Lemmatization	'USER', 'UCL', 'ha', 'ha', 'ha', "n't", 'make', 'laugh', 'also', 'benzema', 'same', 'old', 'struggle', 'Saudi'
Remove word 'USER'	'UCL', 'ha', 'ha', 'ha', "n't", 'make', 'laugh', 'also', 'benzema', 'same', 'old', 'struggle', 'Saudi'

Table 5–5 Pre-processing Technique

- **Shape function**

Shape function shows the number of rows and columns in a dataframe. The following figures show the difference between the dataframe size before and after preprocessing. As shown in figure 5-8 and 5-9.

```
# Before Preprocess
data.shape

(132016, 25)
```

Figure 5–8 Shape before preprocessing

```
# After preprocessing
data.shape

(118298, 1)
```

Figure 5–9 Shape after preprocessing

5.2.4 Deep Learning algorithms

This section explains the Deep Learning algorithms used for sentiment analysis, including their definitions, structures, and implementation steps.

Dataset

For Deep learning algorithms, the dataset included 132,016 posts divided into (103,836 posts for Ronaldo—15,334 posts for Neymar—12,846 posts for Benzema). After cleaning, it decreased to 118,298 posts. To run the algorithms High-RAM, V100 and T4 Graphic processing unit (GPUs) utilized. As shown in the Figure 5-10, the dataset for all Deep Learning algorithms was split 80% into a training set and 20% into a test set.

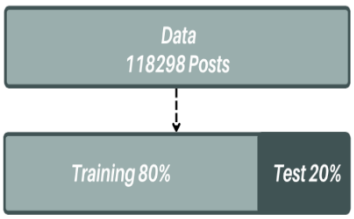


Figure 5–10 Dataset sipliting of DL

Recurrent Neural Network (RNN)

RNN structure

One particular kind of neural network structure used to find patterns in data sequences is called a Recurrent Neural Network (RNN). Data can be numerical time series, handwriting, text, or genomes. Sentiment analysis can be one of RNN's applications since RNN is capable of detecting patterns or relationships in text data and predicting the sentiment. However, it also applies to images handled as a sequence when they break down into a collection of patches. Higher-level applications for RNNs include speech recognition, video tagging, image description generation, and language modeling and generation. RNN differs from Feedforward Neural Networks, also known as Multi-Layer Perceptrons (MLPs), in how information is passed through the network.

Unlike Feedforward Networks, which move data across the network without cycles, RNNs have a unique capability. They can consider not only the current input, Xt but also the previous inputs, X0:t−1. This distinct feature allows RNNs to outperform Feedforward Networks. The corresponding training loss computation and RNN structure are shown in the following Figure 5-11.

A series of input values (x) is translated to a sequence of output values (o) to calculate the difference between o and its training target (y), using the loss function represented by L.

The unnormalized log probabilities, or o, are used when utilizing softmax outputs. Internally, the loss L calculates yˆ = softmax(o) and contrasts it with the target y. In the RNN, the input to hidden connections, hidden-to-hidden recurrent connections, and hidden-to-output connections are determined by the weight matrices U, W, and V in that order. The forward propagation definition for a(t) in this model is given by the equation b + Wh(t-1) + Ux(t).

A(t) is the hidden state at a particular time, whereas b is the bias factor. The weighted sum of the input at the current time step (Ux(t)) and the hidden state from the previous time step (Wh(t-1)) is increased by a fixed number to form the bias term. On the left, recurrent connections represent the RNN and its loss. The same is shown as a computational network unfolding on the right, with each node now connected to a single time-based occurrence. [66] [67].

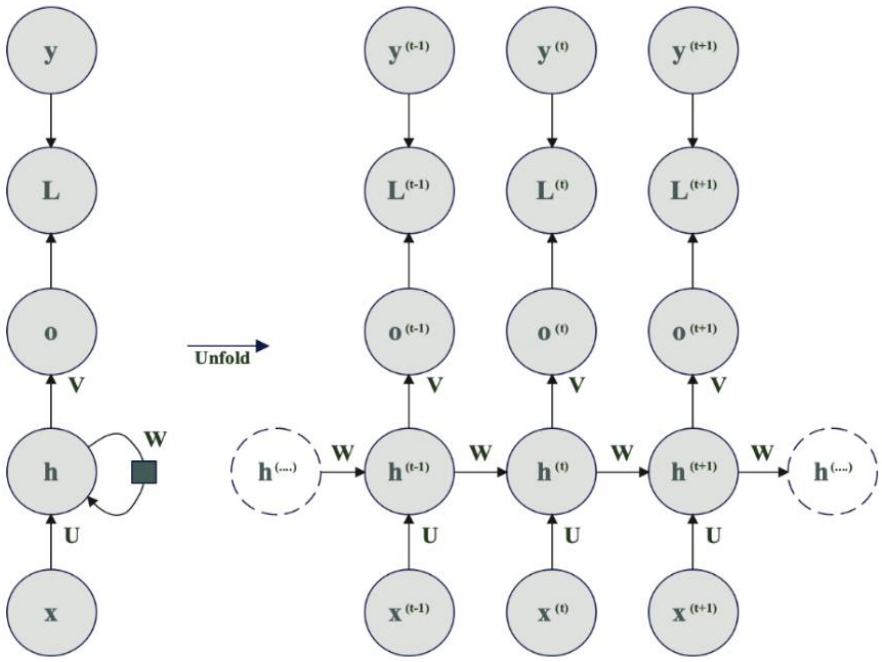


Figure 5–11 RNN structure adapted from [66]

RNN implementation

- Input Data

```
data = pd.read_csv('/content/final_dataset.csv')
```

- Assign Polarity Label

```
# Apply the sentiment analysis function to the 'preprocessed_text_tokenized' column and create a new 'polarity' column
data['polarity'] = data['preprocessed_text_tokenized'].apply(lambda x: TextBlob(x).sentiment.polarity)
# Assign labels based on polarity
data['polarity_label'] = data['polarity'].apply(lambda x: 1 if x > 0 else 0 if x == 0 else -1)
data['Sentiment'] = data['polarity'].apply(lambda x: 'positive' if x > 0 else 'neutral' if x == 0 else 'negative')
```

- Tokenization & Padding the Dataset

```
# Tokenization & Padding the text data
tokenizer = Tokenizer()
tokenizer.fit_on_texts(text)
X_sequence = tokenizer.texts_to_sequences(text)
max_sequence_length = max([len(seq) for seq in X_sequence])
X_padded = pad_sequences(X_sequence, maxlen=max_sequence_length)
```

- Label Encoding (LabelEncoder)

```
# Convert the Sentiment labels to Numerical format
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(sentiment)
y_categorical = to_categorical(y, num_classes=3)  # Assuming 3 sentiment classes
```

- Model Building (SimpleRNN)

```
# Build the RNN model
rnn = Sequential()
rnn.add(Embedding(input_dim=len(tokenizer.word_index)+1, output_dim=64,
input_length=max_sequence_length))
rnn.add(SimpleRNN(64))  # Using a SimpleRNN layer
rnn.add(Dense(3, activation='softmax'))  # 3 output classes for sentiment analysis
```

- Split into training and testing sets

```
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_padded, y_categorical, test_size=0.2, random_state=42)
```

- Model Training

```
# Train the model
rnn.fit(X_train, y_train, epochs=15, batch_size=64, validation_data=(X_test, y_test))
```

- Evaluate the model

```
# Evaluate the model on the test set
y_pred = rnn.predict(X_test)
y_pred_labels = np.argmax(y_pred, axis=1)
y_true_labels = np.argmax(y_test, axis=1)
```

- Evaluate model accuracy

```
# Evaluate the model on the test dataset
test_loss, test_accuracy = rnn.evaluate(X_test, y_test)
print(f'Recurrent Neural Network model accuracy: {test_accuracy}')
```

- Fine-tuning hyperparameters

  The learning rate determines the size of the step taken during the optimization process of training a model. It controls how quickly the model's parameters are adjusted based on the gradients of the *loss function*. While the *batch size* is the number of training examples processed in a single iteration or batch during a model's training. The number of *epochs* is the number of complete passes through the entire training dataset during a model's training [68].

# of training epochs	Batch size	Learning rate (Adam)
15	64	2e-5

Table 5–6 RNN Fine-tuning hyperparameters

- RNN Accuracy

Training set accuracy	Test set accuracy
99.67%	95.60%

Table 5–7 RNN Accuracy

- RNN Results

Positive percentage	Neutral percentage	Negative percentage
47.37%	34.52%	18.11%

Table 5–8 RNN Results

Bidirectional Encoder Representations from Transformers (BERT)

BERT structure

This project utilized BERT-base-uncased, which consists of 12 transformer blocks, each with 12 multi-headed attention layers. With an output dimensionality of 768. And boasts approximately 110 million parameters. The Bert base is designed to generate dense representations of input tokens capturing rich semantic and contextual information. The *"uncased"* aspect means that all text is converted to lowercase during pre-training before tokenization, simplifying the tokenization process and enabling the model to generalize better across different text styles [18]. The embedding layer, as shown in Figure 5-12, consists of token/word embedding, which maps each token into a high-dimensional vector; these word embeddings capture the semantic meaning of each token based on its context within the input text. Position embedding, which encodes the position of each token within the input sequence, helps the model understand the sequential order of the words in the input. Segment embeddings distinguish between tokens belonging to different segments or sentences within the input sequence. The transformer encoder layer comprises multiple transformer encoder blocks stacked on top of each other. Each transformer encoder block consists of two sub-layers: the multi-head self-attention mechanism and the position-wise feed-forward neural network. The multi-head self-attention mechanism allows Bert to capture dependencies between words in both directions within a sentence.

It enables each token to attend to all other tokens in the sequence, capturing contextual information effectively. The multi-head aspect allows the model to focus on different aspects of the input sequence simultaneously, enhancing its ability to learn complex patterns. After the self-attention mechanism, the output passes through a position-wise feed-forward neural network within each encoder block.

This network applies a fully connected feed-forward layer separately to each position in the sequence, enabling the model to capture non-linear relationships between tokens. Residual connections are applied before and after each sub-layer (i.e., self-attention mechanism and feed-forward neural network) .Residual connections help stabilize and speed up the training process by allowing gradients to flow more directly through the network. Layer normalization is also applied after each sub-layer, a technique that normalizes neurons' activations within a neural network layer. It helps stabilize the training

process by reducing the internal covariate shift, which refers to the change in the distribution of layer inputs during training. [69] Lastly, there is a classification layer on top of the pre-trained Bert. It consists of fully connected (dense) layers, a type of neural network layer where each neuron is connected to every neuron in the previous layer; these dense layers transform the aggregated representation obtained from the BERT model into a form suitable for sentiment classification. A Softmax function is used in the output layer of classification models to convert raw model outputs into probability distributions over multiple classes. It takes as input a vector of raw scores produced by the previous dense layers and transforms them into probabilities that sum to 1 [70] [71].

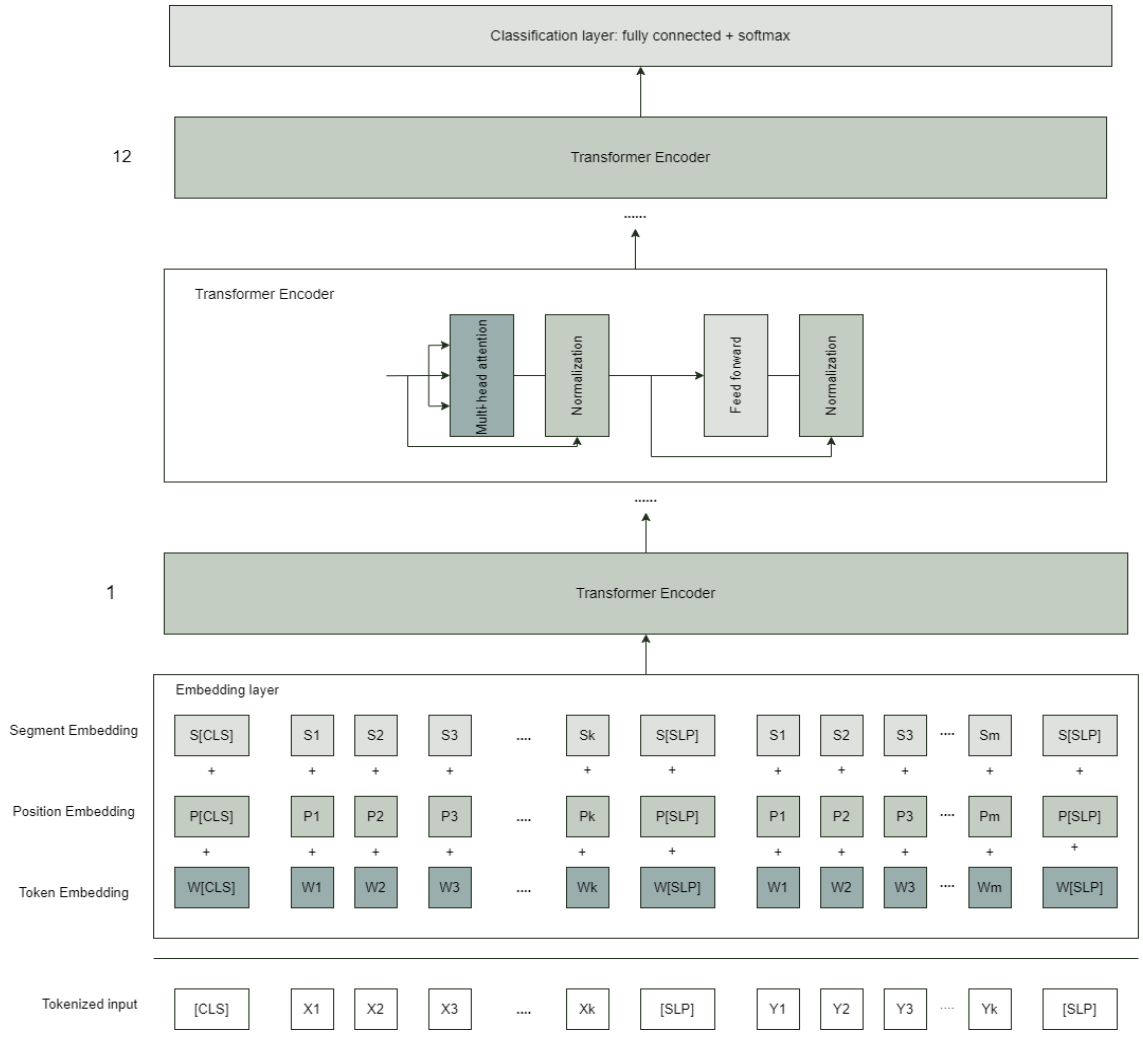


Figure 5–12 BERT base-uncased for text classification Architecture adapted from[71]

Bert Implementation

- Input data

```
# Load the dataset
data = pd.read_csv('/content/final_dataset.csv')
```

- Apply & assign polarity

```
# Apply the sentiment analysis function & create a new 'polarity' column
data['polarity'] = data['preprocessed_text_tokenized'].apply(lambda x: TextBlob(x).sentiment.polarity)
# Assign labels based on polarity
data['polarity_label'] = data['polarity'].apply(lambda x: 1 if x > 0 else 0 if x == 0 else 2)
```

- Split the dataset into training and testing

```
# Split the dataset into training and testing sets
train_data, test_data, train_labels, test_labels = train_test_split(
data['preprocessed_text_tokenized'], data['polarity_label'], test_size=0.2, random_state=42)
```

- Load Bert base-uncased and tokenizer from hugging face transformer library

```
# Load the pre-trained BERT model and tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertForSequenceClassification.from_pretrained('bert-base-uncased', num_labels=3)  # 3 classes
```

- Input data tokenization

```
# Tokenize and format the input data
train_encodings = tokenizer(list(train_data), truncation=True, padding=True, return_tensors='pt')test_encodings =
tokenizer(list(test_data), truncation=True, padding=True, return_tensors='pt')
```

- Training set series into lists conversion

```
# Convert pandas Series to lists
train_input_ids = train_encodings['input_ids'].tolist()
train_attention_mask = train_encodings['attention_mask'].tolist()
train_labels_list = train_labels.tolist()
```

This code segment prepares the data into format that can be fed directly into Bert.

- DataLoader creation

```
# Create PyTorch DataLoader
train_dataset = TensorDataset(torch.tensor(train_input_ids), torch.tensor(train_attention_mask),
torch.tensor(train_labels_list))
train_dataloader = DataLoader(train_dataset, batch_size=16, shuffle=True)
```

This code segment enables the dataset to be loaded in batches and to handle parallelization to send up the training process.

- Testing set series into lists conversion and DataLoader creation

```
# Similar conversion for the test dataset
test_input_ids = test_encodings['input_ids'].tolist()
test_attention_mask = test_encodings['attention_mask'].tolist()
test_labels_list = test_labels.tolist()
test_dataset = TensorDataset(torch.tensor(test_input_ids), torch.tensor(test_attention_mask),
torch.tensor(test_labels_list))
test_dataloader = DataLoader(test_dataset, batch_size=16, shuffle=False)
```

- Optimizer and loss function initialization

```
# Training parameters
optimizer = torch.optim.AdamW(model.parameters(), lr=2e-5)
criterion = torch.nn.CrossEntropyLoss()
```

- Training loop

```
# Training loop
epochs = 1
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model.to(device)
for epoch in range(epochs):
    model.train()
    for batch in DataLoader(train_dataset, batch_size=16, shuffle=True):
        input_ids, attention_mask, labels = batch
        input_ids, attention_mask, labels = input_ids.to(device), attention_mask.to(device), labels.to(device)
        optimizer.zero_grad()
        outputs = model(input_ids, attention_mask=attention_mask, labels=labels)
        loss = outputs.loss
        loss.backward()
        optimizer.step()
```

This code segment checks if there's GPU and utilizes it if it exists. After that, start training the model on batches of data, and then the optimizer updates the model parameters to minimize the loss.

- Training accuracy calculations

```
#Training accuracy
model.eval()
with torch.no_grad():
    train_preds = []
    train_labels_list = []
```

```
    for batch in DataLoader(train_dataset, batch_size=8, shuffle=False):
        input_ids, attention_mask, labels = batch
        input_ids, attention_mask, labels = input_ids.to(device), attention_mask.to(device), labels.to(device)

        outputs = model(input_ids, attention_mask=attention_mask)
        logits = outputs.logits
        preds = torch.argmax(logits, dim=1).cpu().numpy()

        train_preds.extend(preds)
        train_labels_list.extend(labels.cpu().numpy())
train_accuracy = accuracy_score(train_labels_list, train_preds)
print(f'Training Accuracy: {train_accuracy * 100:.2f}%')
```

- Fine-tuning hyperparameters

# of training epochs	Batch size	Learning rate (Adam)
1	16	2e-5

Table 5–9 Bert Fine-tuning hyperparameters

- Bert Accuracy

Training set accuracy	Test set accuracy
97.92%	97.27%

Table 5–10 Bert Accuracy

- Bert Results

Positive percentage	Neutral percentage	Negative percentage
47.13%	34.33%	18.53%

Table 5–11 Bert Results

5.2.5 Machine Learning algorithms

This section explains the machine learning algorithms used in the project, including their definitions structure and implementation steps.

Dataset

For Machine Learning algorithms, the dataset before cleaning included 56,000 posts divided as: (26,000 posts for Ronaldo – 20,000 posts for Neymar – 10,000 posts for Benzema) and after cleaning it decreased to 50,613 posts.

Figure 5–13 Dataset splitting of ML

As shown in the Figure 5-13, the splitting for the dataset for all Machine Learning algorithms was done as: 80% training set and 20% test set.

