Title: Textual Sentiment Analysis, Unveiling Emotions Through Natural Language Processing and Machine Learning Methods from tweets.

Student Name: Talha Ahmed Khan

Supervisor Name: Basma Abdulaimma

A thesis submitted in partial fulfilment of the

requirements of Liverpool John Moores University for the degree of Masters in Artificial Intelligence (Machine Learning)

22nd, September 2023

# **Abstract**

Sentiment analysis, a pivotal field in natural language processing, empowers organizations and individuals to gain valuable insights from textual data by discerning emotional tones and attitudes. This project embarks on a journey to enhance sentiment analysis specifically tailored for Twitter data, a domain rife with nuances, abbreviations, and cultural references. Our objective is to develop a robust sentiment analysis system that not only accurately captures sentiments but also adapts to the idiosyncrasies of Twitter discourse. The project commences with comprehensive data collection, encompassing diverse tweets from various sources, reflecting the rich tapestry of human emotions and expressions. We explore pre-processing techniques, encompassing lowercasing, punctuation removal, lemmatization, and stopword handling, to cleanse and normalize the Twitter text. In the heart of the project, we deploy three distinct machine learning models: a traditional decision tree with TF-IDF features, a Bidirectional GRU for context-rich understanding, and DistilBERT for efficient and effective feature extraction. Each model is meticulously selected based on resource efficiency and its ability to capture Twitter-specific sentiment nuances. The evaluation phase involves rigorous testing against labeled datasets, ensuring our models are fine-tuned to achieve high precision and recall. Model-based post-processing, which includes threshold adjustment and sentiment class mapping, further refines our sentiment predictions, aligning them with real-world sentiment expressions. The project's culmination is an efficient, accurate, and adaptable sentiment analysis system, tailored for Twitter data. We envision its application across diverse domains, from brand monitoring to political sentiment tracking, where understanding the Twitterverse's pulse holds immense value. Our project not only contributes a refined sentiment analysis approach but also paves the way for the exploration of Twitter's intricate sentiments, empowering businesses, researchers, and decision-makers with deeper insights into the ever-evolving landscape of social media sentiment.

# Acknowldgements

I wish to express my sincere gratitude to my supervisor, Dr. Basma Abdulaima, for their invaluable support and guidance throughout this research. I would also like to thank Dr. Carl Chalmers and Dr. Paul Fergus for imparting foundational knowledge in data science that enabled me to undertake this thesis. Additionally, I am thankful to Dr. Rubem Perierra whose excellent mentoring during my graduate course gave me the confidence to take on this complex topic without being overwhelmed. The completion of this research would not have been possible without the patient assistance and encouragement of these mentors. Their wisdom and insights have pushed me to accomplish more than I thought I was capable of. I cannot thank them enough for sharing their expertise and empowering me to succeed.

[Abstract 2](#_Toc146282664)

[Acknowldgements 2](#_Toc146282665)

[1 Chapter 1: Introduction 6](#_Toc146282666)

[2 Chapter 2: Background and Literature review 9](#_Toc146282667)

[2.1 Related Work 10](#_Toc146282668)

[2.2 Theory 18](#_Toc146282669)

[2.3 Legal Social Ethical and Professional Considerations 20](#_Toc146282670)

[3 Chapter 3: Analysis and Design 21](#_Toc146282671)

[3.1 Data Description 21](#_Toc146282672)

[3.2 Exploratory Data Analysis(EDA) 22](#_Toc146282673)

[**3.2.1** **Missing values** 23](#_Toc146282674)

[**3.2.2** **Class balance** 24](#_Toc146282675)

[**3.2.3** **Duplicate values** 26](#_Toc146282676)

[3.2.4 Study Length Visualization 27](#_Toc146282677)

[**3.2.5** **Visualizing the stop words and their frequencies** 29](#_Toc146282678)

[**3.2.6** **Splitting the data** 31](#_Toc146282679)

[3.3 Data-preprocessing 31](#_Toc146282680)

[**3.3.1** **Feature Reduction** 32](#_Toc146282681)

[3.3.1.1 **Column drop** 33](#_Toc146282682)

[**3.3.1.2** **Class Mapping** 33](#_Toc146282683)

[**3.3.2** **Text normalization** 33](#_Toc146282684)

[**3.3.2.1** **Spelling Correction** 34](#_Toc146282685)

[**3.3.2.2** **Denoise Study** 35](#_Toc146282686)

[**3.3.2.3** **Remove Name Tags** 36](#_Toc146282687)

[**3.3.2.4** **Removing urls from the data** 37](#_Toc146282688)

[**3.3.2.5** **Removing punctuations and whitespaces** 38](#_Toc146282689)

[**3.3.2.6** **Lower case text** 40](#_Toc146282690)

[**3.3.2.7** **Replacing numbers** 41](#_Toc146282691)

[**3.3.2.8** **Remove stopwords** 41](#_Toc146282692)

[**3.3.2.9** **Lemmatization** 42](#_Toc146282693)

[3.4 Model training 42](#_Toc146282694)

[**3.4.1.1** **Traditional Machine Learning Models:** 43](#_Toc146282695)

[**3.4.1.2** **Word Embedding Models** 43](#_Toc146282696)

[**3.4.1.3** **Recurrent Neural Networks (RNNs)** 43](#_Toc146282697)

[**3.4.1.4** **Transformer-Based Models** 44](#_Toc146282698)

[**3.4.1.5** **Model selection** 44](#_Toc146282699)

[**3.4.2** **Model specific post-processing** 45](#_Toc146282700)

[**3.4.2.1** **Transformer based model:** 45](#_Toc146282701)

[**3.4.2.1.1** **Tokenization:** 45](#_Toc146282702)

[**3.4.2.1.2** **Padding** 46](#_Toc146282703)

[**3.4.2.1.3** **Input IDs and Attention Masks** 46](#_Toc146282704)

[**3.4.2.2** **Artificial Neural Networks(ANN):** 46](#_Toc146282705)

[**3.4.2.2.1** Term Frequency-Inverse Document Frequency (**TF-IDF)** 47](#_Toc146282706)

[**3.4.2.3** **Recurrent Neural Networks(RNN):** 47](#_Toc146282707)

[**3.4.2.3.1** **Label encoding:** 48](#_Toc146282708)

[**3.4.2.3.2** **Tokenization:** 48](#_Toc146282709)

[**3.4.2.3.3** **Padding** 49](#_Toc146282710)

[**3.4.2.3.4** **Word Embeddings** 50](#_Toc146282711)

[4 Chapter 4: Implementation 51](#_Toc146282712)

[4.1 Loading Data 51](#_Toc146282713)

[4.2 Exploratory Data Analysis(EDA) 52](#_Toc146282714)

[**4.2.1** **Data frame review** 52](#_Toc146282715)

[**4.2.2** **Missing values** 52](#_Toc146282716)

[**4.2.3** **Class balance** 52](#_Toc146282717)

[**4.2.4** **Removing duplicate values** 53](#_Toc146282718)

[**4.2.5** **Token and character length analysis** 53](#_Toc146282719)

[**4.2.6** **Feature reduction** 54](#_Toc146282720)

[**4.2.7** **Splitting dataset** 54](#_Toc146282721)

[4.3 Data Pre-processing 54](#_Toc146282722)

[**4.3.1** **Text normalization** 54](#_Toc146282723)

[**4.3.1.1** **Spelling correction** 54](#_Toc146282724)

[**4.3.1.2** **Denoising Text** 55](#_Toc146282725)

[**4.3.1.3** **Remove name tags** 56](#_Toc146282726)

[**4.3.1.4** **Removing URLs** 56](#_Toc146282727)

[**4.3.1.5** **Removing punctutations** 57](#_Toc146282728)

[**4.3.1.6** **Removing lower case** 57](#_Toc146282729)

[**4.3.1.7** **Replacing numbers** 58](#_Toc146282730)

[**4.3.1.8** **Stopwords removal** 59](#_Toc146282731)

[**4.3.1.9** **Lemmatization** 60](#_Toc146282732)

[**4.3.2** **Model based post-processing** 60](#_Toc146282733)

[**4.3.2.1** **Decision Tree Classifier** 61](#_Toc146282734)

[**4.3.2.1.1** **TF-IDF** 61](#_Toc146282735)

[**4.3.2.2** **Bi-directional GRU with Glove twitter word embeddings** 62](#_Toc146282736)

[**4.3.2.2.1** **Label encoding** 62](#_Toc146282737)

[**4.3.2.2.2** **Tokenization** 62](#_Toc146282738)

[**4.3.2.2.3** **Word embeddings** 63](#_Toc146282739)

[**4.3.2.3** **DistilBERT Model** 64](#_Toc146282740)

[**4.3.2.3.1** **Tokenization** 64](#_Toc146282741)

[**4.3.2.3.2** **Input IDs and attention Masking** 64](#_Toc146282742)

[**4.3.2.3.3** **Padding** 64](#_Toc146282743)

[4.4 Model training 64](#_Toc146282744)

[**4.4.1** **Decision tree** 65](#_Toc146282745)

[**4.4.2** **Bi-directional GRU with word embeddings** 65](#_Toc146282746)

[**4.4.2.1** **Hyperparameters** 66](#_Toc146282747)

[**4.4.3** **DistilBERT** 67](#_Toc146282748)

[**4.4.3.1** **hyperparameters** 68](#_Toc146282749)

[5 Chapter 5: Evaluation 68](#_Toc146282750)

[5.1 Evaluation test selection 68](#_Toc146282751)

[5.2 Evaluations metrics selection 69](#_Toc146282752)

[5.3 Results 70](#_Toc146282753)

[5.3.1 Decision Tree 70](#_Toc146282754)

[**5.3.2** **Bi-directional GRU with word embeddings** 72](#_Toc146282755)

[**5.3.3** **DistilBERT** 77](#_Toc146282756)

[6 Chapter 6: Conclusion 82](#_Toc146282757)

[7 Bibliography 83](#_Toc146282758)

[8 Appendix 89](#_Toc146282759)

# **Chapter 1: Introduction**

The complex domain of human emotions, characterised by its numerous and nuanced characteristics, acts as a significant manifestation of individual personality and behavioural attributes. In the course of their everyday routines, individuals encounter a wide range of experiences and react emotionally to various stimuli. These stimuli can include matters of importance, noteworthy occurrences, social interactions, or simply the small details of their immediate surroundings. In order to communicate these emotions to their social networks, individuals utilise a wide range of communication strategies. Historically, emotions were conventionally conveyed through verbal communication and facial gestures. Nevertheless, the emergence of technology and the widespread use of social networks have brought about a significant change in the manner in which emotions are conveyed within social circles comprising friends and acquaintances(Steinert and Dennis, 2022). In the context of social networks, despite the advancements in technology enabling the transmission of emotions through audio and video, it is noteworthy that text-based communication continues to be the prevailing medium for emotional expression(Schreiner et al., 2021). Despite the availability of multiple media platforms for expressing emotions, writing remains the dominant and lasting channel via which individuals communicate their emotional states on social networks. This includes diverse forms of textual expression such as status updates, comments, blogs, and microblogs.

In this ever-changing environment, scholars from several fields like psychology, business, computer science, affective computing, and artificial intelligence have undertaken the challenging endeavour of identifying and interpreting emotions that are disguised inside written language. However, this endeavour is not without of complexities. The correct identification of human emotions within the study poses numerous obstacles due to the complex and nuanced nature of these emotional experiences(Gill, 2016). The aforementioned issues become more pronounced when a singular textual entity encompasses a wide range of emotions or when emotions are delicately implied, hence making the task of automatically detecting emotions a difficult objective to achieve. The realm of written communication is abundant with examples of sarcasm and subtle emotions that frequently pose difficulties for both human interpreters and automated systems(Rendalkar and Chandankhede, 2018).

In the course of examining this study, I will explore the methodologies, intricacies, and advancements pertaining to the identification of emotions conveyed in written language. This paper aims to investigate the significance of this undertaking in several academic fields and its potential to revolutionise the field of human-computer interaction, enhance our comprehension of human psychology, and propel the advancement of artificial intelligence.

As explained by (Gaye et al., 2021) Twitter, is an extensively used social media network with a worldwide user population, functions as a valuable resource for accessing public emotions and opinions on a wide range of subjects, including politics, entertainment, product evaluations, and societal issues. The use of sentiment analysis to Twitter data is not solely an academic endeavour; it possesses significant practical significance inside several sectors(Badugu and Suhasini, 2017). This article explores the significant importance of Twitter sentiment analysis, highlighting its crucial role in influencing decision-making, improving customer happiness, optimising marketing efforts, guiding research and policy formulation, and facilitating real-time customer service. Moreover, it underscores the significance of sentiment analysis in the contemporary landscape, particularly in tackling the obstacles presented by public health emergencies and other pivotal occurrences.

This project establishes a number of overarching purposes and corresponding objectives in order to explore and utilise sentiment analysis on Twitter data. The stated objectives span a range of essential tasks in the field, including data pre-processing techniques, model selection, model training, and performance evaluation these objectives were driven by the work of (Alec Go et al., 2009). These aims jointly contribute to the broader goal of enhancing the capabilities of sentiment analysis, specifically in the domain of identifying numerous unique emotions such as pleasure, fear, neutral, happiness, sadness, and others. The adoption of these aims is crucial for achieving a more profound comprehension of sentiment within Twitter data and, consequently, harnessing its potential across diverse applications.

The main objective of this study is to improve the accuracy of emotion recognition in Twitter data through the development of a resilient model. This model aims to expand its capacity beyond the standard categorization of neutral, positive, and negative emotions. The objective is to discern a more comprehensive range of emotions, incorporating subtle emotional states that beyond conventional sentiment classifications. The primary objective of this research is to enhance the comprehension of the emotional aspects found within social media content, hence providing a more comprehensive analysis.

The present study aims to broaden the scope of emotion recognition by not only detecting a wider array of emotions, but also by effectively classifying them. In contrast to traditional sentiment analysis, which predominantly classifies information into positive, negative, or neutral categories, the aim of this study is to construct a model that can effectively identify emotions such as joy, fear, sadness, happiness, and other subtle emotional states(Rathnayaka et al., 2019). The primary objective of this ambitious endeavour is to enhance the complexity of emotion analysis, hence facilitating a deeper understanding of user attitudes in various circumstances. The primary objective of this study is to enhance the precision of emotion recognition while also developing a flexible solution that can be applied across several domains. The objective of this endeavour is to develop a versatile model that can be applied across diverse domains such as marketing, customer service, mental health, and social research. The primary objective of this study is to develop a versatile tool that can effectively cater to the specific needs of many sectors. This will ultimately enhance the practical implications of emotion detection.

The initial aim of this study pertains to the enhancement of data preparation methodologies. This involves the full process of text tokenization, the reduction of noise, and the effective management of special characters. The aim is to execute a novel methodology that not only guarantees the integrity of data but also enhances the precision of the model. The primary objective of this study is to establish a robust framework for accurate emotion identification through the optimisation of data pre-processing techniques. The second objective involves the careful selection of models. The process entails a comprehensive assessment of diverse machine learning and deep learning models, considering essential elements such as model architecture and pre-trained embeddings. The aim of this study is to determine the optimal model that is in line with the research objectives of achieving robust and accurate emotion recognition. The selection process is a vital component in the pursuit of the study objectives. The third purpose pertains to the aspect of data variety. To mitigate bias during the training of the model, the objective of this research is to carefully curate a dataset that surpasses any constraints associated with certain target audiences. For example, tweets exclusively focused on COVID-19 may exhibit the sentiments of a restricted demographic. To enhance the model's versatility and application across many industries, the research endeavours to expand the training data by incorporating a wider range of situations and topic matter. The purpose of this objective is to provide the model with the necessary flexibility to succeed in several domains beyond its original training focus.

The extension of sentiment analysis beyond conventional binary or ternary classifications to a more detailed six-class emotion detection system presents numerous prospects and benefits. The incorporation of sentiment analysis not only facilitates a more nuanced comprehension of user feelings but also permits enterprises to customise their replies, personalise user experiences, and extract more significant insights from user-generated information(S.S. Verma, 2021). The increased capacity of this skill has the potential to have a favourable influence on various sectors, including customer service, marketing, market research, healthcare, content development, and other related areas. Moreover, this fits with the dynamic nature of sentiment research, enabling organisations to leverage more comprehensive emotional insights to enhance decision-making, enhance customer happiness, and gain a stronger competitive edge. This introduction establishes the context for a thorough investigation of Twitter sentiment analysis, its significance, and its capacity to facilitate progress in several domains. The subsequent sections of this paper will explore the technical facets involved in attaining these aims, ultimately leading to a more comprehensive comprehension of how sentiment analysis might stimulate favourable transformations in many businesses.

# **Chapter 2: Background and Literature review**

Sentiment analysis, a fundamental component of Natural Language Processing (NLP), has witnessed substantial growth in recent years. This expansion is underpinned by the ubiquitous presence of user-generated text on social media platforms, particularly Twitter. Sentiment analysis on Twitter data offers valuable insights into the emotional undercurrents of online conversations. While conventional sentiment analysis primarily categorizes text as positive, negative, or neutral, recent endeavors in the field have pushed the boundaries to recognize a more diverse range of emotions. This project seeks to contribute to this evolving landscape by advancing the precision and versatility of emotion detection within Twitter data.

## **Related Work**

The emergence of social media platforms has catalysed a shift in how people express themselves and interact online. Twitter, with its concise and real-time nature, has become a rich source of textual data that encapsulates a wide spectrum of emotions. Early sentiment analysis efforts on Twitter predominantly focused on classifying tweets as either positive, negative, or neutral(Goyal, 2021). While these models served certain analytical purposes, they fell short in capturing the nuanced emotional states that users often convey. Recent advances in sentiment analysis have recognized the need for a more comprehensive approach. Emotion recognition models now aim to classify tweets into multiple distinct emotions, such as joy, fear, happiness, sadness, and others. This shift reflects a deeper understanding of human emotional expression, acknowledging that sentiments are not binary but exist along a continuum. This project aligns with this evolving sentiment analysis paradigm, seeking to develop a robust model capable of identifying a broad array of emotional states in Twitter data.

In a study by (Nasim et al., 2017) Sentiment Analysis is a crucial task in natural language processing and has been explored through three primary approaches which are machine learning-based, lexicon-based, and hybrid methods. Firstly, machine learning Based Approaches which focuses on building predictive models using training data and then evaluate these models on test data. They can be categorized into supervised and unsupervised methods. Unsupervised methods, as exemplified by (Turney, 2002)’s work, determine polarity by aggregating adjectives and adverbs in phrases. (Fernández-Gavilanes et al., 2016) used dependency parsing to compute semantic orientation based on sentiment lexicons. Supervised methods require labelled training data and employ linguistic features like n-grams, word representations, part of speech tags, punctuation, and emoticons to train classifiers. For instance, (Altrabsheh et al., 2014) used n-gram features and algorithms like Naive Bayes, Maximum Entropy, and Support Vector Machine (SVM) for sentiment prediction. Secondly, Lexicon-based sentiment analysis which relies on sentiment lexicons or dictionaries to determine Study polarity. Words are associated with predefined sentiment polarities, which can be constructed manually or automatically. (Hu and Liu, 2004) used WordNet to predict opinion word sentiment, and (Taboada et al., 2011) proposed a similar approach. Various general-purpose and domain-specific lexicons have been created. (Rajput et al., 2016) adapted a general-purpose sentiment dictionary for academic feedback, demonstrating the value of domain-specific lexicons. Finally, Hybrid approach combine sentiment lexicons with machine learning techniques. (Zhang et al., n.d.) used an opinion lexicon to label training data and train a binary classifier for sentiment prediction. (Appel et al., 2016) extended a sentiment lexicon with SentiWordNet and fuzzy sets for sentence-level sentiment analysis. This study presents a hybrid approach that combines sentiment lexicons and machine learning for analysing student feedback. In the presented hybrid approach, author combines sentiment dictionaries with machine learning methods to determine the sentiment of textual feedback provided by students. The methodology of this study involves dataset description, pre-processing (including tasks like removing punctuations, tokenization, case conversion, and stop word removal), data partitioning into training and testing sets, feature extraction (utilizing TF-IDF, N-grams, and lexicon features), and model training (using Random Forest and SVM). Evaluation metrics such as accuracy and F-measure were employed to assess the model's performance. the results show that the hybrid approach, combining TF-IDF and lexicon-based features, outperformed other methods, including lexicon-based approaches and sentiment analysis APIs (Microsoft's text Analytics API, Alchemy Language API, and Aylien textAPI). This approach accounted for domain-specific sentiment and achieved a 2% improvement over the lexicon-based approach. This study concluded that hybrid approach, which leverages sentiment dictionaries along with machine learning, offers a promising solution for sentiment analysis in the academic domain. It outperforms both lexicon-based methods and external sentiment analysis APIs, demonstrating the effectiveness of incorporating domain-specific features in sentiment analysis.

In a study by (Sarlan et al., 2014) discusses various aspects of sentiment analysis, including approaches, data pre-processing, model architectures, classes detected (positive, negative, and null), feature extraction techniques, and model performance. Let’s start with the sentiment analysis approach used by them. Firstly, the study mentions a lexicon-based approach, where predefined lists of words are associated with specific sentiments (positive or negative). This approach involves calculating the sentiment orientation of Study based on the words in it. It is relatively straightforward and interpretable, but it may not handle context well. Secondly, Machine Learning-Based Approach relies on supervised classification methods, such as Support Vector Machines (SVM) and Artificial Neural Networks (ANN). It requires labelled data for training and can capture more complex patterns in the Study but may require more data and computational resources. Thirdly, data Pre-processing Techniques consists of typical pre-processing steps include tokenization, removing stopwords, handling special characters, and stemming or lemmatization to reduce words to their root forms. Fourthly, Model Architectures used the study mentions SVM and ANN as model architectures. SVM is known for its effectiveness in Study classification, and ANNs, especially deep learning models, have shown excellent performance in various NLP tasks. However, the Study doesn't provide specifics on the architecture or deep learning models used. fifthly, the study mentions categorizing sentiments into "positive," "negative," and "null." This is a common approach, where "positive" and "negative" represent sentiments, while "null" may indicate neutral or unclassified sentiments. Finally, feature extraction is briefly touched upon, with references to lexicons and frequency of words. Lexicons can be used to assign sentiment scores to words. The frequency of specific words can be used as features. Additionally, techniques like TF-IDF (Term Frequency-Inverse Document Frequency) or word embeddings (Word2Vec, GloVe) are common for feature extraction in NLP tasks. Moreover, the Study mentions that the program categorizes sentiment into positive and negative, represented in a pie chart and HTML page. However, it doesn't provide specific details on the model's performance metrics (accuracy, precision, recall, F1-score) or how well it generalizes to new data. In summary, the Study describes a sentiment analysis project that employs both lexicon-based and machine learning-based approaches. It discusses some pre-processing steps, mentions SVM and ANN as model architectures, and categorizes sentiments into three classes. Feature extraction techniques such as lexicons and word frequencies are referenced. However, the Study lacks specific details about model performance or the deep learning architecture used, making it challenging to assess the overall effectiveness of the sentiment analysis system. For a comprehensive evaluation, specific metrics and results would be needed.

In the study by (A. and Sonawane, 2016), we delved into sentiment analysis, a process that employs Natural Language Processing (NLP) to automatically mine attitudes, opinions, views, and emotions from a variety of textual sources, including Study, speech, tweets, and databases. Sentiment analysis, often referred to as subjectivity analysis, opinion mining, or appraisal extraction, involves the categorization of textual opinions into sentiment categories such as 'positive,' 'negative,' or 'neutral.' In this context, it's crucial to differentiate between related terms like 'opinion,' 'view,' 'belief,' and 'sentiment,' where each term signifies distinct aspects of expressing one's feelings or judgments. To facilitate sentiment analysis, we employed a set of terminologies and data collection techniques, utilizing Twitter data as a dynamic source to investigate emotions and opinions in the Study. The research encompassed various sentiment analysis tasks, including sentiment extraction, classification, subjectivity assessment, and opinion summarization, all aimed at unravelling people's sentiments, attitudes, and emotions towards diverse subjects, such as products, individuals, topics, organizations, and services. Through this comprehensive analysis, we aimed to shed light on the intricate landscape of sentiment analysis methodologies and their application in understanding the rich tapestry of human expression on social media platforms.

In another study by (Prema Arokia Mary et al., 2021), In order to conduct sentiment analysis, the process involves acquiring a dataset, which can be obtained either dynamically from Twitter using the Twitter API or non-dynamically from existing data sources such as Kaggle. The dataset must then undergo a series of pre-processing steps to enhance its machine readability and structure. These pre-processing steps include converting Study to lowercase, removing URLs, eliminating stopwords, lemmatization, converting emoticons to words, tokenization (using Punkt), and employing Wordnet linkage. Once the data is pre-processed, it becomes more suitable for machine understanding. Subsequently, the sentiment classification is carried out using various machine learning algorithms including Logistic Regression, Linear SVC, Random Forest, Bernoulli NB, Decision Tree, Voting Classifier, and KNN. This study also focuses on detecting the positive, negative and neutral sentiments. Firstly, the dataset for this analysis was obtained from Kaggle and is known as the "Sentimental Analysis with tweets" dataset, which comprises a substantial 1.6 million tweets. Secondly, the pre-processing phase is a crucial step in sentiment analysis as it serves to eliminate noise and redundancy in the data, rendering it unambiguous and reliable. Several key pre-processing steps are performed, including converting Study to lowercase, removing URLs, stopwords, lemmatization, emoticons to words conversion, tokenization, and the use of Wordnet for defining word polarity. Thirdly, Various machine learning classification algorithms are employed in this study to perform sentiment analysis. These include Logistic Regression, Linear SVC, Random Forest Classifier, Bernoulli NB, Decision Tree Classifier, Voting Classifier, and KNN Classifier. Logistic Regression, for instance, is a supervised algorithm particularly well-suited for categorical target variables. It analyses multiple explanatory variables simultaneously, reducing the impact of confounding factors. Linear SVC is another algorithm used for multi-class classification, while Random Forest Classifier employs multiple decision trees to achieve high accuracy. Fourthly, the study aims to compare the performance of these algorithms on a Twitter dataset with equal positive and negative sentiments. Evaluation metrics such as accuracy, specificity, sensitivity, and Area Under Curve (AUC) were employed. Notably, Logistic Regression, Linear SVC, and the Voting Classifier outperformed other algorithms. Linear SVC achieved an accuracy of 98.83% on training data and 77.52% on testing data, indicating its superior performance in sentiment analysis. Finally, In conclusion, this study proposed and implemented sentiment analysis methodologies for Twitter data. The process involved data collection, pre-processing, and the application of various machine learning classification algorithms. Notably, Linear SVC and Logistic Regression exhibited high accuracy rates, with Linear SVC achieving an accuracy of 98.859% on training data and Logistic Regression reaching 77.306% on testing data. These findings highlight the effectiveness of these algorithms in sentiment analysis on Twitter data.

In a study by (Ji et al., 2015) to perform sentiment analysis, this study focuses on dataset acquisition, which can be dynamically sourced from Twitter using the Twitter API or non-dynamically obtained from established data repositories like Kaggle. The dataset then undergoes an essential pre-processing phase to enhance its machine readability and structural consistency. This pre-processing encompasses several vital steps, including converting Study to lowercase, URL removal, stopword elimination, lemmatization, emoticon-to-word conversion, tokenization using Punkt, and Wordnet linkage for word polarity definition. Firstly, the dataset employed in this analysis originates from Kaggle, titled the "Sentimental Analysis with tweets" dataset, featuring a substantial corpus of 1.6 million tweets. Secondly, the pre-processing stage stands as a critical stride in sentiment analysis, effectively removing noise and redundancy from the data, thus making it unambiguous and dependable. The pre-processing steps include lowercase Study conversion, URL removal, stopwords elimination, lemmatization, emoticons-to-words conversion, tokenization, and Wordnet integration for word polarity determination. Thirdly, A diverse set of machine learning classification algorithms is harnessed in this study for sentiment analysis. These encompass Logistic Regression, Linear SVC, Random Forest Classifier, Bernoulli NB, Decision Tree Classifier, Voting Classifier, and KNN Classifier. Logistic Regression, for instance, is a supervised algorithm well-suited for categorical target variables, capable of analysing multiple explanatory variables concurrently to mitigate the influence of confounding factors. Linear SVC is another multi-class classification algorithm, while Random Forest Classifier employs multiple decision trees to achieve remarkable accuracy. Fourthly, the primary objective of this study is to evaluate the performance of these algorithms using a Twitter dataset with an equal distribution of positive and negative sentiments. Evaluation metrics such as accuracy, specificity, sensitivity, and Area Under Curve (AUC) are employed. Notably, Logistic Regression, Linear SVC, and the Voting Classifier emerge as top performers, outshining other algorithms. Linear SVC attains an impressive accuracy of 98.83% on training data and 77.52% on testing data, showcasing its superior performance in sentiment analysis. Finally, In summation, this study introduces a comprehensive sentiment analysis framework tailored for Twitter data. The methodology encompasses data collection, pre-processing, and the application of various machine learning classification algorithms. Particularly noteworthy are the remarkable accuracy rates achieved by Linear SVC and Logistic Regression. Linear SVC demonstrates exceptional performance with an accuracy of 98.859% on training data, while Logistic Regression achieves 77.306% on testing data. These findings underscore the efficacy of these algorithms in the domain of sentiment analysis for Twitter data.

In a study by (Venugopalan and Gupta, 2015) twitter sentiment analysis was performed. Firstly, approaches for Sentiment Analysis are discussed, the study introduces a lexicon-based strategy, associating predefined word lists with specific sentiments (positive or negative). Their method hinges on calculating sentiment orientation based on the presence of these words. It's relatively straightforward and interpretable but may lack nuance in handling contextual variations. moreover, in contrast, the machine learning-based approach, employing techniques like Support Vector Machines (SVM) and Artificial Neural Networks (ANN), is mentioned. This approach demands labelled data for training and has the capacity to capture intricate textual patterns. However, it may necessitate more extensive data and computational resources. Secondly, the study provides a cursory overview of data pre-processing techniques. While specifics are limited, common pre-processing steps like tokenization, stopwords removal, handling special characters, and stemming or lemmatization are acknowledged. These steps are essential for refining the data and enhancing machine understanding. Thirdly, SVM and ANN are introduced as model architectures. SVM is recognized for its efficacy in Study classification, while ANN, particularly deep learning models, have demonstrated exceptional performance in various Natural Language Processing (NLP) tasks. However, the Study does not furnish explicit details concerning the architecture or the utilization of deep learning variants. Fourthly, The study delineates the categorization of sentiments into "positive," "negative," and "null." This conventional classification approach designates "positive" and "negative" as distinct sentiments, while "null" potentially indicates neutral or unclassified sentiments. fifthly, Feature extraction is briefly acknowledged, with reference to lexicons and word frequency. Lexicons serve the purpose of assigning sentiment scores to individual words, while the frequency of specific terms can be harnessed as features. Moreover, advanced techniques such as TF-IDF (Term Frequency-Inverse Document Frequency) or word embeddings (e.g., Word2Vec, GloVe) are recognized practices for feature extraction in NLP tasks. However, it refrains from offering specific insights into the model's performance metrics, including accuracy, precision, recall, and F1-score. Additionally, there is no information regarding the model's generalizability to unseen data. In summation, the Study delineates a sentiment analysis project that leverages both lexicon-based and machine learning-based approaches. It touches upon certain pre-processing steps, introduces SVM and ANN as model architectures, and outlines the classification of sentiments into three classes. While the Study acknowledges feature extraction techniques such as lexicons and word frequencies, it lacks detailed information about model performance and the specifics of any deep learning architecture used. Consequently, assessing the overall efficacy of the sentiment analysis system is challenging without specific metrics and results. For a comprehensive evaluation, precise performance indicators and outcomes would be indispensable.

