**FACULTY OF SCIENCE, ENGINEERING, AND COMPUTING**

**School of Science Engineering and Computing**

**MSc DEGREE**

IN

Data Science

**Title: Sentiment Analysis of reviews from twitter using Pre trained, Machine learning and Deep Learning Models**

Name: Tijo Varghese Vinu

ID Number: K2145423

1st Supervisor: Nabajeet Barman

2nd Supervisor: Farzana Rahman

**Kingston University** **London**

**WARRANTY STATEMENT**

This is a student project. Therefore, neither the student nor Kingston University makes any warranty, express or implied, as to the accuracy of the data or conclusion of the work performed in the project and will not be held responsible for any consequences arising out of any inaccuracies or omissions therein.

# **Abstract**

Twitter sentiment analysis refers to the process of using natural language processing and machine learning techniques to analyze and understand the sentiment, or emotional content, of tweets (short messages on the social media platform Twitter). This can be useful for businesses, organizations, and individuals who want to understand how people feel about a particular product, service, or topic, as it can help them gauge public opinion and make informed decisions.

One challenge of performing sentiment analysis on Twitter data is that tweets often contain slang, misspellings, and other types of informal language, which can make them difficult for algorithms to understand. Another challenge is that the context of a tweet may be important for understanding its sentiment, but this can be difficult to capture with automated approaches.

There are several approaches to performing sentiment analysis on tweets, including using pre-trained models or creating a custom model using a labeled dataset. Some common techniques include using lexicons or dictionaries of words with associated sentiment scores, using machine learning algorithms to classify tweets as positive, negative, or neutral based on their content, and using deep learning techniques to analyze the context and sentiment of the entire tweet.

In this project the tweet sample from nltk package is used to train and test Machine learning models like Random Forest, Naïve Bayes, Support vector Machines and also for deep learning LSTM model. Then data from twitter is pulled in real time and is used to predict the sentiment of those and compared it with pretrained models like TextBlob, VADER and roBERTa.

# **Acknowledgement**

Firstly, I would like to express my deepest gratitude to my supervisor, Prof. Nabajeet Barman, for their invaluable guidance, constant encouragement, and support throughout this project. Their expertise in the field of Machine learning, as well as their patience and constructive feedback, greatly contributed to the success of this dissertation.

I am also grateful to my friends and family for their unwavering support and encouragement throughout this journey.

I would also like to thank all the research participants who took the time to participate in this study, without their contributions, this dissertation would not have been possible.

Finally, I would like to acknowledge the support provided by Kingston University, for providing the resources necessary for completing this dissertation.

**Table of Contents**

[**Abstract** 2](#_Toc124276793)

[**Acknowledgement** 3](#_Toc124276794)

[**List of Figures** 6](#_Toc124276795)

[**List of Tables** 8](#_Toc124276796)

[**List of Abbrevations** 9](#_Toc124276797)

[Chapter 1: Introduction 10](#_Toc124276798)

[1.1 Sentiment Analysis 10](#_Toc124276799)

[1.1.1 Motivation 10](#_Toc124276800)

[1.1.2 Background 10](#_Toc124276801)

[1.2 Aims and Objectives: 12](#_Toc124276802)

[1.3 Summary of Thesis Contributions: 12](#_Toc124276803)

[1.4 Thesis Outline: 13](#_Toc124276804)

[1.5 Ethics Relevance & Progress 13](#_Toc124276805)

[**Chapter 2: Literature Review** 14](#_Toc124276806)

[2.1 Sentiment Analysis 14](#_Toc124276807)

[2.2 WordNet 14](#_Toc124276808)

[2.3 Approaches to performing sentiment analysis 15](#_Toc124276809)

[2.4 TextBlob 16](#_Toc124276810)

[2.5 VADER 16](#_Toc124276811)

[2.6 roBERTa 17](#_Toc124276812)

[2.7 Random Forest 17](#_Toc124276813)

[2.8 Naïve Bayes 17](#_Toc124276814)

[2.9 Support Vector machine 18](#_Toc124276815)

[2.10 TF-IDF 19](#_Toc124276816)

[2.11 Recurrent Neural Networks 20](#_Toc124276817)

[**Chapter 3: About the dataset** 24](#_Toc124276818)

[3.1 Getting dataset to train the models 24](#_Toc124276819)

[3.2 Getting dataset to predict the trained models. 25](#_Toc124276820)

[3.1.1 Tweepy 25](#_Toc124276821)

[**Chapter 4: Exploratory Data Analysis** 27](#_Toc124276822)

[**Chapter 5: Overview of Sentiment Analysis Algorithms** 31](#_Toc124276823)

[5.1 TextBlob 31](#_Toc124276824)

[5.2 Vader 32](#_Toc124276825)

[5.3 roBERTa 33](#_Toc124276826)

[5.4 Random Forest 34](#_Toc124276827)

[5.5 Multinomial Naïve Bayes 38](#_Toc124276828)

[5.6 Support vector Machine 39](#_Toc124276829)

[5.6 Bidirectional LSTM model 40](#_Toc124276830)

[**Chapter 6: Model Performance Evaluation Metrics** 44](#_Toc124276831)

[6.2 Confusion Matrix 44](#_Toc124276832)

[6.2 Accuracy 45](#_Toc124276833)

[6.3 Precision 45](#_Toc124276834)

[6.4 Recall 45](#_Toc124276835)

[6.5 F1 Score 46](#_Toc124276836)

[**Chapter 7 : Implementation** 47](#_Toc124276837)

[7.1 Preprocessing 49](#_Toc124276838)

[7.1.1 Convert to lower case: 49](#_Toc124276839)

[7.1.2 Removing numbers, hashtags, user mentions, and other noise: 50](#_Toc124276840)

[7.1.3 Removing special characters, retweets, spaces: 50](#_Toc124276841)

[7.1.4 Removing URLs: 50](#_Toc124276842)

[7.2 Data processing using Natural Language processing (NLP) 50](#_Toc124276843)

[7.2.1 Natural Language Toolkit (NLTK) 51](#_Toc124276844)

[7.3 Lexicon based approach 56](#_Toc124276845)

[7.3.1 Textblob and VADER Sentiment analysis 58](#_Toc124276846)

[7.3.2 roBERTa 61](#_Toc124276847)

[7.4 Machine Learning based approach 62](#_Toc124276848)

[7.4.1 Random Forest 64](#_Toc124276849)

[7.4.2 Multinomial Naïve Bayes 70](#_Toc124276850)

[7.4.3 Support Vector Machines 70](#_Toc124276851)

[7.5 Deep Learning approach 73](#_Toc124276852)

[7.5.1 Bi directional LSTM 73](#_Toc124276853)

[**Chapter 8: Evaluation** 80](#_Toc124276854)

[8.1 Classification Results 80](#_Toc124276855)

[8.1.1 Random Forest 80](#_Toc124276856)

[8.1.2 Multinomial Naïve Bayes 81](#_Toc124276857)

[8.1.3 Support Vector Machine 81](#_Toc124276858)

[8.1.4 Bi LSTM 82](#_Toc124276859)

[8.1.5 Accuracy of TextBlob and VADER 83](#_Toc124276860)

[8.2 Confusion Matrix 86](#_Toc124276861)

[8.2.1 Random Forest 86](#_Toc124276862)

[8.2.2 Multinomial Naïve Bayes 87](#_Toc124276863)

[8.2.2 Support Vector Machine 88](#_Toc124276864)

[8.2.2 Bidirectional LSTM 89](#_Toc124276865)

[8.4 Model Accuracy 90](#_Toc124276866)

[8.3 Sentiment analysis on downloaded tweets 91](#_Toc124276867)

[**Chapter 9. Conclusion** 92](#_Toc124276868)

[**Chapter 10. Future Works** 93](#_Toc124276869)

[**Chapter 11: References** 94](#_Toc124276870)

# **List of Figures**

[Figure 1: Sentiment Analysis 12](#_Toc124276871)

[Figure 2: Twitter Sentiment Analysis 12](#_Toc124276872)

[Figure 3:Different approaches to sentiment analysis 17](#_Toc124276873)

[Figure 4: Modeling pipeline of Machine learning model 19](#_Toc124276874)

[Figure 5: How to use Python's TF-IDF to handle textual data 21](#_Toc124276875)

[Figure 6: A Neural network cell of RNN 22](#_Toc124276876)

[Figure 7: An unrolled recurrent neural network 22](#_Toc124276877)

[Figure 8: Fig: The repeating module in an LSTM with four interacting layers 23](#_Toc124276878)

[Figure 9: Twitter Training data 25](#_Toc124276879)

[Figure 10: Smileys on which the data was collected 25](#_Toc124276880)

[Figure 11: Screenshot of Twitter API dashboard 26](#_Toc124276881)

[Figure 12: First five rows of Twitter data frame 27](#_Toc124276882)

[Figure 13: Senitment distribution in dataset 28](#_Toc124276883)

[Figure 14: Wordcloud of positive tweets file 29](#_Toc124276884)

[Figure 15: Wordcloud of negative tweets file 29](#_Toc124276885)

[Figure 16: Word cloud of real time tweets 30](#_Toc124276886)

[Figure 17: Frequency of most used words in the retrieved tweets 30](#_Toc124276887)

[Figure 18: Decision tree basic structure 35](#_Toc124276888)

[Figure 19: A table created from a zoo dataset in Kaggle to explain decision trees 36](#_Toc124276889)

[Figure 20:Decision tree example 37](#_Toc124276890)

[Figure 21: Bi LSTM working 43](#_Toc124276891)

[Figure 22: Overview of confusion matrix/contingency table. 46](#_Toc124276892)

[Figure 23: Different approaches of Sentiment Analysis 48](#_Toc124276893)

[Figure 24: Basic outline of the project 50](#_Toc124276894)