Support Vector Machine (SVM)

SVM structure

SVM is a supervised machine learning algorithm that has some strategies to be used for multi-classes classification such as one-vs-rest or one-vs-one to effectively compare and classify data into more than two groups. These strategies allow SVM to handle multi-class classification tasks by using multiple binary classifiers [23]. SVM is based on the concept of calculating margins, where it is used to separate groups of data by drawing a line in between because it is a linear machine learning algorithm. The Figure 5-14 below shows the structure of the SVM model and how it works [72].

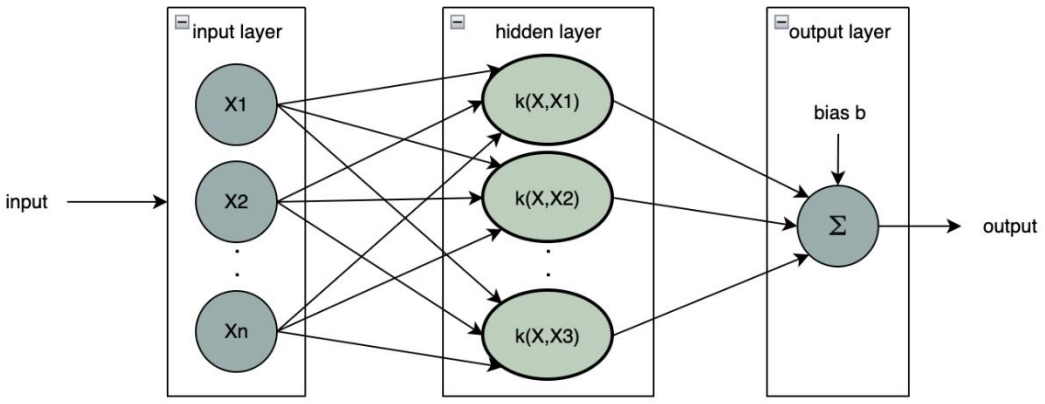


Figure 5–14 SVM structure [72]

SVM implementation

- Input data

```
# Load the preprocessed data
file_path = '/content/10Benzema20Neymar26RonaldoUPDATED.csv'
data = pd.read_csv(file_path, low_memory=False)
```

- Assign polarity label

```
# Assign labels based on polarity
data['polarity_label'] = data['polarity'].apply(lambda x: 'positive' if x > 0 else 'negative' if x < 0 else 'neutral')
data['polarity_label'] = data['polarity_label'].map({'negative': -1,'neutral': 0, 'positive': 1})
```

- Converting text data into a numerical representation using TF-IDF function

```
 # TF-IDF Vectorization
tfidf = TfidfVectorizer(stop_words='english', max_features=5000)
X = tfidf.fit_transform(data['preprocessed_text_tokenized'])
y = data['polarity_label']
```

- Split into training and testing sets

```
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

- SVM classifier

```
# Create the model with the best parameters
svm_best = SVC(C=10, gamma='scale', kernel='linear', probability=True)
```

- Model training.

```
# Train the model
svm_best.fit(X_train, y_train)
```

- Make predictions.

```
# Predict on the test data
y_pred = svm_best.predict(X_test)
```

- SVM Accuracy

Training set accuracy	Test set accuracy
93.9%	88.9%

Table 5–12 SVM Accuracy

- SVM Results

Positive percentage	Neutral percentage	Negative percentage
44.73%	37.89%	17.38%

Table 5–13 SVM Results

eXtreme Gradient Boosting (XGBoost)

XGBoost structure

XGBoost is a machine learning algorithm from gradient boost family that utilizes parallel tree-building methods and incorporates built-in regularization to improve computational efficiency and manage model complexity [25].

The Figure 5-15 below illustrates how XGBoost employs decision trees to generate predictions, the root node is the parent node in the tree that represents the whole dataset, the internal nodes represents the features splits based on a spesfic condition, the leaf nodes are at the bottom of the tree and they represents the final prediction, in case of sentiment analysis the leaf nodes could be the following classes (positive/negative/neutral) [61]. XGboost has the ability to grow in a level-wise manner, which means it grows trees level by level rather than leaf by leaf [57].

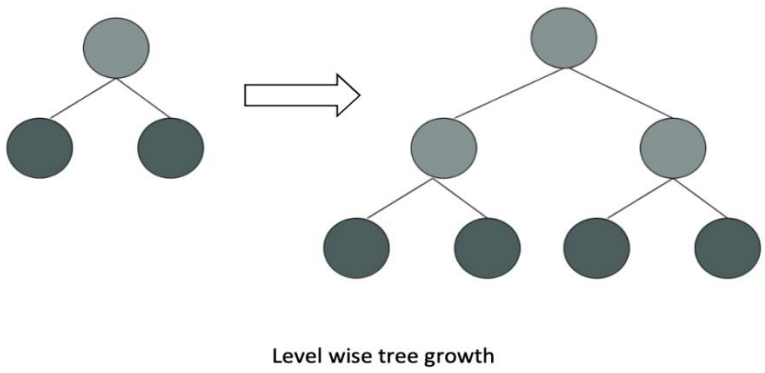


Figure 5–15 XGBoost structure adapted from [57]

XGboost Implementation

- Input Data

```
# Load The dataset

file_path = '/content/10Benzema20Neymar26RonaldoUPDATED1.csv'
data = pd.read_csv(file_path, low_memory=False)
```

- Assign Polarity Label

```
# Apply the sentiment analysis function to the 'preprocessed_text_tokenized' column and create a new 'polarity' column
data['polarity'] = data['preprocessed_text_tokenized'].apply(lambda x: TextBlob(x).sentiment.polarity)
# Assign labels based on polarity
data['polarity_label'] = data['polarity'].apply(lambda x: 1 if x > 0 else 0 if x == 0 else -1)
```

- Split into training and testing sets

```
# Split the dataset into training and testing sets
train_data, test_data, train_labels, test_labels = train_test_split(
    data['preprocessed_text_tokenized'], data['polarity_label'], test_size=0.2, random_state=42)
```

- Feature extraction

```
# Convert text data to feature vectors using CountVectorizer
vectorizer = CountVectorizer(max_features=5000)  # adjust the max_features parameter
train_features = vectorizer.fit_transform(train_data)
test_features = vectorizer.transform(test_data)
```

- Label encoding

```
# Use LabelEncoder to transform labels into consecutive integers
label_encoder = LabelEncoder()
train_labels_encoded = label_encoder.fit_transform(train_labels)
test_labels_encoded = label_encoder.transform(test_labels)
```

- XGboost classifier

```
# Initialize XGBoost model
xgb_model = XGBClassifier()
```

- Model training

```
# Train the model
xgb_model.fit(train_features, train_labels_encoded)
```

- Make predictions

```
# Make predictions on the test set
predictions = xgb_model.predict(test_features)
```

- Evaluate model accuracy

```
# Evaluate the accuracy
accuracy = accuracy_score(test_labels_encoded, predictions)
print(f'Accuracy: {accuracy * 100:.2f}%')
```

- XGboost Accuracy

Training set accuracy	Test set accuracy
92.03%	90.26%

Table 5–14 XGboost Accuracy

- XGboost Results

Positive percentage	Neutral percentage	Negative percentage
44.87%	40.33%	14.80%

Table 5–15 XGboost Results

- XGboost Features

Max features determine the maximum amount of characteristics that can be employed in the training/classification process to characterize texts [73].

The table 5-16 shows how adjusting the *Max_Features* varible change the test set accuracy.

Max_Features	5000	1000	500	100
Test set accuracy	90.26%	90.24%	84.70%	68.22%

Table 5–16 XGboost Max Features

Light Gradient-Boosting Machine (LGBM)

LGBM structure

The LGBM model is a decision-tree-based algorithm from the gradient boost family that divides the input layer's parameters into different parts, thereby constructing the mapping relationship between the inputs and outputs. The Figure 5-16 below shows how LGBM employs decision trees, which is similar to XGBoost, but the difference lies in leaf-wise tree growth instead of the widely used level-wise tree growth. This approach prioritizes tree growth at nodes where the most significant error reduction can be achieved. As a result, the leaf-wise tree algorithm typically generates fewer tree nodes than the level-wise tree algorithm at the same tree depth as XGBoost. Consequently, this reduction in tree nodes leads to a notable acceleration in training time, particularly beneficial when handling large datasets [57].

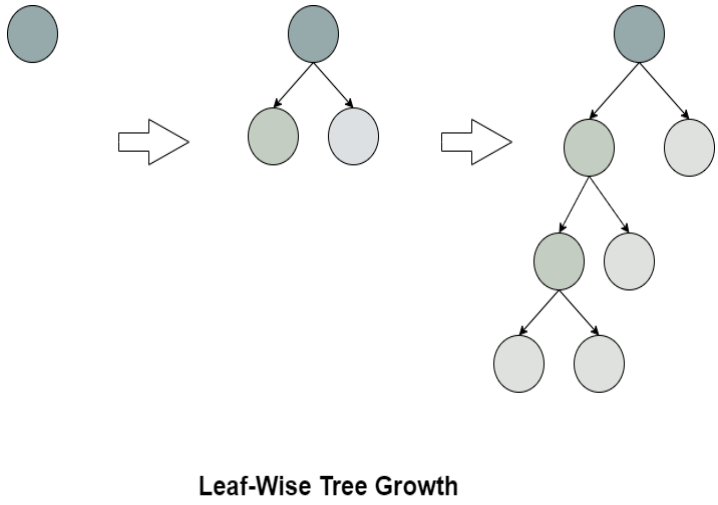


Figure 5–16 LGBM structure adapted from [57]

LGBM Implementation

- Input Data

```
# Load The dataset
file_path = '/content/10Benzema20Neymar26RonaldoUPDATED1.csv'
data = pd.read_csv(file_path, low_memory=False)
```

- Defining the get_polarity() Function

```
# Function to calculate polarity
def get_polarity(text):
 analysis = TextBlob(text)
return analysis.sentiment.polarity
```

This function get_polarity(text) takes a text input, analyzes its sentiment using TextBlob, and returns the polarity score.

- Assign Polarity Label

```
# Assign labels based on polarity
data['polarity_label'] = data['polarity'].apply(lambda x: 'positive' if x > 0 else 'negative' if x < 0 else 'neutral')
data['polarity_label'] = data['polarity_label'].map({'negative': -1,'neutral': 0, 'positive': 1})
data
```

- Converting text data into a numerical representation using TF-IDF

```
# Import the TfidfVectorizer class from scikit-learn and initialize it
tfidf = TfidfVectorizer(stop_words='english', max_features=5000)
# Transform the preprocessed and tokenized text data into TF-IDF matrix.
X = tfidf.fit_transform(data['preprocessed_text_tokenized'])
# Extract the target labels from the DataFrame, which represent the sentiment polarity.
y = data['polarity_label']
```

This code segment prepares the text data and its corresponding labels for machine learning model training.

- Split into training and testing sets

```
# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

- Initialization of LGBM Classifier

```
# Import the LightGBM library
import lightgbm as lgb
# Initialize the LightGBM classifier
clf = lgb.LGBMClassifier()
```

- Model training

```
# Train the classifier on the training data
clf.fit(X_train, y_train)
```

- Make predictions

```
# Predict the results using the trained classifier
y_pred = clf.predict(X_test)
```

- Evaluate model accuracy

```
# Calculating the accuracy score
accuracy = accuracy_score(y_pred, y_test)
# Displaying the accuracy score
print('LightGBM Model accuracy score: {0:0.4f}'.format(accuracy_score(y_test, y_pred)))
```

- LGBM Accuracy

Training set accuracy	Test set accuracy
90.59%	88.81%

Table 5–17 LGBM Accuracy

- LGBM Results

Positive percentage	Neutral percentage	Negative percentage
43.87%	40.47%	15.66%

Table 5–18 LGBM Results

- LGBM Features

Max features determine the maximum amount of characteristics that can be employed in the training/classification process to characterize texts [73].

The Table 5-19 shows how adjusting the *Max_Features* varible change the test set accuracy.

Max_Features	5000	1000	500	100
Test set accuracy	88.81%	85.98%	80.76%	64.94%

Table 5–19 LGBM Max Features

The purpose of utilizing two distinct datasets

The utilization of two different datasets is a common approach in machine learning. Models like support vector machines (SVMs), decision trees, or k-nearest neighbors can perform effectively with small datasets, as they don't require as much data to generalize efficiently. On the other hand, deep learning models, such as convolutional neural networks (CNNs) or recurrent neural networks (RNNs), thrive on large datasets due to their ability to learn

hierarchical representations. They often require a large amount of data to generalize well and avoid overfitting [74].

5.3 I/O Screens

- **Main dashboard**

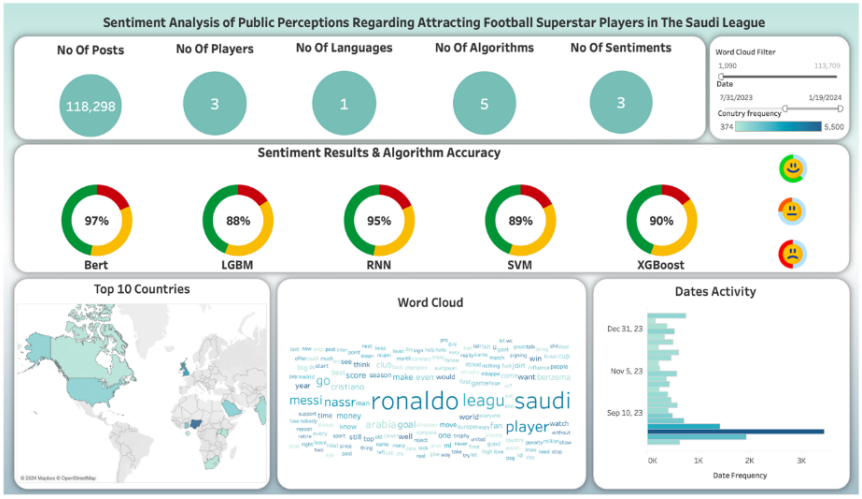


Figure 5–17 Main dashboard

Figure 5-17 shows the main dashboard, which contain all the six main widgets, the first widgets display the general information about the project such as number of posts collected number of players, number of languages, number of algorithms used, and number of sentiments as you can see in Figure 5-18 when the mouse is hoovered into the widget, all the details will be displayed.

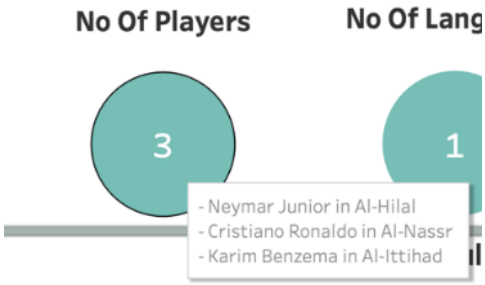


Figure 5–18 number of players widget

The next widget is the sentiment results and algorithm accuracy, which display the accuracy of the algorithms and the percentage of each sentiment. The top 10 countries

display a map that highlight the top 10 active countries in term of the number of posts as shown in Table 5-20.

Country name	Number of posts
Nigeria	5,500
United Kingdom	3,387
Ghana	1,626
Saudi Arabia	1,432
United States	1,312
India	832
Canada	422
South Africa	408
Kenya	407
Uganda	347

Table 5–20 Top 10 countries

Word cloud widget display the most repeated words from the dataset, Table 5-21 show the details of the widgets, which include the top 10 most repeated words and how many time they were repeated.

Word	Number of posts
Ronaldo	113,709
Saudi	92,642
League	47,381
player	36,177
Messi	27,232
Nasser	25,744
Go	24,635
Arabia	17,464
Goal	13,491
Fan	11,150

Table 5–21 Top 10 most repeated words

The Table 5-22 displays all the dates and activities starting from 14 January 2024 to 31 December 2022. The table also includes filters located at the upper right corner, allowing users to filter the results based on word counts, date range, and country frequency.

Date	Frequency
18/07/2023	3898
07/06/2023	3657
15/08/2023	3449
06/06/2023	2840
16/08/2023	2294

Table 5–22 dates activity

Lastly the three buttons in the sentiment results and algorithms accuracy widget navigate the user into the other three dashboards customized for each sentiment.

- **Positive sentiment dashboard**

The Figure 5-19 shows the positive sentiment dashboard that contain the all the widgets that were customized to the positive sentiment. For example, it has a word cloud for the positive words only and a new widget that show a sample of the positive words from the dataset.

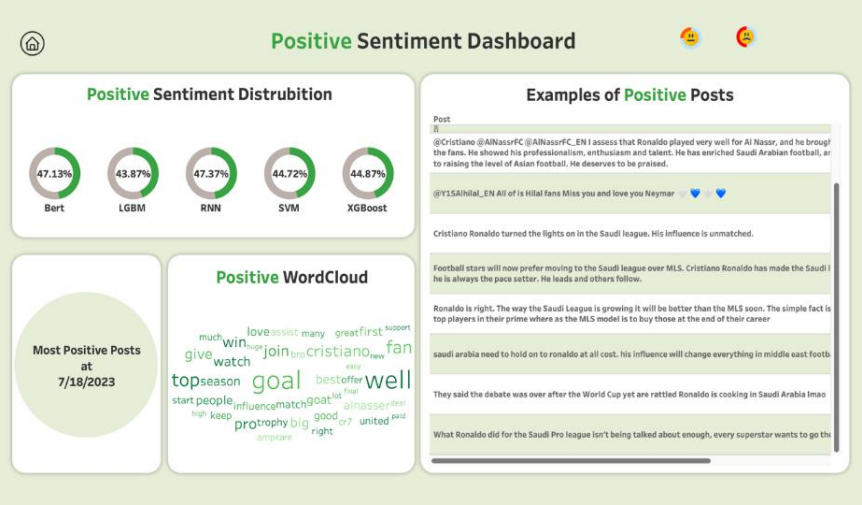


Figure 5–19 Positive sentiment dashboard

- **Neutral sentiment dashboard**

The Figure 5-20 shows the Neutral sentiment dashboard that contain the all the widgets that were customized to the Neutral sentiment. For example, it has a word cloud for the Neutral words only and a new widget that show a sample of the Neutral words from the dataset.

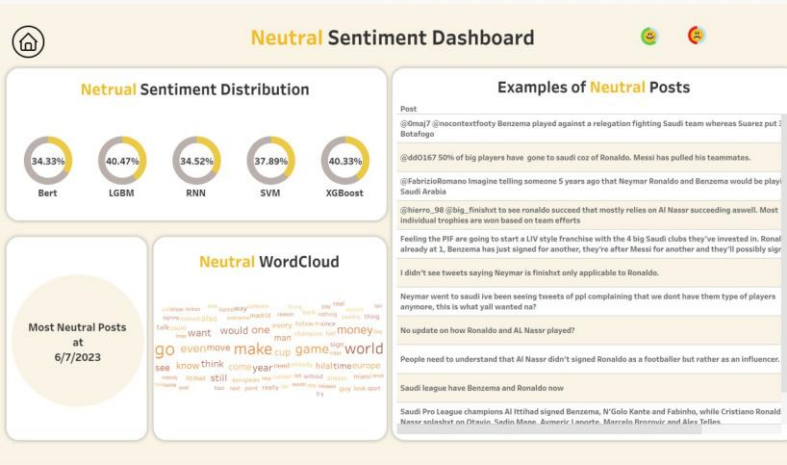


Figure 5–20 Neutral sentiment dashboard

- **Negative sentiment dashboard**

The Figure 5-21 shows the Negative sentiment dashboard that contain the all the widgets that were customized to the Negative sentiment. For example, it has a word cloud for the Neutral words only and a new widget that show a sample of the Negative words from the dataset.