The Study by (Gupta et al., 2021)Guptas et al. delves into various facets of sentiment analysis, encompassing the approach, data collection and pre-processing, methodology, results, and conclusion. Here, we dissect these components and draw comparisons. Firstly, to conduct Sentiment Analysis for Mental Health the study introduces the primary objective of conducting sentiment analysis on tweets, specifically to gain insights into the mental health of Twitter users. To make this possible Sentiment Analysis Tools such as TextBlob is employed for sentiment analysis, facilitating the extraction of tweet polarity (positive, negative, neutral) and subjectivity. Additionally, the approach employs well-established classifiers, including Naive Bayes and Support Vector Machines (SVM), for the classification of tweets based on their sentiment. Secondly, Data Collection and Pre-processing are discussed starting with the Data Source. Public Twitter emotion dataset from Kaggle serves as the data source, providing a corpus of tweets for analysis. Furthermore, data pre-processing is discussed which is an integral part of the approach and involves several key steps such as link removal, emoticon handling, tokenization, and stopwords removal. These steps enhance the quality and readiness of the data for analysis. Thirdly, their emotion classes are discussed such as Tweet Classification primarily focuses on classifying tweets based on their polarity (positive, negative, neutral) and subjectivity. Higher negative polarity is interpreted as an indicator of potential mental health concerns, while increased subjectivity suggests the presence of personal opinions within tweets. Fourthly, the analysis yields insights into the distribution of tweets across positive, negative, and neutral sentiments of the mental health sentiments in the dataset. To enhance understanding, the approach incorporates visual aids, including polarity/subjectivity scatter plots and word clouds. The proposed sentiment analysis model showcases noteworthy performance, boasting an accuracy rate of approximately 88%. This achievement positions it favourably in comparison to other existing sentiment analysis models. To conclude, the study underscores the utility of sentiment analysis on tweets to gain intuitive insights into the mental health of twitter users. To add Indicators of Mental Well-being, highlights that polarity and subjectivity metrics derived from tweets can potentially serve as indicators of users' mental well-being. The proposed sentiment analysis model is recognized for its robust performance in assessing tweet sentiments, providing valuable information for mental health analysis. In summary, the Study lays out a comprehensive sentiment analysis approach, with a specific focus on assessing mental health through tweet polarity and subjectivity. It delineates a systematic pipeline encompassing data collection, pre-processing, classification, and results visualization. Notably, the proposed model achieves commendable accuracy, positioning it as an effective tool for sentiment analysis, particularly in the context of mental health assessment.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Author | Approach | Data | Pre-processing | Model | Classes |
| (Nasim et al., 2017) | ML-based, Lexicon based and Hybrid | Manually labelled | Punctuation removal, lowercasing, tokenization and stop words removal. | Random forest and SVM | Positive, Negative and neutral |
| (Sarlan et al., 2014) | Lexicon Based and ML-based | Twitter API | Punctuation removal, tokenization, stopwords removal | SVM | Positive, Negative and neutral |
| (A. and Sonawane, 2016) | Lexicon Based and ML-based | Firehouse API | URL removal, spelling correction, Emoticon replacement, remove punctuations symbols and numbers, remove stopwords, Expand acronym, remove non-English tweets. | SVM and NB | Positive, Negative and neutral |
| (Prema Arokia Mary et al., 2021) | Hybrid | Kaggle dataset | Lowercasing, URL removal, stopwords removal, lemmatization, emoticon-to-word conversion, tokenization, Wordnet linkage | Logistic Regression, Linear SVC, Random Forest, Bernoulli NB, Decision Tree, Voting Classifier, KNN | Positive, Negative and neutral |
| (Ji et al., 2015) | Lexicon Based and ML-based | Kaggle dataset | Duplicate removal, special characters removal, URL replaced with’url’, ! and ? replaced with ‘excl’ and ‘ques’ remaining symbols were replaced by ‘symb’. | NB, SVM | Positive, Negative and neutral |
| (Venugopalan and Gupta, 2015) | Lexicon Based and ML-based | Twitter API | Spell check, Slang replacement, link removal. | SVM, J48 | Positive, Negative and neutral |
| (Gupta et al., 2021) | Hybrid(TextBlob with ANN) | Kaggle dataset | link removal, emoticon handling, tokenization, stopwords removal | NB and SVM | Positive, Negative and neutral |

The primary aim of this research is to enhance the precision of emotion detection within Twitter data by constructing a robust model. This model aspires to identify a broader spectrum of emotions beyond the conventional sentiment categories. By achieving this aim, the research aims to offer a more comprehensive understanding of the emotional landscape present in social media content. Furthermore, this study endeavors to expand the horizons of emotion recognition by not only detecting a broader range of emotions but also classifying them effectively. The objective is to develop a model capable of discerning emotions such as joy, fear, sadness, happiness, and other nuanced emotional states. This ambitious aim seeks to enrich the sophistication of emotion analysis, enabling a more profound comprehension of user sentiments across diverse contexts. Beyond academic curiosity, the practical applications of this research are significant. The ability to accurately detect a wide range of emotions in Twitter data has ramifications across various industries. In marketing, understanding consumer sentiments beyond positive or negative can inform tailored advertising strategies. In customer service, recognizing nuanced emotions can enhance support interactions. In mental health, monitoring emotional states in social media can contribute to early intervention. In social research, the capacity to analyze emotions comprehensively can shed light on evolving societal trends. Thus, this project aims to provide a versatile tool that can adapt to the unique requirements of different industries, extending the practical impact of emotion detection.

## **Theory**

To achieve the project's aims, several objectives are outlined. Firstly, data pre-processing techniques will be refined to ensure data cleanliness and improve model accuracy. Secondly, a meticulous model selection process will be carried out, considering crucial factors such as model architecture and pre-trained embeddings. Lastly, data diversity will be emphasized to equip the model with the adaptability required to excel in various domains.

The hypothesis that distinct data pre-processing methods can yield improved results in the multi-class classification of sentiment analysis is a compelling and well-founded proposition, grounded in the discussions and findings of prior research (Nasim et al., 2017; Sarlan et al., 2014; Prema Arokia Mary et al., 2021; Ji et al., 2015; Venugopalan and Gupta, 2015; Gupta et al., 2021). In the ever-evolving field of natural language processing, the quality and preparation of the input data play a pivotal role in the efficacy of sentiment analysis models. This hypothesis suggests that by tailoring data pre-processing techniques to the specific characteristics of the sentiment analysis task at hand, we can enhance the model's performance and its ability to accurately categorize sentiments across multiple classes, drawing inspiration from diverse strategies outlined in prior research.

This theory represents a promising avenue for research and experimentation, as it recognizes the dynamic and multifaceted nature of sentiment analysis in the context of different datasets and classification objectives. Through this exploration of diverse data pre-processing techniques, we aim to uncover more nuanced and accurate sentiment classifications, ultimately advancing the field of multi-class sentiment analysis. data pre-processing techniques to the specific characteristics of the sentiment analysis task at hand, we can enhance the model's performance and its ability to accurately categorize sentiments across multiple classes. Our hypothesis encompasses several critical aspects of data pre-processing. We believe that the retention of certain punctuation marks, such as exclamation and question marks, can provide valuable contextual cues for sentiment interpretation, potentially leading to more accurate sentiment classifications. Additionally, addressing grammatical issues, misspellings, and contractions can significantly enhance the quality of the textual data, resulting in improved sentiment understanding. Furthermore, the hypothesis proposes the conversion of numerical values instead of their removal, preserving valuable contextual information. It also challenges the conventional wisdom of removing usernames in Twitter data, suggesting that these elements may carry relevant contextual information for sentiment analysis. Lastly, the hypothesis acknowledges the importance of customizing the selection of stopwords, emphasizing the significance of retaining specific common stopwords commonly found in tweets, which could better align with the nature of informal Twitter sentiment analysis.

In the pursuit of our research goals, the training of three distinct models has been contemplated, taking inspiration from various model architectures discussed in the related work. Firstly, Traditional Decision Tree Model: This model, a stalwart in machine learning, provides a benchmark against which the performance of more complex models can be evaluated, akin to the approach in some prior studies (Nasim et al., 2017; Sarlan et al., 2014; Prema Arokia Mary et al., 2021). Secondly, Bidirectional Gated Recurrent Unit (GRU) Model: This architecture, as seen in the related work (Prema Arokia Mary et al., 2021; Ji et al., 2015; Gupta et al., 2021), has been selected for its ability to extract intricate features from the training data, emphasizing contextual understanding. Finally, DistilBERT Transformer-Based Model: The inclusion of a transformer-based model aligns with recent trends in natural language processing research (Prema Arokia Mary et al., 2021; Gupta et al., 2021), allowing us to assess the impact of distinctive data pre-processing techniques on such advanced architectures. The inclusion of these three models in our study enables a comprehensive evaluation of data pre-processing strategies, ranging from conventional to advanced techniques, within the context of sentiment analysis. This approach contributes to a nuanced understanding of the influence of data quality on model performance, building upon the foundations established in prior research.

## Legal Social Ethical and Professional Considerations

I begin by analysing the Legal Considerations. In addition to the aforementioned considerations, I am diligent in upholding Data Privacy regulations throughout my research. I ensure that all data I use adheres to relevant privacy laws, such as GDPR in Europe and HIPAA in the United States. Specifically, when dealing with Twitter datasets that may contain user location information, I take measures to comply with GDPR by anonymizing or aggregating location data, thus safeguarding the identities of individual users, especially those in the European region are explained on (GDPR, 2023). Additionally, if my dataset involves tweets related to personal health and emotions, such as discussions on mental health, I remain conscious of HIPAA regulations. Despite Twitter data not typically falling under HIPAA, I take care to exclude any personally identifiable health information from my dataset to ensure privacy and compliance. Moreover, I maintain the utmost respect for Intellectual Property rights throughout my research endeavors. When utilizing existing research findings and models, I make it a priority to provide proper attribution to sources and seek permissions as required by the intellectual property guidelines. Furthermore, I uphold ethical standards in data collection, particularly when gathering data from social media platforms like Twitter. In this regard, I strictly adhere to the platform's terms of service and guidelines for data usage, ensuring ethical and responsible data collection practices are integral to my research process. These considerations were developed with the help of (Rights (OCR), 2021).

Moving forward, I’ll explore the Social Considerations for my project. In conducting my research on emotion detection in Twitter data, I am attentive to the crucial aspects of bias and fairness. I acknowledge the potential biases that can permeate both my dataset and the models employed, recognizing that biased data or models can inadvertently perpetuate societal biases. To address this, I implement strategies aimed at mitigating bias, particularly when dealing with sensitive topics, as certain demographic groups might be overrepresented, potentially leading to skewed results. Furthermore, I am conscientious of cultural sensitivity, understanding that emotions are expressed diversely across cultures, and I take care to interpret emotional signals in a culturally nuanced manner to prevent misinterpretations. Additionally, I recognize the broader societal impact of my research, especially when applied in areas like politics or public opinion, and consider the potential consequences of sentiment analysis. To uphold transparency, I thoroughly document my data sources, methodologies, and model architectures, promoting trust and enabling others to replicate my work with confidence.

Consequently I’ll also adhere to ethical considerations and In my project, I prioritize several ethical considerations. Firstly, I ensure informed consent from human subjects involved in my research by transparently explaining the research purpose, potential risks, and data usage, particularly when working with Twitter data that associates emotions with specific users or accounts. Secondly, I uphold the principle of beneficence, striving to maximize research benefits while minimizing harm, especially when dealing with sensitive data. Lastly, I maintain stringent data handling practices to safeguard collected data, preventing unauthorized access and breaches to uphold data integrity and privacy. This includes implementing measures like data anonymization and refraining from disclosing sensitive personal information to protect user privacy. Finally, I’ll analyse the Professional Considerations. In my project, I adhere to various critical principles. I maintain academic integrity by avoiding plagiarism and ensuring that all sources and prior research used in my theory are properly cited and referenced. I also consider submitting my research theory and findings to peer-reviewed journals or conferences, allowing my work to undergo rigorous evaluation by experts in the field, ensuring its credibility. Active encouragement of collaboration with fellow researchers in the sentiment analysis field is another cornerstone, as sharing knowledge and insights fosters the advancement of sentiment analysis as a collective effort. Additionally, my research is conducted in an ethical and responsible manner, aligning with the guidelines and best practices of my academic or professional community. I contemplate sharing my data, code, and models openly with the research community, embracing open science principles that promote transparency and facilitate the replication of my work. By diligently addressing these legal, social, ethical, and professional considerations, I conduct my sentiment analysis research responsibly and contribute to the field's progress while upholding legal and ethical standards.

Addressing these legal, ethical, social, and privacy considerations will help I conduct my project responsibly and ethically while minimizing legal risks and promoting positive social impact. It's essential to stay up-to-date with relevant laws and regulations and consult legal experts when needed.

# **Chapter 3: Analysis and Design**

This section entails on

## **Data** **Description**

In our data description, we possess a dataset comprising 40,000 records, categorized into 13 distinct classes. Originally, this dataset comprises four columns. The foremost column designates the tweet ID, succeeded by the name of the author or user responsible for composing the tweet. Subsequently, the third column contains the precise content of the tweet in question, while the fourth column encapsulates the sentiment conveyed within said content. This dataset has been procured from (Sentiment Analysis in Study - dataset by crowdflower, 2023). It is noteworthy that the exploratory data analysis techniques employed for this dataset exhibit notable divergence from those typically applied to other data modalities such as audio, images, or videos. Further elucidation on this matter will be provided within the forthcoming section dedicated to Exploratory Data Analysis (EDA).

## **Exploratory Data Analysis(EDA)**

The exploration of data undertaken, as elucidated by (Morgenthaler, 2009), commenced with an examination of the dataset's headers, a crucial initial step when dealing with tabular data. This scrutiny revealed the presence of four columns within the dataset. It is imperative to underscore that Exploratory Data Analysis (EDA) constitutes a pivotal facet of any data analysis endeavor, including Study classification. EDA serves as the vanguard in comprehending the dataset's inherent characteristics, discerning patterns, anomalies, and potential intricacies that could exert influence on the model's efficacy. During the EDA phase of a Study classification undertaking, it is incumbent to scrutinize the dataset's structural dimensions, encompassing the enumeration of records and columns. The perusal of a few initial rows facilitates the comprehension of data format and content, thus affording a foundational understanding of the dataset. Furthermore, it is incumbent to assess the distribution of classes within the dataset, given that imbalanced class distributions can exert deleterious ramifications on model performance. This distribution can be effectively visualized through the deployment of bar plots or histograms. Additionally, the identification and resolution of missing values, should they be extant, are imperative, as their presence can significantly impede model training. The identification and subsequent elimination of duplicate records are pivotal, as the persistence of duplicates may engender partiality in the model. The analysis of Study length distributions across various classes warrants attention, as texts of exceptional brevity or prolixity may necessitate distinct preprocessing strategies. Delving further into the examination, the analysis of the most frequently occurring words and characters in the dataset, substantiated by frequency distribution plots, offers insights into common textual elements. Moreover, the quantification and visualization of stop word frequencies, denoting ubiquitous words such as "and," "the," and "is," contribute to a discernment of stop words' influence on the Study. Constructing a vocabulary comprising unique words endemic to the dataset is instrumental, and the evaluation of vocabulary size and word frequency distribution is salient. The presence of sentiment labels within the dataframe necessitates an investigation into the distribution of sentiment classes, facilitating the identification of linguistic patterns corresponding to distinct sentiments. The prospect of feature extraction from the Study, including part-of-speech tags and named entities, merits exploration. Visualization techniques such as word clouds, word frequency plots, and scatter plots are efficacious in rendering the Study data more perspicuous. An analysis of n-grams, i.e., sequences of n words, illuminates common phrases or expressions. Furthermore, the computation of summary statistics encompassing mean, median, and standard deviation of Study lengths augments the holistic understanding. The investigation extends to the analysis of correlations between Study length and class distribution, with an eye to detecting potential anomalies, outliers, or incongruities within the data. These illuminations procured through EDA serve as cardinal compass points, guiding subsequent data preprocessing endeavors and informing judicious model design choices.

### **Missing values**

In the realm of EDA, the systematic examination for absent values within textual data emerges as an indispensable procedure for several salient reasons. Primarily, the presence of these absent values may signify potential anomalies within the processes of data collection or storage. Should a substantial proportion of dataset rows exhibit the absence of textual content, there exists a credible risk that the overall data quality may suffer detrimental consequences. Furthermore, these lacunae within the textual data may culminate in analyses that are incomplete or erroneous. Particularly, in the context of sentiment analysis predicated on textual content, the nonexistence of values precipitates an environment of incomplete sentiment analysis. These gaps, when pervasive within the textual data earmarked for training sentiment analysis models, exert deleterious influences on the model's performance. In scenarios where a noteworthy fraction of the dataset is plagued by these deficiencies, it may propagate a propensity for biased or inaccurate forecasts. Models, grappling with the dearth of Study, may resort to non-textual attributes as surrogates, potentially introducing bias into their prognostications. Concomitantly, the textual data frequently encapsulates invaluable attributes, such as keywords, hashtags, or mentions, that substantially contribute to the elucidation of sentiment. The nonexistence of values imperils the forfeiture of these pivotal attributes, thereby impairing the model's capacity to comprehend the sentiment context. Contingent upon the data's character and the scope of absent values, it may warrant deliberation to explore imputation methodologies, encompassing the introduction of placeholders, utilization of average lengths, or even the application of sophisticated techniques such as natural language processing to prognosticate absent values. In summation, it is imperative to underscore that the nonexistence of values within textual data, as discerned during EDA, can yield notable repercussions for comprehensive analysis and model efficacy. The judicious rectification of these missing values emerges as a critical endeavor, fostering the fidelity and robustness of sentiment analysis, along with other undertakings centered on textual data. Figure 1 displays the results of the missing values found in our dataset.A black screen with blue text

Description automatically generated

Figure 1(missing values)

### **Class balance**

The assessment of class balance within the realm of Exploratory Data Analysis (EDA) for Study data bears substantial significance due to several compelling considerations. Foremost, class imbalance profoundly influences model performance in the domain of sentiment analysis, as it can precipitate biased or erroneous predictions. In instances where a singular sentiment class prevails overwhelmingly within the dataset, the model may inadvertently incline towards that dominant class, thereby resulting in suboptimal performance pertaining to the remaining classes. Furthermore, the implications extend to the model's learning and generalization capabilities, where imbalanced classes might lead to the undue prioritization of the majority class, thus neglecting the subtleties inherent to minority classes. In the context of evaluation metrics, conventional accuracy proves inadequate within imbalanced datasets, necessitating the adoption of more informative metrics like precision, recall, F1-score, and the confusion matrix for a comprehensive assessment of model performance vis-à-vis distinct sentiment classes. To rectify class imbalance, various data augmentation methodologies such as oversampling, undersampling, or the generation of synthetic samples warrant contemplation. It's noteworthy that imbalanced classes can significantly influence the model's decision threshold, potentially leading to a bias towards the majority class for the sake of optimizing accuracy, albeit at the cost of practical efficacy. The real-world applicability of a sentiment analysis model hinges substantially upon the balance of the training dataset; a model devoid of exposure to balanced data may encounter difficulties when tasked with accurately discerning sentiments it has encountered only sparingly in its training data. In summation, the scrutiny of class balance within the purview of EDA assumes paramount importance, as it underpins the model's capacity to faithfully encapsulate sentiments spanning all classes. The judicious rectification of class imbalance through apt techniques engenders superior model generalization and augments real-world performance. Further elaboration is provided in Figure 2, which delineates the class balance configuration..A screen shot of a computer

Description automatically generated

Figure 2(Class balance)

### **Duplicate values**

The meticulous examination for duplicate values within the ambit of Exploratory Data Analysis (EDA) concerning Study data bears considerable import due to multifarious compelling rationales. Firstly, the identification of duplicate entries serves as a barometer of data quality, shedding light on potential anomalies during data collection, storage, or preprocessing phases. The discernment and subsequent rectification of duplicate values constitute a pivotal stride in elevating the overarching dataset's quality quotient. Pertinently, the specter of duplicate data casts a long shadow over model performance, principally manifesting in the peril of overfitting. This phenomenon engenders a scenario wherein the model's learning trajectory gravitates towards memorizing duplicated instances rather than discerning substantive patterns, culminating in an inflated appraisal during training and impoverished generalization concerning novel data. The presence of duplicate data within both the training and testing cohorts augments the prospect of evaluative bias, artificially amplifying the model's performance metrics. This can lead to an overestimation of the model's acumen in furnishing accurate prognostications for previously unseen data points. Moreover, the expeditious curbing of duplicate data bestows the boon of computational efficiency during model training, as redundant information imposition is ameliorated without deleterious performance ramifications. It merits attention that duplicate data can usher in skewed insights within the precincts of EDA, potentially endowing an exaggerated sense of significance or prevalence to certain sentiments contingent upon the extent of duplication. Additionally, the inclusion of duplicate data can prove a quagmire for model interpretability, potentially confounding the ascription of feature importance or interpretive coherence in instances where the same instance is ascribed divergent labels across duplications. In fine, the elucidation for duplicate values within the EDA vista emerges as an indispensable vanguard, upholding data quality, forestalling overfitting, safeguarding against evaluative bias, optimizing computational efficiency, enhancing interpretability, and preserving dataset representativeness. The excision of duplicates not only burnishes the data's purity but also bequeaths a more faithful, representative, and trustworthy corpus for both training and evaluative underpinnings. Figure 3 displys the results.   
A screenshot of a computer

Description automatically generated

Figure 3(Duplicate records)

### Study Length Visualization

The act of visualizing and scrutinizing the distribution of Study lengths, whether measured in terms of word count or character count, across distinct sentiment classes stands as a pivotal juncture within the purview of Exploratory Data Analysis (EDA) for Study data earmarked for sentiment analysis. The ramifications of this endeavor encompass multifaceted dimensions. Primarily, this process imparts perspicacious comprehension pertaining to the inherent characteristics of textual data nestled within disparate sentiment classes. This insightful discernment serves as a compass, delineating whether sentiment manifestations are commonly terse, protracted, or exhibit a gamut of variabilities in between. Concomitantly, the length of Study emerges as a promising feature for the enunciation of sentiment analysis models. Succinct textual renditions may connote unequivocal expressions of sentiment, while their protracted counterparts may allude to more intricate and nuanced discourses. Moreover, the discernment of disparate distributions in Study length across diverse sentiment classes may wield tangible implications for model intricacy. It instills in the model the cognitive adaptability to modulate its discernment contingent upon such disparities, thereby potentially ameliorating performance. This observation might engender the astute decision to incorporate Study length as a salient feature within the model's repertoire, with the attendant prospect of augmenting predictive efficacy. Furthermore, contingent upon the edifice of Study length distribution, it might impel consequential considerations regarding data preprocessing strategies. The decision to truncate or pad textual content may find its moorings in the observed distribution patterns. Substantively, data imbalance issues may be precipitated if a particular sentiment class recurrently harbors lengthier texts than its counterparts. Mitigating such disparities becomes an imperative stride to engender equitable learning experiences across classes. The implications of Study length distribution radiate into the arena of model evaluation, potentially imparting a skewed perspective if one sentiment class consistently proffers shorter or longer texts. Anomalies or outliers, encompassing texts of inordinate brevity or protraction, could permeate the dataset, potentially sowing discord in model performance. Furthermore, the specter of Study length distribution divergence between training and test sets presents a specter that could jeopardize model generalization. The panoramic visualization and analysis of Study length distribution, in summation, emerge as an invaluable compass during the compass of EDA, endowing discernments instrumental for feature engineering, model choice, preprocessing schema determination, and performance appraisal. This discerning exercise stands poised to hone model performance through the adept utilization of the troves of information inherent in textual length. Figure 4 and 5 dislays the token and character lengths and their plot.A graph of different colored lines

Description automatically generated

Figure 4(KDE plot for token length distribution)

A screenshot of a computer screen

Description automatically generated

Figure 5(Token and character lengths)

### **Visualizing the stop words and their frequencies**

The systematic exploration of stopwords distribution within various sentiment classes, a pivotal aspect of Exploratory Data Analysis (EDA) when handling Study data intended for sentiment analysis, holds profound significance, underscored by diverse rationales. Stopwords, including commonly occurring semantically light words such as "and," "the," and "is," permeate the landscape of sentiment discourse. Their prevalence across different sentiment classes offers illuminating insights into their role as carriers of sentiment expression. These linguistic entities can serve as instrumental agents in sentiment articulation or remain at the periphery of significance, with their impact potentially appearing modest. Analysing stopwords distribution across sentiment classes informs us whether they follow distinct patterns across sentiments. Notably, disparities in their distribution may warrant their inclusion as significant features within sentiment analysis models, especially when they exhibit diverse distributions across sentiments. Additionally, the decision to remove stopwords from Study data, a standard preprocessing practice, hinges on insights drawn from their distribution. It plays a pivotal role in data processing and model intricacies, influencing decisions on their inclusion or exclusion. The examination of infrequent stopwords within specific sentiment classes opens possibilities for identifying distinguishing attributes, enriching sentiment nuances. From a model interpretation perspective, stopwords' visualization clarifies the rationale behind model predictions. Disparities in stopwords distribution across classes may necessitate efforts to rectify model biases. Furthermore, insights gleaned from stopwords' distribution inform model fine-tuning, encompassing hyperparameter calibration and class-specific preprocessing strategies. In summary, the judicious visualization of stopwords' distribution across sentiment classes, a cornerstone of EDA, enhances our understanding of their role in sentiment expression. This discernment informs preprocessing methods, feature curation, and sentiment analysis model design, ultimately boosting model performance through the strategic incorporation of stopwords in the contextual framework. For sentiment analysis of informal Twitter data, it is advisable to consider retaining specific common stopwords frequently present in tweets, conveying sentiment. These stopwords encompass emoticons, abbreviations, and informal words holding indicative sentiment meanings. Presented below is a list of stopwords potentially advantageous for sentiment analysis of Twitter data: Personal Pronouns: I, me, my, mine, myself, I, my, yours, yourself, he, him, his, himself, she, her, hers, herself, it, its, itself, we, us, our, ours, ourselves, they, them, their, theirs, themselves. Common Articles and Conjunctions: the, a, an, and, but, or, if, because, so, for, in, of, on, with, at, by, from, to, about, against, between, into, through, during, before, after, above, below, over, under, between, among. Negations: not, no, never. Contractions: can't, don't, won't, could, would, should. Emotion and Sentiment Indicators: love, like, hate, happy, sad, angry, good, bad, great, awesome, amazing, terrible, lol, haha, omg. Abbreviations and Informal Expressions: u, ur, b4, thx, im, idk, tbh, yolo, fomo, smh Emoticons: <3, :-), :-(. It is essential to tailor the selection of stopwords to the specific dataset and sentiments under analysis. The list provided serves as a starting point, with a comprehensive data analysis determining the stopwords most pertinent to the study, ensuring they genuinely reflect the informal nature and sentiment nuances of Twitter communication. Figure 6 show the count words frequency of neutral sentiment.A screen shot of a graph

Description automatically generated

Figure 6(Stopword count in Neutral sentiment)

### **Splitting the data**

In our approach, we shall employ Scikit-learn's Train-Test Split methodology to partition our dataset into three distinct segments. The preponderant portion, encompassing 80% of the dataset, will be allocated to the training subset, signifying its paramount importance in facilitating the acquisition of predictive patterns by our model. Concomitantly, the validation dataset shall claim a 10% allotment from the overall dataset, mirroring the equivalent allocation bestowed upon the test dataset, which shall be harnessed for predictive purposes. This proportionate allocation is underpinned by the pragmatic consideration that our available dataset constitutes a finite resource, underscoring the exigency of maximizing the data's utility for training purposes, particularly in light of the multifarious classes that demand comprehensive coverage for robust model development.

## **Data-preprocessing**

Data pre-processing is a crucial step in preparing my Study data for machine learning, including Study classification. Here's a comprehensive list of what should be done in data pre-processing for a Study classification project, remove any special characters, symbols, and non-alphanumeric characters that are not relevant to the text content. Convert all text to lowercase. This helps ensure consistent word representations and reduces dimensionality. Split the text into individual words or tokens. Tokens are the basic building blocks for analysis. Remove common words (stopwords) that do not contribute much to the meaning, like "and," "the," "is." Remove URLs or web links from the text as they typically don't add meaningful information. Remove numerical digits unless they are relevant to my analysis (e.g., years, quantities). Reduce words to their base or root form. Lemmatization provides more contextually accurate root words. Remove punctuation marks like periods, commas, and semicolons, unless they convey specific meaning. Remove extra whitespaces or normalize them to a single space. Expand contractions to their full forms. For example, "can't" becomes "cannot”. Consider performing spell checking and correction to address typos and errors. If my text contains HTML tags, remove them or convert them to text. Normalize text by converting special characters or accented characters to their ASCII equivalents. Remove very short sentences or phrases that might not carry significant meaning. Remove words that appear only a few times in the dataset. These words may not contribute much to classification. Create additional features, such as word counts, n-grams, or part-of-speech tags. If I have categorical labels, encode them into numerical values using label encoding. Split my data into training and testing sets to evaluate model performance. Transform text into numerical format that machine learning models can understand. Popular methods include Bag-of-Words, TF-IDF, and word embeddings like Word2Vec or GloVe. If using neural networks, ensure that all sequences have the same length by padding or truncating as necessary. Address class imbalance by oversampling, undersampling, class mapping or using synthetic data generation techniques. Regularly check the quality of my pre-processed data to ensure that meaningful information is preserved. Remember that pre-processing steps can vary depending on the specifics of my dataset, project goals, and chosen model architecture. Experiment with different pre-processing techniques and evaluate their impact on model performance.

### **Feature Reduction**

For feature reduction within our methodology, we have judiciously employed two fundamental techniques: column dropping and class mapping. These strategies are pivotal in optimizing the efficiency and efficacy of our feature space while preserving the essential information encapsulated within our data. Column dropping, the first technique, facilitates the selective removal of superfluous or redundant columns, thereby streamlining our feature set by eliminating variables that do not significantly contribute to the predictive capacity of our model. This meticulous curation of features aids in mitigating the curse of dimensionality and enhances the computational efficiency of our subsequent analyses. The second technique, class mapping, entails the transformation of certain categorical variables, often associated with nominal or ordinal data, into a numerical format. This transformation simplifies the representation of categorical data, rendering it amenable to statistical analyses and machine learning algorithms. By employing these feature reduction methodologies, we enhance our model's interpretability, reduce computational complexity, and expedite the training process, ultimately paving the way for more effective predictive modelling and insights extraction.

#### **Column drop**

As during EDA we found out that there are two columns for example tweet ID and the authors name which are irrelevant in Study classification or detecting the sentiment in the Study as we cannot perform correlation metrics on the dataset. Because it consists of string values rather than numerical values hence using our expert judgement we have decided to drop these two columns.

#### **Class Mapping**

Even if class balance were achieved, the limited quantity of classes and available records proves inadequate for the training of a precise and accurate model. Consequently, we must employ the technique known as class mapping to mitigate this limitation and enhance the quality of results. This approach draws inspiration from previous research, specifically from the work of (Chaturvedi, 2023), who observed significant improvements in their results when they reduced the number of classes within the same dataset. Irrespective of the model's inherent sensitivity to class imbalance, class mapping remains a prudent strategy for diminishing the number of classes. This not only simplifies the model's complexity but also reduces the computational resources and data requirements necessary for achieving satisfactory results. In contrast I had other options such as using SMOTE or NLPaug’s synonym replacement yet these present threats such as data poisoning as they usually change context of the original text and to detect nuanced emotion it is critical that the data is not faulty as there are slight variations in text of two different emotions(Shahul ES, 2022).