[Figure 25: Preprocessing tweets on twitter 51](#_Toc124276895)

[Figure 26: Basic Architecture of Sentiment Analysis using NLP 52](#_Toc124276896)

[Figure 27: Steps for Vectorization 53](#_Toc124276897)

[Figure 28: Tokenized tweets 54](#_Toc124276898)

[Figure 29: Lemmatization of tweets 55](#_Toc124276899)

[Figure 30: Stemming of tweets 55](#_Toc124276900)

[Figure 31: Comparison of Tokens, Stemming and Lemmatization 56](#_Toc124276901)

[Figure 32: Example of TextBlob text sentiment classification 58](#_Toc124276902)

[Figure 33: Example of VADER text sentiment classification 59](#_Toc124276903)

[Figure 34: Clean Tweets 60](#_Toc124276904)

[Figure 35: Screenshot of the tweets\_df dataframe after applying TextBlob and VADER sentiment analysis 61](#_Toc124276905)

[Figure 36: Pie chart of VADER vs TextBlob Sentiment Analysis 61](#_Toc124276906)

[Figure 37: tweet text input: 62](#_Toc124276907)

[Figure 38: Tweet preprocessed for roBERTa 62](#_Toc124276908)

[Figure 39: roBERTa model, tokenizer and labels 62](#_Toc124276909)

[Figure 40: Encoding and passing to model 63](#_Toc124276910)

[Figure 41: Sentiment analysed by roBERTa model 63](#_Toc124276911)

[Figure 42: Output of roBERTa model 63](#_Toc124276912)

[Figure 43: Machine learning approaches 64](#_Toc124276913)

[Figure 44: Alphabetical list of part-of-speech tags used in the Penn Treebank Project 66](#_Toc124276914)

[Figure 45: Tokens and their POS tags 67](#_Toc124276915)

[Figure 46: Lemmatised tokens 67](#_Toc124276916)

[Figure 47: Stopwords library 68](#_Toc124276917)

[Figure 48: Tokens after removing stop words 68](#_Toc124276918)

[Figure 49: Combined featureset array 69](#_Toc124276919)

[Figure 50: Features and labels array 69](#_Toc124276920)

[Figure 51: ML models functions 72](#_Toc124276921)

[Figure 52: Model function call and sentiment analysis 73](#_Toc124276922)

[Figure 53: Steps to Implement BiLSTM 74](#_Toc124276923)

[Figure 54: Extracting tweets from .json file 75](#_Toc124276924)

[Figure 55: Dataframe creation for Bi-LSTM model 75](#_Toc124276925)

[Figure 56: Cleaning Dataset for Bi-LSTM 76](#_Toc124276926)

[Figure 57: Function to remove repeating words 76](#_Toc124276927)

[Figure 58: Detokenize words 76](#_Toc124276928)

[Figure 59: Converting text array to 2D numeric arrays (padding) 77](#_Toc124276929)

[Figure 60: Embedding layer 78](#_Toc124276930)

[Figure 61: Single LSTM layer model with respective embedding layer 78](#_Toc124276931)

[Figure 62: Bidirectional layer LSTM model 79](#_Toc124276932)

[Figure 63: Classification result of Random Forest 81](#_Toc124276933)

[Figure 64: Classification result of Multinomial Naïve Bayes 82](#_Toc124276934)

[Figure 65: Classification result of SVM 82](#_Toc124276935)

[Figure 66: Classification result of Bi-LSTM 83](#_Toc124276936)

[Figure 67: Train and test accuracy for BiLSTM 84](#_Toc124276937)

[Figure 68: Highest Train accuracy 84](#_Toc124276938)

[Figure 69: Dataset created using labelled tweets 85](#_Toc124276939)

[Figure 70: RF, TextBlob and Vader sentiment 85](#_Toc124276940)

[Figure 71: Accuracy columns creation for the models 86](#_Toc124276941)

[Figure 72: Data frame with Accuracy columns 86](#_Toc124276942)

[Figure 73: Accuracy calculation of pretrained models 86](#_Toc124276943)

[Figure 74: Accuracy of TextBlob and VADER 86](#_Toc124276944)

[Figure 75: Confusion matrix of Random Forest 87](#_Toc124276945)

[Figure 76: Confusion matrix of Multinomial Naive Bayes 88](#_Toc124276946)

[Figure 77: Confusion matrix of Support Vector Machine 89](#_Toc124276947)

[Figure 78: Confusion matrix of BiLSTM 90](#_Toc124276948)

[Figure 79: Model Accuracy graph 91](#_Toc124276949)

# **List of Tables**

[Table 1: Basic example of Sentiment analysis 48](#_Toc124276950)

# **List of Abbrevations**

NLP - Neuro-Linguistic Programming

NLTK - Natural Language Toolkit

POS - Part Of speech

VADER - Valence Aware Dictionary and sEntiment Reasoner

GDPR - General Data Protection Regulation

TF-IDF - Term Frequency-Inverse Document Frequency

NN - Neural Network

RF - Random Forest

MNB - Multinomial Naive bayes

SVM - Support Vector Machine

RNN - Recurrent Neural network

LSTM - Long short-term memory

BiLSTM - Bidirectional Long Short Term Memory

EDA - Exploratory Data Analysis

# Chapter 1: Introduction

## Sentiment Analysis

### 1.1.1 Motivation

Sentiment analysis, which refers to the capacity of machines and software to perceive and evaluate human emotions, has numerous applications in industries such as e-commerce, marketing, advertising, politics, and market research, to name a few. By utilising text mining algorithms to extract subjective information (Salloum et al., 2018) from a variety of sources, businesses are able to gain a deeper understanding of the public's emotional response to their brand, product, or service. Additionally, these businesses can monitor online conversations.

### 1.1.2 Background

As a result of recent advancements in the field of deep learning, the capacity of algorithms to perform text analysis has improved significantly. Businesses can assess public opinion, conduct market research, monitor the reputation of their brand and products, and gain a deeper understanding of customer experiences by analysing consumer sentiment. Data analytics companies frequently integrate APIs from third-party sentiment analysis providers (*(PDF) SENTIMENT ANALYSIS: USES IN BUSINESS*, n.d.) into their own platforms for customer experience management, social media monitoring, and workforce analytics. This allows businesses to provide customers with valuable information.

Applying machine learning to the process of sentiment analysis has as its primary objective the enhancement and automation of low-level text analytics algorithms. These algorithms, which include Part of Speech tagging, are indispensable to the process of sentiment analysis. For example, data scientists can train a machine learning model to recognise nouns by providing it with a large number of pre-tagged text documents. This enables the model to discover how to classify nouns. Afterwards, the model is able to recognise nouns using supervised and unsupervised machine learning strategies, such as neural networks and deep learning, to recognise nouns.

Diagram

Description automatically generated

Figure 1: Sentiment Analysis

Because Twitter is one of the largest and most popular social media platforms that allows users to post tweets with their followers, I choose to use Twitter API in this instance. It is essential for businesses, organisations, and people to comprehend the sentiment, or emotional content, of tweets in order to evaluate public opinion and make educated judgements, given the daily number of tweets. Herein lies the value of sentiment analysis.Diagram

Description automatically generated

Figure 2: Twitter Sentiment Analysis

Sentiment analysis, also known as opinion mining, is the process of using natural language processing and machine learning techniques to analyze and understand the sentiment of texts, such as tweets. In recent years, there has been a growing interest in applying sentiment analysis to Twitter data, as it can provide valuable insights into the feelings and opinions of Twitter users.

## 1.2 Aims and Objectives:

The aim of this dissertation is to investigate the use of sentiment analysis for Twitter data, with a focus on developing and evaluating effective approaches for accurately classifying the sentiment of tweets. The specific objectives of the study are as follows:

1. To review the existing literature on sentiment analysis for Twitter data by pulling out the data in real time using tweepy which is a python library accessing the Twitter API.
2. To compare and evaluate different approaches to sentiment analysis for Twitter data, including rule-based, machine learning-based, and hybrid approaches.
3. To implement rule based pretrained models like textblob, VADER and roBERTa, to implement hybrid approaches using Machine learning models like Random Forest and Multinomial Naive Bayes and deep learning models like to sentiment analysis for Twitter data.
4. To evaluate the performance of the proposed approach and compare it to other approaches.

## 1.3 Summary of Thesis Contributions:

The main contribution of this thesis is the development and evaluation of sentiment analysis for Twitter data. The proposed approach is the deployment of pretrained, machine learning and deep learning model which will be evaluated on a large dataset of tweets which will be pulled from twitter real time using Twitter API. The results of the evaluation will show which approach outperforms other approaches in terms of classification accuracy.

## 1.4 Thesis Outline:

The remainder of this dissertation is organized as follows:

Chapter 2: Literature Review - This chapter provides an overview of the existing literature on sentiment analysis for Twitter data, including a discussion of different approaches and their strengths and limitations.

Chapter 3: About the dataset - This chapter describes the research design and methods used in the study, including the dataset, the approaches to sentiment analysis compared, and the evaluation metrics used.

Chapter 4: Exploratory Data Analysis - Analysing the datasets and visualising them.

Chapter 5: Overview of Sentiment Analysis Algorithms - Explanation about different sentiment analysis methods used in the project

Chapter 6: Model Performance Evaluation Metrics - Overview of the different evaluation metrics used foe model comparison

Chapter 7: Implementation - Implementation of the models

Chapter 8: Evaluation - This chapter presents the results of the evaluation of the different approaches to sentiment analysis and discusses the implications of the findings.

Chapter 9: Conclusion and Future Work - This chapter summarizes the main contributions of the thesis and discusses directions for future research.