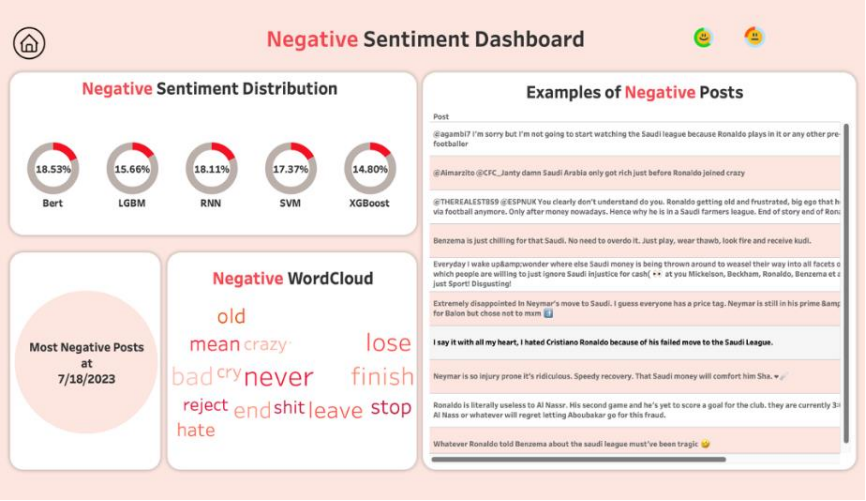


Figure 5–21 Negative sentiment dashboard

5.4 Conclusion

This chapter describes the development of the Sentiment Analysis Dashboard, including the structure and implementation of each deep learning and machine learning algorithms, as well as an I/O screen. Table 5-23 displays the results of all algorithm implementations, with BERT producing the highest accuracy compared to the other algorithms.

Algorithm	Accuracy	Sentiment percentage		
		Positive	Neutral	Negative
BERT	97.27%	47.13%	34.33%	18.53%
RNN	95.60%	47.37%	34.52%	18.11%
XGBoost	90.26%	44.87%	40.33%	14.80%
SVM	88.9%	44.73%	37.89%	17.38%
LGBM	88.81%	43.87%	40.47%	15.66%

Table 5–23 The results of all algorithm implementations

Chapter 6: Testing

6 Testing

Testing involves verifying the functionality of a project by identifying and Correcting errors to guarantee the quality, reliability, and functionality of the final product. This chapter aims to summarize the testing activities conducted and present the findings obtained throughout the testing process.

6.1 Test plan

Testing begins with assessing the dashboard's functionality to ensure its proper operation. Following this, proceed to evaluate the performance of the algorithms using multiple evaluation matrices such as the Confusion matrix, ROC/AUC, F-1 score, Precision, and Recall, among others, to comprehensively gauge their effectiveness (how well they achieve the desired results) and efficiency (how resource-efficient they are). Subsequently, acceptance testing will be employed to analyze user feedback and usability aspects, aiming to optimize the overall user experience.

The testing scope for this project comprises two main areas. Firstly, it involves testing the dashboard functions through functional and user acceptance tests. Secondly, evaluating the performance of the algorithms using evaluation methods.

The following testing types have been considered and performed:

- **Functional testing**

  Focuses on examining the various features and capabilities of the software to ensure that each function behaves as expected and meets the functional requirements [75].

- **Acceptance testing**

  Is conducted by end-users to confirm and approve the software system before transitioning it to the production environment. It occurs in the final testing phase after functional testing [76].

- **Algorithm Performance Evaluation**
  - Confusion Matrix

The *confusion matrix* is a valuable tool for assessing the accuracy of classification models. It depicts a model's correct and incorrect predictions for classification tasks. The matrix is a N × N grid comprising four elements, where N is the total number of target classes.

True positives (TP): The model predicted a positive value, while the actual value was positive.

True negatives (TN): The model predicted a negative value, but the actual value was negative.

False positives (FP) are Type 1 errors, in which the prediction is both positive and false.

False negatives (FN) are Type 2 errors, in which the prediction is negative and false [77].

- ROC/AUC

ROC (Receiver Operating Characteristic) curve and *AUC* (Area Under the Curve) are tools used to evaluate the performance of classification models. The ROC curve plots the true positive rate (sensitivity) against the false positive rate (1 - specificity) at various threshold settings, showing the trade-off between sensitivity and specificity. AUC represents the area under the ROC curve, with a higher AUC indicating better model performance [78].

- Precision

Precision is a helpful metric when FP is more concerning than FN. To assess the model's dependability, it counts the number of accurately predicted positive patterns out of all the expected patterns in a positive class [79].

The Figure 6-1 below shows how Precision measures the positive patterns.

$$Precision = \frac{True\ Positive}{True\ Positive\ +\ False\ Positive}$$

Figure 6–1 Precision equation

- Recall

Recall is a metric that tells how many actual positive cases were correctly predicted by the model, making it useful for cases where FN is more critical than FP. The Recall metric measures the model's ability to correctly identify all positive units in the dataset [79], as illustrated in the Figure 6-2 below.

$$Recall = \frac{True\ Positive}{True\ Positive\ +\ False\ Negative}$$

Figure 6–2 Recall equation

- F-1 score

The *F1-score* is indeed a valuable metric for evaluating classification models. It strikes a balance between precision and recall, making it useful for assessing a model's overall performance in correctly identifying instances across classes [80]. As illustrated in the Figure 6-3 below

$$F-1\ score = 2\ x\ \frac{Precision\ x\ Recall}{Precision\ +Recall}$$

Figure 6–3 F-1 score equation

Schedules of test activities Testing tasks.

In table 6-1, the schedule for testing tasks for the project is placed in terms of 2 weeks (14 days).

Testing type	Duration
Functional testing	
User functional requirements: • View the dashboard • Navigate between different dashboards • Filter result • View data details • Add comments • Downloads the dashboard Admin functional requirements: • Adds data source to the dashboard • Adds dashboard component, filter, and widgets • Customizes dashboard component, filter, and widgets • Deletes dashboard components, filters, and widgets • Navigates between different dashboards • Adds comment to the dashboard • Downloads the dashboard	5 days
Acceptance testing	
Form Setup: • Set up the testing environment. • Define acceptance criteria based on user stories and specifications. • Design the form with relevant fields, validation, and submission. Feedback Collection: • Test the form to check functionality, data validation, and error handling. • Document test results, issues, and areas for improvement. • Gather user feedback on usability, instructions, and ease of use. • Record user suggestions and enhancement requests.	4 days
Algorithm Performance Evaluation	
Conduct various evaluations: • Confusion Matrix analysis. • ROC/AUC (Receiver Operating Characteristic / Area Under the Curve) analysis. • Precision measurement. • Recall measurement. • Calculate the F-1 score, the harmonic mean of precision and recall.	5 days

Table 6–1 Schedule for testing tasks

6.2 Test Cases

In this section, key functionalities will be tested through functional testing. Following that, user feedback from user acceptance testing will be shared. Finally, an assessment will be made on how well the algorithms perform.

6.2.1 Functional testing

This section assesses the implementation of functional requirements outlined in Chapter 3. The test case covers test conditions, actions, expected results, and action result for two primary user roles: user and admin.

User functional requirements:

- View the dashboard

Test condition	Action	Expected result	Action result
The user is invited by the admin and has the privilege of accessing the dashboard.	The user opens the dashboard URL.	The dashboard loads without errors and displays all data and visualization.	Pass
The user is invited by the admin and doesn't have the privilege of accessing the dashboard.	The user opens the dashboard URL.	The dashboard displays an error message indicating that the user must have the right privilege.	

Table 6–2 View the dashboard

- Navigate between different dashboards

Test condition	Action	Expected result	Action result
The dashboard contains buttons for navigation.	The user clicks on the navigation button.	Switch to the selected dashboard without errors.	Pass

Table 6–3 Navigate between different dashboards

- Filter result

Test condition	Action	Expected result	Action result
The dashboard contains a filter option.	The user applies a filter on certain criteria (i.e., Date range)	Visualizations update dynamically based on the applied filters, showing only the relevant data.	Pass

Table 6–4 Filter result

- View data details

Test condition	Action	Expected result	Action result
Data source exists.	The user clicks on the show data source.	The data used to create the dashboard is displayed.	Pass

Table 6–5 View data details

- Add comments

Test condition	Action	Expected result	Action result
Add comment feature.	The user clicks on the add comment button.	A text box is open so the user can input comments.	Pass

Table 6–6 Add comments

- Downloads the dashboard

Test condition	Action	Expected result	Action result
Downloads dashboard feature.	The user clicks on the downloads button.	A prompt allows the user to select the format, e.g., PDF, and the dashboard is downloaded successfully.	Pass

Table 6–7 Downloads the dashboard

Admin functional requirements:

- Adds data source to the dashboard

Test condition	Action	Expected result	Action result
Add data source feature	Admin clicks on "new data source" icon to upload an excel or csv file	New file is added to the dashboard as a data source	Pass

Table 6–8 Adds data source to the dashboard

- Adds dashboard component, filter, and widgets

Test condition	Action	Expected result	Action result
Add dashboard component, filter, or widget feature	Admin drags the desired column from the "data" section to be added to the dashboard as a component or widget then, adds a filter to it	New component is added to be presented on the dashboard	Pass

Table 6–9 Adds dashboard component, filter, and widgets

- Customizes dashboard component, filter, and widgets

Test condition	Action	Expected result	Action result
customize dashboard component, filter, or widget feature	Admin can change the appearance of the dashboard component (map, pie, bar…etc), color, size, font, add a tooltip to it from the "Marks" section	The dashboard component is customized based on the changes done by the admin	Pass

Table 6–10 Customizes dashboard component, filter, and widgets

- Deletes dashboard components, filters, and widgets

Test condition	Action	Expected result	Action result
delete dashboard component, filter, or widget feature	Admin clicks on the dashboard component to be deleted then, clicks on "Remove from dashboard" icon	The dashboard component is no longer presented in the dashboard	Pass

Table 6–11 Deletes dashboard components, filters, and widgets

- Navigates between different dashboards

Test condition	Action	Expected result	Action result
The dashboard contains buttons for navigation.	Admin clicks on the navigation button.	Switch to the selected dashboard without errors.	Pass

Table 6–12 Admin: Navigates between different dashboards

- Adds comment to the dashboard

Test condition	Action	Expected result	Action result
Add comment feature.	Admin clicks on the add comment button.	A text box is open so the user can input comments.	Pass

Table 6–13 Admin: Adds comment to the dashboard

- Downloads the dashboard

Test condition	Action	Expected result	Action result
Downloads dashboard feature.	The user clicks on the downloads button.	A prompt allows the user to select the format, e.g., PDF, and the dashboard is downloaded successfully.	Pass

Table 6–14 Admin: Downloads the dashboard

6.2.2 Acceptance Testing and Result

In order to ensure the effectiveness and usability of the dashboard, acceptance testing session was conducted with 10 participants. The primary objective of the testing was to gather valuable feedback on different aspects of the dashboard, including clarity of information, navigation, ease of use, functionality, and overall satisfaction.

Each participant was provided with access to the dashboard as a user and asked to perform a different tasks and then answer 8 questions, the Table 6-15 and Table 6-16 summarize the results of the acceptance testing.

Question	Result
Q1: Did you find it easy to navigate through different sections of the dashboards?	
Q2: On a scale of 1 to 5, how clear was the presented information on the dashboard?	
Q3: On a scale of 1 to 5, how would you rate the clarity of the colors used in the sentiment analysis visualizations to represent different sentiment categories (positive, negative, neutral)?	
Q4: Did you find these interactive elements (Filters) helpful in exploring the data?	

Table 6–15 Acceptance Testing and Result

Acceptance Testing and Result (cont.)

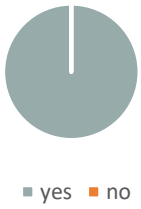
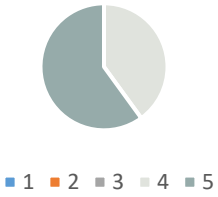
Question	Result
Q5: Were you able to perform all expected tasks without encountering errors?	
Q6: On a scale of 5 to 1, how satisfied are you with the dashboard overall?	
Q7: Are there any features or functionalities you think the dashboard is lacking?	All 9 percipients answered (No), except #P10. #P10 Answer: Adding more filter options. Detailed answers are shown in appendix B.
Q8: Finally, do you have any additional comments, suggestions, or feedback you would like to share about the sentiment analysis dashboard?	All participants agreed that the dashboard is simple, informative, and no suggestions were added.

Table 6–16 Acceptance Testing and Result (cont.)

The feedback gathered from the 10 participants provided valuable insights into the strengths and weaknesses of the dashboard that could be considered now or in the future plans of the project. Participants highlighted areas where the dashboard excelled, such as the clarity of colors and the ease of navigation. They also identified areas for improvement, including suggestions for enhancing certain features and making the interface more user friendly.

6.2.3 Algorithm Performance Evaluation

This section displays the results of algorithms performance evaluation metrics for deep and machine learning algorithms.

- **Confusion Matrix**

In the *confusion matrix*, rows indicate actual labels, while columns show predicted labels. Diagonal elements are correct predictions, while off-diagonal elements are incorrect predictions.

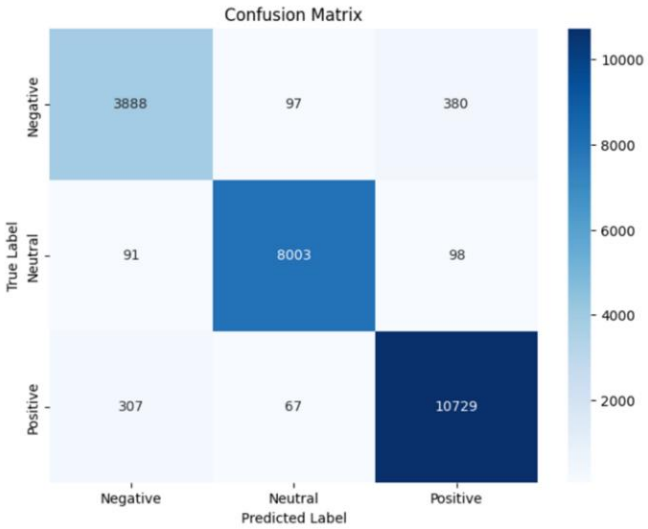


Figure 6–4 RNN Confusion matrix result

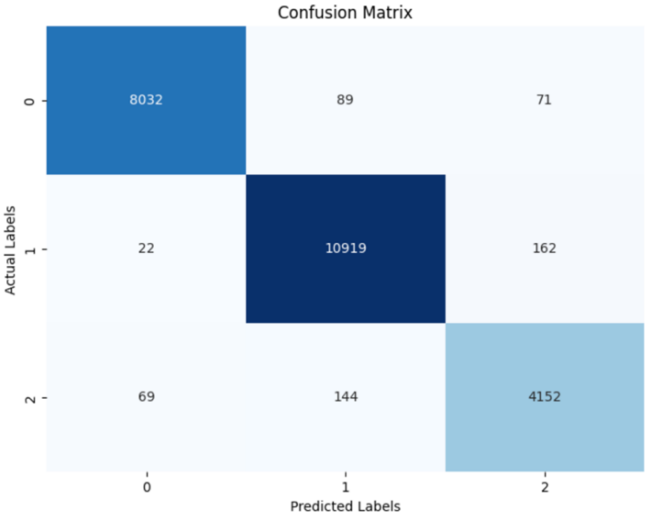


Figure 6–5 BERT Confusion matrix result

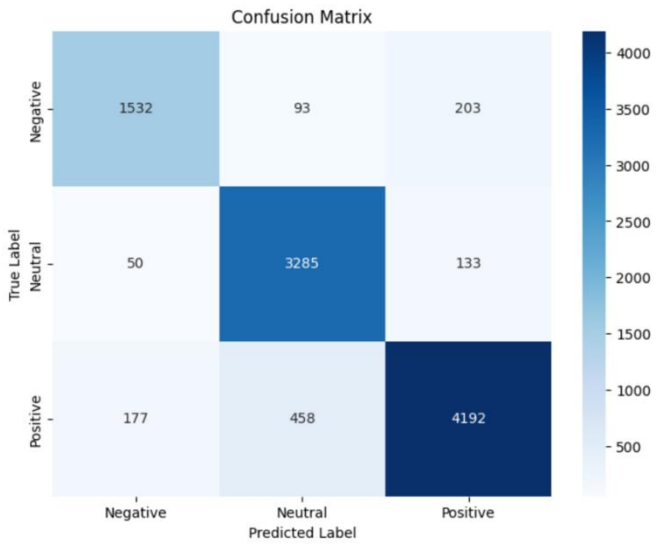


Figure 6–6 SVM Confusion matrix result

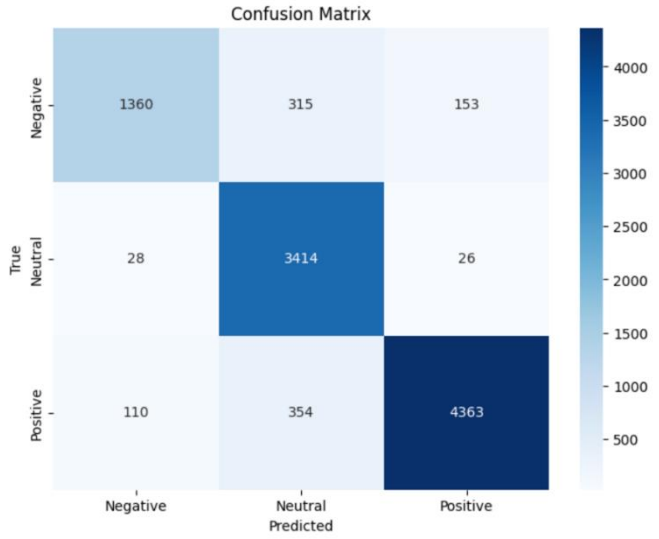


Figure 6–7 XGBoost Confusion matrix result

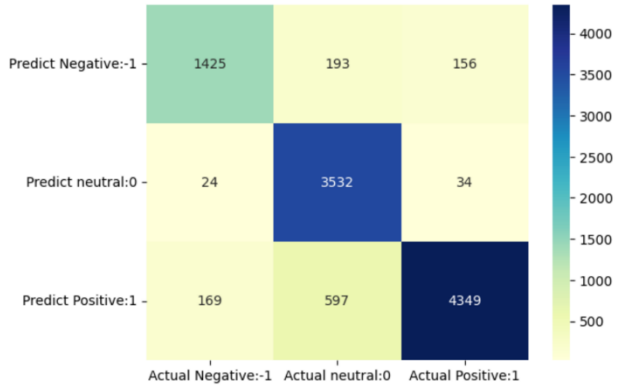


Figure 6–8 LGBM Confusion matrix result

Accuracy

The measure of overall model correctness is accuracy, the fraction of correctly predicted observations out of the total observations. The calculation of *Accuracy* for each algorithm:

BERT: $\frac{True\ positive}{Total\ predictions} = \frac{23003}{23660} \approx 0.97$

RNN: $\frac{True\ positive}{Total\ predictions} = \frac{22620}{23660} \approx 0.95$

XGBoost: $\frac{True\ positive}{Total\ predictions} = \frac{9137}{10123} \approx 0.90$

SVM: $\frac{True\ positive}{Total\ predictions} = \frac{9009}{10123} \approx 0.89$

LGBM: $\frac{True\ positive}{Total\ predictions} = \frac{9306}{10479} \approx 0.88$

- **ROC/AUC**

The *AUC* value represents the overall performance of a classifier. The *ROC* curve shows how TPR and FPR change across different discrimination thresholds. The *ROC/AUC* for each algorithm is listed below:

BERT: Based on the model's attained Ovr *ROC/AUC* score of 1. This score shows how well the BERT model separates each class from the others. As a result, the model performs task classification quite effectively.