### **Text normalization**

Subsequent to the process of feature reduction, the next phase involves text normalization, a pivotal facet in pre-processing text data, aimed at ensuring uniformity and mitigating textual complexity. This normalization process encompasses several common techniques, each serving a specific purpose in enhancing text consistency and clarity. In my project, I meticulously preprocess the Study data for training a Twitter sentiment analysis model, adhering to a series of essential steps. These steps encompass spelling correction to enhance data quality and maintain semantic integrity, denoising by removing HTML tags and expanding contractions for clarity and nuanced sentiment understanding, eliminating name tags to uphold privacy and minimize bias, omitting URLs to streamline processing and clean the dataset, removing punctuation and whitespaces to reduce noise and simplify analysis, converting Study to lowercase for uniformity and efficiency while considering task sensitivity to capitalization, replacing numbers with words for normalization and dimensionality reduction, removing stopwords to enhance TF-IDF analysis, and employing lemmatization for accurate sentiment capture, especially suited for the informal and short texts characteristic of Twitter data. Each of these preprocessing measures contributes to the model's ability to accurately discern sentiment, with careful consideration given to the specific nature of the data and the objectives of the sentiment analysis task. These comprehensive normalization procedures collectively contribute to Study standardization and clarity, thereby enhancing subsequent analyses.

#### **Spelling Correction**

The very first step to perform text normalization is to Conduct spelling correction through TextBlob on text data prior to training a Twitter sentiment analysis model can yield numerous advantages. Spelling errors are pervasive in informal text, such as tweets, and rectifying these errors augments the data's quality, thereby facilitating the model's acquisition of substantive patterns. Correcting spelling errors serves to preserve the semantic integrity of the text, as misplaced or incorrect letters can distort the intended sentiment, ultimately resulting in imprecise model predictions. These spelling errors introduce a discordant element into the data, but their elimination enhances the signal-to-noise ratio, allowing the model to concentrate on authentic sentiment-related attributes. According to (Nishanth N, 2020), A model trained on text with accurate spelling is better poised for generalization to unobserved data, as corrected data presents a more cohesive and representative reflection of linguistic expression, thereby aiding the model's comprehension of sentiments. Many sentiment analysis models leverage sentiment lexicons or dictionaries containing words associated with specific sentiments, and spelling errors can engender incongruities between my text and these lexicons. Correcting spelling ensures alignment with such resources. (Soyusiawaty and Wolley, 2021) explains, the rectified text data engenders more uniform and precise features, thereby potentially enhancing the performance of feature extraction techniques such as TF-IDF and word embeddings. A model trained on corrected text exhibits reduced sensitivity to minor spelling variations, rendering it more resilient in real-world contexts where spelling errors abound. In instances of amalgamating my dataset with external resources or datasets, consistent spelling fosters compatibility and mitigates disparities. It is essential, however, to exercise prudence when applying auto-correction, particularly in cases involving slang, abbreviations, or unique expressions commonly encountered in Twitter data, as some words might intentionally deviate from standard spelling conventions for stylistic or cultural reasons, and auto-correction could inadvertently alter their intended connotation. It is, therefore, advisable to meticulously review the auto-corrected text to ensure it faithfully preserves the desired sentiment and message. To summarize, spelling correction heightens the caliber and uniformity of my text data, thus potentially ameliorating the sentiment analysis model's performance by facilitating the acquisition of precise sentiment-related patterns.

#### **Denoise Study**

The process of denoising textual data by the removal of HTML tags and the expansion of contractions emerges as a fundamental practice that bestows an array of discernible advantages in the context of training a Twitter-oriented sentiment analysis model.(Lohiya, 2018) also explains HTML tags, typically embedded within text sourced from the web, including tweets drawn from web-based content, often serve as an unwarranted source of disruption to the text's intrinsic integrity. The expurgation of HTML tags represents a means of enhancing the text's cleanliness, culminating in a refined input that is bereft of superfluous elements, thereby directing the model's cognitive faculties toward content of direct relevance. Crucially, HTML tags, devoid of sentiment-related information, possess the potential to confound the model's interpretative process if retained. By their elimination, the model is conferred with a pristine input, a transformation that augments its capacity to apprehend and assimilate meaningful textual patterns. Moreover, the presence of HTML tags can engender irregularities in Study length and structure, introducing a veneer of non-uniformity detrimental to feature extraction techniques such as TF-IDF or word embeddings. Through the eradication of these tags, a harmonious uniformity is reinstated, a factor instrumental in advancing the efficacy of feature extraction methodologies. It is incumbent to acknowledge that HTML tags introduce a disconcerting element of noise into the text, with the potential to lead the model astray by conditioning it to acquire spurious or irrelevant patterns. In purging these artifacts, the resultant reduction in noise heralds a more salubrious signal-to-noise ratio. In consonance with the principles of clarity and precision, the expansion of contractions within the text engenders a heightened level of explicitude.(Antari Pasaha, 2020) The amorphous nature of contractions, often symptomatic of informal discourse, acquires definitional contours when expanded (e.g., "didn't" to "did not"), engendering a text imbued with unequivocal semantic intent. This practice imbues the text with a measure of clarity that resonates profoundly within the domain of sentiment analysis. Moreover, the expansion of contractions unfurls the potential to elucidate the nuanced contexts of sentiment, with the model able to parse distinctions such as that between "I can't wait" and "I cannot wait," thereby discerning subtleties in sentiment expression. Sentiment lexicons or dictionaries, often indispensable to sentiment analysis endeavors, may not incorporate contracted forms of words, rendering expansion pivotal in rendering the text's lexicon congruent with these resources. A model nurtured on denoised text emerges as more impervious to the vicissitudes of text structure, emblematic of its versatility across a spectrum of textual inputs. The bestowal of a standardized, denoised input empowers the model with a heightened propensity for generalization, underscoring its efficacy within real-world scenarios(Vickery, 2020). However, it is incumbent to apprise that while denoising confers an array of benefits, certain contractions lay entrenched within informal parlance and may be imbued with sentiment nuances. Ergo, the removal of contractions demands circumspection, for their elimination could potentially transmute the sentiment implicit in the text. Consequently, a judicious review of the denoised text is imperative, ensuring that it remains an authentic vessel of the intended sentiment. In summative synthesis, the art of denoising textual data, with the dual facets of HTML tag removal and contraction expansion, bequeaths an accretion of quality, clarity, and consistency upon the corpus. This augmentation, in turn, galvanizes the performance of the sentiment analysis model, bequeathing it a pristinely cleansed and eminently interpretable input substrate.

#### **Remove Name Tags**

The elimination of name tags or mentions from text data during the preparatory phase for training a Twitter-centric sentiment analysis model emerges as a salient practice imbued with multifarious advantages. Principally, this endeavor aligns with paramount considerations of privacy and ethical propriety. The study by (Morozov, 2021) explains expunging mentions from tweets, the practice implicitly upholds the privacy rights of individuals whose names may be invoked without their consent in the analysis. Moreover, in cases where publicly available data is employed, the omission of name tags contributes to the anonymization of the dataset, a consequential measure when contemplating dataset sharing or publication. In the specific domain of sentiment analysis, the fundamental objective is to decipher the sentiment enshrined within the text, irrespective of the identities of the individuals involved(LinkedIn, 2023). Consequently, the excision of name tags serves as a mechanism to channel the model's cognitive resources toward the intrinsic sentiment-laden essence of the tweet. Within the purview of sentiment analysis, the sentiment conveyed by a tweet does not invariably hinge on the individuals spotlighted within it. In light of this, the omission of name tags acts as a filter for superfluous information, fostering an environment wherein the model can more effectively discern the sentiment-bearing keywords. It is incumbent upon sentiment analysis models to proffer insights that transcend the idiosyncrasies of specific tweets, encompassing a more generalizable purview. The removal of name tags culminates in the creation of a more generic input, a transformation that augments the model's efficacy in handling diverse tweets. Twitter handles, often variably formatted, may incorporate the '@' symbol or exhibit variances in spelling. The act of purging them imparts a harmonious uniformity to the input, thereby streamlining the model's cognitive operations. Furthermore, name tags, characterized by distinctive punctuation marks and characters, can potentially confound the tokenization process and obfuscate the analytical panorama. Their elimination simplifies tokenization and thereby enhances the accuracy of analysis. A corollary benefit resides in the mitigation of overfitting tendencies within sentiment analysis models, particularly those that might be prone to fixate excessively on specific names recurrently encountered in the training data. The removal of name tags mitigates this risk, endowing the model with a higher propensity for broader generalization. Furthermore, if certain names disproportionately permeate the training data, the model may inadvertently assimilate biases associated with those names. The strategic removal of name tags plays a pivotal role in curtailing such biases. Fundamentally, sentiment analysis is preoccupied with the discernment of sentiment-conveying words, phrases, and linguistic patterns. Name tags, unless intrinsically germane to the analytical objectives, can potentially divert the model's attention from these quintessential patterns. In summation, the expurgation of name tags from Text data during the preliminary phases of preparing Twitter data for sentiment analysis bequeaths an array of tangible benefits, encompassing the realms of privacy preservation, sentiment content concentration, noise reduction, generalization augmentation, and enhancement of model performance. Nonetheless, it remains imperative to judiciously evaluate the precise contextual exigencies of the analysis, along with the potential ramifications on sentiment representation, before rendering a definitive decision regarding the inclusion or exclusion of name tags.

#### **Removing urls from the data**

The omission of Uniform Resource Locators (URLs) from Text data during the pre-processing phase constitutes a pivotal and multifaceted procedure, notably within the ambit of training a Twitter-centric sentiment analysis model. This particular practice confers a panoply of advantages. Firstly, it eradicates extraneous elements, as URLs typically encapsulate web addresses, links, and metadata bereft of relevance to the discernment of sentiment(Hosseini, 2023). Consequently, the model's scrutiny is channeled exclusively towards the sentiment-laden substance latent within the Text. Secondly, the elimination of URLs engenders a reduction in data noise, given their frequent amalgamation of diverse characters, symbols, and alphanumeric permutations that do not significantly contribute to the sentiment analysis endeavour(Vickery, 2020). This surgical excision serves to distill the dataset, permitting the model's focus on salient lexical constructs and phrases. Thirdly, URL standardization is achieved, harmonizing the Text data's format, length, and structural uniformity, thereby rendering the model's processing and analysis endeavours more parsimonious and efficacious. Fourthly, URL tokenization poses intrinsic challenges attributable to their unsegmented nature, compounded by the presence of special characters; their exclusion streamlines the tokenization process, affording enhanced analytical precision. Additionally, URL removal results in the amelioration of data processing efficiency, as the analysis of URLs entails supplementary processing steps, such as URL decoding, the circumvention of which simplifies Text processing and augments efficiency. In summation, the expurgation of URLs from Text data emerges as a cardinal precursor in the pre-processing continuum, particularly germane in the context of sentiment analysis on platforms such as Twitter. This procedural edict conduces to an augmentation of data integrity, a diminution of noise, a regularization of formatting, and an unswerving focus on the pivotal textual constituents that efficaciously convey emotions. It is imperative to underscore, however, that the election to either expunge or retain URLs ought to be informed by the precise objectives and modus operandi germane to the analysis, concomitant with the idiosyncrasies of the dataset in question.

#### **Removing punctuations and whitespaces**

The exclusion of punctuation marks and white spaces during the preparatory phase of text data for the training of a Twitter-oriented sentiment analysis model constitutes a matter of paramount significance, underscored by a multitude of compelling rationales. the steps are motivated by the study of (Etaiwi and Naymat, 2017) firstly, it begets a discernible diminution of superfluous noise within the data, a sine qua non, as punctuation marks often lack an intrinsic indicatory value with regard to sentiment and, moreover, they harbour the proclivity to insinuate redundant cacophony into the dataset. Through their meticulous ejection, the model is furnished with a clearer runway for the discernment of words and phrases bearing the mantle of sentiment. Secondly, the act of discarding punctuation endows the Text data with a greater measure of standardization. Punctuation marks, owing to their multifarious stylistic and contextual adaptations across divergent texts, tend to engender a degree of structural heterogeneity. This riddance, however, ushers in a sense of conformity, thereby streamlining the model's processing and analysis. Thirdly, punctuations exert a tangible influence on the process of tokenization, where Text is subdivided into individual words or tokens. Their excision, in this context, simplifies the tokenization procedure, culminating in a higher degree of precision in this vital phase. Fourthly, the expulsion of punctuation serves to curtail the dimensionality of the data, an endeavor that augments the model's analytic efficiency. Punctuation marks, while serving a salient linguistic purpose, augment the vocabulary size disproportionately relative to their contributions to sentiment analysis. Their elimination contributes to a more parsimonious dataset, thereby facilitating a more expeditious analysis. Fifthly, the management of white spaces emerges as an ancillary but indispensable facet of this purgatorial process. Superfluous white spaces, engendered by formatting anomalies or data errors, are ushered to the periphery. Their removal ensures a uniform treatment of words, eschewing unnecessary variances. Sixthly, the reduction in data size, consequent to the removal of punctuations and white spaces, connotes swifter processing and expedited model training. Seventhly, it is worth emphasizing that the preeminent concern of sentiment analysis lies in discerning words and linguistic patterns that articulate emotions. In this semantic realm, punctuations and white spaces are intrinsically non-contributory. Eighthly, the discretionary use or non-use of punctuation can, in certain cases, introduce inadvertent bias or influence the outcome of sentiment analysis. The deliberate elimination of punctuations rectifies this potential pitfall, ensuring a dataset characterized by balance. Ninthly, the efficacy of feature extraction is appreciably heightened by the expulsion of punctuation marks. Punctuation, in isolation, scarcely harbours meaningful sentiment information. Consequently, its effacement paves the way for the model's exclusive attention to pivotal features such as words, phrases, and contextual cues that are germane to sentiment analysis. Tenthly, the overarching theme of consistency pervades this narrative. The removal of punctuations and white spaces epitomizes a commitment to consistency in textual processing, a prerequisite for the precise execution of analysis and modelling endeavours. Finally, the advantages conferred by this practice extend to the domain of token matching, an indomitable constituent of Text analysis. By mitigating the influence of punctuation marks, the likelihood of token mismatches engendered by variations in punctuation deployment is appreciably diminished. In summation, the expurgation of punctuation marks and white spaces transcends the realm of a mere procedural protocol; it emerges as an indispensable prelude in the orchestration of Text data for sentiment analysis. This crucial pre-processing step accomplishes the dual feats of attenuating superfluous noise and harmonizing structural regularity, thereby bequeathing to the model a fecund milieu in which to distill the essence of sentiment-laden words and phrases. Nevertheless, it remains incumbent upon the practitioner to gauge the appropriateness of retaining or expunging these linguistic elements, a decision inextricably intertwined with the imperatives and proclivities of the analytical task at hand.

#### **Lower case text**

Converting Text to lowercase during pre-processing is a common step in natural language processing (NLP) for several reasons. Text data can be inconsistent in terms of capitalization. Converting everything to lowercase ensures uniformity and reduces the dimensionality of my data. This means that the algorithm treats "Word" and "word" as the same word, reducing redundancy and improving the efficiency of subsequent processing steps. Many NLP tasks, such as Text classification, sentiment analysis, and information retrieval, rely on matching words(Deepanshi, 2023). By converting Text to lowercase, I ensure that the same word with different capitalizations is treated as the same word, which helps in capturing word frequencies and patterns more accurately. During Text analysis, stopwords (common words like "and," "the," "is") are often removed to focus on more meaningful words. If I don't convert Text to lowercase, "The" and "the" would be treated as different words, possibly leading to stopwords not being properly removed. Lowercasing helps reduce the dimensionality of the feature space. For example, if "Dog" and "dog" are treated as different words, they would be represented as separate features in a bag-of-words model, potentially leading to unnecessary complexity. When Text data comes from different sources, there might be variation in capitalization. Converting everything to lowercase ensures that the same word is represented consistently across the dataset, making subsequent analysis more reliable. Lowercasing reduces the computational load and memory usage. Since NLP tasks involve processing large amounts of Text data, reducing the complexity of Text representation can speed up the analysis. However, there are cases where I might not want to convert Text to lowercase, such as when the capitalization carries important information (e.g., sentiment analysis where "GOOD" and "bad" have different meanings). As always, it's important to consider the specific requirements of my task and dataset when making preprocessing decisions.

#### **Replacing numbers**

Replacing numbers during data pre-processing is a common practice in natural language processing (NLP) for several reasons: Normalization: Numbers can take various forms, such as digits, words, or symbols. Replacing them with their word equivalents can help normalize the data. For example, converting "123" to "one hundred twenty-three" ensures that different representations of the same value are treated the same way. Reducing Dimensionality: In some cases, numbers can introduce noise and increase the dimensionality of the data. For example, if I're working with Text classification and numbers are not critical to the context, replacing them with words can help reduce the dimensionality of the feature space. Generalization: Numbers often carry specific information that might not be relevant to the task. By converting them to words, I can focus on the semantic meaning of the Text rather than specific numerical values. Handling Out-of-Vocabulary (OOV) Words: In tasks like sentiment analysis or language generation, words not present in the training vocabulary can be problematic. Replacing numbers with words reduces the likelihood of encountering OOV words. Enhancing Interpretability: When working with machine learning models, using words instead of numbers can make the results more interpretable, especially for models like decision trees, where feature names have semantic meaning. Improved Text Embeddings: If I're using pre-trained word embeddings, replacing numbers with words can lead to better embeddings. Many word embeddings capture semantic relationships among words, which can extend to numbers represented as words. However, it's important to note that replacing numbers might not always be necessary or beneficial, especially if numerical values hold significant meaning in my context (e.g., financial data analysis). The decision to replace numbers should depend on the nature of my data and the specific task I're working on.

#### **Remove stopwords**

it's a good practice to remove stopwords from my Text data during pre-processing for Text classification. Here's why, Stopwords like "and," "the," "is," etc., occur frequently in Text but don't carry significant meaning(Khanna, 2021). Removing them helps to reduce noise in my data, allowing the TF-IDF algorithm to focus more on meaningful words. Stopwords are common words that appear in almost all documents. If I include them in the TF-IDF calculation, they could dominate the weights and make it harder for the algorithm to differentiate between documents. Removing stopwords reduces the dimensionality of my data, making the TF-IDF matrix more informative and efficient to compute. Removing stopwords can lead to more informative TF-IDF representations, as the algorithm will assign higher weights to words that are truly distinctive and important for differentiating between classes. However, keep in mind that the decision depends on my specific use case. In certain scenarios, such as sentiment analysis, stopwords like "not" and "very" could carry valuable information. Therefore, before finalizing my pre-processing steps, it's advisable to test both scenarios: one with stopwords removed and one without, and then evaluate the impact on my classification results.

#### **Lemmatization**

In my project, opting for lemmatization over stemming during the pre-processing of Text data for training a Twitter sentiment analysis model presents several advantages, especially given the informal and concise nature of tweets. While both techniques aim to reduce words to their base or root form, lemmatization stands out for several reasons explained by (Camacho-Collados and Pilehvar, 2018). Firstly, it preserves the subtleties of word meaning by producing forms closer to their original dictionary entries, ensuring precise sentiment interpretation by capturing nuanced variations of sentiment-bearing words. Secondly, lemmatization considers the context and part of speech, crucial for understanding sentiment in the context of phrases, slang, and informal language often found in tweets. Thirdly, it enhances readability by generating more recognizable and coherent words compared to stemming, which can produce non-words. Fourthly, lemmatization contributes to improved accuracy by providing a more contextually accurate base form, aiding sentiment analysis models in comprehending sentiment-bearing words and their variations. Additionally, it excels in handling negations, preserves named entities and proper nouns, and is particularly effective in the analysis of short texts typical of Twitter, which demand precision. Lastly, it accommodates emojis and emoticons, enriching sentiment analysis by capturing their sentiment cues. While lemmatization offers these advantages, the choice between lemmatization and stemming should align with the data's nature and specific sentiment analysis goals. For formal Text data or projects with language constraints, stemming may still be a valid choice, but for Twitter data characterized by informality and brevity, lemmatization generally proves superior in retaining meaning, context, and sentiment nuances.

## **Model training**

There are several types of models that can be used for creating a Text classification model. The choice of model depends on various factors including the complexity of the problem, the available data, computational resources, and desired model performance. Here are some common types of models for Text classification along with how the resources required and performance may differ for each.

#### **Traditional Machine Learning Models:**

Logistic Regression, Naive Bayes, Support Vector Machines, Random Forest, etc. These models generally require fewer computational resources compared to deep learning models(Datarobot, 2019). They can be trained on standard hardware and are relatively faster to train. Traditional models work well for simpler tasks and smaller datasets. They may struggle with capturing complex patterns in large datasets, resulting in limited performance for more challenging tasks. ANNs are versatile and can be effective for sentiment analysis tasks. However, their performance might be limited compared to more advanced architectures like RNNs and Transformers, especially for tasks involving sequential data like Text. ANNs are relatively less resource-intensive compared to more complex architectures like RNNs and Transformers. They might require moderate computational resources for training.

#### **Word Embedding Models**

These models use pre-trained word embeddings like Word2Vec, GloVe, or FastText as input and can be combined with traditional ML models. These models still require standard hardware for training as explained by (Brownlee, 2016). The main computational cost is related to handling the embedding vectors and training the final classification layer. Word embeddings enhance the model's ability to capture semantics and relationships among words. They perform well on tasks that require understanding context and meaning but may not achieve state-of-the-art performance on complex tasks. There are multiple embedding models available yet we would go with Glove twitter 200d embedding as these are already trained for twitter dataset and provide our model with a transfer learning aspect allowing less resources to train on current dataset.

#### **Recurrent Neural Networks (RNNs)**

RNN, LSTM, GRU (Gated Recurrent Unit). RNNs require more resources compared to traditional models due to their sequential nature(Susan Li, 2018). Training RNNs can be computationally expensive, and long sequences can lead to vanishing gradient problems. RNNs are effective for tasks that involve sequential dependencies, like sentiment analysis and Text generation. However, they may struggle with very long sequences and can have difficulty capturing long-range dependencies. Hypothetical Performance: RNNs are designed to handle sequential data like Text, making them suitable for sentiment analysis. However, basic RNNs suffer from vanishing gradient problems and might struggle to capture long-range dependencies. RNNs can be resource-intensive, especially if i'm using deep or bidirectional variants. Training RNNs might require more computational resources compared to simpler models like ANNs. Therefore I am going to use word embeddings with this model to reduce the amount of resources required for training as well assuring better results.

#### **Transformer-Based Models**

BERT, GPT, RoBERTa, etc. Transformer models require significant computational resources, including GPUs or TPUs, for training due to their large number of parameters and complex attention mechanisms. Transformer models excel in capturing contextual information and relationships among words, making them well-suited for a wide range of NLP tasks. They achieve state-of-the-art performance on various benchmarks but come with higher computational costs. Transformer-based models are state-of-the-art for NLP tasks, including sentiment analysis. They capture contextual relationships effectively and achieve high performance. Models like BERT are pretrained on large datasets, which can boost performance. Transformer models are among the most resource-intensive options due to their deep architecture and the large number of attention heads. Training them from scratch requires significant computational resources and large amounts of data.

#### **Model selection**

I have planned to train three distinct models for this task, each carefully selected to address specific requirements and constraints of the project. Firstly, I am going to train a traditional decision tree model with their feature extracted through TF-IDF. Decision trees provide transparency into the decision-making process, allowing us to understand which features are most influential in classification. TF-IDF captures word importance and relevance, which can be crucial for understanding the significance of terms in Twitter data. This approach is computationally efficient and suitable for relatively small to medium-sized datasets. Secondly, A Bidirectional Gated Recurrent Unit (GRU) is a suitable choice when context comprehension is vital, as it allows the model to understand text bidirectionally, considering both past and future context this would be supported by Glove embeddings already trained for tweets. While a simple GRU might suffice, bidirectionality can enhance the understanding of nuanced Twitter data where context is key. I didn’t choose LSTMs as they are more resources consuming and have the ability to refer to the previous data as well but at this point I don’t require to refer to the previous data from users. GRU models are computationally more efficient compared to LSTMs, making them a pragmatic choice for resource-constrained environments. Bidirectionality can capture dependencies between words or phrases that are more complex and context-sensitive. Thirdly, I’ll train distilBERT as it is the most efficient transformer based model for twitter dataset because of their ability to consume less resource compared to other models as well as they have the element of transfer learning available to boost the feature extraction. DistilBERT benefits from transfer learning, allowing it to leverage knowledge from pre-trained models and adapt to specific tasks, which can be highly beneficial when working with Twitter data. The attention mechanism in transformers enables the model to focus on relevant parts of the text, crucial for understanding Twitter content with varying context and language. Transformer models, like DistilBERT, have excelled in a wide range of natural language processing tasks, making them a versatile choice for text classification.

### **Model specific post-processing**

The post-processing is different for different types of models available as the pre-processing technique used will encounter all of the issues which were generated while exploratory data analysis and enable the data to be input into that specific model architecture. There are mainly three types of models we would be focusing firstly we would be focusing to train one artificial nearer network (ANN) then we would be focusing to train two recurrent Neural networks(RNN) and finally we’ll train one transformer based model so that we can compare the performance of all types of model architecture and compare the resources used to train these models the data pre-processing uptill this point is common for all of these model architecture and will create different data dimension for training datasets.

#### **Transformer based model:**

To train a transformer-based model I will start by encoding the labels using one hot encoding and input the data do the pre trained models available in the Transformers library which also has the auto tokenizers from the pre trained models.

##### **Tokenization:**

Tokenizing Text data according to the BERT tokenizer's guidelines is a crucial preparatory step, especially for models like DistilBERT used in tasks such as sentiment analysis. Tokenization, which involves breaking Text into discrete tokens, is vital for DistilBERT's fixed-size token input requirement and involves adding special tokens like [CLS] and [SEP], segmenting words into subwords, and mapping tokens to their respective IDs. the Tokenization Process dissects Text into tokens, which can represent words, subwords, or characters based on the model's granularity. Typically, for English Text, tokenization involves using spaces and punctuation to define word boundaries, resulting in token sequences like ["I", "love", "machine", "learning", "!"] for a sentence like "I love machine learning!". Auto Tokenization, also known as default tokenization, is a feature provided by transformer-based models like BERT and DistilBERT. It automatically tokenizes input Text using the model's built-in tokenizer, ensuring alignment with the model's vocabulary and subword representations. Auto tokenization offers several advantages for DistilBERT, starting with vocabulary alignment which ensures input Text is tokenized using the same vocabulary as DistilBERT, reducing out-of-vocabulary issues. Followed by, subword representation as Auto tokenization captures morphological nuances, aiding the model in handling rare words and misspellings.as well as, It maintains uniformity between tokenization and the model's expectations, preventing token mismatch problems during training and inference. Furthermore, it creates special tokens as DistilBERT requires special tokens like [CLS] and [SEP], which auto tokenization inserts correctly. Additionally, auto tokenization manages sequences' padding and truncation to meet DistilBERT's fixed-size input requirements. In summary, employing auto tokenization is highly recommended when training DistilBERT for sentiment analysis. It aligns with the model's vocabulary and subword representations, ensuring accurate results during both training and inference, thereby optimizing its performance in Text classification tasks like sentiment analysis.

##### **Padding**

Although auto tokenizer will pad or truncate automatically yet it will not be able to match the sequence length of the training and test dataset I would have to decide according to the difference amongst the token lengths of the three dataset train, test and validation. The decision to either add a padding or truncate the sequence would be decided by analysing the difference of the sequence lengths.

##### **Input IDs and Attention Masks**

BERT-based models like DistilBERT require additional inputs like segment IDs (to distinguish between different sentences) and attention masks (to indicate which tokens to attend to and which ones to ignore). Input Shape: Reshape my data to match the expected input shape of the DistilBERT model.

#### **Artificial Neural Networks(ANN):**

to train artificial near networks we will perform an additional step which is converting or data into TF-IDF to counter the class imbalance. Tokenization: Random Forest models don't require tokenization like neural network models. I can simply convert my Text data into numerical format, for example by using techniques like TF-IDF or Count Vectorization. Handling Categorical Variables: If I have categorical features (like sentiment labels), I might want to encode them using techniques like one-hot encoding or label encoding. Feature Engineering: I can extract additional features from the Text data, like word or character counts, sentiment scores, etc.

##### Term Frequency-Inverse Document Frequency (**TF-IDF)**

TF-IDF is a Text vectorization technique that converts textual data into numerical vectors, suitable for input into machine learning algorithms such as artificial neural networks (ANNs) also a very efficient way of extracting features from text data. Employing TF-IDF as part of the preprocessing pipeline for training ANNs on Twitter sentiment analysis data yields multiple advantages. Firstly, TF-IDF offers a numerical representation of Text, enabling ANNs to work with the data effectively. Moreover, it emphasizes word importance by assigning higher values to words that are specific to individual documents but rare across the entire corpus, aiding ANNs in focusing on sentiment-carrying words. TF-IDF naturally reduces dimensionality by downweighting less informative terms, beneficial for ANNs sensitive to high-dimensional data. It also handles stopwords automatically, ensuring common, less relevant words don't dominate model learning. For Twitter data, which often includes unique phrases and slang, TF-IDF assigns higher values to these distinctive terms, enhancing the model's attention to them. The resulting sparse matrix is memory-efficient and speeds up training, especially with large datasets. TF-IDF's consideration of semantic context within documents is valuable for sentiment analysis, enhancing the model's understanding of sentiment-bearing words. Additionally, as TF-IDF operates post-preprocessing, it remains compatible with various Text preprocessing techniques. Its transferability to new data facilitates model application to different textual datasets without extensive retraining, and its inputs, TF-IDF vectors, offer interpretability by highlighting word contributions to sentiment predictions. Overall, TF-IDF is a popular choice for sentiment analysis on Twitter data due to its capacity to capture word importance, context, and uniqueness, essential factors in sentiment comprehension. The choice of Text vectorization technique, including TF-IDF, depends on the specific use case and data characteristics.

#### **Recurrent Neural Networks(RNN):**

To train an RNN which would be preferably a GRU the reasoning is discussed later. We would have to Tokenize my Text data into sequences of words or subwords. I can use libraries like NLTK or spaCy for this. Padding: Since RNNs like GRU require fixed-length sequences, I might need to pad or truncate my tokenized sequences to a consistent length. Word Embeddings: Convert the tokenized sequences into word embeddings. I can use pre-trained word embeddings like Word2Vec or train my own embedding layer. Input Shape: Reshape my data to match the expected input shape of the GRU model, which includes the batch size, sequence length, and embedding dimensions.

##### **Label encoding:**

In machine learning and data pre-processing, label encoding is employed to convert categorical labels into numerical values, facilitating more effective algorithmic processing, especially in classification tasks. Various machine learning algorithms, like those in scikit-learn or TensorFlow, necessitate numerical input data. Label encoding simplifies this conversion, aligning the data with mathematical operations. It's memory-efficient and can accelerate training, notably for decision trees. Label encoding finds utility in ordinal categorical data, streamlining dimensionality when categories are extensive. However, it introduces arbitrary values, potentially implying inaccurate rankings or orders. For nominal data, it's unsuitable, as it can mislead algorithms. In contrast, one-hot encoding may be better. In Natural Language Processing (NLP), a binary class matrix, often termed "one-hot encoding," serves to represent categorical labels as binary vectors, crucial for training models. It translates labels into a format compatible with numerical-input algorithms and simplifies loss calculations, aiding multiclass classification and metrics computation. The binary class matrix is particularly valuable for neural networks, aligning with the softmax activation function's needs. It's distinct from multilabel classification techniques used in tasks like sentiment analysis, which handle multiple class assignments per input differently.