## 1.5 Ethics Relevance & Progress

This work has been prepared in accordance with all relevant laws, regulations, and ethical guidelines set forth by Kingston University, including those related to copyright and privacy. Proper citations and references have been used for all sources consulted in the making of this project. The use of open-source software and technologies has been thoroughly considered. The awareness of plagiarism, including the potential consequences, has been gained through participation in plagiarism education. This project adheres to all ethical and legal standards. Additionally, all sources used for this project are freely available and comply with GDPR regulations as no personal data is being collected and hence the project is free from any technical and organisational requirements of GDPR.

# **Chapter 2: Literature Review**

## 2.1 Sentiment Analysis

Sentiment analysis, also known as opinion mining, is the process of using natural language processing and text analysis techniques to identify and extract subjective information from text data. (Pang & Lee, 2008) This task has gained a lot of attention in recent years due to the proliferation of online review systems, social media platforms, and customer feedback forums, which have generated a vast amount of data containing people's opinions, sentiments, and emotions. Machine learning-based approaches, on the other hand, can automatically learn the patterns and features that are indicative of specific sentiments from a large annotated dataset, and can generalize these patterns to classify the sentiment of new, unseen text. These approaches have achieved good results on various sentiment analysis tasks and have been widely used in practice (Kwok & Wang, 2013). However, they still face some challenges, such as the need for large amounts of annotated training data, the difficulty of handling subtle and complex sentiment expressions, and the sensitivity to the choice of model and hyperparameters.

There are several approaches to perform sentiment analysis, ranging from rule-based methods to machine learning-based methods (B. Liu, 2012). Rule-based methods rely on manually crafted dictionaries of words and phrases that are associated with specific sentiments and use these rules to classify the sentiment of a given piece of text (Asghar et al., 2017). While these methods can be effective for simple tasks and small datasets, they are prone to errors and do not scale well to more complex tasks and larger datasets. Phrase dependency parsing is a technique that has been used in the field of sentiment analysis for the purpose of opinion mining (Daniel & Martin, 2023). Dependency grammar is based on the relationship between a head word and its dependents, and phrase dependency parsing aims to balance the information gained from extracting long distance interactions with the information lost due to word-level dependency (Daniel & Martin, 2023). This technique has been found to be effective in extracting opinions from Chinese text, as it allows for the identification of the relationships between phrases and the incorporation of local structures and grammatical categories of phrases into the dependency tree. Overall, phrase dependency parsing has been shown to be a useful tool for sentiment analysis and opinion mining in the context of Chinese language text.

## 2.2 WordNet

WordNet is a large lexical database of English language words and their relationships (Oram, 2001). It has been used in the field of sentiment analysis as a tool for expanding seed sets of named adjectives to classify opinion words as positive or negative. By using WordNet to expand the seed set of named adjectives, researchers can more accurately classify opinion words and improve the performance of their sentiment analysis models (Haddi et al., 2013). In addition to its use in sentiment analysis, WordNet has also been applied in other areas such as natural language processing and information retrieval. Overall, WordNet has been found to be a useful resource for expanding seed sets of named adjectives and improving the performance of sentiment analysis models.

## 2.3 Approaches to performing sentiment analysis

There are several approaches to performing sentiment analysis on Twitter data, including:

1. Rule-based approaches: These use a set of predefined rules or dictionaries to classify texts as positive, negative, or neutral based on the presence of certain words or (Pang et al., n.d.). These approaches are relatively simple to implement and can be effective for identifying broad patterns of sentiment, but they may struggle to capture more subtle or nuanced emotions and may be prone to errors or biases.
2. Machine learning-based approaches: These involve training an algorithm on a set of labeled texts (i.e., texts that have been manually labeled as positive, negative, or neutral) and then using that learning to classify new texts (Nigam et al., 2018). These approaches can be more accurate than rule-based approaches, as they can learn to recognize subtle patterns in the language and consider the context of the text. However, they may require more resources and expertise to implement and may be more sensitive to errors in the training data.
3. Deep learning approach: Deep learning approaches have recently been applied to this task and have achieved state-of-the-art results. One early deep learning approach for sentiment analysis is the use of convolutional neural networks (CNNs) (Kim, 2014).

Another popular deep learning approach for sentiment analysis is the use of recurrent neural networks (RNNs), particularly long short-term memory (LSTM) networks(Dr. G. S. N. Murthy et al., 2020). These models are well-suited for sentiment analysis because they can handle long-term dependencies in the text and have the ability to remember important information for a long period of time. (Hochreiter & Schmidhuber, 1997) introduced the LSTM architecture, which has been widely used for various natural language processing tasks, including sentiment analysis.

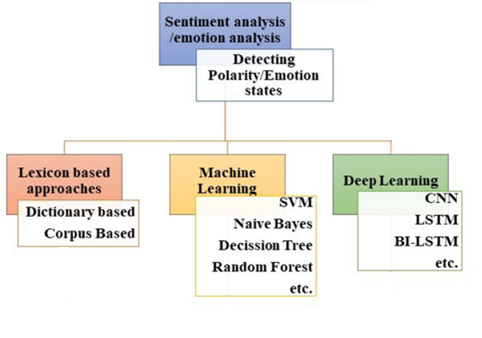


Figure 3:Different approaches to sentiment analysis

Subjectivity analysis and sentence-level sentiment analysis have many similarities. This stage involves analysing each statement to determine if it is neutral, positive, or both. Finding the opinion's target is the goal of the aspect level sentiment analysis. This strategy is based on the idea that every viewpoint has an objective and that an opinion without a target has little value.

## 2.4 TextBlob

Textblob is a Python library (*TextBlob: Simplified Text Processing — TextBlob 0.16.0 Documentation*, n.d.) that provides an easy-to-use interface for performing various natural language processing tasks, including sentiment analysis. Textblob uses a combination of pattern matching and machine learning to assign a sentiment score to text data (Hazarika et al., 2020). One advantage of Textblob is its simplicity and ease of use, but it may not always provide the most accurate sentiment analysis results compared to other methods (Kiritchenko et al., 2014).

## 2.5 VADER

Another widely used sentiment analysis tool is Vader (Valence Aware Dictionary and sEntiment Reasoner) (*SENTIMENTAL ANALYSIS USING VADER. Interpretation and Classification Of… | by Aditya Beri | Towards Data Science*, n.d.). Vader is a rule-based tool that uses a dictionary of words and their associated sentiment scores to assign a sentiment score to text data. One advantage of Vader is its ability to handle social media language, such as emoji and slang, that may not be well-represented in other dictionaries (Hutto & Gilbert, 2014). However, Vader may not perform as well on longer text data and may be less accurate than machine learning-based methods.

## 2.6 roBERTa

Another state-of-the-art method for sentiment analysis is the use of the roBERTa language model (Y. Liu et al., 2019). roBERTa is a transformer-based language model that has been pre-trained on a large dataset of text data and fine-tuned for various natural language processing tasks (Y. Liu et al., 2019)). One advantage of using roBERTa for sentiment analysis is its ability to capture contextual information in the data, leading to improved performance on many tasks. However, roBERTa models require a large amount of computational resources to train and may not be practical for all sentiment analysis tasks (Dai et al., 2021).

Lexicon-based approaches are relatively simple and fast, but may not always provide the most accurate results due to the limited coverage of the lexicon and the inability to capture context and nuances in the language (Kiritchenko et al., 2014). Deep learning approaches, on the other hand, can capture complex patterns in the data and provide improved performance on sentiment analysis tasks (Linder, 2017), but may require a larger amount of data and computational resources to train (Bengio et al., n.d.).

## 2.7 Random Forest

One popular method for sentiment analysis is the use of machine learning algorithms, such as random forests (Liaw & Wiener, 2002). Random forests are a type of ensemble learning method that operate by constructing a multitude of decision trees and combining their predictions (Breiman, 2001). Advantages of using random forests for sentiment analysis include their ability to handle high dimensional data and their robustness to overfitting (Zhang et al., 2019). However, one limitation of random forests is that they can be computationally expensive to train and run (Bengio et al., n.d.).

## 2.8 Naïve Bayes

Another widely used method for sentiment analysis is the multinomial naive Bayes (MNB) algorithm (Domingos & Pazzani, 1997). A study on sentiment analysis was published by(Shah et al., 2016). MNB is a probabilistic classifier that is based on the assumption that features are independent of one another given the class label (Lewis, n.d.). One advantage of MNB is its simplicity and efficiency, making it a popular choice for sentiment analysis tasks (Pang and Lee, 2008). However, MNB can perform poorly when the assumption of independence between features is violated (Nguyen et al., 2016)