RNN: The Ovr *ROC/AUC* value obtained was 0.98, indicating discriminatory solid power in distinguishing between classes. This suggests that the model performs well in classification tasks.

XGBoost: Given that the Ovr *ROC/AUC* value attained was 0.97, the model performs well in classification tasks. This number shows that discrimination has a strong ability to separate classes. Thus, it is possible to conclude that the model performs well in classification tasks.

SVM: Based on the Ovr *ROC/AUC* value of 0.96 achieved by the model, it can be inferred that the model performs well in classification tasks.

LGBM: The model performs well in classification tasks, as evidenced by the obtained Ovr *ROC/AUC* value of 0.95. Therefore, the model does an excellent job of task classification.

- **Precision**

Precision is the proportion of true positives to all positives predicted, indicating the accuracy of positive predictions. The calculation of *Precision* for each algorithm:

BERT: Class Negative = $\frac{8032}{8032+22+69} \approx 0.99$, Class Neutral = $\frac{10919}{10919+89+144} \approx 0.98$, Class Positive = $\frac{4152}{4152+71+162} \approx 0.95$

RNN: Class Negative = $\frac{3888}{3888+91+307} \approx 0.91$, Class Neutral = $\frac{8003}{8003+97+67} \approx 0.98$, Class Positive = $\frac{10729}{10729+98+380} \approx 0.96$

XGBoost: Class Negative = $\frac{1360}{1360+28+110} \approx 0.89$, Class Neutral = $\frac{3414}{3414+315+354} \approx 0.83$, Class Positive = $\frac{4363}{4363+26+153} \approx 0.95$

SVM: Class Negative = $\frac{1532}{1532+50+177} \approx 0.87$, Class Neutral = $\frac{3285}{3285+93+458} \approx 0.86$, Class Positive = $\frac{4192}{4192+133+233} \approx 0.93$

LGBM: Class Negative = $\frac{1425}{1425+24+169} \approx 0.88$, Class Neutral = $\frac{3532}{3532+193+597} \approx 0.82$, Class Positive = $\frac{4349}{4349+34+156} \approx 0.96$

- **Recall**

The proportion of true positive predictions to the total actual positives is known as recall. The calculation of *Recall* for each algorithm:

BERT: Class Negative = $\frac{8032}{8032+89+71} \approx 0.98$, Class Neutral = $\frac{10919}{10919+22+162} \approx 0.98$, Class Positive = $\frac{4152}{4152+69+144} \approx 0.95$

RNN: Class Negative = $\frac{3888}{3888+91+307} \approx 0.91$, Class Neutral = $\frac{8003}{8003+97+67} \approx 0.98$, Class Positive = $\frac{10729}{10729+98+380} \approx 0.96$

XGBoost: Class Negative = $\frac{1360}{1360+153+315} \approx 0.74$, Class Neutral = $\frac{3414}{3414+28+26} \approx 0.97$, Class Positive = $\frac{4363}{4363+110+354} \approx 0.89$

SVM: Class Negative = $\frac{1532}{1532+50+177} \approx 0.87$, Class Neutral = $\frac{3285}{3285+93+458} \approx 0.86$, Class Positive = $\frac{4192}{4192+133+233} \approx 0.93$

LGBM: Class Negative = $\frac{1425}{1425+156+193} \approx 0.80$, Class Neutral = $\frac{3532}{3532+24+34} \approx 0.98$, Class Positive = $\frac{4349}{4349+196+597} \approx 0.85$

- **F-1 Score**

The *F-1 Score* is a measure that combines the *Precision* and *Recall* values, taking into account their relative importance. It is designed to provide a balanced view of both *Precision* and *Recall*. The calculation of *F-1 Score* for each algorithm:

BERT: Class Negative = 2 x $\frac{0.99 \, x \, 0.98}{0.99+0.98} \approx 0.98$, Class Neutral = 2 x $\frac{0.98 \, x \, 0.98}{0.98+0.98} \approx 0.98$, Class Positive = 2 x $\frac{0.95 \, x \, 0.95}{0.95+0.95} \approx 0.95$

RNN: Class Negative = 2 x $\frac{0.91 \, x \, 0.89}{0.91+0.89} \approx 0.90$, Class Neutral = 2 x $\frac{0.98 \, x \, 0.98}{0.98+0.98} \approx 0.98$, Class Positive = 2 x $\frac{0.96 \, x \, 0.97}{0.96+0.97} \approx 0.96$

XGBoost: Class Negative = 2 x $\frac{0.89 \, x \, 0.74}{0.89+0.74} \approx 0.81$, Class Neutral = 2 x $\frac{0.83 \, x \, 0.97}{0.83+0.97} \approx 0.90$, Class Positive = 2 x $\frac{0.95 \, x \, 0.89}{0.95+0.89} \approx 0.92$

SVM: Class Negative = 2 x $\frac{0.87 \, x \, 0.84}{0.87+0.84} \approx 0.85$, Class Neutral = 2 x $\frac{0.86 \, x \, 0.96}{0.86+0.96} \approx 0.89$, Class Positive = 2 x $\frac{0.93 \, x \, 0.87}{0.93+0.87} \approx 0.89$

LGBM: Class Negative = 2 x $\frac{0.88 \, x \, 0.80}{0.88+0.80} \approx 0.83$, Class Neutral = 2 x $\frac{0.98 \, x \, 0.98}{0.98+0.98} \approx 0.89$, Class Positive = 2 x $\frac{0.96 \, x \, 0.97}{0.96+0.97} \approx 0.90$

Algorithm	BERT	RNN	XGBoost	SVM	LGBM
Accuracy	97.65%	95.60%	90.26%	88.9%	88.81%
ROC/AUC	1	0.98	0.97	0.96	0.95
Precision	0.98	0.96	0.91	0.89	0.89
Recall	0.98	0.96	0.90	0.89	0.88
F-1 Score	0.98	0.96	0.90	0.89	0.88

Figure 6–9 Algorithms performance evaluation results

In terms of evaluating algorithm performance results in the Table 6-9 above, Bidirectional Encoder Representations from Transformers (BERT) displayed superior results when compared to all other algorithms.

6.3 Conclusion

In this chapter represents a significant step forward in our project's development. Because conducting different type of testing offered an opportunity to validate the project in different domains and to address the potential issues and the areas of improvement.

Chapter 7: Conclusion

7 Conclusion

In this chapter, the last segment of the study will be defined which summarizes the success of the previous chapters where a sentiment classification of the public perception regarding attrackting football superstar players to Saudi league is now cleary provided using a varity of DL and ML algorithms known for high accuricies in text-based sentiment analysis. The classification is also clearified with a geographical distribution, word cloud that show the most frequent words used, and most frequent dates where people were posting regarding football superstars coming to Saudi league.

7.1 Evaluation

This project achieved successful implementation by utilizing Deep Learning (DL) algorithms and Machine Learning (ML). The performance of these algorithms was evaluated using various metrics, including a confusion matrix, F1 score, recall, precision, and ROC/AUC. The evaluation results are summarized in Table 7-1, providing insights into the effectiveness and efficiency of the employed algorithms in classifying public perception opinions regarding the attraction of football superstar players into the Saudi league.

Algorhim type	Algorithm	Accuracy	Sentiment percentage			Evaluation methods			
			Positive	Neutral	Negative	F1 score	Recall	Precision	ROC/ AUC
DL algorithms	BERT	97.27%	47.13%	34.33%	18.53%	0.98	0.98	0.98	1
	RNN	95.60%	47.37%	34.52%	18.11%	0.96	0.96	0.96	0.98
ML algorithms	XGBoost	90.26%	44.87%	40.33%	14.80%	0.90	0.90	0.91	0.98
	SVM	88.9%	44.73%	37.89%	17.38%	0.89	0.89	0.89	0.96
	LGBM	88.81%	43.87%	40.47%	15.66%	0.88	0.88	0.89	0.95

Table 7–1 Algorthims accuracy and evaluation results

7.2 Future work

As we move forward with our project, our vision include different plans that are made to cover the limitations that were addressed and greatly improve the project's capabilities in a variety of areas. The main future goals of the project include:

- Since our project only cover text-based sentiment analysis, integrating image and video analysis alongside can provide a more comprehensive understanding of public sentiment.
- Analyze other platforms, such as Facebook and Instagram, with X's platform.
- Implementing real-time data processing capabilities to analyze incoming data streams as they occur.

References

[1] N. Turak, "Cristiano Ronaldo signs with Saudi Arabian soccer club Al Nassr for reported record-breaking salary," CNBC, 30 12 2022. [Online]. Available: https://www.cnbc.com/2022/12/30/cristiano-ronaldo-signs-with-saudi-arabian-club-al-nassr-for-reported-record-breaking-salary.html. [Accessed 23 10 2023].

[2] N. Turak, "Saudi soccer league lures Brazilian star Neymar as it seeks to attract ‘exceptional players only’," CNBC, 14 8 2023. [Online]. Available: https://www.cnbc.com/2023/08/14/saudi-soccer-team-al-hilal-lures-brazilian-star-neymar.html. [Accessed 23 10 2023].

[3] "Karim Benzema reportedly signs a deal worth more than $200 million with Saudi Arabia’s Al-Ittihad," CNBC, 7 6 2020. [Online]. Available: https://www.cnbc.com/video/2023/06/07/karim-benzema-reportedly-signs-a-deal-with-saudi-arabias-al-ittihad.html#:~:text=Karim%20Benzema%20reportedly%20signed%20a,CNBC%27s%20Dan%20Murphy%20reports.. [Accessed 23 10 2023].

[4] "Football and business in Saudi Arabia: An era of unlimited potential," [Online]. Available: https://www.arabnews.com/node/2337711/sport.

[5] "Social Media in Sports: Driving Fan Engagement," [Online]. Available: https://www.greenfly.com/blog/social-media-in-sports/.

[6] M. Oldfield, Cristiano Ronaldo, -: Kings Road Publishin, 2009.

[7] M. Gitlin, Neymar: Soccer Superstar, -: The Rosen Publishing Group, Inc, 2018.

[8] R. Smith, "Karim Benzema: Real Madrid's Low-Wattage Galactico," International Herald Tribune, New York, 2020.

[9] "vision 2023 kingdom of saudi arabia," - - 2016 - 2020. [Online]. Available: https://www.vision2030.gov.sa/en/vision-2030/overview/. [Accessed 23 10 2023].

[10] M. j oshi, P. Prajapati , A. Shaikh and V. Vala, A Survey on Sentiment Analysis, India: ResearchGate, 2017.

[11] A. Agarwal, B. Xie, I. Vovsha, O. Rambow and R. Passonneau, Sentiment Analyysis of twitter data, New York: ACL Anthology, 2011.

[12] J. S. V. a. J. F. George, Modern System Analysis and Design, New Jersey: Pearson Education, Inc, 2017.

[13] "The Quality of Life Program Vision 2030," 1 1 2018. [Online]. Available: https://www.vision2030.gov.sa/media/gi3l3js2/qol-en.pdf. [Accessed 3 9 2023].

[14] M. Wankhade, A. C. Sekhara Rao and C. Kulkarni, A survey on sentiment analysis methods, applications, and challengesA survey on sentiment analysis methods, applications, and challenges, India: springer, 2022.

[15] M. R. Minar and J. Naher, Recent Advances in Deep Learning: An Overview, Bangladesh: ArXiv, 2018.

[16] F. Q. Lauzon, An introduction to deep learning, Montreal, QC, Canada: IEEE Xplore, 2012.

[17] S. POUYANFAR, S. S. SADIQ , Y. YAN, H. TIAN, Y. TAO, M. P. REYES and M.-L. SHYU, A Survey on Deep Learning: Algorithms, Techniques, and Applications, Florida: ACM Digital Library, 2018.

[18] B. Muller, "BERT 101 State Of The Art NLP Model Explained," Hugging face, 02 03 2022. [Online]. Available: https://huggingface.co/blog/bert-101#3-bert-model-size--architecture. [Accessed 07 04 2024].

[19] M.-W. C. K. L. K. T. Jacob Devlin, BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, Minneapolis: Association for Computational Linguistics, 2019.

[20] D. K. R. M.P. Geetha a, Improving the performance of aspect based sentiment analysis using, china: Elsevier B.V. on behalf of KeAi Communications Co., Ltd. , 2021.

[21] J. Alzubi, A. Nayyar and A. Kumar, Machine Learning from Theory to Algorithms: An Overview, India: IOP Science , 2018.

[22] B. Mahesh, Machine Learning Algorithms -A Review, ResearchGate: -, 2019.

[23] A. R. I. N. H. a. L. A. Styawati, Comparison of Support Vector Machine and Naïve Bayes on Twitter Data Sentiment Analysis, Bandar Lampung: Journal of Informatics, Journal of IT development, 2021.

[24] A. Shmilovici, Support Vector Machines, -: Springer, 2010.

[25] B. Quinto, next generation machine learning with spark, Carson, A, USA: Apress, 2020.

[26] Available: https://neptune.ai/blog/xgboost-vs-lightgbm#:~:text=In%20XGBoost%2C%20trees%20grow%20depth,documentation%20and%20resolutions%20to%20issues. [Accessed 22 1 2024].

[27] Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=9099543.

[28] A. Sadia, F. Khan and F. Bashir, An Overview of Lexicon-Based Approach For Sentiment Analysis, Pakistan: NED University of Engineering & Technology , 2018.

[29] R. Akbari and S. A. Saadat Neshan, A Combination of Machine Learning and Lexicon Based Techniques for Sentiment Analysis, Iran: IEEE, 2020.

[30] Y. Tian, L. Yang , Y. Sun and D. Liu, Cross-Domain End-To-End Aspect-Based Sentiment Analysis with Domain-Dependent Embeddings, China: Hindawi, 2021.

[31] D. B. A. Syarafina Dewi, TWITTER SENTIMENT ANALYSIS TOWARDS QATAR AS HOST OF THE 2022 WORLD CUP USING TEXTBLOB, Jakarta: Journal of Social Research, 2023.

[32] A. Alhadlaq and A. Alnuaim, A Twitter-Based Comparative Analysis of Emotions and Sentiments of Arab and Hispanic Football Fans, Riyadh: MDPI, 2023.

[33] N. J. Almateg , S. M. BinQasim, J. N. Alshahrani, A. Y. Marghalani and Z. H. Alharbi, Sentiment Analysis of Opinions over Time Toward Saudi Women's Sports, Unitited Kingdom: Springer, 2020, pp. 247-259.

[34] M. R. Z. B. M. P. N. G. Nuno Pombo, Computerised Sentiment Analysis on Social Networks. Two Case Studies: FIFA World Cup 2018 and Cristiano Ronaldo Joining Juventus, Springer International Publishing, 2021.

[35] T. R. Y. A. F. N. W. G. R Ardianto, SENTIMENT ANALYSIS ON E-SPORTS FOR EDUCATION CURRICULUM USING NAIVE BAYES AND SUPPORT VECTOR MACHINE, Jakarta: Jurnal Ilmu Komputer dan Informasi journal , 2020.

[36] R. P. a. K. Passi, Sentiment Analysis on Twitter Data of World Cup Soccer Tournament Using Machine Learning, Canada, 2020.

[37] A. S. Aloufi, sentiment identification in football-specific tweets, 2018.

[38] V. R. David Robinson, The A–Z of Social Research Jargon, Routledge, 2019.

[39] K. Schwalbe, Information Technology Project Managament, Boston: Cengage Learning, 2015.

[40] J. S. Valacich, J. F. George and J. A. Hoffer, Essentials of systems analysis and design, United states: Pearson, 2015.

[41] J. Rasmussen, "Structured Analysis," ScienceDirect, - - 2003. [Online]. Available: https://www.sciencedirect.com/topics/computer-science/structured-analysis. [Accessed 15 11 2023].

[42] S. S. Alhir, Learning UML, O'Reilly Media, 2003.

[43] J. S. Valacich and J. F. George, What is a Data Flow Diagram, United state: Pearson, 2016.

[44] R. E. a. S. B. Navathe, FUNDAMENTALS OF Database Systems 7th edition, PEARSON, 2015.

[45] D. Wall, Multi-Tier Application Programming with PHP Practical Guide for Architects and Programmers, Morgan Kaufmann, 2004.

[46] E. Bressert, SciPy and NumPy: An Overview for Developers, United States: O'Reilly Media, Inc, 2013.

[47] D. Y. Chen, Pandas for Everyone: Python Data Analysis, Addison-Wesley Professional, 2017.

[48] "1," Python Software Foundation, Available: https://docs.python.org/3/library/csv.html. [Accessed 06 04 2024].

[49] "1," Python HTTP, 22 05 2023. Available: https://pypi.org/project/requests . [Accessed 06 04 2024].

[50] "1," Python Software Foundation, Available: https://docs.python.org/3/library/datetime.html. [Accessed 06 04 2024].

[51] N. L. Toolkit. Available: https://www.nltk.org/.

[52] https://docs.python.org/3/library/re.html.

[53] https://docs.python.org/3/library/pickle.html.

[54] https://bedford-computing.co.uk/learning/wp-content/uploads/2015/10/Python-for-Data-Analysis.pdf.

[55] https://www.geeksforgeeks.org/learning-model-building-scikit-learn-python-machine-learning-library/.

[56] S. Loria, "textblob Documentation," 2018.

[57] "Application of the Machine Learning LightGBM Model to the Prediction of the Water Levels of the Lower Columbia River," Available: https://www.mdpi.com/2077-1312/9/5/496.

[58] F. CHOLLET, Deep Learning with Python, SHELTER ISLAND: Manning Publications, 2018.

[59] "1," Python Software Foundation, 28 05 2023. Available: https://pypi.org/project/torch/#more-about-pytorch. [Accessed 06 04 2024].

[60] "1," Hugging face company, Available: https://huggingface.co/docs/transformers/index. [Accessed 06 04 2024].

[61] "An introduction to seaborn," Available: seaborn.pydata.org/tutorial/introduction.html. [Accessed 7 4 2024].

[62] "Introduction to Matplotlib," Available: www.geeksforgeeks.org/python-introduction-matplotlib/. [Accessed 7 4 2024].

[63] "1," SocialData.Tools, Available: https://socialdata.gitbook.io/docs . [Accessed 06 04 2024].

[64] "1," X Developer platform, Available: https://developer.twitter.com/en/docs/twitter-api/tweets/counts/api-reference/get-tweets-counts-recent#:~:text=YYYY-MM-DDTHH%3Amm,first%20second%20of%20the%20minute. [Accessed 16 04 2024].

[65] "PRE-PROCESSING TECHNIQUES," Available: https://dergipark.org.tr/en/download/article-file/2703792.

[66] I. Goodfellow, Y. Bengio and A. Courville, Deep Learning, -: MIT Press, 2015.

[67] R. M. Schmidt, Recurrent Neural Networks (RNNs): A gentle Introduction and Overview, Tübingen, Germany: arXiv, 2019.

[68] Rendyk, "1," Analytics vidhya, 12 01 2024. Available: https://www.analyticsvidhya.com/blog/2021/05/tuning-the-hyperparameters-and-layers-of-neural-network-deep-learning/. [Accessed 16 04 2024].

[69] N. S. N. P. J. U. L. J. A. N. G. Ł. K. I. P. Ashish Vaswani, Attention Is All You Need, Long Beach: Google, 2017.

[70] "Multi-Class Neural Networks: Softmax," Google cloud, Available: https://developers.google.com/machine-learning/crash-course/multi-class-neural-networks/softmax. [Accessed 07 04 2024].