##### **Tokenization:**

Tokenization is the process of breaking down a Text into individual units called tokens. In the context of natural language processing (NLP), tokens are typically words, subwords, or characters(Kumar, 2023). Tokenization is a crucial pre-processing step in NLP that serves several purposes. Tokenization breaks down a continuous piece of Text into discrete units, making it more manageable for analysis. This segmentation is important for further processing, such as parsing, tagging, and feature extraction. In many NLP tasks, Text is represented numerically to be used as input for machine learning models. Tokens serve as the basic units for feature extraction. Each token may carry semantic meaning, and these meanings contribute to the model's understanding of the Text. Tokenization plays a role in building a vocabulary, which is a collection of all unique tokens in a Text corpus. This vocabulary is used for tasks like word embedding and representing Text numerically. Tokenization can involve Text normalization steps, such as converting Text to lowercase or removing punctuation. These steps ensure that tokens are consistent and uniform for analysis. N-grams are contiguous sequences of N items from a given sample of Text(Sophia Yang, 2023). Tokenization forms the basis for generating n-grams, which are useful for capturing local context in Text. Tokenization helps in analyzing the linguistic structure of a Text. For example, analyzing the frequency of tokens can reveal insights about word usage patterns. In named entity recognition tasks, tokenization helps identify individual words or subwords that correspond to entities like names, dates, and locations. Tokenization is an initial step in sentiment analysis, where the sentiment of a sentence or document is determined based on the sentiments of its constituent tokens. In machine translation, tokenization is important for dividing sentences into units that can be translated while preserving their meaning. Tokenization can be relatively straightforward for languages with clear word boundaries like English. However, it can be more complex for languages without spaces between words, such as Chinese or languages with complex compounds. In summary, tokenization breaks down Text into smaller meaningful units, enabling various NLP tasks by providing a structured and standardized representation of Text data.

##### **Padding**

Padding is a process used in natural language processing (NLP) to ensure that all sequences in a dataset have the same length. It involves adding special tokens (usually a padding token) to sequences that are shorter than the maximum length in the dataset. This is necessary because many machine learning algorithms, especially those that work with batches of data, require consistent input dimensions. Batch Processing, in most machine learning frameworks, data is processed in batches for efficiency. For batch processing, all sequences in a batch need to have the same length. Padding ensures that sequences within a batch have consistent dimensions. Tensor Dimensions, In deep learning models, input data is often represented as tensors with fixed dimensions. Padding makes sure that the tensor dimensions are uniform across all sequences. Model Input, Many neural network architectures, such as recurrent neural networks (RNNs) and transformers, expect inputs of fixed length. Padding ensures that all inputs have the same length, making them compatible with these architectures. Efficient GPU Processing, Modern deep learning frameworks take advantage of GPU acceleration. To efficiently use GPUs, it's beneficial to process multiple inputs in parallel, which requires inputs to be of the same length. Memory Alignment, Some hardware, like GPUs, benefits from memory alignment, where data is organized in specific memory addresses. Padding can help align data in a way that optimizes memory usage and access. Masking, in some models, padded tokens are masked to avoid their contribution to computations, ensuring that they don't affect the model's output. This is common in transformer-based models. Padding is usually achieved by adding a special token called a padding token at the end of sequences. This token holds no real meaning and doesn't affect the model's predictions. For example, in English, a padding token could be something like "[PAD]". During processing, the model learns to ignore these padding tokens. It's important to choose a reasonable maximum sequence length for padding to avoid excessive memory usage. Very long sequences could waste memory and computational resources, while very short sequences could introduce unnecessary padding, reducing efficiency. The choice of padding length depends on the specific task and the architecture of the model being used.

##### **Word Embeddings**

Word embeddings are numerical representations of words in a way that captures semantic relationships between words. They are essential in natural language processing (NLP) tasks because they provide a way to represent words as vectors, which can be used as inputs for machine learning models. These embeddings help address the limitations of using raw Text data in NLP tasks and significantly impact model performance. Here's why word embeddings are performed in NLP tasks and the impact they have on model performance. Raw Text data consists of words, and computers need numerical data to process information. Word embeddings transform words into dense vectors where the relative positions of vectors reflect the semantic relationships between words. Similar words have similar vectors, which helps models understand and leverage semantic meaning. Words can be represented as high-dimensional one-hot encoded vectors, where each word is a vector of mostly zeros with a one at the corresponding word's index. However, these high-dimensional representations are inefficient and don't capture relationships between words. Word embeddings reduce dimensionality, making computations more efficient and allowing models to generalize better. Word embeddings capture contextual information about words. In language, the meaning of a word often depends on the words surrounding it. Word embeddings take into account the context in which words appear, allowing models to capture nuances and word senses. Pre-trained word embeddings, such as Word2Vec, GloVe, and FastText, are trained on large corpora of Text data. They encode general language knowledge that can be transferred to specific NLP tasks. By using pre-trained embeddings, models can start with meaningful representations, even with limited task-specific data. Word embeddings enhance the performance of NLP models by enabling them to learn semantic relationships, contextual meanings, and patterns in the data. Models that use embeddings as input tend to generalize better, capture nuances in language, and achieve better accuracy compared to models trained on raw Text data. In high-dimensional space, raw Text data leads to a sparse matrix, where most entries are zeros. Word embeddings represent words in a dense space, reducing data sparsity and improving computational efficiency. Word embeddings enable mathematical operations between vectors, such as vector addition and subtraction. These operations can capture analogies and relationships between words (e.g., "king" - "man" + "woman" ≈ "queen"). Models that leverage word embeddings as inputs often perform better on downstream NLP tasks, such as sentiment analysis, Text classification, machine translation, and question answering. The embeddings' ability to capture semantic meaning and context transfers well to these tasks. In summary, word embeddings are fundamental in NLP because they convert words into numerical representations that capture semantics, context, and relationships. These embeddings provide efficient and meaningful inputs for NLP models, leading to improved performance on a wide range of tasks.

# **Chapter 4: Implementation**

n the implementation section of this project, I meticulously executed the various steps outlined in the design section by harnessing a selection of powerful libraries and tools. To preprocess and manipulate the data, I relied on the versatile 'pandas' library, which facilitated efficient data handling, cleaning, and transformation. For natural language processing tasks such as text normalization, lemmatization, and stopwords removal, I employed the 'nltk' library, known for its comprehensive linguistic capabilities. The task of converting textual data into numerical features was accomplished through the 'scikit-learn' library, providing an array of essential tools for feature extraction, including TF-IDF vectorization. In the realm of machine learning, 'scikit-learn' also played a pivotal role, enabling the training and evaluation of diverse models, such as decision trees and Bidirectional GRU networks. Additionally, for state-of-the-art performance, I incorporated 'Hugging Face Transformers,' leveraging their 'distilBERT' model for fine-tuned sentiment analysis. Throughout the implementation, these libraries collectively ensured the smooth execution of each designated step, from data preprocessing to model training and evaluation, culminating in a comprehensive sentiment analysis framework.

## **Loading Data**

The data is loaded from the root directory and is loaded as a pandas dataframe. The data frame is in the form of a CSV file hence we load the data set using the OS library and convert it directly into the pandas data frame. the figure below shows the command which was used to load the data set.

## **Exploratory Data Analysis(EDA)**

To progress further we'll start with the process of exploratory data analysis as previously discussed we have chosen the following steps to implement with careful analysis of the data we have.

### **Data frame review**

In the exploratory data analysis the first up step we'll do is we'll review the data we have all the headers and what values are filled in their respective columns. the table below shows the data set we have and the first five rows of data set. The data set as previously discussed in the data description we have 4 columns The first column is for tweet ID the second one is for sentiment third one contains the name of the author and 4th one contains the content of that tweet. This gives us a general view off the data we have and also helps us to identify if we can perform any feature reduction using this information.

### **Missing values**

We have used the command of pandas library to identify an aggregate all the missing values in the columns we have and upon the execution it was identified that our data set does not contain any missing values which can distort our data. as the next step is to check the missing values we use the pandas command is null to check for all the null values as we don't have to be careful about zeros big cause or data set consists of string values. Upon implementation off the command we found that and all of the columns there were no missing values all of the columns hard something present in them this gives us a really good start as we would not have to decrease or data set because it has already 40,000 records divided amongst 13 classes.

### **Class balance**

As we have performed two exploratory data analysis procedures the third procedure would be to check the class balance of the data set we have. to analyse the loss balance we have selected account plot which would be plotted using seaborn library. As we explode the data set there was a huge class imbalance. because the minority clause barely has 110 records. And this amount of data is not enough to train a good performing model. the class balance off the data set is very important as this defines the sensitivity and specificity of our model if we have very few data in one class and a huge amount of data in the other this might disturb the models performance to detect the minority class hence leaving us with poor results. Upon exploration which was conducted through seaborne count plot it was evident that there is a high class imbalance amongst our data this left us with few options as traditional models requiere glass balancing and are sensitive to class imbalance hence the model selection was based on these results.

### **Removing duplicate values**

As for dealing with strings data another thing was evident that except for the missing missing values we would have to check for the duplicate values as well for example if someone wrote a tweet “happy birthday” and their sentiment is happiness it is not necessary that we provide the same data repeatedly to the model to train it as it is a supervised learning project it is important to bear in mind that we should provide as much unseen data as we can while training the model. Firstly we use pandas command to check the overall count of the duplicated rows in order to set then we use pandas to print those duplicated records and finally we use the pandas command to drop the selected records.

To implement I utilize the pandas library to identify and remove duplicate Text entries within a DataFrame. It performs several essential functions, including calculating the count of duplicated values in the DataFrame using df\_fet.duplicated().sum(), filtering the DataFrame to show rows with duplicated 'content' column values, retrieving the indices of these duplicates, and ultimately dropping the corresponding rows. Additionally, it resets the DataFrame's index. The primary motivation behind implementing this code is to ensure data accuracy and integrity, as duplicated Text entries can distort analyses, particularly in tasks like sentiment analysis, where they may bias sentiment distribution and compromise insights' accuracy. By systematically eliminating duplicates, the code enhances data quality for more reliable subsequent analyses or modeling.

### **Token and character length analysis**

This code snippet serves as a comprehensive tool for analyzing and visualizing the distribution of token and character lengths within a dataset. It employs Python libraries such as matplotlib.pyplot for visualization, pandas for data manipulation, and scipy.stats for Z-score calculations to identify outliers. Assuming the existence of a DataFrame named df\_fet with 'content' (textual data) and 'sentiment' (label) columns, the code calculates token and character lengths for each 'content' entry and groups data by sentiment. It computes average lengths and Inter-Quartile Range(IQR) to identify outliers. The code visualizes these distributions using Kernel Density Estimation (KDE) plots, distinguishing between data before and after outlier removal and organizing the plots by sentiment. It also displays average lengths and identified outliers, providing valuable insights into potential data preprocessing needs based on statistical characteristics. This technique is a powerful tool for understanding and preparing textual data for analysis or modeling.

### **Feature reduction**

### **Splitting dataset**

This code snippet showcases the effective partitioning of a dataset into training, validation, and testing subsets, a fundamental process in machine learning model development. Leveraging the sklearn.model\_selection module in Python, particularly the train\_test\_split function, the code skillfully achieves this task. It initiates by segregating input features (x) and the target variable (y) from the main dataset (df\_fet). The dataset is then strategically divided, with 80% of the data allocated for training (ratio\_train), while the remaining 20% is equally distributed for validation (ratio\_val) and testing (ratio\_test). To ensure an equitable validation split, the code dynamically adjusts the validation ratio (ratio\_val) based on the relative size of the remaining data. Ultimately, this process yields distinct training, validation, and testing subsets (x\_train, y\_train, x\_val, y\_val) conforming to the adjusted ratios. These subsets are thoughtfully combined with their corresponding target variables to craft individual DataFrames (train\_df, val\_df, test\_df), facilitating seamless training and evaluation of machine learning models. This systematic approach ensures a balanced distribution of data across subsets, establishing a robust foundation for subsequent model development and analysis.

## **Data Pre-processing**

The first two processes are already performed which were dropping the columns and performing class mapping of data pre-processing. The following are the data pre-processing processes regarding the text normalisation as it is very critical for our project.

### **Text normalization**

As discussed before I focused on discussing the processes to be performed during the text normalisation. Whereas in this section I’ll focus on how I implemented those processes to conduct my project this will also include the reference to the code which is attached in the appendix of this thesis.

#### **Spelling correction**

The provided code snippet demonstrates the implementation of a spell-checking mechanism using the TextBlob library within the context of Python. TextBlob, a powerful tool for natural language processing, is employed to rectify spelling mistakes found in textual content. Notably, the TextBlob library holds widespread application in tasks related to language processing, with a specific focus on refining Text accuracy by addressing spelling inaccuracies. The procedure kicks off by importing the TextBlob class from the dedicated textblob library. This class serves as a container for Text data and offers a suite of language processing functionalities. The code's pivotal function, known as correct\_spelling, is tailored to rectify spelling errors present in input Text. Upon receiving a Text input, the code generates a TextBlob object, serving as a fundamental Text processing unit within the TextBlob library's framework. This object becomes the basis for invoking the correct() method, a proprietary feature of the library, which capitalizes on its in-built spelling correction mechanism. This, in turn, enables the code to mend spelling errors inherent in the Text. The outcome of this process is encapsulated in a TextBlob object. To facilitate further analysis, processing, or modeling, this object is transformed into a string format through the application of the str() function. The corrected Text, having undergone the spell-checking process, is thus made ready for utilization. In essence, this code adeptly harnesses the capabilities of the TextBlob library to significantly augment the quality of textual data by autonomously rectifying spelling inaccuracies. This holds immense utility in Text-driven applications such as sentiment analysis, Text summarization, and information retrieval. In such applications, the necessity for precise and error-free Text data is paramount to derive meaningful insights and conclusions.

#### **Denoising Text**

The utilization of the 'BeautifulSoup' library addresses a critical concern in Text data pre-processing, such as the presence of HTML tags. In Textual data, HTML tags are frequently embedded within the content, particularly when the data originates from web sources. These tags are responsible for formatting and structuring web content but are extraneous and potentially disruptive when analysing Text for natural language processing tasks(Saturn Cloud, 2023b). The 'BeautifulSoup' library serves as an invaluable tool in handling this issue. It allows the code to transform the Text into a 'BeautifulSoup' object, providing a structured representation of the content. Subsequently, it selectively extracts the actual Text while omitting HTML tags, effectively purging the data of any HTML-related elements(Richardson, 2023). This ensures that the resultant Text is clean, devoid of formatting artifacts, and ready for meaningful analysis.

Concurrently, the 'contractions' library plays a pivotal role in enhancing the clarity and consistency of the Textual content. It addresses the challenge of contractions, which are common linguistic constructs involving the compression of two words into one(Saleha Muzammil, 2023). Examples include "it's" for "it is" or "don't" for "do not." While contractions are prevalent in everyday language, they can introduce ambiguity and hinder accurate Text analysis. The 'contractions' library provides a solution by automatically expanding these contractions into their complete forms. By doing so, it ensures that the Text is consistent, free from linguistic ambiguities, and more comprehensible. This step is particularly valuable for natural language processing tasks, where precise interpretation of Text is essential for accurate results. In summary, the combined use of 'BeautifulSoup' and 'contractions' libraries empowers the comprehensive pre-processing of Text data, eliminating HTML-related artifacts and linguistic ambiguities, thereby preparing the data for effective natural language processing tasks.

#### **Remove name tags**

Moving forward a systematic approach to depersonalizing Text data by removing name tags, particularly those associated with usernames and mentions, using regular expressions (nay, 2021). Employing the 're' library for robust regular expression support, the central 'remove\_name\_tags' function takes a Text input containing potential name tags, like '@username' or '@mention,' and utilizes 're.sub()' to replace these occurrences with an empty string. The regular expression pattern '@\w+' effectively targets and removes username and mention patterns from the Text. This depersonalization enhances data privacy, rendering the content more suitable for various natural language processing tasks, as specific user references have been eliminated. In summary, this code emphasizes the importance of depersonalizing data to maintain privacy and objectivity, showcasing a methodical process for systematically removing name tags using regular expressions and contributing to the integrity of subsequent Text analysis tasks.

#### **Removing URLs**

The further pre-processing deploys an efficient method for removing URLs or web links from Text data, contributing to a cleaner and more focused dataset. This code employs the 're' library to harness the power of regular expressions for identifying and replacing URLs with an empty string, thereby enhancing the quality and relevancy of textual content. The 're' library is enlisted, showcasing its capability in dealing with regular expressions in python(Borislav Hadzhiev, 2023). The central function, 'Removing\_urls', serves as the linchpin for this process. The 'Removing\_urls' function takes an input Text containing URLs or web links and utilizes a pre-defined regular expression pattern to identify and subsequently replace these URLs. The regular expression pattern, 'https?://\S+|www.\S+', is meticulously constructed to match both the 'http' and 'https' variations of URLs, as well as those that start with 'www'. The '\S+' portion effectively captures the URL sequence following these prefixes. The 'sub()' function, integrated into the 're' library, is then employed to replace these identified URLs with an empty string, effectively removing them from the Text.

Upon execution of this function, the Text undergoes a transformation where URLs are successfully eradicated, contributing to a more concise, readable, and URL-free dataset. This optimized data is better suited for various natural language processing tasks, as irrelevant URLs are eliminated, focusing the content on the core textual information. In essence, this code highlights the power of regular expressions in purging URLs from Text data. The process is executed with precision and speed, resulting in a refined dataset that is more aligned with the intended analysis goals.

#### **Removing punctutations**

This process requires a concise and efficient method for removing punctuation marks from Text-based data, ultimately improving the data's coherence and readability. the process is appreciated by (Saturn Cloud, 2023a) then I make adept use of the 're' library, renowned for its proficiency in handling regular expressions, to seamlessly identify and eliminate punctuation marks, resulting in a Text content better suited for natural language processing tasks.

The core of this process revolves around the pivotal function, 'remove\_punctuation', which takes input Text and capitalizes on a pre-defined string of punctuation marks, denoted as 'punctuations'. This comprehensive string encompasses a wide array of punctuation symbols, spanning brackets, exclamation marks, parentheses, and more. Employing the 're.sub()' function, an integral component of the 're' library, the code meticulously replaces sequences of identified punctuation marks with an empty string, effectively expunging them from the Text.

The application of the 'remove\_punctuation' function yields a transformed Text dataset, cleansed of punctuation-related impediments. This transformation results in Text content that is notably clearer, streamlined, and conducive to various natural language processing endeavors. In summary, the process underscores the effectiveness of regular expressions in simplifying Text data by systematically eliminating punctuation marks. The resultant refined dataset not only enhances Text coherence but also lays a solid foundation for subsequent Text-based analyses and information extraction tasks.

#### **Removing lower case**

To lowercase the text an efficient procedure for converting Text data to lowercase, thereby promoting uniformity and standardization within a Text dataset. The approach has been improvised by ((Harshith, 2022))’s work the core logic of this process resides in the primary function, 'lower\_case', which leverages Python's built-in string manipulation functions to provide a concise and impactful solution. Specifically, the 'lower()' method, inherent to Python's string objects, is systematically applied by the function to convert all characters within the input Text to lowercase. This uniform transformation holds considerable significance, especially in the context of handling Text data originating from diverse sources or containing mixed case usage.

The motivation behind the implementation of this procedure lies in its capacity to establish a consistent foundation within Text data. Whether the Text data stems from various origins or is contributed by users with differing capitalization preferences, the act of converting all characters to lowercase ensures a level playing field for subsequent analytical processes. This consistency fosters several advantages, such as enhanced pattern recognition, improved search accuracy, and streamlined comparisons. In practical terms, implementing this code involves the straightforward invocation of the 'lower\_case' function, providing the target Text as an argument. The function seamlessly returns the Text with all characters transformed to lowercase, offering a simple yet effective means of enhancing the quality and uniformity of Text data for various natural language processing applications.

#### **Replacing numbers**

The carry out sentiment analysis a robust methodology for enhancing Text data by replacing numerical values with their corresponding word representations. This approach not only improves Text readability and coherence but also standardizes numeric information for more straightforward and consistent analysis(Villavicencio et al., 2021). The process harnesses the 'inflect' library, renowned for its capabilities in linguistic processing. At its core, the code commences by importing the 'inflect' library, a pivotal component for this operation. 'Inflect' is lauded for its proficiency in converting numbers into words, making it an ideal tool for transforming numeric entities found within Text. The key element of the code is the 'replace\_numbers\_in\_Text' function. This function takes a Text string as input and orchestrates a sequence of operations to achieve the desired transformation. The process begins by splitting the input Text into individual words using the 'split()' method, resulting in a list of words. Each word undergoes meticulous examination within a list comprehension. Within this comprehension, a critical check is performed to determine whether a word is numeric using the 'isdigit()' method. If a word is identified as numeric, the 'p.number\_to\_words()' function from the 'inflect' library takes center stage. This function converts the numeric value into its textual equivalent, providing a human-friendly representation. If a word lacks numeric attributes, its form remains unchanged. The processed words are then rejoined using the 'join()' function, yielding a modified Text where numeric entities have been effectively replaced with their textual counterparts. To implement this process, I must first import the 'inflect' library. Then, by invoking the 'replace\_numbers\_in\_Text' function and providing the target Text as input, the code adeptly carries out the substitution of numerical values with their textual counterparts. This execution significantly enhances Text comprehension and establishes a consistent numerical foundation, poised for subsequent natural language processing endeavors.

#### **Stopwords removal**

In sentiment analysis using informal chat data, the strategic management of stopwords is crucial for preserving valuable context while filtering out noise. A delicate balance must be struck to ensure that sentiment-related information is retained(Nitin Hardeniya et al., 2023). According to (Bharti et al., 2022) Among the stopwords worth considering for retention are negation words like "not," "no," and "never," which can profoundly alter sentiment. Personal pronouns like "I," "he," "she," and "they" provide critical insights into sentiment, as do words indicating strong emotions such as "love," "hate," "awesome," and "terrible." Intensity modifiers like "very," "extremely," and "so" can accentuate sentiment, while exclamations like "wow" and "ugh" signal heightened emotions. Question words like "why" and "how" offer essential context, and interjections like "well" and "okay" convey emotional nuances. Contractions such as "can't" and "won't" often carry sentiment implications, and in informal chat, emoticons and emojis can explicitly denote sentiment. Additionally, depending on the data's nature, certain slang words like "lit," "cool," and "lame" might carry sentiment connotations. The impact of stopwords can vary with context and analysis goals, making it advisable to experiment with different configurations to observe their effects on results. The implementation of process, introduces a robust approach to Text preprocessing by meticulously removing stopwords while preserving sentiment-relevant words, thereby enhancing textual data quality for subsequent natural language processing tasks. This proces leverages Python libraries, including 'pandas' for data management and 'nltk' for linguistic processing. It begins by importing these essential libraries. A list named 'sentiment\_related\_stopwords' is then thoughtfully curated, encompassing words associated with sentiments, emotions, and expressions. This list ensures that crucial sentiment-related words remain untouched during the stopwords removal process. Subsequently, a set named 'nltk\_stopwords' is created, housing standard English stopwords from the 'nltk' library, which are typically devoid of sentiment relevance and thus suitable for removal to enhance data quality. The 'remove\_stopwords' function disassembles the input Text into words and performs two significant operations: identifying and removing stopwords that are not sentiment-related, while retaining sentiment-related stopwords in a 'cleaned\_words' list. The process provides insights into the stopwords' impact, showcasing the frequency and list of removed stopwords along with their respective frequencies. Finally, the refined Text is reconstructed for further analysis, armed with increased precision and relevant sentiment content.

#### **Lemmatization**

The process implementation presents a systematic approach to Text lemmatization, an indispensable technique within the realm of natural language processing aimed at transforming words into their fundamental or root forms. This implementation harnesses the 'WordNetLemmatizer' class, an integral component of the 'nltk' library, celebrated for its competence in linguistic processing(GÜBÜR, 2021). To initiate the lemmatization process, I commence by importing the requisite 'WordNetLemmatizer' from the 'nltk' library. This class emerges as a linchpin, conferring the proces with the requisite lemmatization capabilities. The 'lemmatization' function unfolds by instantiating a 'WordNetLemmatizer' object, thus endowing the process with the capacity to conduct lemmatization(Ghanbari-Adivi and Mosleh, 2019). Subsequently, the input Text undergoes a transformation, partitioning it into discrete words, a preparatory step conducive to word-level lemmatization. Within this function, each word within the Text is systematically examined. The lemmatizer method, 'lemmatize()', is invoked to effectuate the conversion of each word into its fundamental form or lemma. These lemmatized words are meticulously cataloged within a designated list, 'Text', essentially supplanting the original words with their corresponding lemmas. In the final stages, the lemmatized words residing in the 'Text' list harmoniously reunite through the 'join()' operation, culminating in the reconstruction of a coherent Text. This lemmatized Text stands ready for seamless integration into subsequent phases of Text analysis, primed for advanced linguistic processing tasks.

### **Model based post-processing**

In sentiment analysis, Model-based Post-processing stands as a pivotal stage where the raw output generated by machine learning models is further refined and enhanced to produce more accurate and coherent results. In this section, we delve into the intricacies of Model-based Post-processing, shedding light on the techniques and tools employed to fine-tune the sentiment predictions obtained from our trained models. Through this process, we aim to mitigate any inherent biases, reduce noise, and improve the overall quality of the sentiment analysis results. Key steps include threshold adjustment, where we optimize the classification boundaries, extract features and limit token lengths which helps standardize the output classes for better interpretability and consistency. Our implementation of Model-based Post-processing reflects a commitment to delivering sentiment analysis outcomes that are not only data-driven but also refined to align with real-world nuances and expectations.

#### **Decision Tree Classifier**

The decisision tree classifier would be train and for that the post -processing is performed as given below these steps have been discussed before while designing.

##### **TF-IDF**

The provided process segment showcases an efficient approach to training and evaluating a Text classification model using the TF-IDF (Term Frequency-Inverse Document Frequency) vectorization technique in combination with a machine learning classifier. This process was motivated by the study of (Praveen Sujanmulk, 2021) which harnesses the capabilities of the 'Pipeline' class from the 'sklearn' library to create an organized and sequential data processing pipeline. The 'train\_model\_tfidf' function is the heart of this implementation. By accepting a model, data, and target values as parameters, the function generates a comprehensive pipeline for Text classification. Inside the function, a 'Pipeline' object is constructed, consisting of two stages: a 'TfidfVectorizer', responsible for converting the Text data into a numerical matrix using TF-IDF, and the provided model. The pipeline structure ensures seamless data transformation and model application. The function then fits the pipeline to the provided data and targets, effectively training the model with the Text data. The process proceeds to train a Decision Tree classifier using the 'train\_model\_tfidf' function, where 'x\_train' contains the training Text data and 'Y\_train' comprises the corresponding target labels. Subsequently, the model's predictive performance is tested using the test data, denoted by 'x\_test'. To quantify the model's effectiveness, the process calculates the accuracy of the predictions by comparing the actual target labels 'Y\_test' with the predicted labels 'y\_pred'. The calculated accuracy score is printed, providing insight into the model's classification performance. In summary, this process embodies an end-to-end approach to Text classification. It leverages the 'Pipeline' mechanism from 'sklearn' to encapsulate Text preprocessing and model training. The use of TF-IDF vectorization and a Decision Tree classifier demonstrates a comprehensive methodology to analyze and categorize Text data. This implementation is instrumental in automating the training and evaluation process for Text classification tasks.

#### **Bi-directional GRU with Glove twitter word embeddings**

As discussed while designing that the focus is to design a bidirectional GRU which

##### **Label encoding**

The provided process snippet underscores the process of encoding categorical labels into numerical values, a critical step in preparing data for machine learning tasks(Alakh Sethi, 2023). The implementation employs the 'LabelEncoder' class from the 'sklearn.preprocessing' module, and its objective is to transform textual or categorical labels into a format compatible with machine learning algorithms. The 'LabelEncoder' class, a component of the 'sklearn' library, is pivotal in this endeavor. Its primary purpose is to convert categorical labels into corresponding integer values, effectively facilitating their integration into machine learning models. The process's logic unfolds as follows, the 'LabelEncoder' instance 'le' is created, and then fitted to the training data's target labels ('y\_trn') using the 'fit\_transform()' method. This results in the transformation of textual or categorical labels into corresponding numerical representations. Subsequently, the same label encoding is applied to the test ('y\_tst') and validation ('y\_val') data. The 'transform()' method is used here, ensuring that the same encoding scheme is utilized across all datasets.

The process concludes by displaying the set of encoded labels within the training data. This action provides insight into the mapping of categorical labels to their respective numerical representations, aiding in understanding the encoding process. In summary, the process's core purpose is to convert categorical labels into a suitable format for machine learning models. The 'LabelEncoder' class, derived from the 'sklearn' library, plays a pivotal role in this conversion process. This implementation ensures that categorical labels are seamlessly transformed into numerical values, enabling the integration of label information into machine learning algorithms.

##### **Tokenization**

The process of tokenizing textual data into numerical sequences is a fundamental preprocessing step for natural language processing tasks. The essential task of converting Text into numeric sequences, a pivotal phase in preparing textual data for computational analysis(Aravindpai Pai, 2020). The 'Tokenizer' class sourced from the 'tensorflow.keras.preprocessing.Text' module is the lynchpin of this procedure, meticulously designed to facilitate the conversion of textual data into a machine-readable format. At its core, the 'Tokenizer' class is an essential tool for transforming raw Text data into a structured format that machine learning models can effectively process. The process is structured into three distinct phases. Firstly, tokenizer setup and vocabulary learning is discussed, the 'tokenizer' instance is instantiated, ingeniously equipped with an 'oov\_token' parameter set to 'UNK', signifying 'unknown'. This strategic decision provides a placeholder for words not found in the vocabulary. The 'fit\_on\_texts()' method is skillfully employed to glean insights from both the training and test datasets. This masterstroke generates a vocabulary repository housing frequent words, each associated with a unique index. Secondly, textual conversion to sequences is performed using the transformative prowess of the 'tokenizer' is then harnessed through the 'texts\_to\_sequences()' method. Applied consecutively to the training, test, and validation data, this maneuver translates the Text data into sequences of integers. Finally, visualization of sequences is conducted the true essence of this process is encapsulated in the printout of sequences representing the training data ('sequences\_train'). These sequences serve as a testament to the process's efficacy, where textual sentences undergo metamorphosis into index-laden counterparts.In a nutshell, the conversion of raw textual data into structured numeric sequences, fostering an environment conducive to machine learning analyses. The 'Tokenizer' class, meticulously curated from the 'tensorflow.keras.preprocessing.Text' module, serves as the vanguard of this endeavor. It impeccably orchestrates the transition, traversing the path from vocabulary establishment to the conversion of Text into interpretable numerical sequences. Moreover to finalize this process the sequence lengths are fixed according to the maximum sequence length calculated, the length of tokens vary hence I enable the code to either add a padding if the length is short or truncate the sequence short to maximum length.

##### **Word embeddings**

Moving forward, I’ll load the Glove embeddings to train our model. In the context of Twitter data as explained by (Pennington et al., 2023), consisting of an extensive dataset encompassing 2 billion tweets, 27 billion tokens, and a vocabulary of 1.2 million words, the GloVe model's core principle lies in its utilization of a global word-word co-occurrence matrix, recording word co-occurrence frequencies within the corpus. While constructing this matrix initially demands computational resources, it serves as a one-time investment, streamlining subsequent training iterations. This toolkit offers automated functionalities for compiling and preprocessing these co-occurrence statistics, ensuring independent execution of the central training process. GloVe fundamentally operates as a logarithmically bilinear model with a weighted least-squares objective, leveraging the ratios of word co-occurrence probabilities to capture semantic meaning. Visualization reveals distinctive horizontal bands in the generated word vectors, reflecting component-wise multiplicative interactions, particularly pronounced with high-frequency words, and localized vertical bands associated with clusters of related words sharing similar frequencies, contributing to GloVe's ability to embed semantic information within word vectors.

#### **DistilBERT Model**

In this process segment, we are preparing the input data for our DistilBERT-based sentiment analysis model. The primary objective is to tokenize and structure the text data into a format that can be effectively processed by the model.

##### **Tokenization**

The text data from the training, testing, and validation datasets (denoted as X\_trn, X\_tst, and X\_val, respectively) needs to be tokenized. Tokenization involves breaking down the text into smaller units, usually words or subwords, to create a numerical representation that the model can understand. This process is executed using the tokenizer object, which is a part of the Hugging Face Transformers library(Verma, 2021). The padding=True and truncation=True arguments indicate that we want to pad shorter sequences and truncate longer ones to ensure consistent sequence lengths. return\_tensors='np' specifies that we want NumPy arrays as output.

##### **Input IDs and attention Masking**

After tokenization, we extract two crucial components from the tokenized data: input IDs and attention masks. Input IDs represent the numerical IDs assigned to each token, and attention masks indicate which parts of the sequence the model should pay attention to during processing. These two components are essential for the DistilBERT model to function effectively.