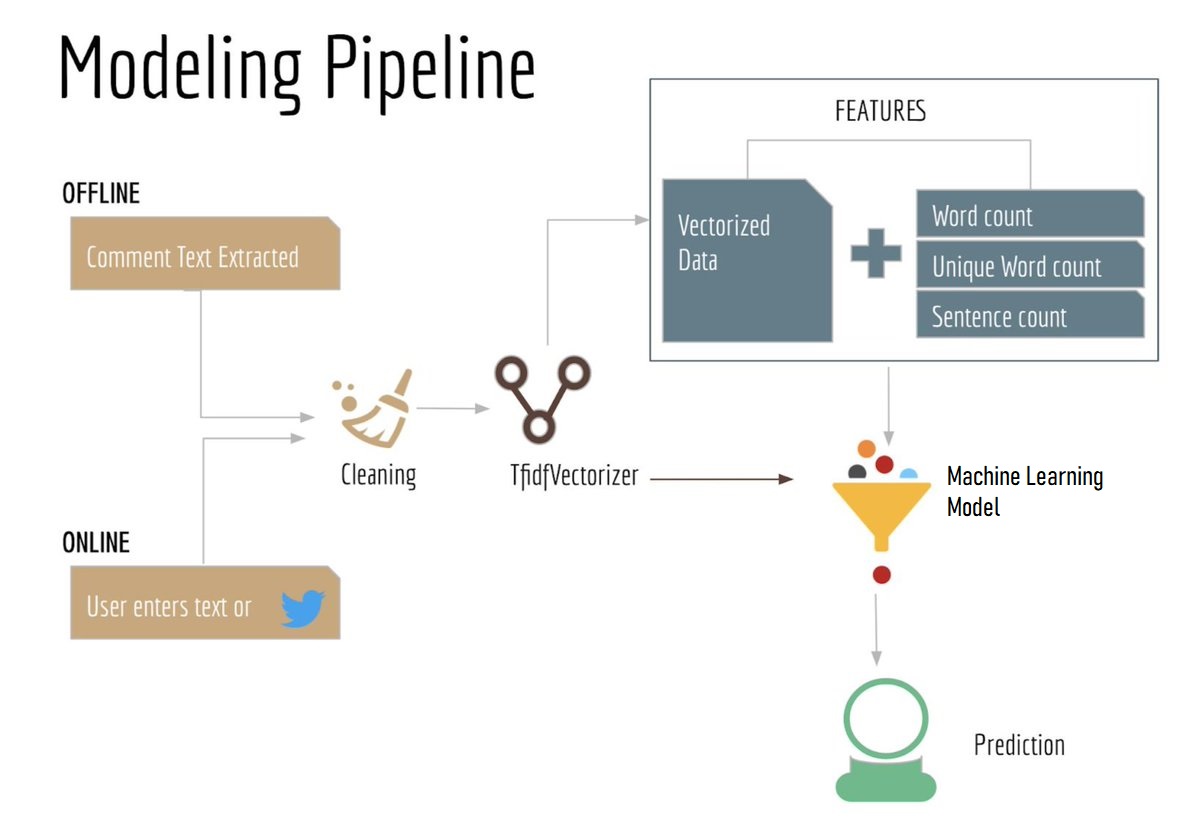


Figure 4: Modeling pipeline of Machine learning model

## 2.9 Support Vector machine

Support Vector Machines (SVMs) are a type of supervised machine learning algorithm that can be used for classification or regression tasks. In sentiment analysis, SVMs have been widely used to classify texts as having positive, negative, or neutral sentiment.(Evgeniou & Pontil, 2001)

One early study on using SVMs for sentiment analysis was published by Pang and Lee in 2002. They used a SVM to classify movie reviews as positive or negative, achieving an accuracy of around 87%.

Another study by (Kim, 2014) compared the performance of SVMs to other machine learning algorithms for sentiment analysis on Twitter data. The study found that SVMs performed better than other algorithms such as naive Bayes and logistic regression.

SVMs have also been used in combination with other techniques to improve the performance of sentiment analysis. For example, (Pang & Lee, 2008) combined SVMs with a technique called opinion mining to classify hotel reviews as positive, negative, or neutral.

## 2.10 TF-IDF

TF-IDF (term frequency-inverse document frequency) is a commonly used method for evaluating the importance of words in a document or a corpus (a collection of documents). It has been widely used in information retrieval, text classification, and other natural language processing tasks (Robertson, n.d.).

TF-IDF weights the importance of each word in a document based on how often it appears in the document, (*Understanding TF-IDF for Machine Learning | Capital One*, n.d.)as well as how often it appears in the corpus as a whole. Words that appear frequently in the document but not in the corpus are considered to be more important than words that appear frequently in both the document and the corpus.

TF-IDF is calculated by multiplying two factors: the term frequency (TF) and the inverse document frequency (IDF). The term frequency is the number of times a term appears in a document, normalized by the total number of terms in the document. The inverse document frequency is a measure of the rarity of a term in a collection of documents, with rarer terms being given higher weights. The formula for calculating the TF-IDF of a term ‘t’ in a document ‘d’ in a collection of documents ‘D’ is:

TF-IDF(t, d, D) = TF(t, d) \* IDF(t, D)

TF-IDF vectorization entails computing the TF-IDF score for each word in your corpus relative to each document and then storing this information in a vector (see image below using documents "A" and "B" as examples).(*How to Process Textual Data Using TF-IDF in Python*, n.d.) Consequently, each document in your corpus would have its own vector, with a TF-IDF score for each and every word in the complete collection of documents. *A* "The car is driven on the road," while B = "The truck is driven on the highway”.

Graphical user interface, application, table, Excel

Description automatically generated

Figure 5: How to use Python's TF-IDF to handle textual data

However, there are also some limitations to using TF-IDF. One issue is that it does not take into account the context in which the terms appear(*A Gentle Introduction To Calculating The TF-IDF Values | by Ann Sebastian | Towards Data Science*, n.d.), which can be important for tasks such as sentiment analysis. It also does not account for the grammatical structure of the text, which can be important for tasks such as language translation.

## 2.11 Recurrent Neural Networks

Recurrent neural networks (RNNs) are a type of neural network that are particularly well-suited for processing sequential data. They have been widely used in a variety of applications, including natural language processing (NLP), speech recognition, and machine translation.

One of the key features of RNNs is the use of a hidden state to store information about previous time steps. This allows the model to capture temporal dependencies in the data, which is essential for tasks such as language translation and speech recognition.

The first RNN model was introduced by Elman in 1990 (Hochreiter & Schmidhuber, 1997)Since then, numerous variants of RNNs have been developed, including long short-term memory (LSTM) networks and gated recurrent units (GRUs).

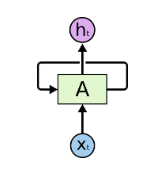


Figure 6: A Neural network cell of RNN

*Fig: In the preceding figure, a neural network cell receives an input Xt and produces a value Ht. The loop then supplies the information to the following cell, and so on. Therefore, the RNNs may be viewed as duplicates of the same network, transmitting information from one to the next.*

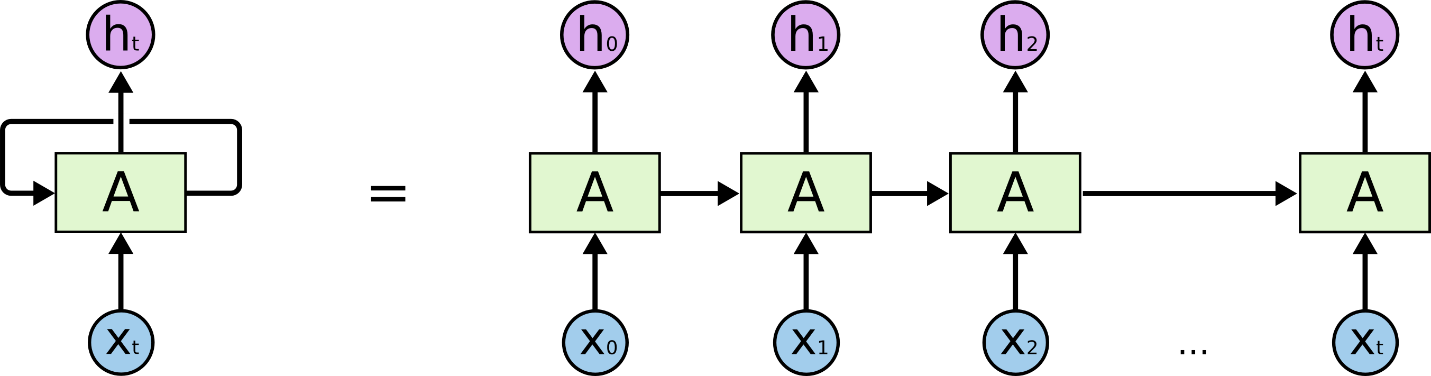


Figure 7: An unrolled recurrent neural network

LSTM networks, introduced by (Hochreiter & Schmidhuber, 1997), are a type of RNN that use gates to control the flow of information through the network. This allows them to selectively remember, forget, and update information in the hidden state, which helps them to avoid the vanishing and exploding gradient problems that can occur when training RNNs. LSTMs have been shown to be effective for tasks such as language translation and language modeling.

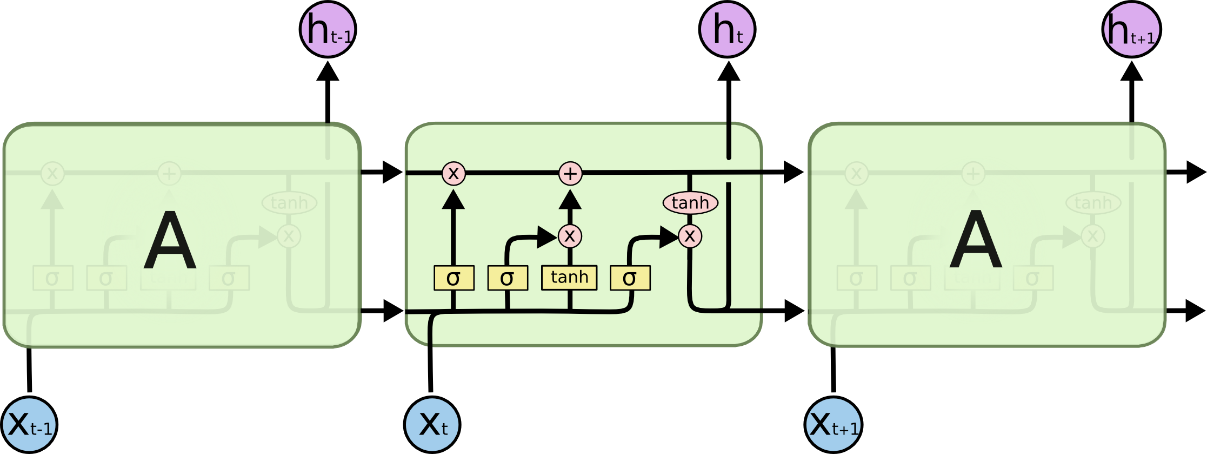


Figure 8: Fig: The repeating module in an LSTM with four interacting layers

.