[71] S. Mohammadi and M. Chapon, Investigating the Performance of Fine-tuned Text Classification Models Based-on Bert, Yanuca Island: IEEE 6th International Conference on Data Science and Systems, 2020.

[72] A. B. S. K. V. C. S. W. Vaibhav Kadam, "Enhancing Surface Fault Detection Using Machine Learning for 3D Printed Products," 5 2021. [Online]. Available: www.researchgate.net/publication/351590398_Enhancing_Surface_Fault_Detection_Using_Machine_Learning_for_3D_Printed_Products#pf8. [Accessed 7 4 2024].

[73] "Max Features," Available: https://help.monkeylearn.com/en/articles/2174109-max-features.

[74] https://www.researchgate.net/profile/Jayme-Barbedo/publication/326957145_Impact_of_dataset_size_and_variety_on_the_effectiveness_of_deep_learning_and_transfer_learning_for_plant_disease_classification/links/5bb4cc56a6fdccd3cb84f98f/Impact-of-dataset-size-.

[75] https://www.opentext.com/what-is/functional-testing#:~:text=Functional%20testing%20is%20a%20type,with%20the%20end%20user's%20expectations..

[76] https://www.guru99.com/user-acceptance-testing.html.

[77] https://www.researchgate.net/publication/355096788_Confusion_Matrix.

[78] https://link.springer.com/article/10.1186/s13040-023-00322-4#Sec6.

[79] https://arxiv.org/pdf/2008.05756.pdf.

[80] https://link.springer.com/chapter/10.1007/978-3-662-44851-9_15.

[81] P. Singh, Machine Learning with PySpark, Bangalore: Pramod Singh, 2022.

[82] K. F. Chen, Offline and Online SVM Performance Analysis, Cambridge: Massachusetts Institute of Technology, 2007.

[83] J. Cervantes, F. Garcia-Lamont , L. Rodríguez-Mazahua and A. Lopez, A comprehensive survey on support vector machine classification: Applications, challenges and trends, Mexico: Science direct, 2020.

[84] V. N. Vanpik, Statistical Learning Theory, -: Wiley, 1998.

[85] Z. Wang and . X. Xue, "Multi-Class Support Vector Machine," in *Support Vector Machine Applications*, USA, Springer, Cham, 2014, pp. 23-48.

[86] J. B. Karl Wiegers, 2013. [Online]. Available: https://www.dirzon.com/file/telegram/HiLCoE%20-

%20DRB1802/Microsoft%20Press%20Software%20Requirements%203%203rd%20Edition%20Aug%202013.pdf. [Accessed 2023].

[87] R. Hoev, "1," 10 November 2018. Available: https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270. [Accessed 29 3 2024].

[88] W. Wang, G. Chakraborty and B. Chakraborty, Predicting the Risk of Chronic Kidney Disease (CKD) Using Machine Learning Algorithm, Japan: ResearchGate, 2020.

Appendix

Appendix A: Usecase high-level description

Use case name	Add comment
Actor	Admin, user
Description	The use case begins when the admin or the user click the add comment button to add a comment in the comments section.

Table 0–1 Add comment

Use case name	Navigate dashboards
Actor	Admin, user
Description	The use case begins when the admin or the user click the navigate buttons to navigate between different dashboards and dashboard components.

Table 0–2 Navigate dashboards

Use case name	Filter Results
Actor	User
Description	The use case begins when the user clicks the filter buttons on the dashboard to filter the presented results.

Table 0–3 Filter Results

Use case name	View data details
Actor	User
Description	The use case begins when the user clicks the data source button to view the data details.

Table 0–4 View data details

Use case name	Download reports
Actor	Admin, user
Description	The use case begins when the admin or the user downloads the dashboard as image, pdf, or PowerPoint.

Table 0–5 Download reports

Use case name	Add Components and Widgets
Actor	Admin
Description	The use case begins when the admin expands the dashboard's functionality by adding new components and widgets, enhancing the user experience and analytical capabilities.

Table 0–6 Add Components and Widgets

Use case name	Delete Component and Widgets
Actor	Admin
Description	The use case begins when the admin removes unnecessary or unneeded components and widgets from the dashboard.

Table 0–7 Delete Component and Widgets

Use case name	Customize Component and Widgets
Actor	Admin
Description	The use case begins when the admin customizes the appearance and behavior of components and widgets on the dashboard, tailoring them to specific needs and preferences.

Table 0–8 Customize Component and Widgets

Use case name	View dashboards
Actor	user
Description	The use case begins when the user clicks the present button to view the dashboards.

Table 0–9 View dashboards

Use case name	Add data source
Actor	Admin
Description	The use case begins when the admin clicks on the add data source button to add an excel file as a csv file to be the data source for the dashboard.

Table 0–10 Add data source

Appendix B: Acceptance testing resultes

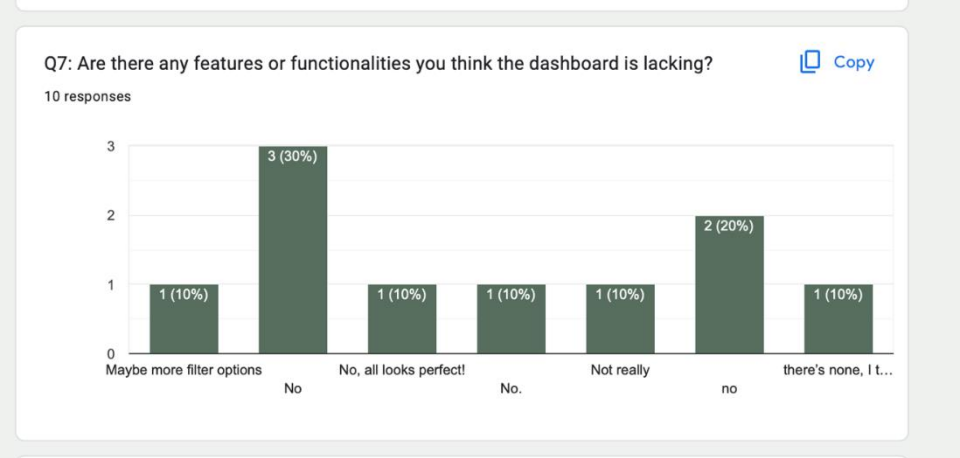


Figure 0–1 acceptance testing Q7 results

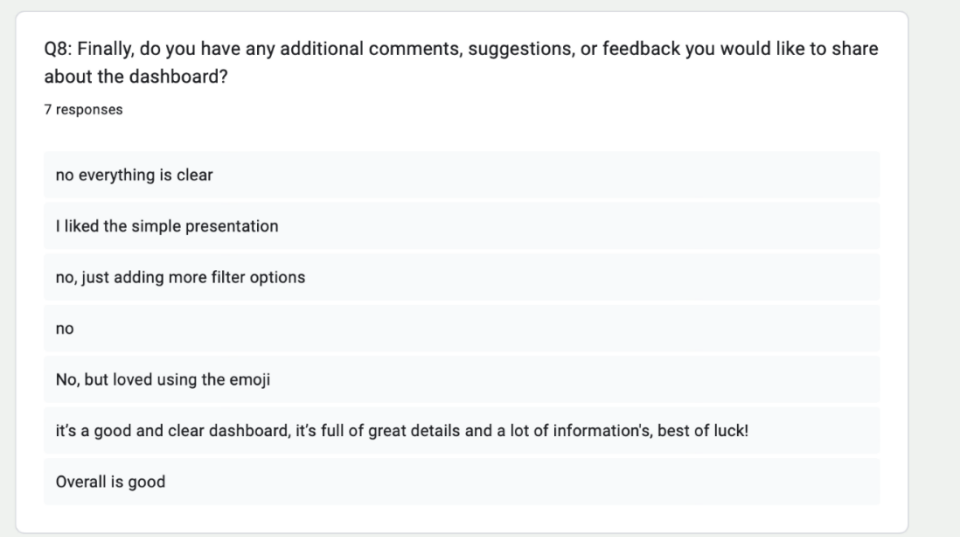


Figure 0–2 acceptance testing Q8 result

Appendix C: Source code

- **Data collection:**

```
import csv
import requests
from datetime import datetime
# output filename to save the data
filename = '/content/Dataset (R) v3.csv'

# 1672491661 = 31.12.2022 (Ronaldo)
# 1692061261 = 15.08.2023 (Neymar)
# 1686013261 = 06.06.2023 (Benzema)
start_with_timestamp = 1682608686

# Ronaldo + Al Nassr
query = "Ronaldo AND (\"Al Nassr\" OR Alnassr OR Alnasser OR \"Al Nasser\" OR
KSA OR Saudi) -filter:retweets -filter:links lang:en"
end_at_timestamp = 1672491661

# Neymar Junior + All Hilal
query = "Neymar OR \"Neymar Junior\" AND (\"Al Hilal\" OR AlHilal OR Hilal OR
KSA OR Saudi) -filter:retweets -filter:links lang:en"
end_at_timestamp = 1692061261

# Karim Benzema + AlIttihad
query = "Benzema OR \"Karim Benzema\" AND (\"Al Ittihad\" OR AlIttihad OR
Ittihad OR KSA OR Saudi) -filter:retweets -filter:links lang:en"
end_at_timestamp = 1686013261


# API endpoint and parameters to fetch tweets
api_key = '54|gNr9GhykCQp2Tva2FOjV5goEAzOT1C9w7Mmt5Yqrfce543fe'
api_endpoint = 'https://api.socialdata.tools/twitter/search'
# Global variable to track total number of tweets exported
tweets_count = 0

# Converts an ISO 8601 formatted timestamp from a tweet dictionary into a Unix
timestamp.
def get_timestamp(tweet):
    datetime_obj = datetime.fromisoformat(tweet['tweet_created_at'][:-1])
    return int(datetime_obj.timestamp())
# Process each tweet returned by the API to remove unnecessary fields
def prepare_tweet(tweet):

    # Remove entities
```

```
    if 'entities' in tweet:
        del tweet['entities']
    # Remove quoted_statsus
    if 'quoted_status' in tweet:
        del tweet['quoted_status']

    # Remove retweeted_status
    if 'retweeted_status' in tweet:
        del tweet['retweeted_status']

    # Flatten user object
    if 'user' in tweet:
        user = {}
        for key, value in tweet['user'].items():
            new_key = "user_" + key
            user[new_key] = value
        tweet.update(user)
        del tweet['user']

    return tweet
def get_search_results(query):

    # Set up the headers with the API key
    headers = {
        'Authorization': f'Bearer {api_key}',
    }
    # Prepare the request payload
    params = {
        'query': query,
    }
    print("Processing query:", query)

    # Execute the request
    # If something fails - don't stop execution of this script
    try:
        response = requests.get(api_endpoint, params=params, headers=headers)
    except:
        print('ERROR: API request failed. Will retry')
        return []

    # Return the tweets provided by the API
    if response.status_code == 200:
        response_data = response.json()
        return response_data['tweets']
    else:
```

```
        print('ERROR: Failed to send POST request. Status code:',
response.status_code)
        return []
# Open CSV file in append mode to record the tweets
with open(filename, 'a', newline='') as csvfile:
    writer = csv.writer(csvfile)

    # Recursively request tweets from the API
    should_continue = True
    while should_continue:
        if start_with_timestamp is None:
            search_query = query
        else:
            search_query = query + " until_time:" + str(start_with_timestamp)

        # Execute search query
        tweets = get_search_results(search_query)
# Repeat the following for each tweet in the dataset provided by the API
        for tweet in tweets:

            # Remove unnecessary fields from tweet
            processed_tweet = prepare_tweet(tweet)

            # Write row to csv
            writer.writerow([
                processed_tweet['tweet_created_at'],
                processed_tweet['id_str'],
                processed_tweet['full_text'],
                processed_tweet['in_reply_to_status_id_str'],
                processed_tweet['in_reply_to_user_id_str'],
                processed_tweet['in_reply_to_screen_name'],
                processed_tweet['user_id_str'],
                processed_tweet['user_name'],
                processed_tweet['user_screen_name'],
                processed_tweet['user_location'],
                processed_tweet['user_followers_count'],
                processed_tweet['user_friends_count'],
                processed_tweet['user_listed_count'],
                processed_tweet['user_favourites_count'],
                processed_tweet['user_statuses_count'],
                processed_tweet['user_created_at'],
                processed_tweet['quoted_status_id_str'],
                processed_tweet['is_quote_status'],
                processed_tweet['quote_count'],
                processed_tweet['reply_count'],
```

```
                processed_tweet['retweet_count'],
                processed_tweet['favorite_count'],
                processed_tweet['lang'],
                processed_tweet['views_count'],
                processed_tweet['bookmark_count'],
            ])
            tweets_count += 1

        print("Total tweets saved:", str(tweets_count), "  Current
start_with_timestamp =", str(start_with_timestamp))
        if len(tweets) > 0:
            earliest_tweet = min(tweets, key=lambda x: x['id'])
            start_with_timestamp = get_timestamp(earliest_tweet) - 1
        else:
# If for some reason received 0 tweets - try again with time = T-60 seconds
# There is a bug in Twitter search when sometimes it returns 0 tweets even
though some older tweets are available
            start_with_timestamp = start_with_timestamp - 60

# Stop execution if reached tweet older than end_at_timestamp variable
        if start_with_timestamp < end_at_timestamp:
            should_continue = False
            print("Finished")
```

Output:

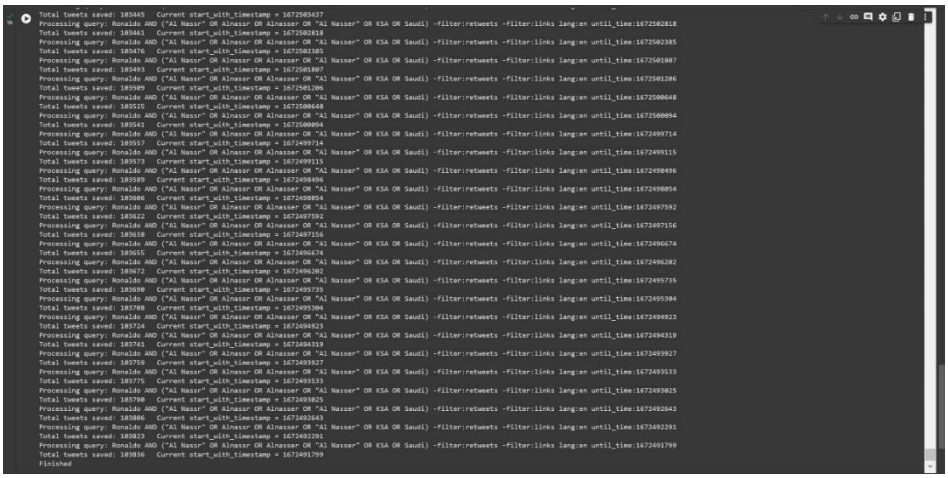


Figure 0–3 Screenshot of Ronaldo query

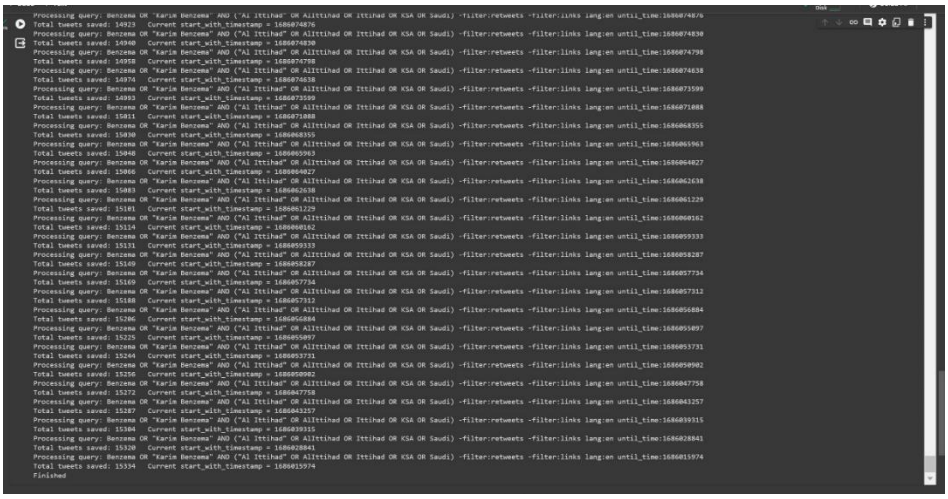


Figure 0–4 Screenshot of Benzema query

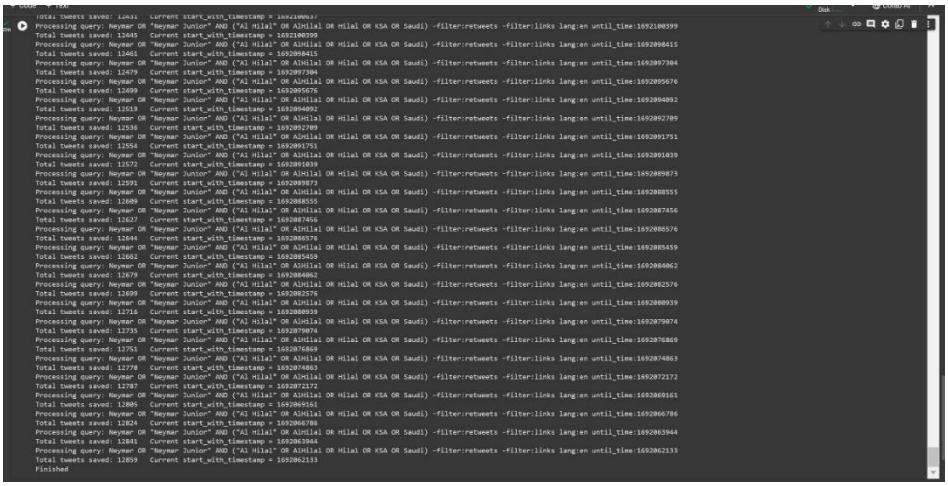


Figure 0–5 Screenshot of Neymar query

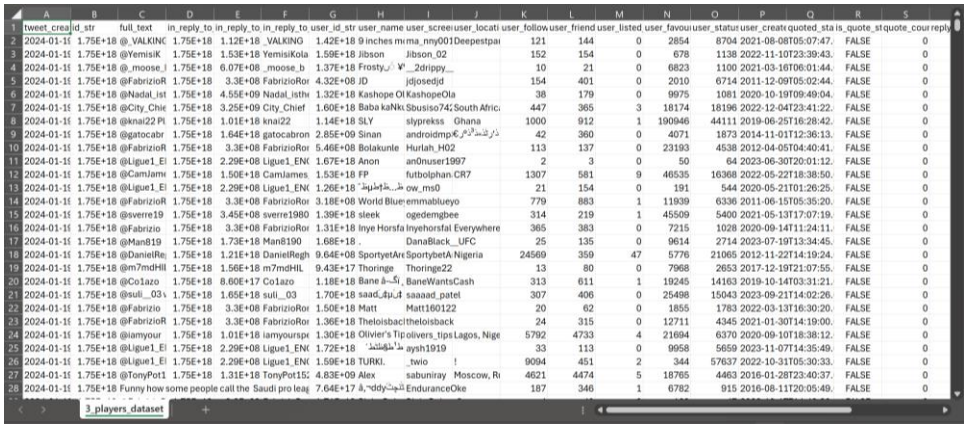


Figure 0–6 Screenshot of the CSV file

- **Data Preprocessing:**

```
#importing req. Lib.
import pandas as pd
import numpy as np
import pickle
import re

#Nltk
import nltk
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.tokenize import TweetTokenizer
from nltk.stem import SnowballStemmer
from nltk.corpus import wordnet

nltk.download('wordnet')
nltk.download('stopwords')
nltk.download('averaged_perceptron_tagger')
```