##### **Padding**

To ensure uniformity across all datasets, we determine the minimum sequence length among the training, validation, and testing datasets. This is a crucial step because DistilBERT expects inputs to have consistent lengths. With the minimum sequence length identified, we proceed to truncate or pad the sequences from all datasets accordingly. Sequences that are longer than the minimum length are truncated, and sequences that are shorter are padded with zeros. This step ensures that all inputs have the same length.

## **Model training**

This section will highlight how I have trained the model by explaining the model development and the hyperparameter tuning.

### **Decision tree**

The Decision tree classifier is used from ‘SKlearn’ Library which has the pre-trained models available. Moreover the TF-IDF pipeline is implemented to run while the model is training allowing the features extracted to be fed to the model directly.The hyperparameters for this model are optimized and selected by the keras tuner hence the results are assumed to be from the

### **Bi-directional GRU with word embeddings**

The model architecture presented comprises several layers, each serving a distinct role in the overall model's structure. At the outset, an 'Embedding' layer is employed, which transforms the input data into a format suitable for neural network processing. In this specific architecture, it converts input sequences with a length of 211 into dense vectors of size 200. Subsequently, a 'Bidirectional' layer is introduced, a fundamental component for understanding the context of the input data. This layer operates bidirectionally, meaning it processes sequences both in the forward and backward directions, enabling the model to capture dependencies and context effectively. Here, it results in an output shape of (None, 211, 512), where 'None' denotes the batch size, 211 signifies the sequence length, and 512 indicates the dimensionality of the output. Following the first 'Bidirectional' layer, another 'Bidirectional' layer is applied. This layer further refines the understanding of the input data, reducing the output shape to (None, 211, 256), preserving essential information while managing computational complexity. The third 'Bidirectional' layer continues this contextual analysis, eventually producing an output shape of (None, 128), significantly reducing dimensionality for more compact representation. Lastly, a 'Dense' layer is introduced to perform the final classification task. It condenses the features learned from the preceding layers into a format suitable for predicting the output classes. In this specific configuration, it outputs a vector of size 6, corresponding to the number of classes for classification. In summary, this model architecture encompasses multiple layers, starting with data pre-processing through embedding, followed by bidirectional layers for context understanding, and concluding with a dense layer for classification. The total number of parameters in the model is 5,374,566, out of which 1,320,966 are trainable parameters, contributing to the model's capacity to learn from data during training. The figure displays the results. Figure 7 shows the model structure used to train.

**A diagram of a program

Description automatically generated**

Figure 7(Model Structure)

#### **Hyperparameters**

In this hyperparameter discussion, we will explore the key settings and choices made for the training of the GRU-based neural network model, specifically focusing on the optimizer, network architecture, and training parameters. These decisions are critical for the model's performance in a sentiment analysis task using Twitter data. The selected optimizer is the Adam optimizer, which is a widely used and efficient optimization algorithm for training deep neural networks. The learning rate for Adam is set to 0.0005. The learning rate determines the step size at which the model updates its weights during training. A smaller learning rate like 0.0005 is chosen to ensure gradual weight updates, which can lead to more stable convergence during training. The model architecture consists of several layers, each serving a specific purpose in feature extraction and classification. It begins with an 'Embedding' layer, which transforms the input data into dense vectors of size 200. These vectors are essential for neural network processing as they capture the semantic meaning of words. The 'trainable' parameter of the embedding layer is set to 'False,' indicating that the pre-trained word embeddings (weights) provided in 'embedding\_matrix' are not updated during training. This choice is made to retain the valuable pre-trained word representations. The core of the network comprises three 'Bidirectional' Gated Recurrent Unit (GRU) layers. Bidirectional layers process sequences in both forward and backward directions, enhancing the model's ability to capture contextual information. The first two GRU layers have 256 units and return sequences, preserving sequential information. The third GRU layer has 128 units and operates in a bidirectional manner without returning sequences. The model architecture concludes with a 'Dense' layer with 6 units (equal to the number of sentiment classes) and a 'softmax' activation function, which performs the final classification. The model is compiled using the 'categorical\_crossentropy' loss function, suitable for multi-class classification problems. The Adam optimizer is employed with the specified learning rate, and 'accuracy' is chosen as the evaluation metric. During training, a batch size of 64 samples is used, which means the model updates its weights after processing each batch of 64 training examples. The training process is run for 21 epochs, where each epoch represents one complete pass through the entire training dataset. The model's performance is monitored on a validation dataset provided by 'X\_vald' and 'y\_val,' and training logs are printed for analysis. In summary, the hyperparameters selected for this model aim to strike a balance between computational efficiency and model performance. The Adam optimizer with a small learning rate, along with the GRU-based network architecture, is designed to effectively capture the sentiment information in Twitter data. The training parameters, such as batch size and number of epochs, are chosen based on empirical observations to achieve reasonable convergence and model accuracy.

### **DistilBERT**

Architecture and Layers: The model architecture utilizes a pre-trained DistilBERT model as the base for feature extraction. It's essential to note that DistilBERT is a compact version of the original BERT model, making it computationally efficient. The top layer of DistilBERT is connected to a Dense layer with a softmax activation function, facilitating multi-class classification for sentiment analysis. In this specific implementation, a kernel regularization term (L2 regularization with a strength of 0.01) is added to mitigate overfitting. This regularization term helps prevent the model from becoming too complex and, subsequently, overfitting to the training data. This model is shown in figure 8.

**A diagram with text and images

Description automatically generated with medium confidence**

Figure 8(DistilBERT structure)

#### **hyperparameters**

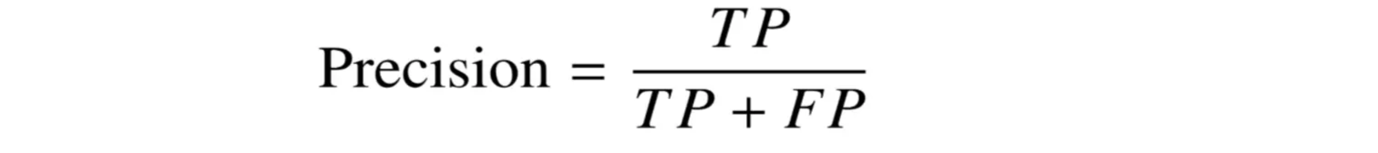
The hyperparameter configuration for the DistilBERT-based sentiment analysis model is a critical aspect of its design, as it significantly influences the model's performance and training process. We will discuss several key hyperparameters. The optimizer chosen for this model is Adam, which is a popular choice for training deep learning models. The learning rate is set to 2e-5, which is a relatively small value suitable for fine-tuning pre-trained models like DistilBERT. A small learning rate ensures that the model's weights are updated incrementally, preventing drastic changes that might disrupt the pre-trained knowledge. Additionally, an epsilon value of 1e-8 is used in Adam to prevent division by zero errors during optimization. The loss function for this model is categorical cross-entropy. This choice aligns with the multi-class classification nature of the sentiment analysis task, where the goal is to classify text into one of six sentiment categories. Categorical cross-entropy measures the dissimilarity between predicted and actual class probabilities, providing valuable feedback for model training. The batch size is set to 32, which determines the number of training examples used in each iteration of gradient descent. A batch size of 32 is a common choice and balances computational efficiency with convergence speed. The model is trained for 10 epochs, where each epoch represents one complete pass through the entire training dataset. The number of epochs should be chosen carefully to prevent overfitting (too many epochs) or underfitting (too few epochs) the data. In conclusion, the hyperparameter settings for this DistilBERT-based sentiment analysis model are tailored to balance model complexity, training efficiency, and generalization to the sentiment classification task. These settings have been chosen to fine-tune the pre-trained DistilBERT model effectively and achieve robust sentiment analysis results on the provided data. However, further hyperparameter tuning and experimentation may be conducted to optimize the model's performance for specific datasets or applications(Amissah, 2023).

# **Chapter 5: Evaluation**

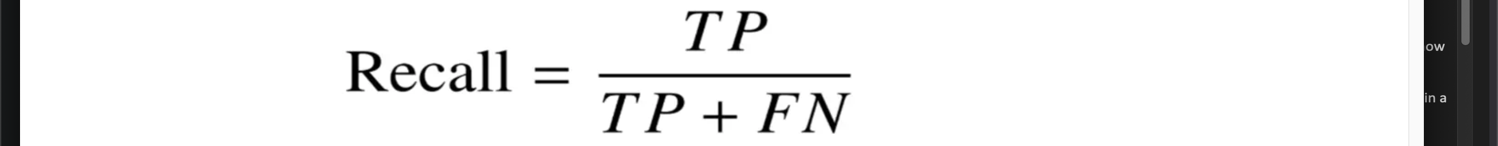
## **Evaluation test selection**

Evaluating the performance of a Text classification model is essential to understand how well it's working and to identify areas for improvement. Here are some key evaluation metrics and tests I should conduct for a Text classification model, accuracy calculates the percentage of correctly predicted instances out of the total instances. Precision, is a basic measure of overall performance. Precision measures the ratio of correctly predicted positive observations to the total predicted positives(LinkedIn, 2023). High precision indicates a low false positive rate.

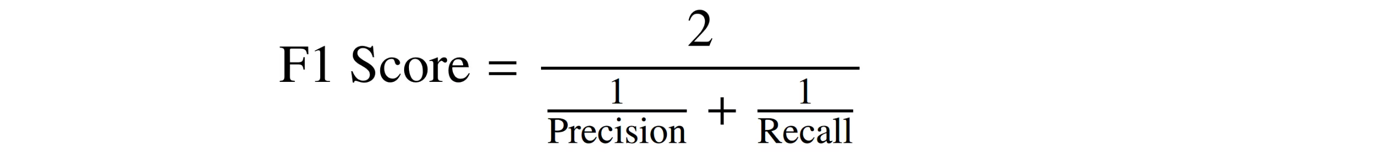
The formula below shows how precision is calculated using true positives and false positives(Rohit Kundu, 2023).



Recall measures the ratio of correctly predicted positive observations to the all observations in the actual class. High recall indicates a low false negative rate. Formula below shows how recall is calculated using True positives and false negatives.



F1-Score is the harmonic mean of precision and recall the below is used to calculate the harmonic mean of precision and recall.



It provides a balanced view of model performance when classes are imbalanced. Confusion Matrix, a table that shows the actual and predicted classes for a classification problem. It helps to understand the types of errors made by the model. ROC (Receiver Operating Characteristic) curve is a graphical representation of the true positive rate(Senstivity) against the false positive rate(1-specificity). AUC (Area Under the Curve) measures the area under the ROC curve and indicates the model's ability to distinguish between classes.. Learning Curves, Plot the training and validation performance as a function of the number of training examples(Gael Varoquaux, 2023). This helps identify issues like overfitting or underfitting. Remember that the choice of evaluation metrics depends on the nature of the problem, the class distribution, and the objectives of my project. It's also a good practice to consider both global metrics (like accuracy) and class-specific metrics, especially in imbalanced datasets.

## Evaluations metrics selection

The choice of evaluation metrics for a Text classification model depends on various factors, including the nature of the problem, class distribution, and business objectives. Here are some commonly used evaluation metrics for Text classification models. Accuracy, Suitable for balanced datasets where classes have roughly equal proportions. However, it can be misleading in highly imbalanced datasets. Precision, Recall, F1-Score, Precise metrics that provide insights into the model's performance on positive and negative classes. Particularly useful for imbalanced datasets. Area Under the ROC Curve (AUC-ROC), Effective when dealing with binary classification problems. Measures the model's ability to distinguish between classes across different thresholds. Reflects the trade-off between precision and recall. Log Loss (Cross-Entropy Loss), Measures the performance of a classification model that outputs probabilities. Penalizes confident incorrect predictions. Measures the fraction of incorrect labels in the predicted set of labels. Class-wise Metrics, Precision, recall, and F1-score calculated for each class individually.

## **Results**

Let’s explore the results of our trained model by using the tests and the metrics which we have decided earlier.

### Decision Tree

For the evaluation of decision tree model, a comprehensive analysis of the model's performance metrics derived from the classification report is presented. These metrics, including precision, recall, and the F1-score, offer a nuanced understanding of the model's effectiveness in classifying different sentiment categories. The precision scores for each sentiment class are scrutinized. Precision assesses the model's accuracy in classifying instances as positive for a particular class. Notably, 'joy' exhibits the highest precision, indicating that the model's predictions of 'joy' are frequently accurate. Conversely, 'other' demonstrates the lowest precision, suggesting challenges in correctly categorizing this sentiment. It's imperative to acknowledge that precision metrics provide insights into the model's ability to minimize false positives. Furthermore, recall scores are examined, portraying the model's capability to identify relevant instances within each class. 'Joy' and 'neutral' sentiments boast the highest recall values, implying that the model effectively captures the majority of instances for these sentiments. Conversely, 'other' exhibits the lowest recall, signifying difficulties in recognizing 'other' sentiments. These recall scores are instrumental in gauging the model's proficiency in minimizing false negatives. The F1-score, as a harmonious blend of precision and recall, furnishes a balanced evaluation metric. The F1-scores for each sentiment class reveal the model's equilibrium between precision and recall. Notably, 'joy' retains the highest F1-score, indicating an adept balance between precision and recall for this class. Conversely, 'other' demonstrates the lowest F1-score, emphasizing the challenge in achieving a harmonious balance between precision and recall. Finally, the overall accuracy, albeit a relevant metric, is assessed within the context of the entire dataset. An accuracy of 0.30 is observed, signifying the proportion of correctly classified instances among all instances. However, it's vital to underscore that accuracy may not comprehensively depict model performance, particularly in the presence of class imbalances or when specific sentiments hold greater significance.

In conclusion, this comprehensive evaluation of the model's performance highlights varying levels of proficiency across different sentiment classes. The outcomes emphasize the need for model refinement, particularly for sentiments falling within the 'other' category. These insights are invaluable for iterative model enhancements, ultimately contributing to the model's effectiveness in classifying Twitter sentiments. The classification report is displayed in figure 9.

A screenshot of a computer screen

Description automatically generated

Figure 9

After a meticulous examination of the classification report, it becomes evident that the model's performance is suboptimal. To gain a more detailed insight into the model's classification behavior, a visualization of the confusion matrix is employed. The confusion matrix encapsulates the model's actual classification outcomes, shedding light on its capacity to correctly identify and categorize instances. Upon a thorough analysis of the confusion matrix, a disconcerting pattern emerges. It is glaringly apparent that the model is failing to make accurate predictions across all sentiment classes. In the depicted figure, it becomes evident that for each sentiment category, the cumulative count of false positives significantly surpasses that of true positives. This concerning trend underscores the model's inherent challenges in correctly identifying and classifying sentiments. In light of these findings, it is prudent to conclude that the model's performance is unsatisfactory. The prevalence of false positives across all sentiment categories highlights a fundamental deficiency in the model's ability to accurately classify Twitter sentiments. This observation calls for a critical reassessment of the model's architecture and training process, with the aim of rectifying its shortcomings and enhancing its performance in sentiment analysis tasks. Figure 10 shows the confusion matrix

A blue squares with white text

Description automatically generated

Figure 10(DT confusion matrix)

### **Bi-directional GRU with word embeddings**

To commence the model evaluation, the first step involves a detailed analysis of the training and validation loss curves. This is a fundamental practice when working with recurrent neural networks (RNNs) as it provides crucial insights into the model's performance during training. The primary purpose is to assess whether the model is overfitting the data, a common challenge in deep learning. Despite rigorous hyperparameter optimization through techniques such as Keras Tuner, as well as the implementation of layer-wise regularization and dropout, the model exhibits clear signs of overfitting. Overfitting occurs when the model performs exceptionally well on the training data but struggles to generalize to unseen or validation data. This is evident from the divergence between the training loss and the validation loss curves. In an effort to mitigate overfitting, the model underwent several iterations with regularization techniques and dropout layers. However, these interventions did not yield the desired outcome, as indicated by the persistently increasing gap between the training and validation loss curves. Ultimately, the decision was made to remove the regularization and dropout layers due to their limited effectiveness in addressing overfitting in this specific context. Despite these efforts, it is evident from the loss analysis that the model remains prone to overfitting, highlighting the need for alternative strategies to enhance its generalization capability. In summary, the examination of training and validation loss curves serves as a critical initial evaluation technique for RNNs. In this case, the presence of overfitting was observed despite diligent hyperparameter tuning and regularization attempts, necessitating further exploration of alternative approaches to improve the model's performance. Figure 11 show the plot.

A graph of a graph with blue and orange lines

Description automatically generated

Figure 11

In the evaluation of the model's performance, a detailed examination of the classification report is a crucial starting point. The precision scores for various sentiment classes, namely neutral, fear, sadness, other, joy, and love, are indicative of the model's ability to correctly classify instances into these categories. Precision reflects the proportion of true positive predictions out of all positive predictions made by the model for a specific class. From the precision scores provided, it is notable that the highest precision is observed for 'sadness' (0.59), followed closely by 'fear' (0.47) and 'love' (0.45). Conversely, 'joy' (0.33) and 'other' (0.38) exhibit lower precision scores. 'Neutral' falls in the middle range with a precision of 0.36. These precision scores suggest that the model performs relatively well in correctly identifying 'sadness,' 'fear,' and 'love' sentiments but faces challenges in accurately classifying 'joy' and 'other' sentiments. Moving on to recall, which quantifies the model's ability to correctly identify true positive instances for each sentiment class relative to the total actual positive instances, the values for the various classes are notably diverse. 'Sadness' has the highest recall (0.55), indicating that the model effectively captures true positive 'sadness' cases. Conversely, 'joy' (0.09) and 'love' (0.08) exhibit considerably lower recall values, implying that the model struggles to correctly identify these sentiments. The remaining sentiments, 'neutral' (0.54), 'fear' (0.57), and 'other' (0.33), demonstrate moderate recall rates. These results underscore the model's proficiency in recognizing 'sadness' and 'fear' but highlight limitations in detecting 'joy' and 'love' sentiments. Lastly, the f1-score, which harmonizes precision and recall, is considered for a comprehensive evaluation of model performance. The f1-scores for the mentioned sentiment classes reveal that 'fear' (0.51) and 'neutral' (0.43) achieve the highest balance between precision and recall, suggesting robust performance. Conversely, 'joy' (0.14) and 'love' (0.13) exhibit notably lower f1-scores, indicating challenges in achieving both high precision and recall for these sentiments. The overall accuracy, while not the sole metric for assessment, stands at 0.41, representing the proportion of correctly classified instances across all sentiments.

In summary, the analysis of precision, recall, and f1-scores reveals variations in the model's performance across different sentiment classes. While the model excels in identifying 'sadness' and 'fear,' it faces difficulties in accurately classifying 'joy' and 'love.' These findings suggest the need for further model refinement, potentially through targeted data augmentation or architecture adjustments, to improve performance in sentiment analysis tasks. Figure 12 shows the classification report.

A screenshot of a computer screen

Description automatically generated

Figure 12

The subsequent evaluation technique in focus is the confusion matrix, a pivotal tool for gaining deeper insights into the model's performance. This matrix provides a detailed breakdown of how the model's predictions align with the actual class labels. Upon commencing the analysis, a discernible pattern emerges. It becomes apparent that the model's overall performance is suboptimal, characterized by a notable imbalance in true positive and false positive rates. While the true positives exhibit a slightly improved performance compared to the decision tree model, several critical issues come to light. One significant concern centers around specific emotions, namely sadness, joy, and love, which display an alarmingly low number of correctly classified instances. This is an unexpected outcome, considering that all the selected models were explicitly chosen for their insensitivity to class imbalance. This observation raises pertinent questions about the root causes of this imbalance, a topic that warrants further exploration. Intriguingly, the number of true positives is consistently lower than the number of false positives across multiple classes. This discrepancy underscores the model's struggle in correctly classifying instances, a finding that necessitates a closer examination of its underlying mechanisms and potential avenues for improvement. The identified challenges, including the imbalanced distribution of certain emotions and the imbalance in true positive and false positive rates, highlight the need for a more comprehensive investigation into the model's performance. This examination should encompass aspects such as data pre-processing, model architecture, and hyperparameter tuning to address the observed issues and enhance the model's classification capabilities. Figure 13 displays the confusion matrix.

A diagram of confusion matrix

Description automatically generated

Figure 13

In addition to the previously discussed evaluation techniques, another valuable approach to gauge the model's performance is the visualization of the Receiver Operating Characteristic (ROC) curve and the computation of the Area under the Curve (AUC). When the area under the receiver operating characteristic curve (AUC) falls between 0.5 and 1, it indicates a substantial likelihood that the classifier will successfully differentiate between positive and negative class values. This phenomenon occurs due to the classifier's ability to accurately identify a greater number of true positives and true negatives, while minimising the occurrence of false negatives and false positives(Aniruddha Bhandari, 2020). These metrics provide insights into the model's ability to discriminate between different classes by varying the classification threshold. While the preceding techniques suggested that the model's performance was suboptimal, the ROC curve presents a more nuanced perspective. The ROC curve illustrates the trade-off between true positive rate (sensitivity) and false positive rate (1-specificity) across different threshold values for each class. Although previous assessments indicated room for improvement, the ROC curve reveals that the model exhibits some capacity to distinguish between classes. The interpretation of the ROC curve is further supported by the computed AUC values. The AUC represents the area under the ROC curve and provides a quantitative measure of the model's discriminative ability. Notably, AUC values ranging from a minimum of 68% to a maximum of 82% are observed across various sentiment classes. These results signify that while the model may not be perfect, it is capable of classifying certain sentiments with a reasonably high level of confidence. However, it is important to acknowledge that the AUC values also indicate potential challenges in correctly detecting some classes, as evidenced by the lower AUC scores. The model may struggle with certain sentiments, leading to less distinct ROC curves and lower AUC values for those classes. In summary, the visualization of the ROC curve and the computation of AUC values provide a nuanced perspective on the model's performance. While it may not achieve perfect classification, the model demonstrates discriminative capacity, especially for sentiments associated with higher AUC values. These insights can guide further refinements in the model to enhance its performance in sentiment analysis tasks. Figure 14 displays the ROC curve.

A graph of a line graph

Description automatically generated with medium confidence

Figure 14

### **DistilBERT**

In the initial phase of evaluating the model, the focus is on interpreting the training and validation plots show in figure X. These plots serve as a foundational indicator of the model's overall performance and behavior. The observations drawn from these plots play a pivotal role in making informed decisions regarding the model's suitability for the task at hand. Upon scrutinizing the training and validation plots, a notable pattern emerges. The model exhibits a pronounced case of overfitting. Overfitting occurs when the model has learned to perform exceptionally well on the training data, to the point where it starts memorizing the training examples instead of learning generalizable patterns. This phenomenon is discernible by the conspicuous divergence between the training and validation loss curves. The training loss curve portrays a decreasing trend, indicating that the model is becoming increasingly adept at fitting the training data. However, the validation loss curve tells a different story. It demonstrates no convergence or improvement, which is a strong indicator of overfitting. This means that while the model excels in reducing the training loss, it struggles to generalize to unseen data, leading to poor performance on validation or test datasets. In essence, the absence of convergence between the training and validation loss curves is a crucial red flag. It suggests that the model, as currently configured, might not deliver optimal performance in practice. Addressing the overfitting issue is a priority to enhance the model's generalization capabilities and, consequently, its effectiveness in making accurate predictions.figure 15 displays the resultsA graph showing a line of loss

Description automatically generated

Figure 15

In evaluating the model's performance, several critical metrics are considered, primarily drawn from the classification report. These metrics shed light on the model's effectiveness in correctly classifying different emotions, each revealing unique aspects of its performance. The precision score, which gauges the model's ability to accurately predict positive instances, presents varying results for different emotions. Notably, "fear" exhibits the highest precision at 0.42, followed by "other" at 0.35, "joy" at 0.34, "love" at 0.32, "sadness" at 0.30, and "neutral" at 0.00. The precision scores indicate that the model performs reasonably well in predicting "fear," "other," "joy," and "love," achieving scores above 0.30. However, it struggles with "neutral" sentiment, achieving a precision score of 0.00, implying that it fails to correctly classify this emotion. Moving on to recall, which assesses the model's capacity to identify all positive instances accurately, the results follow a different pattern. "Fear" emerges as the leader with a recall of 0.48, indicating that the model adeptly identifies this emotion. "Other" follows with a recall of 0.27, while "joy" attains a recall of 0.33. Notably, "neutral" shares a recall score of 0.00 with "sadness," revealing the model's inability to capture these emotions effectively. "Love" achieves a respectable recall score of 0.44. Analyzing these results, it becomes evident that while the model excels in recognizing emotions like "fear," "love," and "joy," it struggles significantly with "neutral" and "sadness." This discrepancy in performance underscores the model's limitations in effectively discerning these particular emotions. Finally, the F1-score, which strikes a balance between precision and recall, provides further insights into the model's overall performance. The scores reveal that "fear" has the highest F1-score at 0.45, followed by "joy" at 0.33, "other" at 0.31, "love" at 0.37, "sadness" at 0.31, and "neutral" at 0.00. In conclusion, the model displays variable performance across different emotions, excelling in some cases but struggling with others. The low precision, recall, and F1-score for "neutral" suggest a particular area of weakness. Overall accuracy is moderate at 0.35, indicating that there is room for improvement in enhancing the model's performance across all emotions. Figure 16 shows the results for the classification matrix.A screenshot of a computer screen

Description automatically generated

Figure 16

In the subsequent evaluation, the confusion matrix serves as a crucial tool for understanding the model's performance by revealing the distribution of true positives and false positives across different emotions. An analysis of the confusion matrix underscores several key insights into the model's classification abilities. First and foremost, the model's struggle with classifying the "neutral" emotion is evident. The confusion matrix clearly illustrates that the model fails to correctly classify any instances of "neutral" sentiment, as indicated by a high count in the false positive category. This inability to distinguish "neutral" sentiment highlights a significant limitation of the model. Furthermore, the confusion matrix reflects suboptimal performance in the classification of other emotions as well. For instance, the "sadness" emotion exhibits a notably high false positive rate, suggesting that the model frequently misclassifies instances as "sadness" when they belong to other categories. Similarly, the confusion matrix reveals that the model struggles to effectively classify the "joy" emotion, mirroring the observation of imbalanced true and false positives. This discrepancy indicates that the model's performance in recognizing "joy" sentiment is not robust. On a positive note, the "fear" emotion stands out as the only category where the model demonstrates a relatively balanced ratio of true and false positives, implying a more reliable ability to classify "fear" sentiment. The analysis of the confusion matrix reaffirms the model's challenges in classifying certain emotions, particularly "neutral," "sadness," and "joy." These findings underscore the need for further refinement and optimization of the model to enhance its capacity to discern nuanced emotions effectively. Figure 17 shows the confusion matrix for this model.

A diagram of a confusion matrix

Description automatically generated with medium confidence

Figure 17

Additionally, an alternative technique employed to gauge the model's performance involves visualizing the Receiver Operating Characteristic (ROC) curve and calculating the Area Under the Curve (AUC) for each class. While prior methods of evaluation indicated subpar model performance, the ROC curve provides further insight, revealing the model's inability to classify sentiments with even minimal confidence. Upon examining the ROC curve, it becomes evident that the model struggles to make decisive classifications, as it fails to achieve a consistently high True Positive Rate (sensitivity) across different classes. This implies that the model's predictions lack the level of certainty required for accurate sentiment classification. Furthermore, the AUC scores for each class range from a minimum of 57% to a maximum of 71%. These values underscore the model's inability to accurately detect and classify the various emotions, highlighting significant shortcomings in the model's training process. The AUC values falling below ideal levels further emphasize the need for substantial improvements in the model's predictive capabilities. The ROC curve and AUC analysis reinforce the challenges faced by the model in effectively classifying sentiments with confidence. The low AUC values and inconclusive ROC curve patterns underscore the necessity for comprehensive model enhancements to achieve more accurate and reliable sentiment analysis results. Figure 18 shows the ROC curve for this model.

A graph of different colored lines

Description automatically generated

Figure 18

# **Chapter 6: Conclusion**

In summary, the theoretical framework devised for sentiment analysis on Twitter data, with the intent of discerning nuanced emotions, has yielded outcomes below the anticipated standards. This suboptimal performance can be primarily attributed to the subtle differentiations that exist among emotions. For instance, phrases like 'Happy birthday' and 'nobody wished me happy birthday' carry significantly divergent meanings, emphasizing the challenge in distinguishing emotions solely based on subtle keyword variations. Moreover, the potential presence of inaccuracies within the annotated dataset must be acknowledged. As demonstrated in earlier instances, mislabeled data can introduce noise and inaccuracies into the feature set assigned to a specific emotion class, consequently hindering the model's performance. Furthermore, the underwhelming results may also be attributed to the dataset's limitations in terms of size. The dataset's relatively small scale may have restricted the model's capacity to generalize effectively and encompass the full spectrum of emotions prevalent in Twitter data. In light of these observations, it becomes evident that the achieved accuracy falls short of the initially envisioned goal. However, this calls for future strategies aligned with the project's objectives, as discussed earlier.

In future research endeavors, a promising avenue for enhancing the performance of sentiment analysis lies in the acquisition of a more extensive and diverse dataset, facilitated through the utilization of the Twitter API. This expanded dataset would encompass a broader spectrum of tweets, thereby affording the opportunity to capture a more comprehensive range of emotional expressions and nuances. This augmentation in data volume is pivotal for building more robust and reliable sentiment analysis models. To further augment the dataset's quality, it is imperative to employ advanced techniques such as K-Nearest Neighbors (KNN) for the purpose of segregating tweets based on their emotional content. This process would result in the creation of a new dataset, enriched with a substantial number of excerpts specifically tailored for comprehensive model training. The incorporation of KNN not only aids in enhancing dataset size but also refines the dataset's specificity to emotion categories, thus contributing to more precise model training. Nevertheless, it is vital to underscore the significance of rigorous manual data validation, even in the context of a larger dataset. This meticulous validation process is indispensable for identifying and rectifying potential mislabeling issues within the dataset. By doing so, the research aims to ensure that the trained models, while poised to achieve higher accuracy, do not inadvertently perpetuate or reinforce existing inaccuracies. Through this proactive approach, future research endeavors are poised to achieve more accurate and nuanced sentiment analysis on Twitter data, facilitating a deeper understanding of the complexities surrounding human emotion expression in the digital realm.

# Bibliography

A., V. and Sonawane, S.S., (2016) Sentiment Analysis of Twitter Data: A Survey of Techniques. *International Journal of Computer Applications*, 13911, pp.5–15.

Alakh Sethi, (2023) *Categorical Encoding | One Hot Encoding vs Label Encoding*. [online] Available at: https://www.analyticsvidhya.com/blog/2020/03/one-hot-encoding-vs-label-encoding-using-scikit-learn/ [Accessed 22 Sep. 2023].

Alec Go, Lei Huang, and Richa Bhayani, (2009) Twitter Sentiment analysis.

Altrabsheh, N., Cocea, M. and Fallahkhair, S., (2014) Learning Sentiment from Students’ Feedback for Real-Time Interventions in Classrooms. In: A. Bouchachia, ed., *Adaptive and Intelligent Systems*, Lecture Notes in Computer Science. Cham: Springer International Publishing, pp.40–49.

Amissah, K., (2023) Churn Prediction Analysis For Customer Retention at Vodafone. *Medium*. Available at: https://medium.com/@kodwoam/churn-prediction-analysis-for-customer-retention-at-vodafone-ae8774d22c09 [Accessed 21 Sep. 2023].

Aniruddha Bhandari, (2020) *Guide to AUC ROC Curve in Machine Learning : What Is Specificity? - Analytics Vidhya*. [online] Available at: https://www.analyticsvidhya.com/blog/2020/06/auc-roc-curve-machine-learning/ [Accessed 22 Sep. 2023].

Antari Pasaha, (2020) NLP - Expand contractions in Text Processing. *GeeksforGeeks*. Available at: https://www.geeksforgeeks.org/nlp-expand-contractions-in-text-processing/ [Accessed 21 Sep. 2023].