Long Short-Term Memory (LSTM) networks are a type of recurrent neural network (RNN) that are able to capture long-term dependencies in data. They are especially useful for tasks that involve sequential data, such as language translation, speech recognition, and time series forecasting.

One of the key features of LSTMs is the ability to control the flow of information through the use of gates(Yeturu, 2020), which allow the model to selectively remember, forget, and update information in the hidden state. This helps the model to avoid the vanishing and exploding gradient problems that can occur when training RNNs, and allows it to handle long-term dependencies more effectively.

Bi-directional long short-term memory (bi-LSTM) (Schuster & Paliwal, 1997) is a type of recurrent neural network that is able to process sequential data in both forward and backward directions (Graves & Schmidhuber, n.d.). Bi-LSTM has been used for sentiment analysis tasks on Twitter (Lai et al., 2015), with some studies finding that it outperforms other methods (Jin et al., 2016). One advantage of using bi-LSTM for sentiment analysis is its ability to capture long-term dependencies in the data (Hochreiter & Schmidhuber, 1997). However, bi-LSTM models can be computationally expensive to train (Gers et al., 2000) and may not always be the best choice for sentiment analysis tasks.

In comparing the different methods for Twitter sentiment analysis, it is important to consider their performance and the trade-offs between accuracy, computational efficiency, and simplicity (Feder Cooper et al., 2021). While machine learning-based methods, such as random forests and MNB, may provide more accurate results (Soliman, 2020), they may be more computationally expensive and require more data to train (Wang and Wang, 2014). On the other hand, rule-based methods, such as Vader and Textblob, are simpler and faster to run (Hutto and Gilbert, 2014; Liao et al., 2013), but may not always provide the highest accuracy ((Kiritchenko et al., 2014)). Bi-LSTM and roBERTa may also provide improved performance on sentiment analysis tasks but may be even more computationally expensive and require more data for training(Li et al., 2016)

Overall, the choice of method for Twitter sentiment analysis will depend on the specific requirements of the task and the resources available (Burnap & Williams, 2015). (Sarker, 2021), while rule-based methods may be more appropriate for smaller scale tasks or when computational resources are limited .

In conclusion, this literature review has presented an overview of several methods for performing Twitter sentiment analysis, including random forests (Liaw & Wiener, 2002), multinomial naive Bayes (Domingos & Pazzani, 1997), Textblob ((Hazarika et al., 2020), Vader (Hutto and Gilbert, 2014), bi-LSTM (Schuster & Paliwal, 1997), and roBERTa (Liu et al., 2019). Each method has its own advantages and limitations and the best choice of method will depend on the specific requirements of the task and the resources available (Kwok & Wang, 2013). Further research is needed to compare and evaluate the performance of these methods on a variety of Twitter sentiment analysis tasks (Ranco et al., 2015)

In summary, there are various techniques that have been applied to Twitter sentiment analysis, including machine learning-based methods, rule-based methods, lexicon-based approaches, and deep learning approaches. Each technique has its own advantages and limitations, and the best choice of method will depend on the specific requirements of the task and the resources available. Further research is needed to compare and evaluate the performance of these methods on a variety of Twitter sentiment analysis tasks.

# **Chapter 3: About the dataset**

## 3.1 Getting dataset to train the models

To train the classic machine learning models and deep learning LSTM model we have used twitter sample from nltk package.

This Twitter data is in the ntlk corpora folder, under twitter\_samples. They are in .json format. There are two files which we use for training and they’re positive tweets file and negative tweets. Both the files contain 5000 tweets each. The way the data was collected was, the tweets were classified negative or positive based on smileys. So if the tweet contained ‘: )’ it was classified as positive. This will have ramifications for us later.

Graphical user interface, text, application

Description automatically generated

Figure 9: Twitter Training data

Scatter chart

Description automatically generated

Figure 10: Smileys on which the data was collected

## 3.2 Getting dataset to predict the trained models.

The data is pulled from twitter in real time using tweepy api after creating a developer account in twitter and setting up an api and have collected 9000 tweets for sentiment analysis. The tweets are stored into a dataframe with following headings 'Datetime', 'Tweet', 'Username', 'Retweets', 'Followers’. Also the tweets are pulled from the accounts with minimum of 500 followers for credibility.

### Tweepy

The tweepy Python package provides access to the Twitter API. It helps developers to construct apps that can fetch and send tweets, manage user accounts, and more. You may execute different activities using tweepy, such as:

* The process of searching for tweets using keywords or hashtags.
* Retrieve the most recent tweets from the timeline of a person.
* Posting tweets or retweets
* Follow and unfollow other people
* Administration of user lists
* Having access to user profiles and data

Tweepy is user-friendly and well-documented, making it a popular option among developers who wish to create Twitter-based apps. It's compatible with Python versions 2.7 and 3.



Figure 11: Screenshot of Twitter API dashboard

There are 4 parameters to authorize tweepy to access Twitter API which are consumer key, consumer secret, access token and access token secret which are obtained from the Twitter developer portal dashboard. A function is defined to create a data frame to import the tweets from twitter by passing the search word ’review’ and the limit is set as 9000 for tweets.

Following is the first five rows of the dataset created by pulling the tweets from twitter using tweepy:

Graphical user interface, text, application, email

Description automatically generated

Figure 12: First five rows of Twitter data frame

# **Chapter 4: Exploratory Data Analysis**

A dataset was created using the positive and negative twitter samples from the nltk corpura package. They were in .json format from which the text data or tweets which are in string format was extracted from the files, tokenized, cleaned and stored into two separate positive and negative feature set arrays mapping the respective sentiment label to the tweets. Both the positive and negative feature set arrays were combined together and they were stored in a data frame.

Chart, bar chart, treemap chart

Description automatically generated

Figure 13: Senitment distribution in dataset

In each category, 5,000 tweets will be collected. Then, the training list will have 7,500 tweets and the test list will contain 1,500.

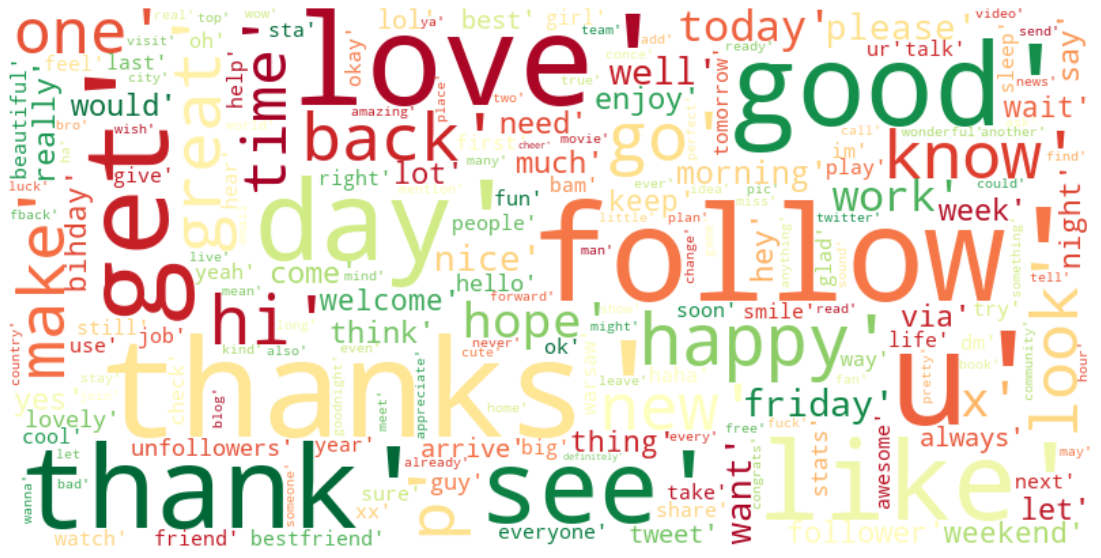


Figure 14: Wordcloud of positive tweets file



Figure 15: Wordcloud of negative tweets file

The word ‘review’ was used as our tweet search parameter to obtain the tweets which could be reviews of products or services. The following word cloud was formed from the most used words in the tweets without the stopwords.

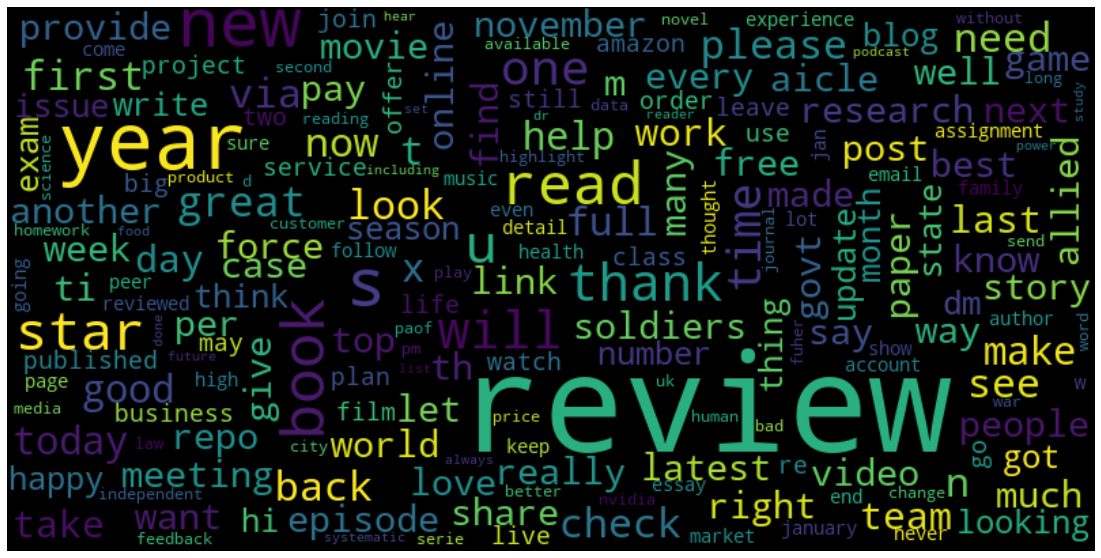


Figure 16: Word cloud of real time tweets

The below graph was formed from the retrieved tweets to find the frequency of most used words in those tweets.