Output:

```
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data...
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
True
```

Figure 0–7 Prprocessing libraries

```
# Load the collected dataset
file_path = '/content/3_players_dataset.csv'
data = pd.read_csv(file_path, low_memory=False)

# Before Preprocess
data.shape

# Checking the first 10 Rows
data.head(10)

# Checking the last 10 Rows
data.tail(10)
# viewing the columns in our dataset
data.columns
# Viewing the info about our dataset
data.info()
# Checking unique values
data.nunique()
# Checking null values in our dataset
data.isnull().sum()

# Parsing date strings in a DataFrame, converts them to ISO 8601 format, and handles exceptions.
```

```
from dateutil import parser
for i, date_str in enumerate(data['tweet_created_at']):
    try:
        datetime_obj = parser.parse(date_str)
        iso8601_str = datetime_obj.isoformat()
        data.at[i, 'tweet_created_at'] = iso8601_str
    except ValueError:
        data.at[i, 'tweet_created_at'] = None
# Converting 'tweet_created_at' to human-readable date-time format and extracting date:
data['tweet_created_at'] = pd.to_datetime(data['tweet_created_at']).dt.date
# Converting 'tweet_created_at' to standard datetime format

data['tweet_created_at'] = pd.to_datetime(data['tweet_created_at'])

print(data['tweet_created_at'].head())

data['tweet_created_at'].min()

#checking uniques values in tweet_created columns

data['tweet_created_at'].nunique()
numberoftweets = data.groupby('tweet_created_at').size()
numberoftweets.dtype
numberoftweets

data.isna().sum()

print("Percentage null or na values in df")

((data.isnull() | data.isna()).sum() * 100 / data.index.size).round(2)
del data['quoted_status_id_str']
del data['user_created_at']
del data['user_location']
del data['in_reply_to_status_id_str']
del data['in_reply_to_screen_name']
del data['in_reply_to_screen_name']
del data['user_screen_name']
del data['user_followers_count']
del data['user_friends_count']
del data['user_listed_count']
del data['user_favourites_count']
del data['user_statuses_count']
del data['is_quote_status']
del data['quote_count']
del data['retweet_count']
del data['favorite_count']
del data['bookmark_count']
del data['views_count']
del data['user_id_str']
del data['lang']
```

```
del data['reply_count']
del data['in_reply_to_user_id_str']
data.info()

data.head(10)

data.tail(10)

data.isnull().sum()

# Convert the 'full_text' column in the DataFrame 'data' to string data type
data['full_text'] = data['full_text'].astype("string")
# Defining dictionary containing all emojis with their meanings
emojis = {':)': 'smile', ':-)': 'smile', ';d': 'wink', ':-E': 'vampire', ':(': 'sad',
          ':-(': 'sad', ':-<': 'sad', ':P': 'raspberry', ':O': 'surprised',
          ':-@': 'shocked', ':@': 'shocked',':-$': 'confused', ':\\': 'annoyed',
          ':#': 'mute', ':X': 'mute', ':^)': 'smile', ':-&': 'confused', '$_$': 'greedy',
          '@@': 'eyeroll', ':-!': 'confused', ':-D': 'smile', ':-0': 'yell', 'O.o': 'confused',
          '<(-_-)>': 'robot', 'd[-_-]b': 'dj', ":'-)": 'sadsmile', ';)': 'wink',
          ';-)': 'wink', 'O:-)': 'angel','O*-)': 'angel','(:-D': 'gossip', '=^.^=': 'cat'}
def preprocess(textdata):
    processedText = []
    # Defining regex patterns.
    urlPattern = r"((http://)[^ ]*|(https://)[^ ]*|( www\.)[^ ]*)"
    userPattern = r'@[^\s]+'
    alphaPattern = r"[^a-zA-Z0-9]"
    sequencePattern = r"(.)\1\1+"
    seqReplacePattern = r"\1\1"
    for tweet in textdata:
        if pd.notnull(tweet):  # Check for missing values
            tweet = tweet.lower()

# Replace all URLs with 'URL'
            tweet = re.sub(urlPattern, 'URL', tweet)

            # Replace @USERNAME with 'USER'
            tweet = re.sub(userPattern, 'USER', tweet)

            # Remove emojis
            for emoji in emojis.keys():
                tweet = tweet.replace(emoji, '')

            # Remove all non-alphabetic characters
            tweet = re.sub(alphaPattern, " ", tweet)

            # Replace 3 or more consecutive letters by 2 letters
            tweet = re.sub(sequencePattern, seqReplacePattern, tweet)

            processedText.append(tweet)
```

```
        else:
            processedText.append('')

    return processedText
processed_tweets = preprocess(data['full_text'])
# Assign the preprocessed tweets stored in the 'processed_tweets' list to a new column named
'preprocessed_text' in the DataFrame 'data'.

data['preprocessed_text'] = processed_tweets

# Handle missing values
data.dropna(inplace=True)
# Handle Duplicate values
data.drop_duplicates(inplace=True)
# Remove < 3 words in full text rows
data= data[data['preprocessed_text'].apply(lambda x: len(x.split()) > 2)]
# Display the updated dataset
print(data.head(10))
del data['tweet_created_at']
del data['full_text']
del data['id_str']
data.shape
data.info()

data.nunique()

def get_wordnet_pos(word):
    tag = nltk.pos_tag([word])[0][1][0].upper()
    tag_dict = {"J": wordnet.ADJ, "N": wordnet.NOUN, "V": wordnet.VERB, "R": wordnet.ADV}
    return tag_dict.get(tag, wordnet.NOUN)

# Function to tokenize, remove stop words, and lemmatize text data
def preprocess_text(df, text_column):
    tokenizer = TweetTokenizer()
    stop_words = set(stopwords.words('english'))
    lemmatizer = WordNetLemmatizer()

    df[text_column + '_tokenized'] = df[text_column].apply(tokenizer.tokenize)
    df[text_column + '_tokenized'] = df[text_column + '_tokenized'].apply(lambda tokens:
[lemmatizer.lemmatize(token, get_wordnet_pos(token)) for token in tokens if token.lower() not in
stop_words])

 # Remove instances of the word 'user'
    df[text_column + '_tokenized'] = df[text_column + '_tokenized'].apply(lambda tokens: [token for token in
tokens if token.lower() != 'user'])
    df[text_column + '_tokenized'] = df[text_column + '_tokenized'].apply(lambda tokens: [token for token in
tokens if not token.isdigit()])
    return df
```

```
# Function to save the DataFrame to a new CSV file
def save_to_csv(df, output_filename):
    df.to_csv(output_filename, index=False)
    print(f"Processed data saved to {output_filename}")
# Load the data
file_path ='/finaldatasetpreprocessed.csv'
data = pd.read_csv(file_path, low_memory=False)

# Explore the data
print("Original Data:")
print(data.head())
print("\nData Info:")
print(data.info())

# Specify the text column in the DataFrame
text_column = 'preprocessed_text'

# Preprocessing pipeline
data = preprocess_text(data, text_column)
# Explore the preprocessed data
print("\nPreprocessed Data:")
print(data.head())

# Save the processed data to a new CSV file
output_filename = 'finalpreprocesseddataset.csv'
save_to_csv(data, output_filename)
del data['preprocessed_text']
data.dropna(inplace=True)
data.drop_duplicates(inplace=True)
data = data[data['preprocessed_text_tokenized'].apply(lambda x: len(x) > 2)]
# After preprocessing
data.shape
# Save the processed data to a new CSV file
output_filename = 'final_dataset.csv'
save_to_csv(data, output_filename)
print("Dataset have been saved successfully to", output_filename, "file")
```

- **Algorithm's codes:**
  - **BERT**

```
!pip install torch transformers
```

Output:

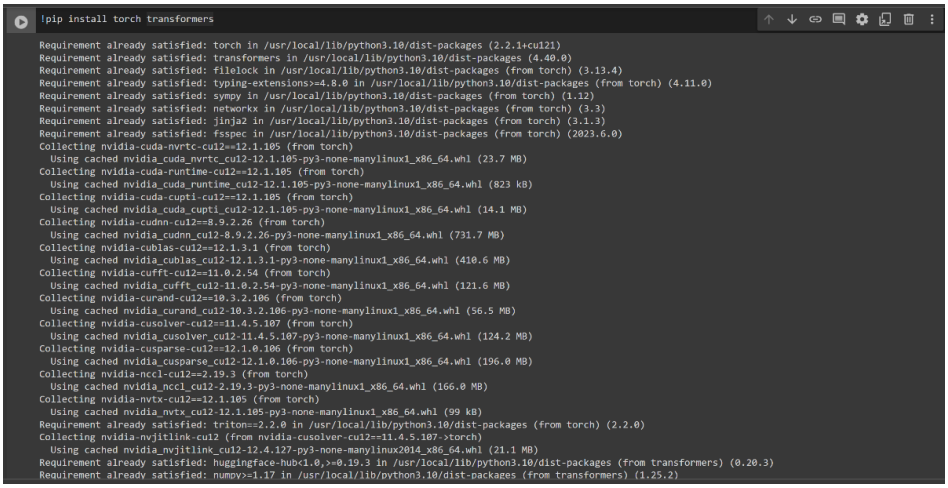


Figure 0–8 Torch transformers1

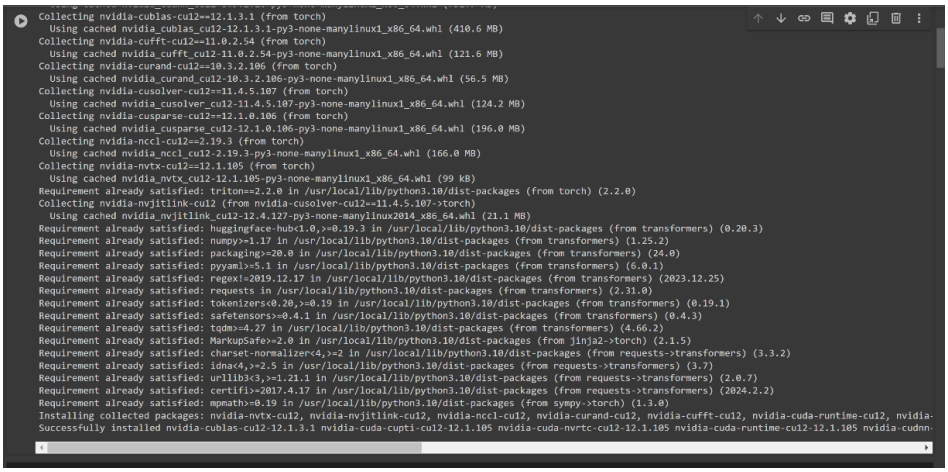


Figure 0–9 Torch transformers2

```
import torch
import numpy as np
import pandas as pd
import seaborn as sns
from textblob import TextBlob
from sklearn.metrics import roc_auc_score
from transformers import BertTokenizer, BertForSequenceClassification
from sklearn.model_selection import train_test_split
from torch.utils.data import TensorDataset, DataLoader
import torch.optim as optim
import torch.nn as nn
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, confusion_matrix,
classification_report, accuracy_score, f1_score, precision_score, recall_score,
roc_curve, auc

# Load the dataset
data = pd.read_csv('/content/final_dataset.csv')

# Apply the sentiment analysis function & create a new 'polarity' column
data['polarity'] = data['preprocessed_text_tokenized'].apply(lambda x:
TextBlob(x).sentiment.polarity)

# Assign labels based on polarity
data['polarity_label'] = data['polarity'].apply(lambda x: 1 if x > 0 else 0 if
x == 0 else 2)

# Split the dataset into training and testing sets
train_data, test_data, train_labels, test_labels = train_test_split(
    data['preprocessed_text_tokenized'], data['polarity_label'], test_size=0.2,
random_state=42)

# Load the pre-trained BERT model and tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased',
force_download=True)
model = BertForSequenceClassification.from_pretrained('bert-base-uncased',
num_labels=3)  # 3 classes

# Tokenize and format the input data
train_encodings = tokenizer(list(train_data), truncation=True, padding=True,
return_tensors='pt')
test_encodings = tokenizer(list(test_data), truncation=True, padding=True,
return_tensors='pt')

# Convert pandas Series to lists
```

```
train_input_ids = train_encodings['input_ids'].tolist()
train_attention_mask = train_encodings['attention_mask'].tolist()
train_labels_list = train_labels.tolist()

# Create PyTorch DataLoader
train_dataset = TensorDataset(torch.tensor(train_input_ids),
torch.tensor(train_attention_mask), torch.tensor(train_labels_list))

# Similar conversion for the test dataset
test_input_ids = test_encodings['input_ids'].tolist()
test_attention_mask = test_encodings['attention_mask'].tolist()
test_labels_list = test_labels.tolist()

test_dataset = TensorDataset(torch.tensor(test_input_ids),
torch.tensor(test_attention_mask), torch.tensor(test_labels_list))


train_dataloader = DataLoader(train_dataset, batch_size=16, shuffle=True)
test_dataloader = DataLoader(test_dataset, batch_size=16, shuffle=False)

# Training parameters
optimizer = torch.optim.AdamW(model.parameters(), lr=2e-5)
criterion = torch.nn.CrossEntropyLoss()
```

Output:

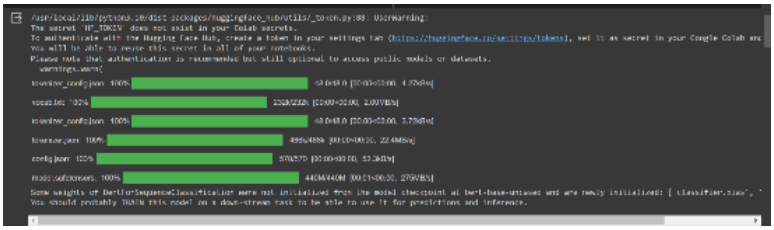


Figure 0–10 Training parameters

```
# Training loop
epochs = 1
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model.to(device)

for epoch in range(epochs):
    model.train()
    for batch in DataLoader(train_dataset, batch_size=16, shuffle=True):
        input_ids, attention_mask, labels = batch
        input_ids, attention_mask, labels = input_ids.to(device),
attention_mask.to(device), labels.to(device)

        optimizer.zero_grad()
```

```
        outputs = model(input_ids, attention_mask=attention_mask,
labels=labels)
        loss = outputs.loss
        loss.backward()
        optimizer.step()

#Training accuracy
model.eval()
with torch.no_grad():
    train_preds = []
    train_labels_list = []
    for batch in DataLoader(train_dataset, batch_size=8, shuffle=False):
        input_ids, attention_mask, labels = batch
        input_ids, attention_mask, labels = input_ids.to(device),
attention_mask.to(device), labels.to(device)

        outputs = model(input_ids, attention_mask=attention_mask)
        logits = outputs.logits
        preds = torch.argmax(logits, dim=1).cpu().numpy()

        train_preds.extend(preds)
        train_labels_list.extend(labels.cpu().numpy())

train_accuracy = accuracy_score(train_labels_list, train_preds)
print(f'Training Accuracy: {train_accuracy * 100:.2f}%')
```

Output:

```
Training Accuracy: 97.92%
```

Figure 0–11 Bert Training Accuracy

```
# Evaluate on the test set
model.eval()
all_preds = []
all_labels = []

with torch.no_grad():
    for batch in test_dataloader:
        input_ids, attention_mask, labels = batch
        input_ids, attention_mask, labels = input_ids.to(device),
attention_mask.to(device), labels.to(device)

        outputs = model(input_ids, attention_mask=attention_mask)
        logits = outputs.logits
        _, preds = torch.max(logits, dim=1)
```

```
        all_preds.extend(preds.cpu().numpy())
        all_labels.extend(labels.cpu().numpy())

# Save the test predictions in a new CSV file
result_df = pd.DataFrame({
    'Text': test_data,
    'Actual_Labels': all_labels,
    'Predicted_Labels': all_preds
})

result_df.to_csv('/content/test_predictions_BERT.csv', index=False)

# Calculate and print metrics
accuracy = accuracy_score(all_labels, all_preds)
print(f'Testing Accuracy: {accuracy * 100:.2f}%')
```

Output:

```
Testing Accuracy: 97.26%
```

Figure 0–12 Bert Testing Accuracy

```
from sklearn.preprocessing import label_binarize
# Get predicted probabilities from the model
model.eval()
predicted_probabilities = []

with torch.no_grad():
    for batch in test_dataloader:
        input_ids, attention_mask, labels = batch
        input_ids, attention_mask, labels = input_ids.to(device),
attention_mask.to(device), labels.to(device)

        outputs = model(input_ids, attention_mask=attention_mask)
        logits = outputs.logits
        probabilities = torch.softmax(logits, dim=1)  # Convert logits to
probabilities

        predicted_probabilities.extend(probabilities.cpu().numpy())

# Binarize the test labels to create multi-class ROC curves
test_labels_binarized = label_binarize(test_labels, classes=[0, 1, 2])

# Generate ROC curves and calculate AUC for each class
fpr = dict()  # False Positive Rates
tpr = dict()  # True Positive Rates
```

```
roc_auc = dict()  # AUC scores for each class

for i in range(3):
    fpr[i], tpr[i], _ = roc_curve(test_labels_binarized[:, i], [prob[i] for
prob in predicted_probabilities])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Plot the ROC curves
plt.figure()
colors = ['blue', 'red', 'green']
labels = ['Class 0', 'Class 1', 'Class 2']
for i, color, label in zip(range(3), colors, labels):
    plt.plot(fpr[i], tpr[i], color=color, lw=2, label=f'{label} (AUC =
{roc_auc[i]:.2f})')

plt.plot([0, 1], [0, 1], 'k--', lw=2)  # Dashed line for a random classifier
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve for Multi-Class Sentiment Analysis')
plt.legend(loc='lower right')
plt.show()

# Print the AUC scores
for i, auc_score in roc_auc.items():
    print(f'Class {i} AUC score: {auc_score:.2f}')
```

Output:

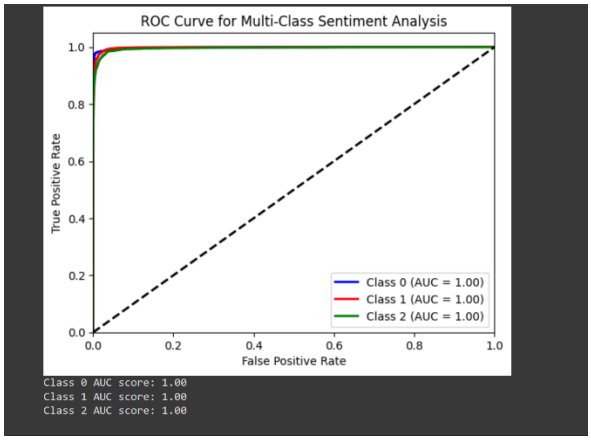


Figure 0–13 Bert ROC/AUC

```
# Confusion Matrix
conf_matrix = confusion_matrix(all_labels, all_preds)
print('Confusion Matrix:')
print(conf_matrix)
```

Output:

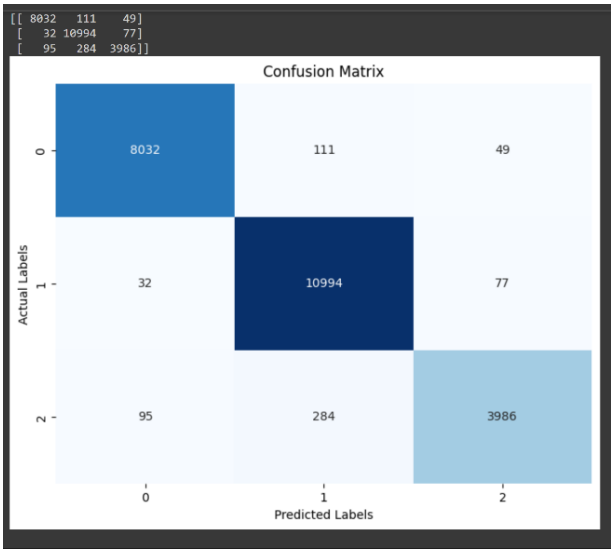


Figure 0–14 Bert Confusion Matrix

```
# Classification Report
print('Classification Report:')
print(classification_report(all_labels, all_preds, target_names=['Class 0',
'Class 1', 'Class 2']))
```