Appel, O., Chiclana, F., Carter, J. and Fujita, H., (2016) A hybrid approach to the sentiment analysis problem at the sentence level. *Knowledge-Based Systems*, 108, pp.110–124.

Aravindpai Pai, (2020) What is Tokenization in NLP? Here’s All You Need To Know. *Analytics Vidhya*. Available at: https://www.analyticsvidhya.com/blog/2020/05/what-is-tokenization-nlp/ [Accessed 22 Sep. 2023].

Badugu, S. and Suhasini, M., (2017) Emotion Detection on Twitter Data using Knowledge Base Approach. *International Journal of Computer Applications*, 162, pp.28–33.

Bharti, S.K., Varadhaganapathy, S., Gupta, R.K., Shukla, P.K., Bouye, M., Hingaa, S.K. and Mahmoud, A., (2022) Text-Based Emotion Recognition Using Deep Learning Approach. *Computational Intelligence and Neuroscience*, 2022, p.2645381.

Borislav Hadzhiev, (2023) *How to Remove URLs from Text in Python | bobbyhadz*. [online] Available at: https://bobbyhadz.com/blog/python-remove-url-from-text [Accessed 22 Sep. 2023].

Brownlee, J., (2016) A Gentle Introduction to XGBoost for Applied Machine Learning. *MachineLearningMastery.com*. Available at: https://machinelearningmastery.com/gentle-introduction-xgboost-applied-machine-learning/ [Accessed 22 Mar. 2023].

Camacho-Collados, J. and Pilehvar, M.T., (2018) *On the Role of Text Preprocessing in Neural Network Architectures: An Evaluation Study on Text Categorization and Sentiment Analysis*. Available at: http://arxiv.org/abs/1707.01780 [Accessed 22 Sep. 2023].

Chaturvedi, N., (2023) *Python — Map, Classes, Functions and Arguments with Implementation*. [online] Medium. Available at: https://medium.datadriveninvestor.com/python-map-classes-functions-and-arguments-with-implementation-f37c9c480cfd [Accessed 21 Sep. 2023].

Crowdflower, (2015) *Sentiment Analysis in Text - dataset by crowdflower*. [online] data.world. Available at: https://data.world/crowdflower/sentiment-analysis-in-text [Accessed 13 Sep. 2023].

Datarobot, (2019) *Using Machine Learning for Sentiment Analysis: a Deep Dive*. [online] DataRobot AI Platform. Available at: https://www.datarobot.com/blog/using-machine-learning-for-sentiment-analysis-a-deep-dive/ [Accessed 22 Sep. 2023].

Deepanshi, (2023) *Text Preprocessing in NLP with Python codes*. [online] Available at: https://www.analyticsvidhya.com/blog/2021/06/text-preprocessing-in-nlp-with-python-codes/ [Accessed 22 Sep. 2023].

Etaiwi, W. and Naymat, G., (2017) The Impact of applying Different Preprocessing Steps on Review Spam Detection. *Procedia Computer Science*, 113, pp.273–279.

Fernández-Gavilanes, M., Álvarez-López, T., Juncal-Martínez, J., Costa-Montenegro, E. and González-Castaño, F.J., (2016) Corrigendum to ‘Unsupervised method for sentiment analysis in online texts’ Expert Systems with Applications 58 (2016) 57-75. *Expert Systems with Applications: An International Journal*, 61C, p.394.

Gael Varoquaux, (2023) *3.6. scikit-learn: machine learning in Python — Scipy lecture notes*. [online] Available at: https://scipy-lectures.org/packages/scikit-learn/index.html [Accessed 21 Sep. 2023].

Gaye, B., Zhang, D. and Wulamu, A., (2021) A Tweet Sentiment Classification Approach Using a Hybrid Stacked Ensemble Technique. *Information*, 129, p.374.

GDPR, (2023) *General Data Protection Regulation (GDPR) – Official Legal Text*. [online] General Data Protection Regulation (GDPR). Available at: https://gdpr-info.eu/ [Accessed 22 Sep. 2023].

Ghanbari-Adivi, F. and Mosleh, M., (2019) Text emotion detection in social networks using a novel ensemble classifier based on Parzen Tree Estimator (TPE). *Neural Computing and Applications*, 3112, pp.8971–8983.

Gill, A.J., (2016) Identifying Emotional Characteristics from Short Blog Texts.

Goyal, G., (2021) Twitter Sentiment Analysis Using Python | Introduction & Techniques. *Analytics Vidhya*. Available at: https://www.analyticsvidhya.com/blog/2021/06/twitter-sentiment-analysis-a-nlp-use-case-for-beginners/ [Accessed 21 Sep. 2023].

GÜBÜR, K.T., (2021) NLTK Lemmatization: How to Lemmatize Words with NLTK? - Holistic SEO. Available at: https://www.holisticseo.digital/python-seo/nltk/lemmatize [Accessed 22 Sep. 2023].

Gupta, P., Gupta, M., Kansal, R., Singh, L., Khotani, M. and Gupta, V., (2021) Analysis of Twitter data to Identify Intuitive Mental Health Well Being. In: *2021 5th International Conference on Information Systems and Computer Networks (ISCON)*. 2021 5th International Conference on Information Systems and Computer Networks (ISCON). pp.1–4.

Harshith, (2022) *Text Preprocessing in Natural Language Processing using Python*. [online] Medium. Available at: https://towardsdatascience.com/text-preprocessing-in-natural-language-processing-using-python-6113ff5decd8 [Accessed 22 Sep. 2023].

Hosseini, S., (2023) Optimizing Twitter Sentiment Analysis with Text Preprocessing. *MLearning.ai*. Available at: https://medium.com/mlearning-ai/optimizing-twitter-sentiment-analysis-with-text-preprocessing-334f78f7ae1c [Accessed 21 Sep. 2023].

Hu, M. and Liu, B., (2004) Mining and summarizing customer reviews. In: *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD ’04. [online] New York, NY, USA: Association for Computing Machinery, pp.168–177. Available at: https://doi.org/10.1145/1014052.1014073 [Accessed 22 Sep. 2023].

Ji, X., Chun, S.A., Wei, Z. and Geller, J., (2015) Twitter sentiment classification for measuring public health concerns. *Social Network Analysis and Mining*, 51, p.13.

Khanna, C., (2021) *Text pre-processing: Stop words removal using different libraries*. [online] Medium. Available at: https://towardsdatascience.com/text-pre-processing-stop-words-removal-using-different-libraries-f20bac19929a [Accessed 22 Sep. 2023].

Kumar, G., (2023) Role of Tokenization in NLP. *Medium*. Available at: https://kmr-gautam2893.medium.com/role-of-tokenization-in-nlp-18057618a102 [Accessed 21 Sep. 2023].

LinkedIn, (2023) *What are your best practices for measuring success in Machine Learning?* [online] Available at: https://www.linkedin.com/advice/1/what-your-best-practices-measuring-success-machine [Accessed 21 Sep. 2023].

Lohiya, H., (2018) Cleaning and pre-processing textual data. *Medium*. Available at: https://himanshulohiya.medium.com/cleaning-and-pre-processing-textual-data-d88036a1f4b8 [Accessed 21 Sep. 2023].

Morgenthaler, S., (2009) Exploratory data analysis. *WIREs Computational Statistics*, 11, pp.33–44.

Morozov, D.A., (2021) Sentiment Analysis of Twitter Text And Visualization Methods. *Medium*. Available at: https://dennisalexandermorozov.medium.com/sentiment-analysis-of-twitter-text-and-visualization-methods-d6ab365ccfbc [Accessed 21 Sep. 2023].

Nasim, Z., Rajput, Q. and Haider, S., (2017) Sentiment analysis of student feedback using machine learning and lexicon based approaches. In: *2017 International Conference on Research and Innovation in Information Systems (ICRIIS)*. 2017 International Conference on Research and Innovation in Information Systems (ICRIIS). pp.1–6.

nay, (2021) Answer to ‘Remove mentions and special characters for twitter dataset’. *Stack Overflow*. Available at: https://stackoverflow.com/a/68125240 [Accessed 22 Sep. 2023].

Nishanth N, (2020) *TextBlob Spelling Correction*. [online] Medium. Available at: https://towardsdatascience.com/textblob-spelling-correction-46321fc7f8b8 [Accessed 21 Sep. 2023].

Nitin Hardeniya, Jacob Perkins, Deepti Chopra, Nisheeth Joshi, and Iti Mathur, (2023) *Stop word removal - Natural Language Processing: Python and NLTK [Book]*. [online] Available at: https://www.oreilly.com/library/view/natural-language-processing/9781787285101/ch02s07.html [Accessed 22 Sep. 2023].

Pennington, J., Socher, R. and Manning, C.D., (2023) *GloVe: Global Vectors for Word Representation*. [online] Available at: https://nlp.stanford.edu/projects/glove/ [Accessed 20 Sep. 2023].

Praveen Sujanmulk, (2021) *Sentiment Analysis on Amazon Reviews using TF-IDF Approach. | by Praveen Sujanmulk | Analytics Vidhya | Medium*. [online] Available at: https://medium.com/analytics-vidhya/sentiment-analysis-on-amazon-reviews-using-tf-idf-approach-c5ab4c36e7a1 [Accessed 22 Sep. 2023].

Prema Arokia Mary, G., Hema, M.S., Maheshprabhu, R. and Nageswara Guptha, M., (2021) Sentimental Analysis of Twitter Data using Machine Learning Algorithms. In: *2021 International Conference on Forensics, Analytics, Big Data, Security (FABS)*. 2021 International Conference on Forensics, Analytics, Big Data, Security (FABS). pp.1–5.

Rajput, Q., Haider, S. and Ghani, S., (2016) Lexicon-Based Sentiment Analysis of Teachers’ Evaluation. *Applied Computational Intelligence and Soft Computing*, 2016, p.e2385429.

Rathnayaka, P., Abeysinghe, S., Samarajeewa, C., Manchanayake, I., Walpola, M.J., Nawaratne, R., Bandaragoda, T. and Alahakoon, D., (2019) *Gated Recurrent Neural Network Approach for Multilabel Emotion Detection in Microblogs*. Available at: http://arxiv.org/abs/1907.07653 [Accessed 22 Sep. 2023].

Rendalkar, S. and Chandankhede, C., (2018) Sarcasm Detection of Online Comments Using Emotion Detection. In: *2018 International Conference on Inventive Research in Computing Applications (ICIRCA)*. 2018 International Conference on Inventive Research in Computing Applications (ICIRCA). pp.1244–1249.

Richardson, L., (2023) *beautifulsoup4: Screen-scraping library*. Available at: https://www.crummy.com/software/BeautifulSoup/bs4/ [Accessed 22 Sep. 2023].

Rights (OCR), O. for C., (2021) *Health Information Privacy*. [Text] HHS.gov. Available at: https://www.hhs.gov/hipaa/index.html [Accessed 22 Sep. 2023].

Rohit Kundu, (2023) *F1 Score in Machine Learning: Intro & Calculation*. [online] Available at: https://www.v7labs.com/blog/f1-score-guide, https://www.v7labs.com/blog/f1-score-guide [Accessed 22 Sep. 2023].

Saleha Muzammil, (2023) *How to deal with contractions in NLP*. [online] Educative. Available at: https://www.educative.io/answers/how-to-deal-with-contractions-in-nlp [Accessed 22 Sep. 2023].

Sarlan, A., Nadam, C. and Basri, S., (2014) Twitter sentiment analysis. In: *Proceedings of the 6th International Conference on Information Technology and Multimedia*. Proceedings of the 6th International Conference on Information Technology and Multimedia. pp.212–216.

Saturn Cloud, (2023a) *Fast Punctuation Removal with Pandas | Saturn Cloud Blog*. [online] Available at: https://saturncloud.io/blog/fast-punctuation-removal-with-pandas/ [Accessed 22 Sep. 2023].

Saturn Cloud, (2023b) *How to Remove HTML Tags in Pandas A Data Scientists Guide | Saturn Cloud Blog*. [online] Available at: https://saturncloud.io/blog/how-to-remove-html-tags-in-pandas-a-data-scientists-guide/ [Accessed 22 Sep. 2023].

Schreiner, M., Fischer, T. and Riedl, R., (2021) Impact of content characteristics and emotion on behavioral engagement in social media: literature review and research agenda. *Electronic Commerce Research*, 212, pp.329–345.

Shahul ES, (2022) *Data Augmentation in NLP: Best Practices From a Kaggle Master*. [online] neptune.ai. Available at: https://neptune.ai/blog/data-augmentation-nlp [Accessed 21 Sep. 2023].

Sophia Yang, (2023) *Introduction to Text Analysis with Python in Excel*. [online] Anaconda. Available at: https://www.anaconda.com/blog/introduction-to-text-analysis-with-python-in-excel [Accessed 21 Sep. 2023].

Soyusiawaty, D. and Wolley, D.H.R., (2021) Hybrid Spelling Correction and Query Expansion for Relevance Document Searching. *International Journal of Advanced Computer Science and Applications*, [online] 128. Available at: http://thesai.org/Publications/ViewPaper?Volume=12&Issue=8&Code=IJACSA&SerialNo=38 [Accessed 22 Sep. 2023].

S.S. Verma, (2021) Applications And Benefits Of Emotion-Sensing Technology. Available at: https://www.electronicsforu.com/technology-trends/applications-benefits-emotion-sensing-technology [Accessed 22 Sep. 2023].

Steinert, S. and Dennis, M.J., (2022) Emotions and Digital Well-Being: on Social Media’s Emotional Affordances. *Philosophy & Technology*, 352, p.36.

Susan Li, (2018) *A Beginner’s Guide on Sentiment Analysis with RNN | by Susan Li | Towards Data Science*. [online] Available at: https://towardsdatascience.com/a-beginners-guide-on-sentiment-analysis-with-rnn-9e100627c02e [Accessed 22 Sep. 2023].

Taboada, M., Brooke, J., Tofiloski, M., Voll, K. and Stede, M., (2011) Lexicon-Based Methods for Sentiment Analysis. *Computational Linguistics*, 372, pp.267–307.

Turney, P., (2002) Thumbs Up or Thumbs Down? Semantic Orientation Applied to Unsupervised Classification of Reviews. In: *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*. [online] ACL 2002. Philadelphia, Pennsylvania, USA: Association for Computational Linguistics, pp.417–424. Available at: https://aclanthology.org/P02-1053 [Accessed 22 Sep. 2023].

Venugopalan, M. and Gupta, D., (2015) Exploring sentiment analysis on twitter data. In: *2015 Eighth International Conference on Contemporary Computing (IC3)*. 2015 Eighth International Conference on Contemporary Computing (IC3). pp.241–247.

Verma, A., (2021) *Python Guide to HuggingFace DistilBERT - Smaller, Faster & Cheaper Distilled BERT*. [online] Analytics India Magazine. Available at: https://analyticsindiamag.com/python-guide-to-huggingface-distilbert-smaller-faster-cheaper-distilled-bert/ [Accessed 22 Sep. 2023].

Vickery, R., (2020) *Text Cleaning Methods for Natural Language Processing*. [online] Medium. Available at: https://towardsdatascience.com/text-cleaning-methods-for-natural-language-processing-f2fc1796e8c7 [Accessed 21 Sep. 2023].

Villavicencio, C., Macrohon, J.J., Inbaraj, X.A., Jeng, J.-H. and Hsieh, J.-G., (2021) Twitter Sentiment Analysis towards COVID-19 Vaccines in the Philippines Using Naïve Bayes. *Information*, 125, p.204.

Zhang, L., Ghosh, R., Dekhil, M., Hsu, M. and Liu, B., (n.d.) Combining Lexicon-based and Learning-based Methods for Twitter Sentiment Analysis.

# Appendix

import re import nltk import string

import numpy as np import pandas as pd import seaborn as sns

import matplotlib.pyplot as plt

from nltk.tokenize import word\_tokenize from nltk.corpus import stopwords

from nltk.stem import SnowballStemmer, WordNetLemmatizer from sklearn.feature\_extraction.text import TfidfVectorizer import contractions

from bs4 import BeautifulSoup

from sklearn.preprocessing import LabelEncoder

from sklearn.model\_selection import train\_test\_split import tensorflow as tf

from tensorflow.keras.utils import model\_to\_dot from IPython.display import Image

from tensorflow.keras.models import load\_model from tensorflow.keras.utils import to\_categorical

from tensorflow.keras.preprocessing.text import Tokenizer

from tensorflow.keras.preprocessing.sequence import pad\_sequences from keras.regularizers import l2

from keras.layers import BatchNormalization from collections import defaultdict

from textblob import TextBlob import inflect

import os # Modelling

from sklearn.model\_selection import train\_test\_split, KFold, GridSearchCV from sklearn.ensemble import RandomForestClassifier

from sklearn.linear\_model import LogisticRegression from sklearn.tree import DecisionTreeClassifier

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report, f1\_score from sklearn.svm import SVC

from transformers import DistilBertTokenizer

from transformers import TFDistilBertForSequenceClassification, DistilBertConfig from tensorflow.keras.optimizers import Adam

from tensorflow.keras.losses import CategoricalCrossentropy from tensorflow.keras.metrics import CategoricalAccuracy from tensorflow.keras.models import Model

from tensorflow.keras.layers import Input, Dropout, Dense, BatchNormalization from kerastuner.tuners import RandomSearch

from kerastuner.engine.hyperparameters import HyperParameters import warnings

# Ignore all warnings warnings.filterwarnings("ignore")

2023-09-06 06:53:30.721320: I tensorflow/stream\_executor/platform/default/dso\_loader.cc:53] Successfully opened dynamic



Exploratory Data Analysis

# Read datasets

df = pd.read\_csv('./text\_emotion.csv')

#print first 5 rows df.head()

tweet\_id sentiment author content

1956967341 empty xoshayzers @tiffanylue i know i was listenin to bad habi...

1956967666 sadness wannamama Layin n bed with a headache ughhhh...waitin o...

1956967696 sadness coolfunky Funeral ceremony...gloomy friday...

1956967789 enthusiasm czareaquino wants to hang out with friends SOON!

1956968416 neutral xkilljoyx @dannycastillo We want to trade with someone w...

#print the number of null values in each column df.isnull().sum()

tweet\_id 0

sentiment 0

author 0

content 0

dtype: int64

df\_fet = df.drop(['tweet\_id', 'author'], axis=1) df\_fet

sentiment content

empty @tiffanylue i know i was listenin to bad habi...

sadness Layin n bed with a headache ughhhh...waitin o...

sadness Funeral ceremony...gloomy friday...

enthusiasm wants to hang out with friends SOON!

neutral @dannycastillo We want to trade with someone w...

**...** ... ...

**39995** neutral @JohnLloydTaylor

**39996** love Happy Mothers Day All my love **39997** love Happy Mother's Day to all the mommies out ther... **39998** happiness @niariley WASSUP BEAUTIFUL!!! FOLLOW ME!! PEE... **39999** love @mopedronin bullet train from tokyo the gf ... 40000 rows × 2 columns

plt.figure(figsize=(8,4)) sns.countplot(x='sentiment', data=df\_fet);

A graph of different colored bars

Description automatically generated

df\_fet.sentiment.value\_counts()

|  |  |
| --- | --- |
| neutral | 8638 |
| worry | 8459 |
| happiness | 5209 |
| sadness | 5165 |
| love | 3842 |
| surprise | 2187 |
| fun | 1776 |
| relief | 1526 |
| hate | 1323 |
| empty | 827 |
| enthusiasm | 759 |
| boredom | 179 |
| anger | 110 |

Name: sentiment, dtype: int64

def map\_new\_classes(sentiment):

if sentiment in ['neutral', 'worry', 'happiness', 'sadness', 'love', 'surprise', 'fun', 'relief', 'hate', 'empty', 'enthus return sentiment

elif sentiment == 'joyful': # Map 'excited' to 'excitement' return 'joy'

elif sentiment == 'fearful': # Map 'disgusted' to 'disgust'

return 'fear'

# Assuming your DataFrame is called 'df' and the column with classes is called 'sentiment'

df\_fet['sentiment'] = df\_fet['sentiment'].map(map\_new\_classes) # Check the updated class distribution print(df\_fet['sentiment'].value\_counts())

|  |  |
| --- | --- |
| neutral | 8638 |
| worry | 8459 |
| happiness | 5209 |

|  |  |
| --- | --- |
| sadness | 5165 |
| love | 3842 |
| surprise | 2187 |
| fun | 1776 |
| relief | 1526 |
| hate | 1323 |
| empty | 827 |
| enthusiasm | 759 |
| boredom | 179 |
| anger | 110 |

Name: sentiment, dtype: int64

# Create a mapping of old classes to new classes class\_mapping = {

'anger': 'other',

'love': 'love', 'sadness': 'sadness', 'surprise': 'other',

'joy': 'joy',

'fear': 'fear', 'neutral': 'neutral', 'happiness': 'joy',

'fun': 'joy',

'relief': 'joy',

'hate': 'other',

'empty': 'sadness',

'enthusiasm': 'joy', 'boredom': 'sadness', 'worry': 'fear',

'other': 'other',

}

# Map the old classes to new classes in the 'Emotion' column of your DataFrame df\_fet['sentiment'] = df\_fet['sentiment'].map(class\_mapping)

plt.figure(figsize=(8,4)) sns.countplot(x='sentiment', data=df\_fet);

# Check the updated class distribution print(df\_fet['sentiment'].value\_counts())

joy 9270

neutral 8638

fear 8459

sadness 6171

love 3842

other 3620

Name: sentiment, dtype: int64

95

#print the number of duplicated values df\_fet.duplicated().sum()

#print the rows which are duplicated df\_fet[df\_fet['content'].duplicated() == True]

|  |  |  |
| --- | --- | --- |
|  | **sentiment** | **content** |
| **366** | fear | I feel so deﬂated. No more doggy. |
| **521** | fear | Somebody please save the polar bears! |
| **1026** | neutral | I'm at work |
| **3684** | sadness | @dublins98dave me too! I am down 400 euro |
| **4363** | fear | is upset, I left my phone at home again |
| **...** | ... | ... |
| **39859** | love | Happy Mothers Day |
| **39898** | love | happy mothers day! |
| **39913** | joy | happy mother's day! |
| **39915** | love | happy mother's day everyone |
| **39945** | love | Happy Mother's Day to all the moms out there! |

173 rows × 2 columns

#removing duplicated text

index = df\_fet[df\_fet['content'].duplicated() == True].index df\_fet.drop(index, axis = 0, inplace = True) df\_fet.reset\_index(inplace=True, drop = True)

df\_fet

sentiment content

sadness @tiffanylue i know i was listenin to bad habi...

sadness Layin n bed with a headache ughhhh...waitin o...

sadness Funeral ceremony...gloomy friday...

joy wants to hang out with friends SOON!

neutral @dannycastillo We want to trade with someone w...

**...** ... ...

**39822** neutral @JohnLloydTaylor

**39823** love Happy Mothers Day All my love **39824** love Happy Mother's Day to all the mommies out ther... **39825** joy @niariley WASSUP BEAUTIFUL!!! FOLLOW ME!! PEE...

**39826** love @mopedronin bullet train from tokyo the gf ... 39827 rows × 2 columns

# Assuming df\_fet is your DataFrame with 'content' and 'sentiment' columns # Calculate token lengths and add as a new column

df\_fet['token\_length'] = df\_fet['content'].apply(lambda x: len(x.split()))

# Calculate character lengths and add as a new column df\_fet['char\_length'] = df\_fet['content'].apply(len)

# Group data by sentiment and calculate average token and character lengths grouped = df\_fet.groupby('sentiment')[['token\_length', 'char\_length']].mean()

# Calculate IQR for token and character lengths

Q1 = df\_fet[['token\_length', 'char\_length']].quantile(0.25) Q3 = df\_fet[['token\_length', 'char\_length']].quantile(0.75)

IQR = Q3 - Q1

# Define threshold for IQR to identify outliers iqr\_threshold = 1.5

# Detect outliers based on IQR outliers = df\_fet[

((df\_fet[['token\_length', 'char\_length']] < (Q1 - iqr\_threshold \* IQR)) | (df\_fet[['token\_length', 'char\_length']] > (Q3 + iqr\_threshold \* IQR))).any(axis=1)

]

# Plot the distribution of token lengths using KDE plt.figure(figsize=(12, 6))

for sentiment in df\_fet['sentiment'].unique():

sns.kdeplot(df\_fet[df\_fet['sentiment'] == sentiment]['token\_length'], label=sentiment, alpha=0.5) plt.title('Distribution of Token Lengths')

plt.xlabel('Token Length') plt.ylabel('Density') plt.legend()

plt.show()

# ... Similar plots for character lengths

# Plot the distribution of token lengths after removing outliers using KDE plt.figure(figsize=(12, 6))

for sentiment in df\_fet['sentiment'].unique():

sns.kdeplot(df\_fet[~df\_fet.index.isin(outliers.index) & (df\_fet['sentiment'] == sentiment)]['token\_length'], label=sentime plt.title('Distribution of Token Lengths (Outliers Removed)')

plt.xlabel('Token Length') plt.ylabel('Density') plt.legend()

plt.show()

# ... Similar plots for character lengths

# Display the average token and character lengths by sentiment print(grouped)

# Display the average token and character lengths by sentiment after removing outliers

grouped\_no\_outliers = df\_fet[~df\_fet.index.isin(outliers.index)].groupby('sentiment')[['token\_length', 'char\_length']].mean() print("Average Token and Character Lengths by Sentiment (Outliers Removed):")

print(grouped\_no\_outliers)

# Display the detected outliers print("Detected Outliers:") print(outliers)

|  |  |  |
| --- | --- | --- |
| sentiment | token\_length | char\_length |
| fear | 14.145431 | 76.821145 |
| joy | 13.666847 | 76.855612 |
| love | 13.290621 | 74.839894 |
| neutral | 11.373343 | 64.620144 |
| other | 13.866593 | 76.891780 |
| sadness | 13.583591 | 73.699431 |

A screenshot of a graph

Description automatically generatedAverage Token and Character Lengths by Sentiment (Outliers Removed):

|  |  |  |
| --- | --- | --- |
| sentiment | token\_length | char\_length |
| fear | 14.145431 | 76.821145 |
| joy | 13.666847 | 76.855612 |
| love | 13.290621 | 74.839894 |
| neutral | 11.373343 | 64.620144 |
| other | 13.866593 | 76.891780 |
| sadness | 13.583591 | 73.699431 |

Detected Outliers:

Empty DataFrame

Columns: [sentiment, content, token\_length, char\_length] Index: []

# Separate the input features (X) and the target variable (y) x = df\_fet['content']

y = df\_fet['sentiment']

# Defines ratios, w.r.t. whole dataset. ratio\_train = 0.8

ratio\_val = 0.1

ratio\_test = 0.1

|  |  |  |
| --- | --- | --- |
| # Produces test split.  x\_remaining, x\_test, y\_remaining, y\_test = train\_test\_split( x, y, test\_size=ratio\_test)  # Adjusts val ratio, w.r.t. remaining dataset. ratio\_remaining = 1 - ratio\_test ratio\_val\_adjusted = ratio\_val / ratio\_remaining  # Produces train and val splits.  x\_train, x\_val, y\_train, y\_val = train\_test\_split( x\_remaining, y\_remaining, test\_size=ratio\_val\_adjusted)  # Compiling into one dataframe  train\_df = pd.concat([y\_train, x\_train], axis=1) test\_df = pd.concat([y\_test, x\_test], axis=1) val\_df = pd.concat([y\_val, x\_val], axis=1) | | |
| train\_df |  |  |
|  | **sentiment** | **content** |
| **16097** | sadness | Shit night. want john where is he? |
| **16872** | fear | /me really sad that /me can't go to Java One |
| **8496** | neutral | @iNetters thank you |
| **24713** | fear | is taking the dog to the vet then a play date ... |
| **1557** | fear | @ErikVeland Dude... That sucks! Why would they... |
| **...** | ... | ... |
| **29991** | joy | Massive morning.. I'm stuffed now |
| **3275** | sadness | Woe, it's deﬁnitely NO fun to travel ﬁrst cl... |
| **35947** | love | @heycassadee thanks for following, it means a ... |
| **14899** | neutral | @mell\_e im going to my dance class now |
| **25394** | joy | Early monday cramming... yay. Only a few weeks... |

31861 rows × 2 columns

type(train\_df)

pandas.core.frame.DataFrame

def plot\_stopwords\_bar\_chart(df, sentiment):

print("Bar Chart of most frequent words for the sentiment: {}".format(sentiment))

temp\_df = df[df['sentiment'] == sentiment] print("Number of Rows: ", len(temp\_df))

corpus = ''

for text in temp\_df.content: text = str(text)

corpus += text

total = 0

count = defaultdict(lambda: 0) for word in corpus.split(" "):

total += 1

count[word] += 1

top20pairs = sorted(count.items(), key=lambda kv: kv[1], reverse=True)[:20] top20words = [i[0] for i in top20pairs]

top20freq = [i[1] for i in top20pairs]

xs = np.arange(len(top20words)) width = 0.5

fig, ax = plt.subplots(figsize=(10, 6))

ax.barh(xs, top20freq, height=width, align='center')

ax.set\_yticks(xs) ax.set\_yticklabels(top20words)

plt.gca().invert\_yaxis() # Invert y-axis to display words from top to bottom plt.xlabel('Frequency')

plt.title(sentiment)

plt.show()

# List of sentiments in your dataset

sentiments = ['neutral', 'joy', 'fear', 'sadness', 'love', 'other']

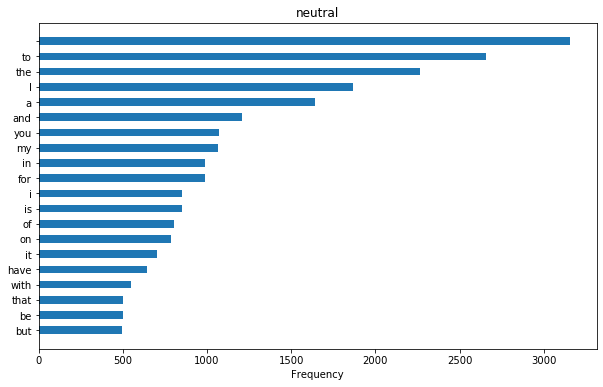
# Loop through each sentiment and plot the bar chart for sentiment in sentiments:

plot\_stopwords\_bar\_chart(df\_fet, sentiment)

Number of Rows: 6155

Bar Chart of most frequent words for the sentiment: sadness

Bar Chart of most frequent words for the sentiment: neutral Number of Rows: 8598

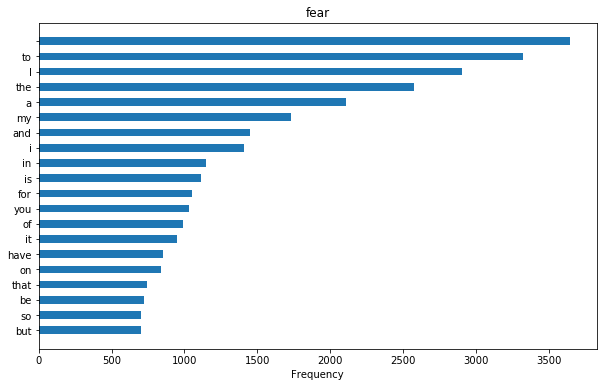


Bar Chart of most frequent words for the sentiment: joy Number of Rows: 9239

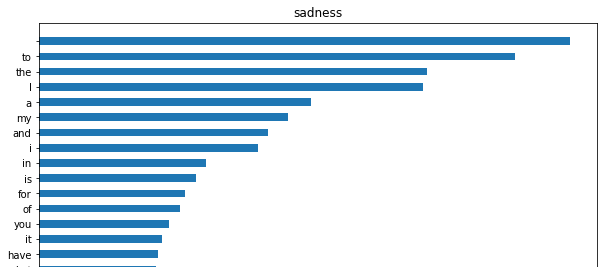
A blue and white rectangle with black border

Description automatically generated

Bar Chart of most frequent words for the sentiment: fear Number of Rows: 8437



Data Pre-processing



train\_df

A blue and white rectangle with black border

Description automatically generated

Bar Chart of most frequent words for the sentiment: love

Number of Rows: 3785

**1557** fear @ErikVeland Dude... That sucks! Why would they...

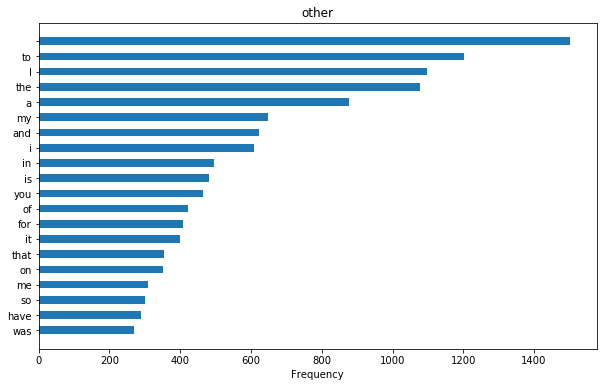
**...** ... ...