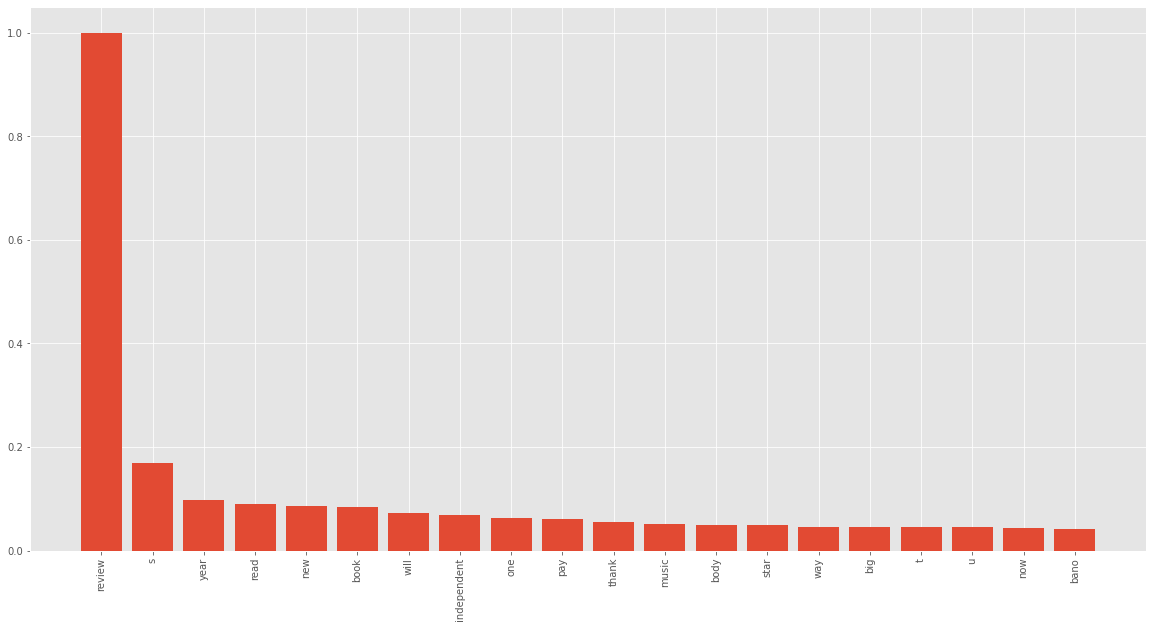


Figure 17: Frequency of most used words in the retrieved tweets

The data needs to be divided into a train and test component consisting of both positive and negative tweets since we shouldn't train and test on the same data set as this would cause some major bias concerns. Once the model has been trained, that portion should be tested. This process is known as supervised machine learning since the computer is being taught what is good and negative from the data.

# **Chapter 5: Overview of Sentiment Analysis Algorithms**

This chapter explains about the different pretrained, machine learning and deep learning algorithms used for Sentiment Analysis of tweets with ‘review’ keyword which are as follows:

1. TextBlob
2. Vader
3. roBERTa
4. Random Forest
5. Naïve Bayes
6. Support Vector Machine
7. LSTM

## 5.1 TextBlob

TextBlob is a popular Python library that provides simple, intuitive methods for working with text data. One of the features of TextBlob is that it includes a pre-trained sentiment analysis model, which can be used to classify text as positive, negative, or neutral. This model is useful for a variety of applications, such as analyzing customer reviews, social media posts, and other types of text data to understand the sentiment of the authors.

To use the pre-trained sentiment analysis model in TextBlob, you first need to install the library. Once you have TextBlob installed, you can use the TextBlob function to create a TextBlob object from a piece of text, and then call the sentiment method on that object to get a sentiment score. The sentiment score is a tuple of two values, a polarity score and a subjectivity score. The polarity score is a value between -1 and 1, with -1 being the most negative and 1 being the most positive. The subjectivity score is a value between 0 and 1, with 0 being very objective and 1 being very subjective.

It's important to note that the pre-trained sentiment analysis model in TextBlob is not perfect, and it may not always produce accurate results. This is because sentiment analysis is a complex task that involves understanding the context and nuance of language, which can be difficult for a machine to do. In some cases, you may want to fine-tune the model by training it on a dataset of your own, or you may want to use a different sentiment analysis tool altogether.

There are a few different approaches that can be used for sentiment analysis, and the pre-trained model in TextBlob uses a combination of techniques. One approach is to use a lexicon, which is a list of words and their associated sentiment scores. The model in TextBlob uses a lexicon of over 8,000 words, which have been manually annotated with sentiment scores. The model calculates the overall sentiment of a piece of text by summing the sentiment scores of the individual words in the text.

Another approach is to use machine learning algorithms to train a model to predict the sentiment of text. This can be done by feeding the model a large dataset of labeled text, where the labels indicate the sentiment of the text. The model can then learn to identify patterns in the data that are indicative of positive, negative, or neutral sentiment. The pre-trained model in TextBlob uses a combination of lexicon-based and machine learning-based techniques to classify text.

Overall, the pre-trained sentiment analysis model in TextBlob is a useful tool for quickly and easily classifying text as positive, negative, or neutral. While it is not perfect, it can be a good starting point for many applications, and can be fine-tuned or supplemented with other techniques as needed.

## 5.2 Vader

Vader (Valence Aware Dictionary and sEntiment Reasoner) is a popular pre-trained sentiment analysis model that is specifically designed to analyze the sentiment of social media text. It is implemented in Python and is available as a part of the NLTK (Natural Language Toolkit) library.

One of the key features of Vader is that it is specifically designed to handle the challenges of analyzing social media text, which can be very different from other types of text. For example, social media text often includes slang, abbreviations, and other types of informal language, which can be difficult for a traditional sentiment analysis model to understand. Vader is able to handle these challenges by using a lexicon of over 7,000 words and punctuation marks that have been manually annotated with sentiment scores. It also takes into account the context in which these words and punctuation marks are used, using advanced rules-based techniques to determine the overall sentiment of a piece of text.

To use Vader, you first need to install the NLTK library. Once you have NLTK installed, you can use the SentimentIntensityAnalyzer class to create a sentiment analyzer object. Then, you can call the polarity\_scores method on the analyzer object, passing in a piece of text as an argument. The polarity\_scores method will return a dictionary of scores, including a compound score that is a normalized, weighted composite of the other scores. The compound score is a value between -1 and 1, with -1 being the most negative, 0 being neutral, and 1 being the most positive.

In addition to the compound score, the polarity\_scores method also returns scores for positive, negative, and neutral sentiment. These scores can be used to get a more detailed understanding of the sentiment of a piece of text. For example, if a piece of text has a high positive score and a low negative score, it is likely to be overall positive in sentiment, even if the compound score is not particularly high.

Vader is widely used and has been shown to be effective at analyzing the sentiment of social media text. However, like any pre-trained model, it is not perfect, and it may not always produce accurate results. In some cases, you may want to fine-tune the model by training it on a dataset of your own, or you may want to use a different sentiment analysis tool altogether.

## 5.3 roBERTa

roBERTa (from "Robustly Optimized BERT Approach") is a pre-trained language model developed by Facebook AI that has been trained on a large dataset of unannotated text. It is based on the BERT (Bidirectional Encoder Representations from Transformers) model, which is a transformer-based architecture that has achieved state-of-the-art results on a variety of natural language processing tasks.

One of the key features of roBERTa is that it is trained using a much larger dataset and more computational resources than BERT, which makes it significantly more powerful and accurate. In particular, roBERTa is trained on a dataset of over 160GB of text, which is more than ten times larger than the dataset used to train BERT. It is also trained using a more efficient training method, which allows it to learn from the data more effectively.

To use roBERTa for sentiment analysis, you first need to install the transformers library, which provides an implementation of the model in Python. Once you have transformers installed, you can use the RobertaModel class to create a roBERTa model, and then use the predict method to generate predictions for a piece of text. The predict method returns a tuple of two values: a tensor of predicted labels and a tensor of predicted probabilities.

It's important to note that roBERTa was not specifically trained for sentiment analysis, and it may not always produce accurate results when used for this task. In order to use roBERTa for sentiment analysis, you will need to fine-tune the model on a dataset of labeled text, where the labels indicate the sentiment of the text. This can be done using supervised learning techniques, where you provide the model with a large dataset of labeled text and train it to predict the labels based on the text.

Overall, roBERTa is a powerful pre-trained language model that can be used for a wide variety of natural language processing tasks, including sentiment analysis. While it may not always produce accurate results out of the box, it can be fine-tuned and customized to fit the specific needs of your application.

## 5.4 Random Forest

Random Forest is a machine learning algorithm that can be used for a variety of tasks, including sentiment analysis. In sentiment analysis, a Random Forest model is trained to predict the sentiment of a piece of text based on the words and other features in the text.

To train a Random Forest model for sentiment analysis, you first need to gather a dataset of labeled text, where the labels indicate the sentiment of the text (e.g. positive, negative, neutral). You can then use this dataset to train the model to predict the labels based on the text.

A random forest is an ensemble learning method that trains a large number of decision trees and combines their predictions to make a more accurate and stable prediction. Random forests are less prone to overfitting than decision trees, because they train multiple decision trees on different subsets of the data and average their predictions.

Diagram

Description automatically generated

Figure 18: Decision tree basic structure

A decision tree is a tree-like flowchart structure that makes a decision based on the values of the data's characteristics.