Output:

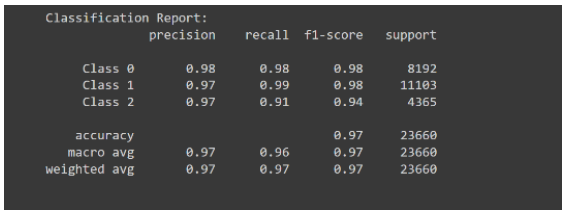


Figure 0–15 Bert Classification Report

```
test_predictions = pd.read_csv('/content/test_predictions_BERT.csv')

positive_count = (test_predictions['Actual_Labels'] == 1).sum()
negative_count = (test_predictions['Actual_Labels'] == 2).sum()
neutral_count = (test_predictions['Actual_Labels'] == 0).sum()

total_count = len(test_predictions)

# Calculate the percentages
```

```
positive_percentage = (positive_count / total_count) * 100
negative_percentage = (negative_count / total_count) * 100
neutral_percentage = (neutral_count / total_count) * 100

print(f"Positive Percentage: {positive_percentage:.2f}%")
print(f"Negative Percentage: {negative_percentage:.2f}%")
print(f"Neutral Percentage: {neutral_percentage:.2f}%")
```

Output:

```
Positive Percentage: 46.93%
Negative Percentage: 18.45%
Neutral Percentage: 34.62%
```

Figure 0–16 Sentiment distribution of Bert

- **RNN**

```
#Import req.libraries
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from textblob import TextBlob
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, SimpleRNN, Dense
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.utils import to_categorical
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import confusion_matrix, classification_report, roc_auc_score, roc_curve, auc
from textblob import TextBlob

data = pd.read_csv('/content/final_dataset.csv')
# Apply the sentiment analysis function to the 'preprocessed_text_tokenized' column and create a new
'polarity' column
data['polarity'] = data['preprocessed_text_tokenized'].apply(lambda x: TextBlob(x).sentiment.polarity)
# Assign labels based on polarity
data['polarity_label'] = data['polarity'].apply(lambda x: 1 if x > 0 else 0 if x == 0 else -1)
data['Sentiment'] = data['polarity'].apply(lambda x: 'positive' if x > 0 else 'neutral' if x == 0 else 'negative')
data.to_csv('/content/final_dataset_labeled.csv', index=False)

del data['polarity']
del data['polarity_label']
data.shape
text = data['preprocessed_text_tokenized'].values
sentiment = data['Sentiment'].values

# Tokenization & Padding the text data
tokenizer = Tokenizer()
```

```
tokenizer.fit_on_texts(text)
X_sequence = tokenizer.texts_to_sequences(text)
max_sequence_length = max([len(seq) for seq in X_sequence])
X_padded = pad_sequences(X_sequence, maxlen=max_sequence_length)

# Convert the Sentiment labels to Numerical format
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(sentiment)
y_categorical = to_categorical(y, num_classes=3)  # Assuming 3 sentiment classes

# Build the RNN model
rnn = Sequential()
rnn.add(Embedding(input_dim=len(tokenizer.word_index)+1, output_dim=64,
input_length=max_sequence_length))
rnn.add(SimpleRNN(64))  # Using a SimpleRNN layer
rnn.add(Dense(3, activation='softmax'))  # 3 output classes for sentiment analysis
#Compile the RNN model
rnn.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_padded, y_categorical, test_size=0.2, random_state=42)
# Train the model
rnn.fit(X_train, y_train, epochs=15, batch_size=64, validation_data=(X_test, y_test))
```

Output:

```
Epoch 1/15
1479/1479 [==============================] - 1003s 677ms/step - loss: 0.8543 - accuracy: 0.6042 - val_loss: 0.4305 - val_accuracy: 0.8466
Epoch 2/15
1479/1479 [==============================] - 968s 655ms/step - loss: 0.2134 - accuracy: 0.9330 - val_loss: 0.1632 - val_accuracy: 0.9511
Epoch 3/15
1479/1479 [==============================] - 970s 656ms/step - loss: 0.1077 - accuracy: 0.9686 - val_loss: 0.1422 - val_accuracy: 0.9567
Epoch 4/15
1479/1479 [==============================] - 962s 650ms/step - loss: 0.1272 - accuracy: 0.9584 - val_loss: 0.2352 - val_accuracy: 0.9291
Epoch 5/15
1479/1479 [==============================] - 965s 652ms/step - loss: 0.0842 - accuracy: 0.9760 - val_loss: 0.1501 - val_accuracy: 0.9551
Epoch 6/15
1479/1479 [==============================] - 964s 652ms/step - loss: 0.0744 - accuracy: 0.9782 - val_loss: 0.1768 - val_accuracy: 0.9551
Epoch 7/15
1479/1479 [==============================] - 965s 652ms/step - loss: 0.0550 - accuracy: 0.9841 - val_loss: 0.1630 - val_accuracy: 0.9557
Epoch 8/15
1479/1479 [==============================] - 977s 661ms/step - loss: 0.0444 - accuracy: 0.9870 - val_loss: 0.1881 - val_accuracy: 0.9577
Epoch 9/15
1479/1479 [==============================] - 969s 655ms/step - loss: 0.0441 - accuracy: 0.9870 - val_loss: 0.1667 - val_accuracy: 0.9566
Epoch 10/15
1479/1479 [==============================] - 966s 653ms/step - loss: 0.0306 - accuracy: 0.9908 - val_loss: 0.1800 - val_accuracy: 0.9603
Epoch 11/15
1479/1479 [==============================] - 977s 660ms/step - loss: 0.0296 - accuracy: 0.9913 - val_loss: 0.1781 - val_accuracy: 0.9540
Epoch 12/15
1479/1479 [==============================] - 971s 656ms/step - loss: 0.0273 - accuracy: 0.9921 - val_loss: 0.1775 - val_accuracy: 0.9606
Epoch 13/15
1479/1479 [==============================] - 971s 657ms/step - loss: 0.0494 - accuracy: 0.9861 - val_loss: 0.3831 - val_accuracy: 0.8998
Epoch 14/15
1479/1479 [==============================] - 975s 659ms/step - loss: 0.0325 - accuracy: 0.9906 - val_loss: 0.1912 - val_accuracy: 0.9566
Epoch 15/15
1479/1479 [==============================] - 969s 655ms/step - loss: 0.0260 - accuracy: 0.9926 - val_loss: 0.1883 - val_accuracy: 0.9560
```

Figure 0–17 RNN train

```
# Evaluate the model on the test set
y_pred = rnn.predict(X_test)
y_pred_labels = np.argmax(y_pred, axis=1)
y_true_labels = np.argmax(y_test, axis=1)
```

Output:

```
740/740 [==============================] - 52s 70ms/step
```

Figure 0–18 Evaluate the model on the test set

```
# Print the model accuracy
_, training_accuracy = rnn.evaluate(X_train, y_train)  # Evaluate the model on the training data and get the accuracy
_, test_accuracy = rnn.evaluate(X_test, y_test)  # Evaluate the model on the test data and get the accuracy
print(f'Training Set Accuracy: {training_accuracy * 100 :.4f}')
print(f'Test Set Accuracy: {test_accuracy * 100 :.4f}')
```

Output:

```
2958/2958 [==============================] - 211s 71ms/step - loss: 0.0132 - accuracy: 0.9968
740/740 [==============================] - 53s 71ms/step - loss: 0.1883 - accuracy: 0.9560
Training Set Accuracy: 99.6756
Test Set Accuracy: 95.6044
```

Figure 0–19 RNN training & testing accuracy

```
# Compute Confusion Matrix
conf_matrix = confusion_matrix(y_true_labels, y_pred_labels)
print("Confusion Matrix:")
print(conf_matrix)

# Plot confusion matrix using seaborn heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues' ,
            xticklabels=['Negative', 'Neutral', 'Positive'],
            yticklabels=['Negative', 'Neutral', 'Positive'])
plt.title('Confusion Matrix')
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.show()
```

Output:

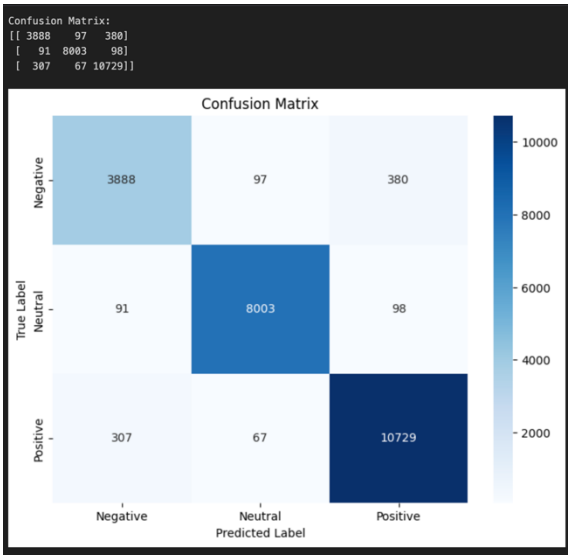


Figure 0–20 RNN Confusion Matrix

```
# ROC/AUC
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(3):  # Assuming 3 classes
    fpr[i], tpr[i], _ = roc_curve(y_test[:, i], y_pred[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Print the ROC/AUC for the model
auc_score = roc_auc_score(y_test, y_pred, multi_class='ovr')  # Assuming 'ovr' for multi-class
print(f"ROC AUC Score: {auc_score:.4f}")

# Plot ROC curve
plt.figure()
plt.plot(fpr[2], tpr[2], color='darkorange', lw=2, label='ROC curve (area = %0.2f)' % roc_auc[2])
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()
```

Output:

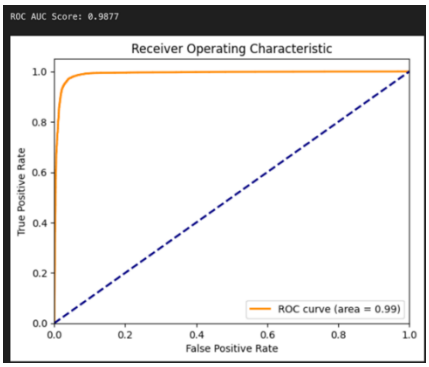


Figure 0–21 RNN ROC/AUC

```
# Classification Report
class_report = classification_report(y_true_labels, y_pred_labels)
print("Classification Report:")
print(class_report)
```

Output:

```
Classification Report:
              precision    recall  f1-score   support

           0       0.91      0.89      0.90      4365
           1       0.98      0.98      0.98      8192
           2       0.96      0.97      0.96     11103

    accuracy                           0.96     23660
   macro avg       0.95      0.94      0.95     23660
weighted avg       0.96      0.96      0.96     23660
```

Figure 0–22 RNN Classification Report

```
# Create a DataFrame with true and predicted labels
df = pd.DataFrame({
   'True Label': y_true_labels,
   'Predicted Label': y_pred_labels
   })
# Define the file path for saving the CSV file
file_path = 'RNN_Predict_result.csv.csv'

# Save the DataFrame to a new CSV file
df.to_csv(file_path, index=False)
print(f"The true labels and predicted labels have been saved to {file_path}.")

df1 = pd.read_csv('/content/RNN_Predict_result.csv.csv')
# Count the occurrences of each sentiment label
sentiment_counts = df1['Predicted Label'].value_counts()

# Print the count of each sentiment label
print("Count of Neutral Sentiment :{}".format(sentiment_counts[1]))
print("Count of Positive Sentiment : {}".format(sentiment_counts[2]))
print("Count of Negative Sentiment : {}".format(sentiment_counts[0]))

count_of_negative = 4286
count_of_neutral = 8167
count_of_positive = 11207

# Calculate the total count of all sentiments
total_count = count_of_neutral + count_of_positive + count_of_negative

# Calculate the percentage of each sentiment label
neutral_percentage = (count_of_neutral / total_count) * 100
positive_percentage = (count_of_positive / total_count) * 100
negative_percentage = (count_of_negative / total_count) * 100

# Print the percentage of each sentiment label
print("Neutral Sentiment: {:.2f}%".format(neutral_percentage))
print("Positive Sentiment: {:.2f}%".format(positive_percentage))
print("Negative Sentiment: {:.2f}%".format(negative_percentage))
```

Output:

```
Neutral Sentiment: 34.52%
Positive Sentiment: 47.37%
Negative Sentiment: 18.11%
```

Figure 0–23 Sentiment distribution of RNN

- **SVM**

```
# Install necessary libraries
!pip install seaborn matplotlib
import pandas as pd
from textblob import TextBlob
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix, roc_auc_score
import seaborn as sns
import matplotlib.pyplot as plt

# Function to save the DataFrame to a new CSV file
def save_to_csv(df, output_filename):
    df.to_csv(output_filename, index=False)
    print(f"Processed data saved to {output_filename}")

# Load the preprocessed data
file_path = '/content/10Benzema20Neymar26RonaldoUPDATED.csv'
data = pd.read_csv(file_path, low_memory=False)

# Specify the tokenized text column in your DataFrame
text = 'preprocessed_text_tokenized'

# Function to calculate polarity and assign labels
def get_polarity(text):
    analysis = TextBlob(text)
    return analysis.sentiment.polarity

# Apply the sentiment analysis function to the 'text_data' column and create a
new 'polarity' column
data['polarity'] = data[text].apply(get_polarity)

# Assign labels based on polarity
data['polarity_label'] = data['polarity'].apply(lambda x: 'positive' if x > 0
else 'negative' if x < 0 else 'neutral')
data['polarity_label'] = data['polarity_label'].map({'negative': -1,'neutral':
0, 'positive': 1})

# TF-IDF Vectorization
tfidf = TfidfVectorizer(stop_words='english', max_features=5000)
X = tfidf.fit_transform(data['preprocessed_text_tokenized'])
```

```
y = data['polarity_label']

# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# SVM model with GridSearchCV
svm_model = SVC()
param_grid = {'C': [0.1, 1, 10], 'gamma': ['scale', 'auto'], 'kernel': ['rbf',
'linear']}
grid = GridSearchCV(svm_model, param_grid, refit=True, verbose=2)
grid.fit(X_train, y_train)

print(grid.best_params_)
print(grid.best_estimator_)

# Create the model with the best parameters
svm_best = SVC(C=10, gamma='scale', kernel='linear', probability=True)

# Train the model
svm_best.fit(X_train, y_train)

# Predict on the test data
y_pred = svm_best.predict(X_test)

# Calculate accuracy on the training set
y_train_pred = svm_best.predict(X_train)
train_accuracy = accuracy_score(y_train, y_train_pred) * 100
print("Training Accuracy:", train_accuracy, "%")

# Calculate accuracy on the testing set
test_accuracy = accuracy_score(y_test, y_pred) * 100
print("Testing Accuracy:", test_accuracy, "%")

# Save the test predictions data
test_predictions_df = pd.DataFrame({'Text': data.loc[y_test.index, text],
                                    'Actual_Labels': y_test,
                                    'Predicted_Labels': y_pred})
save_to_csv(test_predictions_df, 'SVM_test_predictions.csv')
```

Output:

```
Fitting 5 folds for each of 12 candidates, totalling 60 fits
[CV] END .....................C=0.1, gamma=scale, kernel=rbf; total time= 5.1min
[CV] END .....................C=0.1, gamma=scale, kernel=rbf; total time= 5.5min
[CV] END .....................C=0.1, gamma=scale, kernel=rbf; total time= 5.1min
[CV] END .....................C=0.1, gamma=scale, kernel=rbf; total time= 5.2min
[CV] END .....................C=0.1, gamma=scale, kernel=rbf; total time= 5.5min
[CV] END ..................C=0.1, gamma=scale, kernel=linear; total time= 3.5min
[CV] END ..................C=0.1, gamma=scale, kernel=linear; total time= 3.5min
[CV] END ..................C=0.1, gamma=scale, kernel=linear; total time= 3.4min
[CV] END ..................C=0.1, gamma=scale, kernel=linear; total time= 3.5min
[CV] END ..................C=0.1, gamma=scale, kernel=linear; total time= 3.6min
[CV] END .....................C=0.1, gamma=auto, kernel=rbf; total time= 4.4min
[CV] END .....................C=0.1, gamma=auto, kernel=rbf; total time= 4.5min
[CV] END .....................C=0.1, gamma=auto, kernel=rbf; total time= 4.5min
[CV] END .....................C=0.1, gamma=auto, kernel=rbf; total time= 4.6min
[CV] END .....................C=0.1, gamma=auto, kernel=rbf; total time= 4.6min
[CV] END ..................C=0.1, gamma=auto, kernel=linear; total time= 3.7min
[CV] END ..................C=0.1, gamma=auto, kernel=linear; total time= 3.6min
[CV] END ..................C=0.1, gamma=auto, kernel=linear; total time= 3.6min
[CV] END ..................C=0.1, gamma=auto, kernel=linear; total time= 3.6min
[CV] END ..................C=0.1, gamma=auto, kernel=linear; total time= 3.7min
[CV] END .......................C=1, gamma=scale, kernel=rbf; total time= 9.8min
[CV] END .......................C=1, gamma=scale, kernel=rbf; total time= 9.4min
[CV] END .......................C=1, gamma=scale, kernel=rbf; total time= 9.5min
[CV] END .......................C=1, gamma=scale, kernel=rbf; total time= 9.6min
[CV] END .......................C=1, gamma=scale, kernel=rbf; total time= 9.5min
[CV] END ...................C=1, gamma=scale, kernel=linear; total time= 3.1min
[CV] END ...................C=1, gamma=scale, kernel=linear; total time= 3.1min
[CV] END ...................C=1, gamma=scale, kernel=linear; total time= 3.1min
[CV] END ...................C=1, gamma=scale, kernel=linear; total time= 3.1min
[CV] END ...................C=1, gamma=scale, kernel=linear; total time= 3.1min
[CV] END .......................C=1, gamma=auto, kernel=rbf; total time= 4.8min
[CV] END .......................C=1, gamma=auto, kernel=rbf; total time= 4.8min
[CV] END .......................C=1, gamma=auto, kernel=rbf; total time= 4.8min
[CV] END .......................C=1, gamma=auto, kernel=rbf; total time= 4.8min
[CV] END .......................C=1, gamma=auto, kernel=rbf; total time= 4.9min
[CV] END ...................C=1, gamma=auto, kernel=linear; total time= 3.2min
[CV] END ...................C=1, gamma=auto, kernel=linear; total time= 3.2min
[CV] END ...................C=1, gamma=auto, kernel=linear; total time= 3.1min
[CV] END ...................C=1, gamma=auto, kernel=linear; total time= 3.2min
```