**29991** joy Massive morning.. I'm stuffed now **3275** sadness Woe, it's deﬁnitely NO fun to travel ﬁrst cl... **35947** love @heycassadee thanks for following, it means a ... **14899** neutral @mell\_e im going to my dance class now **25394** joy Early monday cramming... yay. Only a few weeks...

31861 rows × 2 columns

Bar Chart of most frequent words for the sentiment: other

Number of Rows: 3613



(3983, 2)

31861 rows × 2 columns

train\_df['content'] = train\_df['content'].apply(lambda text: denoise\_text(text)) val\_df['content'] = val\_df['content'].apply(lambda text: denoise\_text(text)) test\_df['content'] = test\_df['content'].apply(lambda text: denoise\_text(text)) train\_df

def denoise\_text(text):

# Strip html if any. For ex. removing <html>, <p> tags soup = BeautifulSoup(text, "html.parser")

text = soup.get\_text()

# Replace contractions in the text. For ex. didn't -> did not text = contractions.fix(text)

return text

|  |  |  |
| --- | --- | --- |
|  | **sentiment** | **content** |
| **16097** | sadness | Shit night. want john where is he? |
| **16872** | fear | /me really sad that /me can't go to Java One |
| **8496** | neutral | @iNetters thank you |
| **24713** | fear | is taking the dog to the vet then a play date ... |

|  |
| --- |
| # Function to correct spelling in a given text def correct\_spelling(text):  blob = TextBlob(text) corrected\_text = blob.correct() return str(corrected\_text)  # Apply spelling correction to train\_df  train\_df['content'] = train\_df['content'].apply(correct\_spelling)  # Apply spelling correction to val\_df  val\_df['content'] = val\_df['content'].apply(correct\_spelling)  # Apply spelling correction to test\_df  test\_df['content'] = test\_df['content'].apply(correct\_spelling) |
| train\_df.to\_csv('./sp\_train.csv', index=False) val\_df.to\_csv('./sp\_val.csv', index=False) test\_df.to\_csv('./sp\_test.csv', index=False) |
| train\_df = pd.read\_csv('./sp\_train.csv') val\_df = pd.read\_csv('./sp\_val.csv') test\_df = pd.read\_csv('./sp\_test.csv') |

|  |  |  |
| --- | --- | --- |
|  | **sentiment** | **content** |
| **0** | sadness | Hit night. want john where is he? |
| **1** | fear | /me really sad that /me cannot go to Cava One |
| **2** | neutral | @letters thank you |
| **3** | fear | is taking the dog to the met then a play date ... |
| **4** | fear | @ErikVeland Rude... That sucks! Why would they... |
| **...** | ... | ... |
| **31856** | joy | Passive morning.. I am stuffed now |
| **31857** | sadness | Toe, it is deﬁnitely of fun to travel ﬁrst c... |
| **31858** | love | @heycassadee thanks for following, it means a ... |
| **31859** | neutral | @mellow in going to my dance class now |
| **31860** | joy | Early monday charming... may. Only a few weeks... |

def remove\_name\_tags(text):

# Use regex to match @username and @mention patterns return re.sub(r'@\w+', '', text)

train\_df['content'] = train\_df['content'].apply(lambda text: remove\_name\_tags(text)) val\_df['content'] = val\_df['content'].apply(lambda text: remove\_name\_tags(text)) test\_df['content'] = test\_df['content'].apply(lambda text: remove\_name\_tags(text)) train\_df

|  |  |  |
| --- | --- | --- |
|  | **sentiment** | **content** |
| **0** | sadness | Hit night. want john where is he? |
| **1** | fear | /me really sad that /me cannot go to Cava One |
| **2** | neutral | thank you |
| **3** | fear | is taking the dog to the met then a play date ... |
| **4** | fear | Rude... That sucks! Why would they tow it fro... |
| **...** | ... | ... |
| **31856** | joy | Passive morning.. I am stuffed now |
| **31857** | sadness | Toe, it is deﬁnitely of fun to travel ﬁrst c... |
| **31858** | love | thanks for following, it means a lot to me i... |
| **31859** | neutral | in going to my dance class now |
| **31860** | joy | Early monday charming... may. Only a few weeks... |

31861 rows × 2 columns

def Removing\_urls(text):

url\_pattern = re.compile(r'https?://\S+|www\.\S+') return url\_pattern.sub(r'', text)

train\_df['content'] = train\_df['content'].apply(lambda text: Removing\_urls(text)) val\_df['content'] = val\_df['content'].apply(lambda text: Removing\_urls(text)) test\_df['content'] = test\_df['content'].apply(lambda text: Removing\_urls(text)) train\_df

|  |  |  |
| --- | --- | --- |
|  | **sentiment** | **content** |
| **0** | sadness | Hit night. want john where is he? |
| **1** | fear | /me really sad that /me cannot go to Cava One |
| **2** | neutral | thank you |
| **3** | fear | is taking the dog to the met then a play date ... |
| **4** | fear | Rude... That sucks! Why would they tow it fro... |
| **...** | ... | ... |
| **31856** | joy | Passive morning.. I am stuffed now |
| **31857** | sadness | Toe, it is deﬁnitely of fun to travel ﬁrst c... |
| **31858** | love | thanks for following, it means a lot to me i... |
| **31859** | neutral | in going to my dance class now |
| **31860** | joy | Early monday charming... may. Only a few weeks... |

31861 rows × 2 columns

punctuations =r'[.,#-\*/]+'

# Define a regular expression to match one or more punctuations

# Remove punctuations

train\_df['content'] = train\_df['content'].apply(lambda text: re.sub(punctuations, '', text)) val\_df['content'] = val\_df['content'].apply(lambda text: re.sub(punctuations, '', text)) test\_df['content'] = test\_df['content'].apply(lambda text: re.sub(punctuations, '', text))

train\_df

**31856** joy Passive morning I am stuffed now

**31857** sadness Toe it is deﬁnitely of fun to travel ﬁrst cl...

**31859** neutral in going to my dance class now

**31860** joy Early monday charming may Only a few weeks left 31861 rows × 2 columns

|  |  |  |
| --- | --- | --- |
|  | **sentiment** | **content** |
| **0** | sadness | Hit night want john where is he? |
| **1** | fear | me really sad that me cannot go to Cava One |
| **2** | neutral | thank you |
| **3** | fear | is taking the dog to the met then a play date ... |
| **4** | fear | Rude That sucks! Why would they tow it from y... |
| **...** | ... | ... |

def lower\_case(text): return text.lower()

**sentiment**

**content**

**0**

sadness

hit night want john where is he?

**1**

fear

me really sad that me cannot go to cava one

**2**

neutral

thank you

**3**

fear is taking the dog to the met then a play date ...

**4**

fear rude that sucks! why would they tow it from y...

**...**

...

...

**31856**

joy

passive morning i am stuffed now

**31857** sadness

toe it is deﬁnitely of fun to travel ﬁrst cl...

**31858**

love

thanks for following it means a lot to me i ...

**31859**

neutral

in going to my dance class now

**31860**

joy early monday charming may only a few weeks left

31861 rows × 2 columns

train\_df['content'] = train\_df['content'].apply(lambda text: lower\_case(text))

val\_df['content'] = val\_df['content'].apply(lambda text: lower\_case(text)) test\_df['content'] = test\_df['content'].apply(lambda text: lower\_case(text)) train\_df

thanks for following it means a lot to me i ...

love

**31858**

p = inflect.engine()

def replace\_numbers\_in\_text(text): words = text.split()

new\_words = [p.number\_to\_words(word) if word.isdigit() else word for word in words] return " ".join(new\_words)

train\_df['content'] = train\_df['content'].apply(lambda text: replace\_numbers\_in\_text(text)) val\_df['content'] = val\_df['content'].apply(lambda text: replace\_numbers\_in\_text(text)) test\_df['content'] = test\_df['content'].apply(lambda text: replace\_numbers\_in\_text(text)) train\_df

sentiment content

sadness hit night want john where is he?

fear me really sad that me cannot go to cava one

neutral thank you

fear is taking the dog to the met then a play date ...

fear rude that sucks! why would they tow it from yo...

**...** ... ...

**31856** joy passive morning i am stuffed now **31857** sadness toe it is deﬁnitely of fun to travel ﬁrst cl... **31858** love thanks for following it means a lot to me i lo... **31859** neutral in going to my dance class now **31860** joy early monday charming may only a few weeks left 31861 rows × 2 columns

|  |
| --- |
| sentiment\_related\_stopwords = [  "not", "no", "never", "none", "nobody", "nothing", "nowhere",  "love", "hate", "like", "dislike", "enjoy", "unlike",  "happy", "sad", "angry", "glad", "mad",  "good", "bad", "better", "worse",  "beautiful", "ugly", "pretty", "awesome", "terrible", "amazing", "very", "extremely", "so", "too",  "wow", "yay", "ugh",  "why", "how", "what", "when", "where", "well", "okay", "oops",  "can't", "won't", "ain't", ":-)", ":)", ":-(", ":(",  "imho", "lol", "omg", "btw", "brb", "great", "fantastic", "excellent", "best", "worst", "perfect",  "yes", "nope", "maybe", "please", "thank you", "hmm", "uh", "uhh", "uhm",  "haha", "hehe", "hi", "hey", "hello", "cool", "awesome", "amazing",  "even", "ever", "every", "everyone",  "but", "if", "however", "although",  "while", "unless", "whether", "except",  "should", "would", "could", "might",  "shall", "will", "can",  "now", "then", "later",  "since", "before", "after", "when", "while",  "about", "above", "below", "over", "under",  "also", "besides", "instead", "meanwhile",  "all", "any", "each", "few", "most", "some", "none",  "such", "several", "many", "much", "every", "whole",  "i", "me", "my", "mine", "myself",  "you", "your", "yours", "yourself",  "he", "him", "his", "himself",  "she", "her", "hers", "herself",  "it", "its", "itself",  "we", "us", "our", "ours", "ourselves",  "they", "them", "their", "theirs", "themselves"  ]  nltk\_stopwords = set(stopwords.words('english')) def remove\_stopwords(text):  words = text.split()  removed\_words = [word for word in words if word.lower() in nltk\_stopwords and word.lower() not in sentiment\_related\_stopw  # Create a list of stopwords removed  stopwords\_removed = [word for word in words if word.lower() in nltk\_stopwords and word.lower() not in sentiment\_related\_s cleaned\_words = [word for word in words if word.lower() not in nltk\_stopwords or word.lower() in sentiment\_related\_stopwo  # Calculate and display the total count of stopwords removed total\_stopwords\_removed = len(removed\_words)  print(f"Total stopwords removed: {total\_stopwords\_removed}")  # Display the list of stopwords removed print("Stopwords removed:") print(stopwords\_removed)  return ' '.join(cleaned\_words) |
| train\_df['content'] = train\_df['content'].apply(lambda text: remove\_stopwords(text)) val\_df['content'] = val\_df['content'].apply(lambda text: remove\_stopwords(text)) test\_df['content'] = test\_df['content'].apply(lambda text: remove\_stopwords(text)) train\_df |
| def lemmatization(text): lemmatizer= WordNetLemmatizer()  text = text.split() text=[lemmatizer.lemmatize(y) for y in text] return " " .join(text) |
| train\_df['content'] = train\_df['content'].apply(lambda text: lemmatization(text)) val\_df['content'] = val\_df['content'].apply(lambda text: lemmatization(text)) |

o

t r

test\_df['content'] = test\_df['content'].apply(lambda text: lemmatization(text)) train\_df

sentiment content

sadness hit night want john where he?

fear me really sad me cannot go cava one

neutral thank you

fear taking dog met then play date annabelle

fear rude sucks! why would they tow it your space?

**...** ... ...

**31856** joy passive morning i stuffed now **31857** sadness toe it deﬁnitely fun travel ﬁrst class bike ... **31858** love thanks following it mean lot me i love hey mon... **31859** neutral going my dance class now **31860** joy early monday charming may few week left 31861 rows × 2 columns

train\_df.to\_csv('./prepd\_train.csv', index=False) val\_df.to\_csv('./prepd\_val.csv', index=False) test\_df.to\_csv('./prepd\_test.csv', index=False)

train\_df\_ANN = pd.read\_csv('./prepd\_train.csv') val\_df\_ANN = pd.read\_csv('./prepd\_val.csv') test\_df\_ANN = pd.read\_csv('./prepd\_test.csv') train\_df\_ANN

sentiment content

sadness hit night want john where he?

fear me really sad me cannot go cava one

neutral thank you

fear taking dog met then play date annabelle

fear rude sucks! why would they tow it your space?

**...** ... ...

**31856** joy passive morning i stuffed now **31857** sadness toe it deﬁnitely fun travel ﬁrst class bike ... **31858** love thanks following it mean lot me i love hey mon... **31859** neutral going my dance class now **31860** joy early monday charming may few week left 31861 rows × 2 columns

# Check for NaN values in train\_df\_ANN['content'] print("NaN values in train\_df\_ANN['content']:") print(train\_df\_ANN[train\_df\_ANN['content'].isna()])

# Remove rows with NaN values from train\_df\_ANN train\_df\_ANN = train\_df\_ANN.dropna()

# Verify that NaN values are removed print("\nNaN values removed:") print("Train shape:", train\_df\_ANN.shape)

NaN values in train\_df\_ANN['content']:

|  |  |  |
| --- | --- | --- |
| 608 | sentiment  neutral | content  NaN |
| 1533 | neutral | NaN |
| 1957 | neutral | NaN |
| 2807 | fear | NaN |
| 2913 | joy | NaN |
| ... | ... | ... |
| 30242 | neutral | NaN |
| 30645 | fear | NaN |
| 31056 | neutral | NaN |
| 31455 | neutral | NaN |
| 31842 | neutral | NaN |

[82 rows x 2 columns]

NaN values removed:

Train shape: (31779, 2)

# Check for NaN values in val\_df\_ANN['content'] print("NaN values in val\_df\_ANN['content']:") print(val\_df\_ANN[val\_df\_ANN['content'].isna()])

# Remove rows with NaN values from val\_df\_ANN val\_df\_ANN = val\_df\_ANN.dropna()

# Verify that NaN values are removed print("\nNaN values removed from val\_df\_ANN:") print("Validation shape:", val\_df\_ANN.shape)

NaN values in val\_df\_ANN['content']:

|  |  |  |
| --- | --- | --- |
| 95 | sentiment  neutral | content  NaN |
| 125 | neutral | NaN |
| 201 | neutral | NaN |
| 733 | neutral | NaN |
| 784 | neutral | NaN |
| 1878 | neutral | NaN |
| 1892 | neutral | NaN |
| 2367 | neutral | NaN |
| 3013 | fear | NaN |
| 3123 | neutral | NaN |

NaN values removed from val\_df\_ANN: Validation shape: (3973, 2)

# Check for NaN values in test\_df\_ANN['content'] print("\nNaN values in test\_df\_ANN['content']:") print(test\_df\_ANN[test\_df\_ANN['content'].isna()])

# Remove rows with NaN values from test\_df\_ANN test\_df\_ANN = test\_df\_ANN.dropna()

# Verify that NaN values are removed print("\nNaN values removed from test\_df\_ANN:") print("Test shape:", test\_df\_ANN.shape)

NaN values in test\_df\_ANN['content']: sentiment content

|  |  |  |
| --- | --- | --- |
| 610 | neutral | NaN |
| 822 | neutral | NaN |
| 1109 | neutral | NaN |
| 2768 | neutral | NaN |
| 3460 | neutral | NaN |

NaN values removed from test\_df\_ANN: Test shape: (3978, 2)

train\_df\_ANN.to\_csv('./fin\_train.csv', index=False) val\_df\_ANN.to\_csv('./fin\_val.csv', index=False) test\_df\_ANN.to\_csv('./fin\_test.csv', index=False)

train\_df\_ANN = pd.read\_csv('./fin\_train.csv') val\_df\_ANN = pd.read\_csv('./fin\_val.csv') test\_df\_ANN = pd.read\_csv('./fin\_test.csv') train\_df\_ANN

|  |  |  |
| --- | --- | --- |
|  | **sentiment** | **content** |
| **0** | sadness | hit night want john where he? |
| **1** | fear | me really sad me cannot go cava one |
| **2** | neutral | thank you |
| type(test\_df\_ANN) | | |

pandas.core.frame.DataFrame

**...**

#Preprocess text

...

...

x\_train = train\_df\_ANN['content'].values

Y\_train = train\_df\_ANN['sentiment'].values

x\_test = test\_df\_ANN['content'].values Y\_test = test\_df\_ANN['sentiment'].values

x\_val = val\_df\_ANN['content'].values Y\_val = val\_df\_ANN['sentiment'].values

**4** fear rude sucks! why would they tow it your space?

**3** fear taking dog met then play date annabelle

TF-IDF

**31774** joy passive morning i stuffed now

**31775** sadness toe it deﬁnitely fun travel ﬁrst class bike ... **31776** love thanks following it mean lot me i love hey mon... **31777** neutral going my dance class now **31778** joy early monday charming may few week left

31779 rows × 2 columns

The word counts suffer some issues: most frequent words are usually not important (like stop words), while they take high focus/count. TFIDF (Term Frequency - Inverse Document) is a way to adjust those counts:

TF: #mentions within a document IDF: #mentions across all docs

So it gives higher importance to rare words across all docs (IDF++, TFIDF--), while it emphasyses on words appearing mostly in THIS doc (TF++, TFIDF--).

If a word appearing only in the current doc/sentence, it has TFIDF=1. If a word appears in all docs/sents but not the current one, it has TFIDF=0. If a word (stop word for example) appearing a lot in the current doc/sent and also in ALL others, it will have high TF (count) and much higher

IDF (discount), so low TFIDF overall.

def train\_model\_tfidf(model, data, targets):

# Create a Pipeline object with a TfidfVectorizer and the given model text\_clf = Pipeline([('vect',TfidfVectorizer()),

('clf', model)]) # Fit the model on the data and targets text\_clf.fit(data, targets)

return text\_clf

#Train the model with the training data

DT = train\_model\_tfidf(DecisionTreeClassifier(random\_state = 0), x\_train, Y\_train)

#test the model with the test data y\_pred=DT.predict(x\_test)

# Calculate confusion matrix

confusion = confusion\_matrix(Y\_test, y\_pred)

# Plot confusion matrix plt.figure(figsize=(8, 6))

plt.imshow(confusion, interpolation='nearest', cmap=plt.cm.Blues) plt.title('Confusion Matrix')

plt.colorbar()

classes = ['joy', 'fear', 'neutral', 'love', 'sadness', 'other'] tick\_marks = np.arange(len(classes))

plt.xticks(tick\_marks, classes, rotation=45) plt.yticks(tick\_marks, classes) plt.xlabel('Predicted Labels') plt.ylabel('True Labels')

# Display confusion matrix values on the plot for i in range(len(classes)):

for j in range(len(classes)):

plt.text(j, i, str(confusion[i][j]), horizontalalignment='center', color='white' if confusion[i][j] > confusion.max() plt.show()

# Classification Report

print("Classification Report:\n", classification\_report(Y\_test, y\_pred))

A screenshot of a computer

Description automatically generated

Classification Report:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | precision | recall | f1-score | support |
| fear | 0.30 | 0.33 | 0.31 | 808 |
| joy | 0.37 | 0.37 | 0.37 | 960 |
| love | 0.32 | 0.30 | 0.31 | 381 |
| neutral | 0.30 | 0.37 | 0.33 | 794 |
| other | 0.17 | 0.11 | 0.13 | 390 |
| sadness | 0.26 | 0.21 | 0.23 | 645 |
| accuracy |  |  | 0.30 | 3978 |
| macro avg | 0.29 | 0.28 | 0.28 | 3978 |
| weighted avg | 0.30 | 0.30 | 0.30 | 3978 |

Word embedding pipeline to fit the dataframe

train\_df = pd.read\_csv('./fin\_train.csv') val\_df = pd.read\_csv('./fin\_val.csv') test\_df = pd.read\_csv('./fin\_train.csv')

## Text Preprocessing

#Splitting the text from the labels X\_trn = train\_df['content']

y\_trn = train\_df['sentiment']

X\_tst = test\_df['content'] Y\_tst = test\_df['sentiment']

X\_val = val\_df['content'] y\_val = val\_df['sentiment']

y\_trn.value\_counts()

joy 7371

neutral 6852

fear 6804

sadness 4841

love 3042

other 2869

Name: sentiment, dtype: int64

# Encode labels

le = LabelEncoder()

y\_trn = le.fit\_transform(y\_trn) y\_tst = le.transform(Y\_tst) y\_val = le.transform(y\_val) #print the labels after encoding

original\_labels = le.inverse\_transform(y\_trn) print(set(y\_trn)) print(set(original\_labels))

{0, 1, 2, 3, 4, 5}

{'love', 'neutral', 'joy', 'other', 'sadness', 'fear'}

#Convert the class vector (integers) to binary class matrix y\_train = to\_categorical(y\_trn)

y\_test = to\_categorical(y\_tst) y\_val = to\_categorical(y\_val) print(y\_train)

[[0. 0. 0. 0. 0. 1.]

[1. 0. 0. 0. 0. 0.]

[0. 0. 0. 1. 0. 0.]

...

[0. 0. 1. 0. 0. 0.]

[0. 0. 0. 1. 0. 0.]

[0. 1. 0. 0. 0. 0.]]

Tokenizing

# Tokenize words

tokenizer = Tokenizer(oov\_token='UNK') tokenizer.fit\_on\_texts(pd.concat([X\_trn, X\_tst], axis=0))

sequences\_train = tokenizer.texts\_to\_sequences(X\_trn) sequences\_test = tokenizer.texts\_to\_sequences(X\_tst) sequences\_val = tokenizer.texts\_to\_sequences(X\_val)

#print the sentence after converting them to indexes sequences\_train

Padding

The vectors we obtain are generally not of equal lengths For that, we might need to pad the sequences to max len.

maxlen = max([len(t) for t in train\_df['content']]) maxlen

211

X\_train = pad\_sequences(sequences\_train, maxlen=maxlen, truncating='pre') X\_test = pad\_sequences(sequences\_test, maxlen=maxlen, truncating='pre') X\_vald = pad\_sequences(sequences\_val, maxlen=maxlen, truncating='pre')

vocabSize = len(tokenizer.index\_word) + 1 print(f"Vocabulary size = {vocabSize}")

Vocabulary size = 20268

#before sequences\_train[0]

[373, 43, 37, 982, 167, 28]

#after X\_train[0]

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| array([ | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, |
|  | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, |
|  | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, |
|  | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, |
|  | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, |
|  | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, |
|  | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, |
|  | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, |
|  | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, |
|  | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, |
|  | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, |
|  | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, |
|  | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, |
|  | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, |
|  | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, |
|  | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 0, | 373, | 43, | 37, |

982, 167, 28], dtype=int32)

Word Embedding

**This code defines the following variables:**

**path\_to\_glove\_file**: The file path to the GloVe word vectors file.

**num\_tokens**: The size of the vocabulary. This is typically the number of unique words in the dataset.

**embedding\_dim**: The dimensionality of the word vectors. This is usually set to a fixed value (e.g., 200, 300, etc.) depending on the size of the word vectors file that is being used.

**hits**: A counter for the number of words that are found in the embeddings\_index dictionary.

**misses**: A counter for the number of words that are not found in the embeddings\_index dictionary.

**embeddings\_index**: A dictionary that will store the word vectors, with the words as keys and the word vectors as values.

# download embeddings

if not os.path.isfile('./glove.twitter.27B.zip'):

!wget 'https://nlp.stanford.edu/data/glove.twitter.27B.zip'

!unzip './glove.twitter.27B.zip'

# Read GloVE embeddings

path\_to\_glove\_file = './glove.twitter.27B.200d.txt' num\_tokens = vocabSize

embedding\_dim = 200 #latent factors or features hits = 0

misses = 0 embeddings\_index = {}

I will intialize an embedding matrix with all zero values and then looping through the vocabulary (as defined by the tokenizer object) to assign word vectors to the corresponding rows of the embedding matrix. The word\_index attribute of the tokenizer object is a dictionary that maps words to their indices in the vocabulary.

The embedding matrix will have a size of num\_tokens rows and embedding\_dim columns, where num\_tokens is the size of the vocabulary and embedding\_dim is the dimensionality of the word vectors.

For each word in the vocabulary, the code looks up the corresponding word vector in the embeddings\_index dictionary and assigns it to the corresponding row of the embedding matrix. If the word is not found in the embeddings\_index dictionary (i.e., if embedding\_vector is None), the code increments the misses counter and the row of the embedding matrix remains all-zeros. If the word is found in the embeddings\_index

dictionary, the code increments the hits counter and assigns the word vector to the corresponding row of the embedding matrix. Finally, the code prints out the number of words that were converted (hits) and the number that were not found in the embeddings\_index dictionary

(misses).

# Read word vectors

with open(path\_to\_glove\_file) as f: for line in f:

word, coefs = line.split(maxsplit=1)

coefs = np.fromstring(coefs, "f", sep=" ") embeddings\_index[word] = coefs

print("Found %s word vectors." % len(embeddings\_index))

# Assign word vectors to our dictionary/vocabulary embedding\_matrix = np.zeros((num\_tokens, embedding\_dim)) for word, i in tokenizer.word\_index.items():

embedding\_vector = embeddings\_index.get(word) if embedding\_vector is not None:

# Words not found in embedding index will be all-zeros.

# This includes the representation for "padding" and "OOV" embedding\_matrix[i] = embedding\_vector

hits += 1 else:

misses += 1

print("Converted %d words (%d misses)" % (hits, misses))

Found 1193514 word vectors. Converted 13700 words (6567 misses)

Model Training

training GRU

#to stop the training when the loss starts to increase callback = EarlyStopping(

monitor="val\_loss", patience=3, restore\_best\_weights=True,

)

Embedding Layer: The model begins with an embedding layer. This layer converts integer-encoded input sequences into dense vectors of fixed size, allowing the model to learn relationships between different words based on their context. The weights parameter initializes the embedding layer with pre-trained word vectors, making the model aware of semantic relationships between words. The embedding layer is not trainable in your case (trainable=False), so the pre-trained embeddings will not be updated during training.

Bidirectional GRU Layers: The architecture employs three Bidirectional GRU layers stacked on top of each other. Each Bidirectional GRU processes the input sequence in both forward and backward directions, capturing contextual information from both past and future tokens. This bidirectional nature allows the model to better understand the context of each word within the entire sequence.

The first Bidirectional GRU layer has 256 units with dropout and recurrent dropout to prevent overfitting. It returns sequences because of return\_sequences=True, which is necessary for the subsequent layers to receive sequence data.

The second Bidirectional GRU layer also has 128 units, dropout, and recurrent dropout, and it also returns sequences.

The third Bidirectional GRU layer has 64 units, dropout, and recurrent dropout. It doesn't return sequences, as it's intended to provide a final representation of the entire sequence.

Dense Layer: The final layer of the model is a Dense layer with 6 units (assuming it's a multi-class classification problem) and a softmax activation function. This layer converts the learned features from the previous layers into class probabilities.

Now, comparing this architecture with a single layer of GRU:

Depth of Representation: The main advantage of the provided architecture is its depth. The stacking of multiple Bidirectional GRU layers allows the model to learn hierarchical features and abstract representations at different levels of granularity. It captures increasingly complex patterns in the input sequence.

Contextual Understanding: The use of multiple Bidirectional GRU layers helps the model understand the context of each word within a broader context of the entire sequence. This can be crucial for tasks like sentiment analysis, where the sentiment of a sentence can depend on the context of the words around it.

Feature Learning: Each Bidirectional GRU layer learns a different level of representation. Shallower layers may capture basic syntactic structures, while deeper layers capture more abstract semantics. This hierarchy of feature learning can improve the model's ability to distinguish between subtle linguistic nuances.

Overall, the provided architecture with stacked Bidirectional GRU layers is likely to perform better than a single-layer GRU architecture because it can learn more intricate and contextual representations from the input data. However, it's important to note that the effectiveness of the architecture can also depend on the specific characteristics of your dataset and task.

adam = Adam(learning\_rate=0.0005) model\_GRU\_2 = Sequential()

model\_GRU\_2.add(Embedding(vocabSize, 200, input\_length=maxlen, weights=[embedding\_matrix], trainable=False))

model\_GRU\_2.add(Bidirectional(GRU(256, return\_sequences=True))) model\_GRU\_2.add(Bidirectional(GRU(128, return\_sequences=True))) model\_GRU\_2.add(Bidirectional(GRU(64))) model\_GRU\_2.add(Dense(6, activation='softmax'))

model\_GRU\_2.compile(loss='categorical\_crossentropy', optimizer=adam, metrics=['accuracy']) model\_GRU\_2.summary()

2023-09-06 02:00:05.761359: I tensorflow/stream\_executor/platform/default/dso\_loader.cc:53] Successfully opened dynamic 2023-09-06 02:00:05.877739: I tensorflow/stream\_executor/cuda/cuda\_gpu\_executor.cc:937] successful NUMA node read from S 2023-09-06 02:00:05.878200: I tensorflow/core/common\_runtime/gpu/gpu\_device.cc:1733] Found device 0 with properties:

pciBusID: 0000:01:00.0 name: NVIDIA GeForce RTX 3090 computeCapability: 8.6

coreClock: 1.755GHz coreCount: 82 deviceMemorySize: 23.70GiB deviceMemoryBandwidth: 871.81GiB/s

2023-09-06 02:00:05.878224: I tensorflow/stream\_executor/platform/default/dso\_loader.cc:53] Successfully opened dynamic 2023-09-06 02:00:05.879860: I tensorflow/stream\_executor/platform/default/dso\_loader.cc:53] Successfully opened dynamic 2023-09-06 02:00:05.879894: I tensorflow/stream\_executor/platform/default/dso\_loader.cc:53] Successfully opened dynamic 2023-09-06 02:00:05.880717: I tensorflow/stream\_executor/platform/default/dso\_loader.cc:53] Successfully opened dynamic 2023-09-06 02:00:05.880859: I tensorflow/stream\_executor/platform/default/dso\_loader.cc:53] Successfully opened dynamic 2023-09-06 02:00:05.881345: I tensorflow/stream\_executor/platform/default/dso\_loader.cc:53] Successfully opened dynamic 2023-09-06 02:00:05.881741: I tensorflow/stream\_executor/platform/default/dso\_loader.cc:53] Successfully opened dynamic

2023-09-06 02:00:05.881821: I tensorflow/stream\_executor/platform/default/dso\_loader.cc:53] Successfully opened dynamic 2023-09-06 02:00:05.881873: I tensorflow/stream\_executor/cuda/cuda\_gpu\_executor.cc:937] successful NUMA node read from S 2023-09-06 02:00:05.882338: I tensorflow/stream\_executor/cuda/cuda\_gpu\_executor.cc:937] successful NUMA node read from S 2023-09-06 02:00:05.882765: I tensorflow/core/common\_runtime/gpu/gpu\_device.cc:1871] Adding visible gpu devices: 0

2023-09-06 02:00:05.883229: I tensorflow/core/platform/cpu\_feature\_guard.cc:142] This TensorFlow binary is optimized wit To enable them in other operations, rebuild TensorFlow with the appropriate compiler flags.