Each of these events results in more nodes that branch off into other potential outcomes. This gives it the appearance of a tree. Their action is complete when a subset contains identical values for the target variable or when spitting no longer contributes to the accuracy of predictions. Hopefully, the bottom of the tree has a choice for target feature values. Decision Trees are among the most popular machine learning algorithms due to their understandability and straightforwardness.

Additionally, decision trees have downsides due to their propensity to overfit, which may be mitigated by adopting Random Forest.

Table

Description automatically generated with low confidence

Figure 19: A table created from a zoo dataset in Kaggle to explain decision trees

Table

Description automatically generated with low confidence

Diagram

Description automatically generated with low confidence

Diagram

Description automatically generated

Figure 20:Decision tree example

So multiple decision trees are combined to make a Random Forest which makes it less prone to overfitting which is a phenomenon that occurs when a machine learning model is trained too well on the training data and performs poorly on new, unseen data. This can happen when the model is too complex for the given training data, and it ends up learning patterns that are specific to the training data but do not generalize to new examples.

In other words, an overfitted model has learned the "noise" in the training data rather than the underlying relationship that is being modeled. As a result, the model performs well on the training data, but it does not generalize well to new, unseen examples.

There are a few different ways to represent the text data for use in a Random Forest model. One common approach is to use a bag-of-words representation, where each piece of text is represented as a vector of the counts of each word in the text. This can be done using a technique called term frequency-inverse document frequency (TF-IDF), which represents each word in a text as a weighted value that reflects the importance of the word in the text. The weight of a word is calculated as:

TF-IDF = TF \* IDF

where TF (term frequency) is the number of times the word appears in the text, and IDF (inverse document frequency) is a measure of how rare the word is in the dataset. The IDF of a word is calculated as:

IDF = log(N / DF)

where N is the total number of documents in the dataset and DF is the number of documents in which the word appears.

Another approach is to use a word embedding representation, where each word is represented as a dense vector of real values. These vectors can be learned from the data using techniques such as word2vec or GloVe. In word2vec, the vectors for each word are learned by training a neural network to predict the surrounding words given a center word. The vectors for the words are the weights of the neural network. In GloVe, the vectors for each word are learned by training a model to predict the co-occurrence probabilities of the words in the dataset. The vectors for the words are the learned parameters of the model.

Once the text data has been represented, you can use it to train a Random Forest model using any of the standard machine learning libraries, such as scikit-learn in Python. A Random Forest model is an ensemble of decision trees, where each tree is trained to make predictions based on a random subset of the features in the data. The final prediction made by the model is the average of the predictions made by each tree.

The decision trees in a Random Forest model are trained using a variant of the CART (Classification and Regression Tree) algorithm, which involves recursively partitioning the data into smaller and smaller subsets based on a feature threshold. At each step in the tree, the algorithm chooses the feature and the threshold that result in the greatest reduction in impurity. The impurity of a node in the tree is calculated using the Gini index, which is defined as:

Gini = 1 - ∑(p^2)

where p is the proportion of the samples in the node that belong to each class. The Gini index ranges from 0 (perfect purity) to 1 (maximum impurity).

## 5.5 Multinomial Naïve Bayes

Multinomial Naive Bayes is a machine learning algorithm that can be used for a variety of tasks, including classification and regression. It is a simple probabilistic classifier based on the idea of applying Bayes' theorem with strong independence assumptions between the features.

In a classification problem, the goal is to predict the class (also called the label or the output) of an input instance based on the values of a set of features (also called the predictors or the inputs). The class is one of a finite set of predefined classes. For example, in a sentiment analysis task, the class could be positive, negative, or neutral, and the features could be the words in a piece of text.

Bayes' theorem states that the probability of an event occurring can be calculated by multiplying the prior probability of the event by the likelihood of the event given some evidence. In the context of classification, the event is the class of the input instance, and the evidence is the values of the features. The theorem can be written as:

P(C|X) = (P(X|C) \* P(C)) / P(X)

where P(C|X) is the posterior probability of the class given the features (i.e. the probability of the class given the evidence), P(X|C) is the likelihood of the features given the class (i.e. the probability of the evidence given the class), P(C) is the prior probability of the class (i.e. the probability of the class before considering the evidence), and P(X) is the prior probability of the features (i.e. the probability of the evidence before considering the class).

Multinomial Naive Bayes makes the assumption that the features are independent of each other given the class. This means that the likelihood of the features can be calculated by multiplying the probabilities of each feature independently. The likelihood can be written as:

P(X|C) = ∏ P(x\_i|C)

where x\_i is the i-th feature, and C is the class.

Under this independence assumption, the posterior probability of the class can be calculated using the above equation for Bayes' theorem by replacing the likelihood with the product of the individual feature probabilities.

The prior probability of the class, P(C), can be calculated by counting the number of instances of each class in the training dataset and normalizing the counts by the total number of instances. The prior probability of the features, P(X), is typically not used in the classification process and can be ignored.

To make predictions with the Multinomial Naive Bayes model, you can calculate the posterior probability of each class for a given input instance, and then choose the class with the highest probability as the prediction

# 5.6 Support vector Machine

Support Vector Machines (SVMs) are a powerful and popular method for supervised learning tasks such as classification and regression. The basic idea behind SVMs is to find the maximum margin linear classifier, a concept that is closely related to the idea of a support vector.

SVMs work by finding the best boundary or hyperplane that separates the different classes in the data. The best boundary is the one that maximizes the margin, which is the distance between the boundary and the closest data points from each class. These closest data points are called support vectors. The boundary that maximizes the margin is also the one that is the most robust to errors, as it gives the most space between the different classes.

The process of finding the maximum margin boundary can be formulated as an optimization problem that is solved using a method called quadratic programming. The optimization problem seeks to minimize the square of the norm of the weight vector while ensuring that all data points are correctly classified. The constraints in the problem are set up in such a way that the margin is maximized.

SVMs can also be used for non-linearly separable data by using the so-called kernel trick, a method that maps the input data into a higher-dimensional space where it becomes linearly separable. The kernel trick is achieved by defining a kernel function that computes the dot product of the mapped feature vectors, without explicitly computing the coordinates of the mapped points. This allows SVMs to be applied to a wide range of data, including data that is not easily represented by a simple linear boundary.

Once the model is trained, it can be used to make predictions on new data by projecting it into the higher-dimensional space and finding the side of the boundary where it lies. The prediction will be based on the class label of the nearest point on the boundary, which is also called a decision boundary.

To evaluate the performance of the model, common techniques include using evaluation metrics such as accuracy, precision, and recall, and using methods such as k-fold cross-validation to estimate the generalization performance of the model.

In summary, SVMs are a supervised learning method that finds the best boundary or hyperplane that separates the different classes in the data by maximizing the margin, which is the distance between the boundary and the closest data points from each class. With kernel trick, it also work on non-linearly separable data by mapping the input data into a higher-dimensional space where it becomes linearly separable. Once the model is trained, it can be used to make predictions on new data and performance can be evaluated using evaluation metrics and cross-validation techniques.

# 5.6 Bidirectional LSTM model

A Bidirectional LSTM (BiLSTM) is a variant of LSTM that processes the input sequence in two directions: forward and backward. A Long Short-Term Memory (LSTM) is a type of recurrent neural network (RNN) that is capable of learning long-term dependencies in sequential data. An RNN is a type of neural network that processes sequential input by using loops to allow the network to maintain a state that depends on the previous inputs. This allows the network to capture patterns in the data that may span long periods of time.

An LSTM network is composed of LSTM cells, which are similar to traditional artificial neurons but with a more complex structure. Each LSTM cell has three gates: an input gate, an output gate, and a forget gate. The gates are used to control the flow of information into and out of the cell.

The input gate determines which input values should be passed through to the cell state. It does this by using a sigmoid activation function, denoted by the symbol σ, to calculate a value between 0 and 1 for each input, with 0 indicating that the input should be completely blocked and 1 indicating that it should be passed through unchanged. The input gate can be calculated using the following equation:

i\_t = σ(W\_i \* [h\_{t-1}, x\_t] + b\_i)

where i\_t is the input gate at time step t, W\_i is the weight matrix for the input gate, h\_{t-1} is the previous hidden state, x\_t is the input at time step t, and b\_i is the bias term for the input gate. The brackets [h\_{t-1}, x\_t] indicate that the hidden state and input are concatenated together.

The output gate determines which values in the cell state should be passed on as output. It does this by using a sigmoid activation function in a similar way to the input gate. The output gate can be calculated using the following equation:

o\_t = σ(W\_o \* [h\_{t-1}, x\_t] + b\_o)

where o\_t is the output gate at time step t, W\_o is the weight matrix for the output gate, h\_{t-1} is the previous hidden state, x\_t is the input at time step t, and b\_o is the bias term for the output gate.

The forget gate determines which values in the cell state should be forgotten. It does this by using a sigmoid activation function to calculate a value between 0 and 1 for each value in the cell state, with 0 indicating that the value should be completely forgotten and 1 indicating that it should be retained.

The cell state is a continuous vector that stores information from the past. It is modified by the input, output, and forget gates at each time step. The cell state can be thought of as a "memory" of the past input that is retained over time.

The LSTM cell also has an output, which is used to make predictions based on the input and cell state. The output is calculated by combining the input, cell state, and output gate using an activation function, such as tanh or ReLU.

To make a prediction using an LSTM network, you pass the input through the network one time step at a time, starting with the initial cell state and hidden state. At each time step, the input, previous hidden state, and previous cell state are used to calculate the current hidden state and cell state using the input, output, and forget gates. The hidden state and cell state are then used to calculate the output of the network for that time step.

A BiLSTM network consists of two LSTM networks, one that processes the input sequence from start to end and another that processes it from end to start. The outputs of the two networks are combined to make the final prediction.