Figure 0–24 SVM training1

```
[CV] END ......................C=10, gamma=scale, kernel=rbf; total time=11.6min
[CV] END ......................C=10, gamma=scale, kernel=rbf; total time=11.9min
[CV] END ......................C=10, gamma=scale, kernel=rbf; total time=12.2min
[CV] END ......................C=10, gamma=scale, kernel=rbf; total time=12.2min
[CV] END ......................C=10, gamma=scale, kernel=rbf; total time=12.6min
[CV] END ..................C=10, gamma=scale, kernel=linear; total time= 5.7min
[CV] END ..................C=10, gamma=scale, kernel=linear; total time= 5.4min
[CV] END ..................C=10, gamma=scale, kernel=linear; total time= 5.3min
[CV] END ..................C=10, gamma=scale, kernel=linear; total time= 5.4min
[CV] END ..................C=10, gamma=scale, kernel=linear; total time= 5.3min
[CV] END ......................C=10, gamma=auto, kernel=rbf; total time= 4.9min
[CV] END ......................C=10, gamma=auto, kernel=rbf; total time= 5.0min
[CV] END ......................C=10, gamma=auto, kernel=rbf; total time= 5.0min
[CV] END ......................C=10, gamma=auto, kernel=rbf; total time= 4.7min
[CV] END ......................C=10, gamma=auto, kernel=rbf; total time= 4.7min
[CV] END ..................C=10, gamma=auto, kernel=linear; total time= 5.2min
[CV] END ..................C=10, gamma=auto, kernel=linear; total time= 5.1min
[CV] END ..................C=10, gamma=auto, kernel=linear; total time= 5.1min
[CV] END ..................C=10, gamma=auto, kernel=linear; total time= 5.2min
[CV] END ..................C=10, gamma=auto, kernel=linear; total time= 5.1min
{'C': 1, 'gamma': 'scale', 'kernel': 'linear'}
SVC(C=1, kernel='linear')
Training Accuracy: 93.91192669613969 %
Testing Accuracy: 88.9953571075768 %
Processed data saved to SVM_test_predictions.csv
```

Figure 0–25 SVM training2

```
# Visualize the confusion matrix
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues",
            xticklabels=['Negative', 'Neutral', 'Positive'],
            yticklabels=['Negative', 'Neutral', 'Positive'])
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()

# Print the confusion matrix
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))

# Print a classification report to see precision, recall, and F1-score
print("\nClassification Report:")
print(classification_report(y_test, y_pred))

# Calculate and print the AUC using the one-vs-rest (OvR) strategy
y_prob = svm_best.predict_proba(X_test)
auc = roc_auc_score(y_test, y_prob, multi_class='ovr')
print("AUC:", auc)
```

Output:

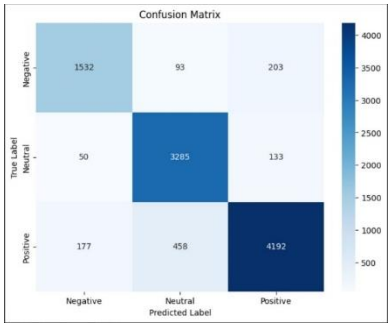


Figure 0–26 SVM Confusion Matrix

```
Confusion Matrix:
[[1532   93  203]
 [  50 3285  133]
 [ 177  458 4192]]

Classification Report:
              precision    recall  f1-score   support

          -1       0.87      0.84      0.85      1828
           0       0.86      0.95      0.90      3468
           1       0.93      0.87      0.90      4827

    accuracy                           0.89     10123
   macro avg       0.88      0.88      0.88     10123
weighted avg       0.89      0.89      0.89     10123
```

Figure 0–27 Classification report

```
AUC: 0.9609136418650263
```

Figure 0–28 SVM AUC result

```
import pandas as pd

# Read the test predictions file
data = pd.read_csv("/content/SVM_test_predictions.csv")

# Extract the "Predicted_Labels" column
predicted_labels = data["Predicted_Labels"]

# Count the occurrences of each label
positive_count = (predicted_labels == 1).sum()
negative_count = (predicted_labels == -1).sum()
neutral_count = (predicted_labels == 0).sum()

# Calculate the percentage of each label category
total_samples = len(predicted_labels)
positive_percentage = (positive_count / total_samples) * 100
negative_percentage = (negative_count / total_samples) * 100
neutral_percentage = (neutral_count / total_samples) * 100

# Display the results
print("Positive Percentage: {:.2f}%".format(positive_percentage))
print("Negative Percentage: {:.2f}%".format(negative_percentage))
print("Neutral Percentage: {:.2f}%".format(neutral_percentage))
```

Output:

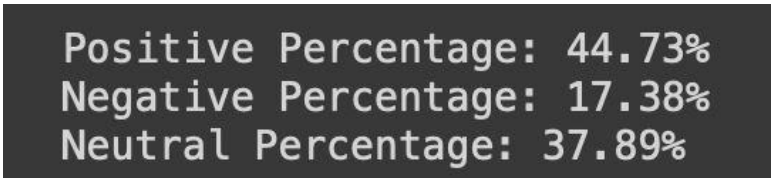


Figure 0–29 Sentiment distribution of SVM

- **XGBOOST**

```
pip install xgboost scikit-learn pandas
```

Output:

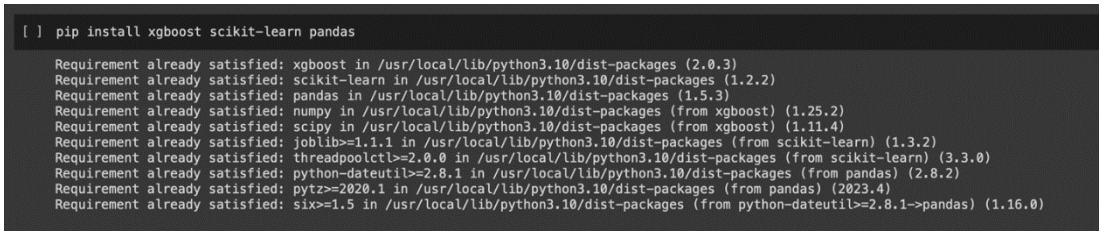


Figure 0–30 xgboost scikit-learn pandas

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
from xgboost import XGBClassifier
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import LabelEncoder
from textblob import TextBlob


# Load The dataset
data = pd.read_csv('/content/10Benzema20Neymar26RonaldoUPDATED.csv')


# Apply the sentiment analysis function to the 'preprocessed_text_tokenized'
column and create a new 'polarity' column
data['polarity'] = data['preprocessed_text_tokenized'].apply(lambda x:
TextBlob(x).sentiment.polarity)


# Assign labels based on polarity
data['polarity_label'] = data['polarity'].apply(lambda x: 1 if x > 0 else 0 if
x == 0 else -1)


# Split the dataset into training and testing sets
train_data, test_data, train_labels, test_labels = train_test_split(
    data['preprocessed_text_tokenized'], data['polarity_label'], test_size=0.2,
random_state=42
)




# Convert text data to feature vectors using CountVectorizer
vectorizer = CountVectorizer(max_features=5000)  # Adjusting The max_features
train_features = vectorizer.fit_transform(train_data)
test_features = vectorizer.transform(test_data)




# Use LabelEncoder to transform labels into consecutive integers
label_encoder = LabelEncoder()
train_labels_encoded = label_encoder.fit_transform(train_labels)
test_labels_encoded = label_encoder.transform(test_labels)


# Initialize XGBoost model
xgb_model = XGBClassifier()
```

```
# Train the model
xgb_model.fit(train_features, train_labels_encoded)

# Make predictions on the test set
predictions = xgb_model.predict(test_features)
# Evaluate the accuracy
accuracy = accuracy_score(test_labels_encoded, predictions)
print(f'Accuracy: {accuracy * 100:.2f}%')
# Evaluate the accuracy on the training set
train_predictions = xgb_model.predict(train_features)
train_accuracy = accuracy_score(train_labels_encoded, train_predictions)

# Print the scores on training and test set
print(f'Training Set Accuracy: {train_accuracy * 100:.2f}%')
print(f'Test Set Accuracy: {accuracy * 100:.2f}%')
```

Output:

```
Training Set Accuracy: 92.03%
Test Set Accuracy: 90.26%
```

Figure 0–31 XGBoost training and testing accuracy

```
from sklearn.metrics import precision_score, recall_score, f1_score

# Calculate precision, recall, and F1-score on the test set
precision = precision_score(test_labels_encoded, predictions,
average='weighted')
recall = recall_score(test_labels_encoded, predictions, average='weighted')
f1 = f1_score(test_labels_encoded, predictions, average='weighted')


print(f'Weighted Precision on Test Set: {precision:.4f}')
print(f'Weighted Recall on Test Set: {recall:.4f}')
print(f'Weighted F1-score on Test Set: {f1:.4f}')
```

Output:

```
Weighted Precision on Test Set: 0.9084
Weighted Recall on Test Set: 0.9026
Weighted F1-score on Test Set: 0.9016
```

Figure 0–32 XGBoost classification report

```
from sklearn.metrics import confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
# Generate confusion matrix
conf_matrix = confusion_matrix(test_labels_encoded, predictions)
# Print the confusion matrix
print("Confusion Matrix:")
print(conf_matrix)

# Visualize confusion matrix using seaborn heatmap
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt='d', cmap='Blues',
xticklabels=['Negative', 'Neutral', 'Positive'],
            yticklabels=['Negative', 'Neutral', 'Positive'])
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix')
plt.show()
```

Output:

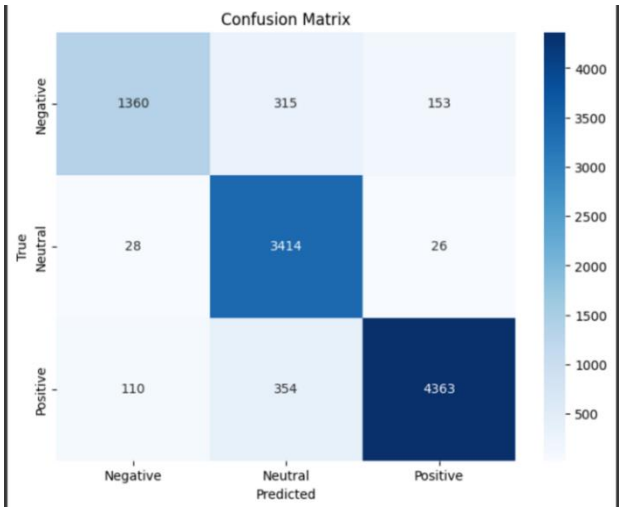


Figure 0–33 XGBoost Confusion Matrix

```
from sklearn.preprocessing import label_binarize
from sklearn.metrics import roc_auc_score, roc_curve, auc

# Binarize the labels for multi-class AUC
test_labels_bin = label_binarize(test_labels_encoded, classes=[0, 1, 2])  #
Adjust the classes based on problem
# Get predicted probabilities for each class on the test set
test_probs = xgb_model.predict_proba(test_features)


# Calculate AUC scores for each class
auc_scores = []
for i in range(test_labels_bin.shape[1]):
```

```
    auc_scores.append(roc_auc_score(test_labels_bin[:, i], test_probs[:, i]))

# Print AUC scores for each class
for i, auc_score in enumerate(auc_scores):
    print(f'AUC Score (Class {i}): {auc_score:.4f}')


# Plot ROC curves for each class
plt.figure(figsize=(8, 6))
for i in range(test_labels_bin.shape[1]):
    fpr, tpr, _ = roc_curve(test_labels_bin[:, i], test_probs[:, i])
    plt.plot(fpr, tpr, lw=2, label=f'ROC curve (Class {i}) (area =
{auc_scores[i]:.4f})')


plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curves for Multi-Class')
plt.legend(loc='lower right')
plt.show()
```

Output:

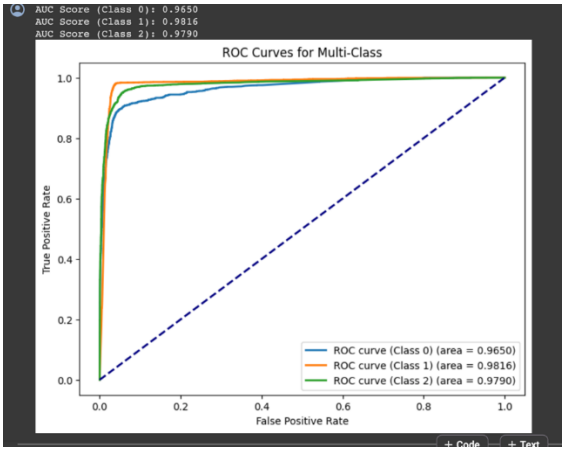


Figure 0–34XGBoost ROC/AUC

- **LGBM**

```
import pandas as pd
from textblob import TextBlob
import matplotlib.pyplot as plt # data visualization
import seaborn as sns # statistical data visualization
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
# Load The dataset
file_path = '/content/10Benzema20Neymar26RonaldoUPDATED1.csv'
data = pd.read_csv(file_path, low_memory=False)

# Specify the tokenized text column in DataFrame
text = 'preprocessed_text_tokenized'

# Function to calculate polarity
def get_polarity(text):
    analysis = TextBlob(text)
    return analysis.sentiment.polarity

# Apply the sentiment analysis function to the 'text_data' column and create
a new 'polarity' column
data['polarity'] = data[text].apply(get_polarity)
# Assign labels based on polarity
data['polarity_label'] = data['polarity'].apply(lambda x: 'positive' if x >
0 else 'negative' if x < 0 else 'neutral')

data['polarity_label'] = data['polarity_label'].map({'negative': -
1,'neutral': 0, 'positive': 1})
data

# Import the TfidfVectorizer class from scikit-learn and initialize it
tfidf = TfidfVectorizer(stop_words='english', max_features=5000)
# Transform the preprocessed and tokenized text data into TF-IDF matrix.
X = tfidf.fit_transform(data['preprocessed_text_tokenized'])
# Extract the target labels from the DataFrame, which represent the
sentiment polarity.
y = data['polarity_label']

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Import the LightGBM library
import lightgbm as lgb
```

```
# Initialize the LightGBM classifier
clf = lgb.LGBMClassifier()

# Train the classifier on the training data
clf.fit(X_train, y_train)
```

Output:

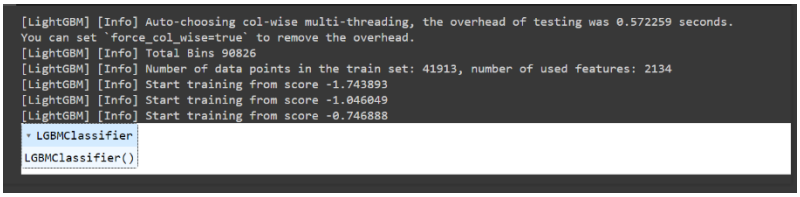


Figure 0–35 LightGBM library

```
# Predict the results using the trained classifier
y_pred = clf.predict(X_test)
# Importing the accuracy_score function from the scikit-learn library
from sklearn.metrics import accuracy_score

# Calculating the accuracy score
accuracy = accuracy_score(y_pred, y_test)

# Create a DataFrame with Text, Actual_Labels, and Predicted_Labels columns
result_df = pd.DataFrame({
    'Text': X_test,
    'Actual_Labels': y_test,
    'Predicted_Labels': y_pred
})

# Save the DataFrame to a CSV file
result_df.to_csv("/content/test_predictionsLGBM.csv", index=False)


# Displaying the accuracy score
print('LightGBM Model accuracy score:
{0:0.4f}'.format(accuracy_score(y_test, y_pred)))

#Compare train and test set accuracy
y_pred_train = clf.predict(X_train)
print('Training-set accuracy score: {0:0.4f}'.
format(accuracy_score(y_train, y_pred_train)))

# print the scores on training and test set
print('Training set score: {:.4f}'.format(clf.score(X_train, y_train)))
print('Test set score: {:.4f}'.format(clf.score(X_test, y_test)))
```

Output:

```
Training set score: 0.9059
Test set score: 0.8881
```

Figure 0–36LGBM Training and testing accuracy

```
# Print the Confusion Matrix and slice it into four pieces
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred)
print('Confusion matrix\n\n', cm)
print('\nTrue Positives(TP) = ', cm[0,0])
print('\nTrue Negatives(TN) = ', cm[1,1])
print('\nFalse Positives(FP) = ', cm[0,1])
print('\nFalse Negatives(FN) = ', cm[1,0])

# visualize confusion matrix with seaborn heatmap
cm_matrix = pd.DataFrame(data=cm, columns=['Actual Negative:-1', 'Actual
neutral:0', 'Actual Positive:1'],
                                 index=['Predict Negative:-1', 'Predict
neutral:0', 'Predict Positive:1'])

sns.heatmap(cm_matrix, annot=True, fmt='d', cmap='YlGnBu')
```

Output:

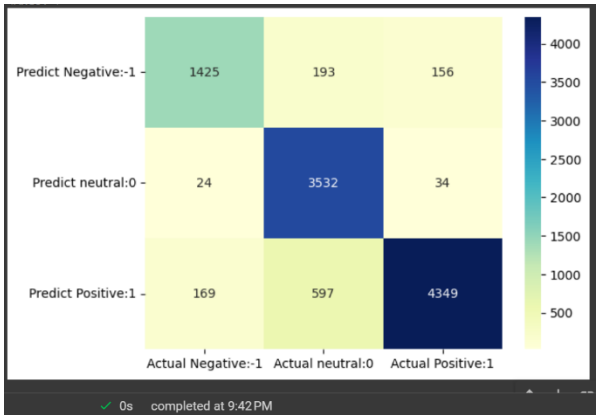


Figure 0–37 LGBM Confusion Matrix

```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

Output:

	precision	recall	f1-score	support
-1	0.88	0.80	0.84	1774
0	0.82	0.98	0.89	3590
1	0.96	0.85	0.90	5115
accuracy			0.89	10479
macro avg	0.89	0.88	0.88	10479
weighted avg	0.90	0.89	0.89	10479

Figure 0–38 LGBM Classification report

```
from sklearn.metrics import precision_score, recall_score, f1_score

# Calculate precision, recall, and F1-score on the test set
precision = precision_score(y_test, y_pred, average='weighted')
recall = recall_score(y_test, y_pred, average='weighted')
f1 = f1_score(y_test, y_pred, average='weighted')

print(f'Weighted Precision on Test Set: {precision:.4f}')
print(f'Weighted Recall on Test Set: {recall:.4f}')
print(f'Weighted F1-score on Test Set: {f1:.4f}')
```

Output:

```
Weighted Precision on Test Set: 0.8968
Weighted Recall on Test Set: 0.8881
Weighted F1-score on Test Set: 0.8879
```

Figure 0–39 Precision, recall, F1-score

```
from sklearn.metrics import roc_auc_score

# Predict the probabilities for each class
y_pred_prob = clf.predict_proba(X_test)

# Compute OvR ROC AUC
roc_auc_ovr = roc_auc_score(y_test, y_pred_prob, multi_class='ovr')
print('OvR ROC AUC Score: {0:0.4f}'.format(roc_auc_ovr))
```

Output:

```
OvR ROC AUC Score: 0.9598
```

Figure 0–40 LGBM ROC/AUC

```
# Read the test predictions file
data = pd.read_csv("/content/test_predictionsLGBM.csv")

# Extract the "Predicted_Labels" column
predicted_labels = data["Predicted_Labels"]

# Count the occurrences of each label
positive_count = (predicted_labels == 1).sum()
negative_count = (predicted_labels == -1).sum()
neutral_count = (predicted_labels == 0).sum()

# Calculate the percentage of each label category
total_samples = len(predicted_labels)
positive_percentage = (positive_count / total_samples) * 100
negative_percentage = (negative_count / total_samples) * 100
neutral_percentage = (neutral_count / total_samples) * 100
```

```
# Display the results
print("Positive Percentage: {:.2f}%".format(positive_percentage))
print("Negative Percentage: {:.2f}%".format(negative_percentage))
print("Neutral Percentage: {:.2f}%".format(neutral_percentage))
```

Output:

```
Positive Percentage: 43.87%
Negative Percentage: 15.66%
Neutral Percentage: 40.47%
```

Figure 0–41 Sentiment distribution of LGBM