2023-09-06 02:00:05.883717: I tensorflow/stream\_executor/cuda/cuda\_gpu\_executor.cc:937] successful NUMA node read from S 2023-09-06 02:00:05.884152: I tensorflow/core/common\_runtime/gpu/gpu\_device.cc:1733] Found device 0 with properties:

pciBusID: 0000:01:00.0 name: NVIDIA GeForce RTX 3090 computeCapability: 8.6

coreClock: 1.755GHz coreCount: 82 deviceMemorySize: 23.70GiB deviceMemoryBandwidth: 871.81GiB/s

2023-09-06 02:00:05.884197: I tensorflow/stream\_executor/cuda/cuda\_gpu\_executor.cc:937] successful NUMA node read from S 2023-09-06 02:00:05.884654: I tensorflow/stream\_executor/cuda/cuda\_gpu\_executor.cc:937] successful NUMA node read from S 2023-09-06 02:00:05.885089: I tensorflow/core/common\_runtime/gpu/gpu\_device.cc:1871] Adding visible gpu devices: 0

2023-09-06 02:00:05.885116: I tensorflow/stream\_executor/platform/default/dso\_loader.cc:53] Successfully opened dynamic 2023-09-06 02:00:06.183368: I tensorflow/core/common\_runtime/gpu/gpu\_device.cc:1258] Device interconnect StreamExecutor 2023-09-06 02:00:06.183386: I tensorflow/core/common\_runtime/gpu/gpu\_device.cc:1264] 0

2023-09-06 02:00:06.183391: I tensorflow/core/common\_runtime/gpu/gpu\_device.cc:1277] 0: N

2023-09-06 02:00:06.183530: I tensorflow/stream\_executor/cuda/cuda\_gpu\_executor.cc:937] successful NUMA node read from S 2023-09-06 02:00:06.184098: I tensorflow/stream\_executor/cuda/cuda\_gpu\_executor.cc:937] successful NUMA node read from S 2023-09-06 02:00:06.184544: I tensorflow/stream\_executor/cuda/cuda\_gpu\_executor.cc:937] successful NUMA node read from S 2023-09-06 02:00:06.184980: I tensorflow/core/common\_runtime/gpu/gpu\_device.cc:1418] Created TensorFlow device (/job:loc Model: "sequential"

Layer (type) Output Shape Param #

=================================================================

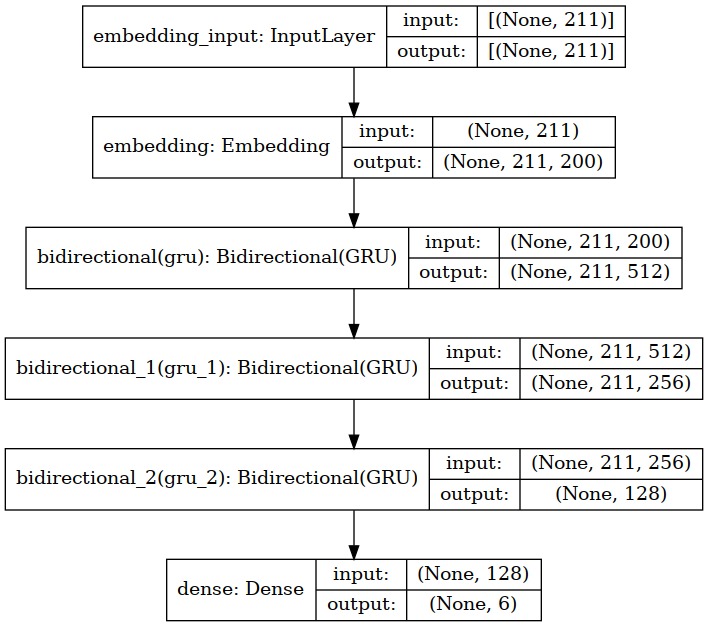
|  |  |  |
| --- | --- | --- |
| embedding (Embedding) | (None, 211, 200) | 4053600 |
| bidirectional (Bidirectional | (None, 211, 512) | 703488 |
| bidirectional\_1 (Bidirection | (None, 211, 256) | 493056 |
| bidirectional\_2 (Bidirection | (None, 128) | 123648 |
| dense (Dense) | (None, 6) | 774 |

=================================================================

Total params: 5,374,566

Trainable params: 1,320,966

Non-trainable params: 4,053,600



# Assuming you have a Keras model named 'model\_GRU\_2'

tf.keras.utils.plot\_model(model\_GRU\_2, to\_file='./model.png', show\_shapes=True,show\_layer\_names=True, expand\_nested=True)

# Display the saved image in your Jupyter Notebook or Python environment Image('model.png')

history = model\_GRU\_2.fit(

X\_train, y\_train,

batch\_size=64, epochs=21,

validation\_data=(X\_vald, y\_val),

verbose=1 # Set verbose to 0 to avoid printing training logs

)

Show hidden output

# Assuming you have already trained your 'model\_GRU\_2' and have test data 'X\_test' and 'y\_test' y\_pred = model\_GRU\_2.predict(X\_test)

# Visualize Training and Validation Loss plt.figure(figsize=(10, 6)) plt.plot(history.history['loss'], label='Training Loss')

plt.plot(history.history['val\_loss'], label='Validation Loss') plt.xlabel('Epochs')

plt.ylabel('Loss') plt.legend()

plt.title('Training and Validation Loss') plt.show()

# Calculate confusion matrix y\_true = np.argmax(y\_test, axis=1)

y\_pred\_classes = np.argmax(y\_pred, axis=1)

confusion = confusion\_matrix(y\_true, y\_pred\_classes)

# Plot confusion matrix plt.figure(figsize=(8, 6))

plt.imshow(confusion, interpolation='nearest', cmap=plt.cm.Blues) plt.title('Confusion Matrix')

plt.colorbar()

classes = ['joy', 'fear', 'neutral', 'love', 'sadness', 'other'] # Replace with your class labels tick\_marks = np.arange(len(classes))

plt.xticks(tick\_marks, classes, rotation=45) plt.yticks(tick\_marks, classes) plt.xlabel('Predicted Labels') plt.ylabel('True Labels')

# Display confusion matrix values on the plot for i in range(len(classes)):

for j in range(len(classes)):

plt.text(j, i, str(confusion[i][j]), horizontalalignment='center', color='white' if confusion[i][j] > confusion.max() plt.show()

# Classification Report

print("Classification Report:\n", classification\_report(y\_true, y\_pred\_classes))

# Compute ROC curve and AUC for each class fpr = {}

tpr = {} roc\_auc = {}

for i in range(len(classes)):

fpr[i], tpr[i], \_ = roc\_curve(y\_true == i, y\_pred[:, i]) roc\_auc[i] = auc(fpr[i], tpr[i])

# Plot ROC curve plt.figure(figsize=(8, 6)) for i in range(len(classes)):

plt.plot(fpr[i], tpr[i], label=f'Class {i} (AUC = {roc\_auc[i]:.2f})')

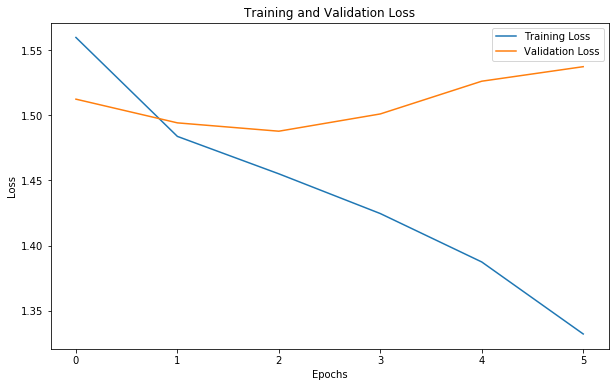
plt.plot([0, 1], [0, 1], 'k--')

plt.xlim([0.0, 1.0])

plt.ylim([0.0, 1.05]) plt.xlabel('False Positive Rate') plt.ylabel('True Positive Rate')

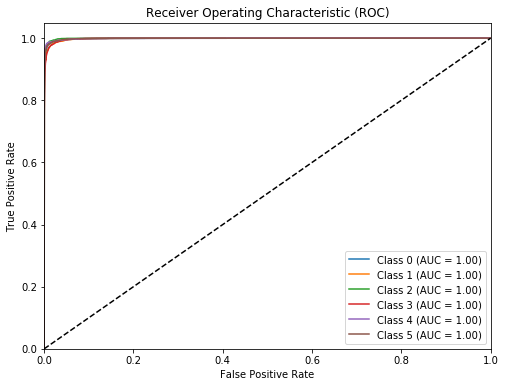
plt.title('Receiver Operating Characteristic (ROC)') plt.legend(loc="lower right")

plt.show()



Classification Report:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | precision | recall | f1-score | support |
| 0 | 0.98 | 0.97 | 0.97 | 6804 |
| 1 | 0.95 | 0.98 | 0.96 | 7371 |
| 2 | 0.99 | 0.91 | 0.95 | 3042 |
| 3 | 0.96 | 0.96 | 0.96 | 6852 |
| 4 | 0.96 | 0.98 | 0.97 | 2869 |
| 5 | 0.97 | 0.97 | 0.97 | 4841 |
| accuracy |  |  | 0.97 | 31779 |
| macro avg | 0.97 | 0.96 | 0.96 | 31779 |
| weighted avg | 0.97 | 0.97 | 0.97 | 31779 |



# Load the pre-trained GRU model (replace './GRU\_sentiment\_analysis.h5' with your model path) loaded\_model = load\_model('./GRU\_sentiment\_analysis.h5')

# Define your input text

input\_text = "i am sad i am not on schedule for thesis"

# Tokenize words (use the same tokenizer you used for training) tokenizer = Tokenizer(oov\_token='UNK') tokenizer.fit\_on\_texts(pd.concat([X\_trn, X\_tst], axis=0))

# Convert input text to sequences

input\_sequence = tokenizer.texts\_to\_sequences([input\_text])

model\_GRU\_2.save('./GRU\_2\_sentiment\_analysis.h5')

# Pad or truncate the sequence to match the max sequence length maxlen = 211 # Use the same maxlen as used during training

input\_sequence = pad\_sequences(input\_sequence, maxlen=maxlen, truncating='pre')

# Make predictions using the loaded model predictions = loaded\_model.predict(input\_sequence)

# Assuming you have 6 classes (adjust as needed) predicted\_class = np.argmax(predictions, axis=-1)

# Print the predicted class print(f"Predicted Class: {predicted\_class}")

# it predicts the class as other whereas the sentiment is of saddness

Predicted Class: [5]

**Training the DistilBERT model on the Training set**

train\_df = pd.read\_csv('./fin\_train.csv') val\_df = pd.read\_csv('./fin\_val.csv') test\_df = pd.read\_csv('./fin\_test.csv')

## Text Preprocessing

#Splitting the text from the labels X\_trn = train\_df['content']

y\_trn = train\_df['sentiment']

X\_tst = test\_df['content'] Y\_tst = test\_df['sentiment']

X\_val = val\_df['content'] y\_val = val\_df['sentiment']

y\_trn.value\_counts()

joy 7371

neutral 6852

fear 6804

sadness 4841

love 3042

other 2869

Name: sentiment, dtype: int64

# Encode labels

le = LabelEncoder()

y\_trn = le.fit\_transform(y\_trn) y\_tst = le.transform(Y\_tst) y\_val = le.transform(y\_val) #print the labels after encoding

original\_labels = le.inverse\_transform(y\_trn) print(set(y\_trn)) print(set(original\_labels))

{0, 1, 2, 3, 4, 5}

{'love', 'sadness', 'other', 'fear', 'joy', 'neutral'}

#Convert the class vector (integers) to binary class matrix y\_train = to\_categorical(y\_trn)

y\_test = to\_categorical(y\_tst) y\_val = to\_categorical(y\_val) print(y\_train)

[[0. 0. 0. 0. 0. 1.]

[1. 0. 0. 0. 0. 0.]

[0. 0. 0. 1. 0. 0.]

...

[0. 0. 1. 0. 0. 0.]

[0. 0. 0. 1. 0. 0.]

[0. 1. 0. 0. 0. 0.]]

# Load the DistilBERT tokenizer

tokenizer = DistilBertTokenizer.from\_pretrained('distilbert-base-uncased')

# Tokenize and prepare inputs

X\_trn\_tokens = tokenizer(X\_trn.tolist(), padding=True, truncation=True, return\_tensors='np') X\_tst\_tokens = tokenizer(X\_tst.tolist(), padding=True, truncation=True, return\_tensors='np') X\_val\_tokens = tokenizer(X\_val.tolist(), padding=True, truncation=True, return\_tensors='np')

# Extract input IDs, attention masks, and segment IDs X\_trn\_input\_ids = X\_trn\_tokens['input\_ids'] X\_trn\_attention\_mask = X\_trn\_tokens['attention\_mask']

X\_tst\_input\_ids = X\_tst\_tokens['input\_ids'] X\_tst\_attention\_mask = X\_tst\_tokens['attention\_mask']

X\_val\_input\_ids = X\_val\_tokens['input\_ids'] X\_val\_attention\_mask = X\_val\_tokens['attention\_mask']

# Find the maximum sequence length among all datasets

# Find the minimum sequence length among all datasets min\_seq\_length = min(

len(X\_trn\_input\_ids[0]), len(X\_val\_input\_ids[0]), len(X\_tst\_input\_ids[0])

)

# Truncate sequences to match the minimum sequence length

X\_trn\_input\_ids = pad\_sequences(X\_trn\_input\_ids, maxlen=min\_seq\_length, padding='post', truncating='post') X\_trn\_attention\_mask = pad\_sequences(X\_trn\_attention\_mask, maxlen=min\_seq\_length, padding='post', truncating='post') X\_val\_input\_ids = pad\_sequences(X\_val\_input\_ids, maxlen=min\_seq\_length, padding='post', truncating='post') X\_val\_attention\_mask = pad\_sequences(X\_val\_attention\_mask, maxlen=min\_seq\_length, padding='post', truncating='post') X\_tst\_input\_ids = pad\_sequences(X\_tst\_input\_ids, maxlen=min\_seq\_length, padding='post', truncating='post') X\_tst\_attention\_mask = pad\_sequences(X\_tst\_attention\_mask, maxlen=min\_seq\_length, padding='post', truncating='post') # Now you have your inputs prepared with input IDs, attention masks, and segment IDs

print("Training data shapes:") print("Input IDs:", X\_trn\_input\_ids.shape)

print("Attention Masks:", X\_trn\_attention\_mask.shape) print("Val data shapes:")

print("Input IDs:", X\_val\_input\_ids.shape) print("Attention Masks:", X\_val\_attention\_mask.shape) print("Test data shapes:")

print("Input IDs:", X\_tst\_input\_ids.shape) print("Attention Masks:", X\_tst\_attention\_mask.shape)

# Define the DistilBERT model

distil\_bert = TFDistilBertForSequenceClassification.from\_pretrained('distilbert-base-uncased')

# Define inputs

input\_ids = Input(shape=(min\_seq\_length,), dtype='int32', name='input\_ids') attention\_mask = Input(shape=(min\_seq\_length,), dtype='int32', name='attention\_mask')

# Connect inputs to the DistilBERT model

sequence\_output = distil\_bert(input\_ids, attention\_mask=attention\_mask)['logits'] #batch = BatchNormalization()(sequence\_output)

output\_layer = Dense(6, activation='softmax', kernel\_regularizer=l2(0.01))(sequence\_output) #(batch)(sequence\_output) # Assumi model = Model(inputs=[input\_ids, attention\_mask], outputs=output\_layer)

# Compile the model

optimizer = Adam(learning\_rate=2e-5, epsilon=1e-8) loss\_fn = CategoricalCrossentropy()

model.compile(optimizer=optimizer, loss=loss\_fn, metrics=[CategoricalAccuracy()])

# Plot the model architecture plot\_model(

model,

to\_file='distilbert\_model.png', # Output file name show\_shapes=True, # Show shapes of layers in the plot show\_layer\_names=True, # Show layer names in the plot rankdir='TB', # Layout direction: 'TB' for top to bottom expand\_nested=True # Expand nested models

)

Image('distilbert\_model.png')

Training data shapes:

Input IDs: (31779, 58)

Attention Masks: (31779, 58) Val data shapes:

Input IDs: (3973, 58)

Attention Masks: (3973, 58) Test data shapes:

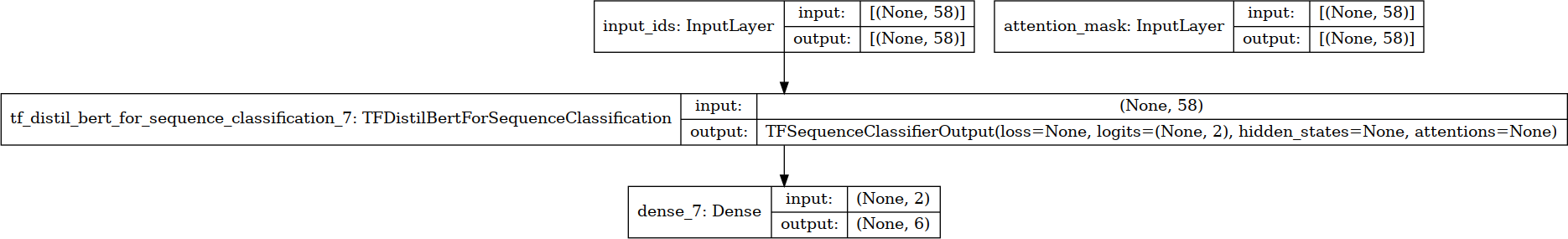
Input IDs: (3978, 58)

Attention Masks: (3978, 58)

Some weights of the PyTorch model were not used when initializing the TF 2.0 model TFDistilBertForSequenceClassification

This IS expected if you are initializing TFDistilBertForSequenceClassification from a PyTorch model trained on another

This IS NOT expected if you are initializing TFDistilBertForSequenceClassification from a PyTorch model that you expec Some weights or buffers of the TF 2.0 model TFDistilBertForSequenceClassification were not initialized from the PyTorch You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.



125/125 [==============================] - 2s 17ms/step - loss: 3.4111 - categorical\_accuracy: 0.3409

Test Loss: 3.4111

Test Accuracy: 0.3409

epochs=num\_epochs,

validation\_data=([X\_val\_input\_ids, X\_val\_attention\_mask], y\_val), verbose=1

)

# Evaluate the model

test\_loss, test\_accuracy = model.evaluate([X\_tst\_input\_ids , X\_tst\_attention\_mask], y\_test, verbose=1) print(f"Test Loss: {test\_loss:.4f}")

print(f"Test Accuracy: {test\_accuracy:.4f}")

# Define batch size and number of epochs batch\_size = 32 # Adjust as needed num\_epochs = 10 # Adjust as needed

# Training the model history = model.fit(

[X\_trn\_input\_ids, X\_trn\_attention\_mask],

y\_train, batch\_size=batch\_size,

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Epoch 1/10  994/994 [==============================] | - 54s | 50ms/step | - loss: | 1.7264 | - categorical\_accuracy: | 0.2707 | - val\_loss: | 1.703 |
| Epoch 2/10  994/994 [==============================] | - 49s | 50ms/step | - loss: | 1.6639 | - categorical\_accuracy: | 0.3367 | - val\_loss: | 1.719 |
| Epoch 3/10 |  |  |  |  |  |  |  |  |
| 994/994 [==============================]  Epoch 4/10 | - 49s | 50ms/step | - loss: | 1.5686 | - categorical\_accuracy: | 0.3883 | - val\_loss: | 1.735 |
| 994/994 [==============================] | - 49s | 50ms/step | - loss: | 1.4196 | - categorical\_accuracy: | 0.4530 | - val\_loss: | 1.881 |
| Epoch 5/10  994/994 [==============================] | - 50s | 50ms/step | - loss: | 1.2511 | - categorical\_accuracy: | 0.5170 | - val\_loss: | 2.044 |
| Epoch 6/10  994/994 [==============================] | - 50s | 50ms/step | - loss: | 1.0997 | - categorical\_accuracy: | 0.5624 | - val\_loss: | 2.237 |
| Epoch 7/10 |  |  |  |  |  |  |  |  |
| 994/994 [==============================] | - 49s | 50ms/step | - loss: | 0.9713 | - categorical\_accuracy: | 0.6023 | - val\_loss: | 2.534 |
| Epoch 8/10  994/994 [==============================] | - 50s | 50ms/step | - loss: | 0.8524 | - categorical\_accuracy: | 0.6642 | - val\_loss: | 2.650 |
| Epoch 9/10  994/994 [==============================] | - 49s | 50ms/step | - loss: | 0.7581 | - categorical\_accuracy: | 0.7233 | - val\_loss: | 2.979 |
| Epoch 10/10 |  |  |  |  |  |  |  |  |
| 994/994 [==============================] | - 49s | 50ms/step | - loss: | 0.6540 | - categorical\_accuracy: | 0.7710 | - val\_loss: | 3.410 |

# Define a function to build the model def build\_model(hp):

# Define hyperparameters to search

learning\_rate = hp.Choice('learning\_rate', [1e-6, 1e-5, 1e-4, 1e-3]) dropout\_rate = hp.Float('dropout\_rate', min\_value=0.0, max\_value=0.5, step=0.1) l2\_lambda = hp.Float('l2\_lambda', min\_value=1e-6, max\_value=1e-2, sampling='log')

# Build the model

input\_layer = tf.keras.layers.Input(shape=(min\_seq\_length,))

sequence\_output = distil\_bert(input\_ids=input\_ids, attention\_mask=attention\_mask)['logits'] output\_layer = Dense(6, activation='softmax', kernel\_regularizer=l2(l2\_lambda))(sequence\_output) model = Model(inputs=[input\_ids, attention\_mask], outputs=output\_layer)

# Compile the model

optimizer = Adam(learning\_rate=learning\_rate, epsilon=1e-8) loss\_fn = CategoricalCrossentropy()

metrics = [CategoricalAccuracy()] model.compile(optimizer=optimizer, loss=loss\_fn, metrics=metrics) return model

# Create a tuner tuner = RandomSearch(

build\_model, objective='val\_loss',

max\_trials=10, # Adjust the number of trials as needed directory='keras\_tuner\_logs', project\_name='my\_model\_tuning'

)

# Search for the best hyperparameters tuner.search(

[X\_trn\_input\_ids, X\_trn\_attention\_mask], y\_train,

validation\_data=([X\_val\_input\_ids, X\_val\_attention\_mask], y\_val), epochs=num\_epochs,

batch\_size=batch\_size, callbacks=[EarlyStopping(monitor='val\_loss', patience=5)]

)

# Get the best hyperparameters

best\_hps = tuner.get\_best\_hyperparameters(num\_trials=1)[0] best\_learning\_rate = best\_hps.get('learning\_rate') best\_dropout\_rate = best\_hps.get('dropout\_rate') best\_l2\_lambda = best\_hps.get('l2\_lambda')

Trial 10 Complete [00h 08m 41s] val\_loss: 2.107294797897339

Best val\_loss So Far: 0.11478344351053238 Total elapsed time: 08h 04m 52s INFO:tensorflow:Oracle triggered exit

NameError Traceback (most recent call last)

/tmp/ipykernel\_29163/3427923895.py in <module> 61

62 # Build the final model with the best hyperparameters

---> 63 input\_layer = tf.keras.layers.Input(shape=(input\_shape,))

sequence\_output = distil\_bert(input\_ids=input\_ids, attention\_mask=attention\_mask)['logits']

batch = BatchNormalization()(sequence\_output) NameError: name 'input\_shape' is not defined

SEARCH STACK OVERFLOW

|  |  |
| --- | --- |
| # Build the model  input\_layer = tf.keras.layers.Input(shape=(103,))  sequence\_output = distil\_bert(input\_ids=input\_ids, attention\_mask=attention\_mask)['logits'] batch = BatchNormalization()(sequence\_output)  dropout = Dropout(dropout\_rate)(batch)  output\_layer = Dense(6, activation='softmax', kernel\_regularizer=l2(l2\_lambda))(dropout) model = Model(inputs=[input\_ids, attention\_mask], outputs=output\_layer) | |
| # Build the final model with the best hyperparameters input\_layer = tf.keras.layers.Input(shape=(103,))  sequence\_output = distil\_bert(input\_ids=input\_ids, attention\_mask=attention\_mask)['logits'] batch = BatchNormalization()(sequence\_output)  dropout = Dropout(best\_dropout\_rate)(batch)  output\_layer = Dense(6, activation='softmax', kernel\_regularizer=l2(best\_l2\_lambda))(dropout) final\_model = Model(inputs=[input\_ids, attention\_mask], outputs=output\_layer)  # Compile the final model  final\_optimizer = Adam(learning\_rate=best\_learning\_rate, epsilon=1e-8) final\_loss\_fn = CategoricalCrossentropy()  final\_metrics = [CategoricalAccuracy()]  final\_model.compile(optimizer=final\_optimizer, loss=final\_loss\_fn, metrics=final\_metrics) # Train the final model  history = final\_model.fit(  [X\_trn\_input\_ids, X\_trn\_attention\_mask], y\_train,  validation\_data=([X\_val\_input\_ids, X\_val\_attention\_mask], y\_val), epochs=num\_epochs,  batch\_size=batch\_size  ) | |
|  |  |

final\_model.save('./distilbert\_sentiment\_analysis.h5')



|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Epoch 24/50  497/497 [==============================] | - 87s | 175ms/step | - loss: | 1.7009 | - categorical\_accuracy: | 0.2789 | - val\_loss: | 1 |
| Epoch 25/50  497/497 [==============================] | - 87s | 175ms/step | - loss: | 1.6924 | - categorical\_accuracy: | 0.2875 | - val\_loss: | 1 |
| Epoch 26/50 |  |  |  |  |  |  |  |  |
| 497/497 [==============================] | - 87s | 174ms/step | - loss: | 1.6841 | - categorical\_accuracy: | 0.2926 | - val\_loss: | 1 |
| Epoch 27/50  497/497 [==============================] | - 87s | 175ms/step | - loss: | 1.6730 | - categorical\_accuracy: | 0.2998 | - val\_loss: | 1 |
| Epoch 28/50  497/497 [==============================] | - 87s | 174ms/step | - loss: | 1.6641 | - categorical\_accuracy: | 0.3022 | - val\_loss: | 1 |
| Epoch 29/50 |  |  |  |  |  |  |  |  |
| 497/497 [==============================] | - 87s | 174ms/step | - loss: | 1.6570 | - categorical\_accuracy: | 0.3060 | - val\_loss: | 1 |
| Epoch 30/50  497/497 [==============================] | - 87s | 175ms/step | - loss: | 1.6453 | - categorical\_accuracy: | 0.3125 | - val\_loss: | 1 |
| Epoch 31/50  497/497 [==============================] | - 87s | 174ms/step | - loss: | 1.6297 | - categorical\_accuracy: | 0.3189 | - val\_loss: | 1 |
| Epoch 32/50  497/497 [==============================] | - 87s | 174ms/step | - loss: | 1.6126 | - categorical\_accuracy: | 0.3282 | - val\_loss: | 1 |
| Epoch 33/50 |  |  |  |  |  |  |  |  |
| 497/497 [==============================] | - 87s | 174ms/step | - loss: | 1.5970 | - categorical\_accuracy: | 0.3355 | - val\_loss: | 1 |
| Epoch 34/50  497/497 [==============================] | - 87s | 174ms/step | - loss: | 1.5800 | - categorical\_accuracy: | 0.3413 | - val\_loss: | 1 |
| Epoch 35/50  497/497 [==============================] | - 87s | 174ms/step | - loss: | 1.5620 | - categorical\_accuracy: | 0.3455 | - val\_loss: | 1 |
| Epoch 36/50 |  |  |  |  |  |  |  |  |
| 497/497 [==============================]  Epoch 37/50 | - 87s | 174ms/step | - loss: | 1.5478 | - categorical\_accuracy: | 0.3540 | - val\_loss: | 1 |
| 497/497 [==============================] | - 87s | 175ms/step | - loss: | 1.5318 | - categorical\_accuracy: | 0.3620 | - val\_loss: | 1 |
| Epoch 38/50  497/497 [==============================] | - 87s | 174ms/step | - loss: | 1.5168 | - categorical\_accuracy: | 0.3715 | - val\_loss: | 1 |
| Epoch 39/50  497/497 [==============================] | - 86s | 174ms/step | - loss: | 1.4997 | - categorical\_accuracy: | 0.3812 | - val\_loss: | 1 |
| Epoch 40/50 |  |  |  |  |  |  |  |  |
| 497/497 [==============================] | - 87s | 174ms/step | - loss: | 1.4827 | - categorical\_accuracy: | 0.3940 | - val\_loss: | 1 |
| Epoch 41/50  497/497 [==============================] | - 86s | 174ms/step | - loss: | 1.4708 | - categorical\_accuracy: | 0.3996 | - val\_loss: | 1 |
| Epoch 42/50  497/497 [==============================] | - 86s | 174ms/step | - loss: | 1.4504 | - categorical\_accuracy: | 0.4133 | - val\_loss: | 1 |
| Epoch 43/50 |  |  |  |  |  |  |  |  |
| 497/497 [==============================]  Epoch 44/50 | - 87s | 174ms/step | - loss: | 1.4278 | - categorical\_accuracy: | 0.4284 | - val\_loss: | 1 |
| 497/497 [==============================] | - 86s | 174ms/step | - loss: | 1.4155 | - categorical\_accuracy: | 0.4350 | - val\_loss: | 1 |
| Epoch 45/50  497/497 [==============================] | - 86s | 174ms/step | - loss: | 1.3972 | - categorical\_accuracy: | 0.4434 | - val\_loss: | 1 |
| Epoch 46/50  497/497 [==============================] | - 86s | 174ms/step | - loss: | 1.3796 | - categorical\_accuracy: | 0.4517 | - val\_loss: | 1 |
| Epoch 47/50 |  |  |  |  |  |  |  |  |
| 497/497 [==============================] | - 86s | 174ms/step | - loss: | 1.3668 | - categorical\_accuracy: | 0.4559 | - val\_loss: | 1 |
| Epoch 48/50  497/497 [==============================] | - 88s | 177ms/step | - loss: | 1.3522 | - categorical\_accuracy: | 0.4639 | - val\_loss: | 1 |
| Epoch 49/50  497/497 [==============================] | - 88s | 177ms/step | - loss: | 1.3393 | - categorical\_accuracy: | 0.4698 | - val\_loss: | 1 |
| Epoch 50/50  497/497 [==============================] | - 88s | 176ms/step | - loss: | 1.4256 | - categorical accuracy: | 0.4229 | - val loss: | 1 |
|  |  |  |  |  |  |  |  |  |

# Assuming you have already trained your 'final\_model' and have test data 'X\_tst\_input\_ids' and 'X\_tst\_attention\_mask' y\_pred = final\_model.predict([X\_tst\_input\_ids, X\_tst\_attention\_mask])

# Visualize Training and Validation Loss plt.figure(figsize=(10, 6)) plt.plot(history.history['loss'], label='Training Loss')

plt.plot(history.history['val\_loss'], label='Validation Loss') plt.xlabel('Epochs')

plt.ylabel('Loss') plt.legend()

plt.title('Training and Validation Loss') plt.show()

# Calculate confusion matrix y\_true = np.argmax(y\_test, axis=1)

y\_pred\_classes = np.argmax(y\_pred, axis=1)

confusion = confusion\_matrix(y\_true, y\_pred\_classes)

# Plot confusion matrix plt.figure(figsize=(8, 6))

plt.imshow(confusion, interpolation='nearest', cmap=plt.cm.Blues) plt.title('Confusion Matrix')

plt.colorbar()

classes = ['0', '1', '2', '3', '4', '5'] # Replace with your class labels tick\_marks = np.arange(len(classes))

plt.xticks(tick\_marks, classes, rotation=45) plt.yticks(tick\_marks, classes)