Diagram

Description automatically generated

Figure 21: Bi LSTM working

BiLSTM is useful for tasks that involve understanding the context of the input sequence. For example, in a natural language processing task, understanding the meaning of a word in a sentence may depend on the words that come before and after it. By processing the input sequence in both directions, a BiLSTM network can capture the dependencies between words in both directions

In a BiLSTM network, the forward and backward LSTM networks process the input sequence independently and generate their own hidden states and cell states. The final prediction is made by combining the outputs of the two networks. There are several ways to combine the outputs, such as concatenating them or summing them.

# **Chapter 6: Model Performance Evaluation Metrics**

There are many different metrics that can be used to evaluate the performance of a machine learning model. Here are a few of the most common ones:

1. Confusion Matrix
2. Accuracy
3. Precision
4. Recall
5. F1 Score

## 6.2 Confusion Matrix

A confusion matrix is a table that is used to evaluate the performance of a classification model. It compares the predicted class labels with the true class labels and counts the number of correct and incorrect predictions.

The confusion matrix contains four quadrants:

1. True Positives (TP): These are the cases where the model predicted the positive class, and it was correct.
2. False Positives (FP): These are the cases where the model predicted the positive class, but it was incorrect.
3. False Negatives (FN): These are the cases where the model predicted the negative class, but it was incorrect.
4. True Negatives (TN): These are the cases where the model predicted the negative class, and it was correct.

From the counts in the confusion matrix, a variety of evaluation metrics can be computed, such as accuracy, precision, recall, and F1 score. The confusion matrix is a useful tool for understanding the performance of a classification model, and for identifying areas where the model can be improved.

Table

Description automatically generated

Figure 22: Overview of confusion matrix/contingency table.

## 6.2 Accuracy

This is the number of correct predictions made by the model, divided by the total number of predictions. It is a good metric to use when the target classes are well balanced (i.e., there is roughly the same number of examples for each class).

Accuracy = (TP + TN) / (TP + TN + FP + FN)

## 6.3 Precision

This is the number of true positive predictions made by the model, divided by the total number of positive predictions made by the model. Precision is a good metric to use when the goal is to minimize false positives, such as in the case of a spam filter.

Precision = TP / (TP + FP)

## 6.4 Recall

This is the number of true positive predictions made by the model, divided by the total number of actual positive examples in the data. Recall is a good metric to use when the goal is to minimize false negatives, such as in the case of a cancer diagnosis model.

Recall = TP / (TP + FN)

## 6.5 F1 Score

This is the harmonic mean of precision and recall. It is a good metric to use when you want to balance precision and recall.

F1 Score = 2 \* Precision \* Recall / (Precision + Recall)

# **Chapter 7 : Implementation**

Twitter sentiment analysis is the classification of tweets according to their sentiment using natural language processing and machine learning techniques (positive, neutral, or negative). There are several methods for implementing Twitter sentiment analysis, including employing pretrained models, machine learning models, and deep learning models.

|  |  |
| --- | --- |
| This is great. | Positive |
| I don’t like this food | Negative |
| Maybe I’m mad but I’m now the proud owner of a potentially #bendy #iPhone7, it’s so much bigger than the #4s | Positive |
| I am glad you are here | Positive |
| I’m not sure I want it. It’s too big to git in my back pocket! Lol #iPhone7 | Negative |
| What is this about? | Neutral |

Table 1: Basic example of Sentiment analysis

In this study, sentiment analysis is conducted utilising two primary methods: lexicon-based analysis and machine learning.

Diagram

Description automatically generated

Figure 23: Different approaches of Sentiment Analysis

Using a pretrained model, which is a model that has already been trained on a big dataset and can be used to categorise tweets without additional training, is one option. Available models for Twitter sentiment analysis include the TextBlob library and the VADER library. These models are simple to use and can produce satisfactory outcomes, but they may not be as accurate as models trained expressly for the job at hand.

Use a machine learning model, which is a model that has been trained on a dataset and can learn to categorise tweets based on the data's properties. Several types of machine learning methods, including Random Forest and Multinomial Naive Bayes can be utilised for Twitter sentiment analysis. These models require a labelled training dataset and may be fine-tuned to increase their performance.

Using a deep learning model, which is a form of machine learning model built of numerous layers of artificial neural networks, is the third strategy. Deep learning models can understand complicated data patterns and achieve cutting-edge performance on a variety of tasks, including Twitter sentiment analysis. However, they demand a huge quantity of data and computer resources to train, and their implementation may be more challenging than that of other models.



Figure 24: Basic outline of the project

## 7.1 Preprocessing

Preprocessing tweets for sentiment analysis entails cleaning and preparing tweets for analysis in order to extract valuable information and insights. This may entail a variety of steps, including:

### 7.1.1 Convert to lower case:

Convert all the words in tweet to lowercase to standardize.

### 7.1.2 Removing numbers, hashtags, user mentions, and other noise:

These elements do not contribute to the sentiment of the tweet and can be removed to make the tweet easier to analyze. So everything after ‘@’,’#’ (also called as hashtag), ‘&’ symbols and if it starts with letters from a-z and numbers from 0-9 is replaced with empty space.

### 7.1.3 Removing special characters, retweets, spaces:

Special characters, such as emojis and punctuation marks, ‘RT’ prefix on the start of a tweet which denotes a retweet or a forwarded tweet, empty spaces followed by ‘RT’ can be removed or replaced with a space to make the tweet easier to analyze.

### 7.1.4 Removing URLs:

URLs do not contribute to the sentiment of the tweet and can be removed.

Diagram

Description automatically generated

Figure 25: Preprocessing tweets on twitter

## 7.2 Data processing using Natural Language processing (NLP)

Natural language processing, sometimes known as NLP, is an area of study within the disciplines of computer science, artificial intelligence, and linguistics that focuses on the interactions that take place between computers and human languages. The objective of natural language processing, often known as NLP, is to create methods that will enable computers to read, comprehend, and produce human language in order to improve communication between humans and machines.

NLP may be used for a broad variety of purposes, such as the translation of languages, the summary of texts, the study of emotions, and the development of content. Techniques from the field of natural language processing (NLP) are utilised in a wide range of industries, including as healthcare, banking, and customer service, in order to examine and comprehend enormous amounts of text data and derive useful insights.

NLP is a sophisticated and multidimensional area, and it draws on a range of disciplines, including linguistics, computer science, and psychology, amongst others. Study into natural language processing (NLP) has made great headway in recent years, and it remains an important area of research and development today.



Figure 26: Basic Architecture of Sentiment Analysis using NLP

### 7.2.1 Natural Language Toolkit (NLTK)

The Natural Language Toolkit (NLTK) is a free, open-source library for natural language processing (NLP) in Python. It provides a wide range of tools and resources for working with human language data, including methods for tokenization, stemming, and lemmatization, as well as tools for building and parsing grammars.

NLTK was developed by Steven Bird and Edward Loper at the University of Pennsylvania and has been widely adopted by researchers and practitioners in the NLP community. It is designed to be easy to use and has extensive documentation and examples, making it a useful resource for those new to NLP.

In addition to its core functionality, NLTK also includes a large collection of text data and corpora, as well as pre-trained models for a variety of NLP tasks. These resources make it an ideal tool for teaching and learning about NLP, as well as for developing and testing new NLP algorithms.

Overall, NLTK is an important resource for anyone interested in natural language processing and is widely used in academic and industry settings.

First step in performing text analysis is vectorization which is a consists of tokenization, lemmatization, stemming, stop words and normalization



Figure 27: Steps for Vectorization

1. Tokenization:

Tokenization is the process of breaking a stream of text up into individual words, phrases, symbols, or other meaningful elements, known as tokens. These tokens can then be analyzed and processed by a computer, usually for the purpose of interpreting or manipulating human language in some way. Tokenization is a fundamental task in natural language processing (NLP), as it allows NLP algorithms to work with raw text as input and identify the underlying structure and meaning of the text. It is a key step in many NLP applications, including information retrieval, machine translation, and text classification.

‘word\_tokenize’ and ‘TweetTokenizer’ are two functions used to tokenize texts in the Python Natural Language Toolkit (NLTK) library in this project.

‘word\_tokenize’ is a function that uses the Punkt tokenization algorithm to tokenize a given text into words. Punkt is a widely-used, unsupervised machine learning algorithm that was trained on a large dataset of unannotated text, and is designed to identify word boundaries in a general-purpose language such as English. ‘word\_tokenize’ is a robust and widely-used function that can handle a variety of text types and languages, and is a good choice for many NLP tasks.

‘TweetTokenizer’, on the other hand, is a specialized tokenization function that is designed specifically for tokenizing social media texts such as tweets. This function is designed to handle the unique characteristics of social media texts, such as the use of emojis, hashtags, and user mentions, and is able to preserve these elements as individual tokens rather than breaking them up into smaller units. ‘TweetTokenizer’ is useful for NLP tasks involving social media texts, but may not be appropriate for other types of texts.

In summary, ‘word\_tokenize’ and ‘TweetTokenizer’ are both useful tools for tokenizing texts in Python, but they are tailored to different types of texts and should be chosen appropriately for the task at hand.

Following is the example of tokenization performed on the tweets downloaded using tweepy:

Text

Description automatically generated with medium confidence

Figure 28: Tokenized tweets

1. Lemmatization and Stemming:

Lemmatization and stemming are two techniques used in natural language processing (NLP) to extract base forms of words. These base forms, known as lemmas and stems, can be used to identify the underlying lexical structure of a text, as well as to facilitate various NLP tasks such as information retrieval, machine translation, and text classification.

Lemmatization is the process of reducing a word to its base form, known as the lemma. This base form is often the word's dictionary form, and is used to represent the word in its most general, dictionary-defined sense. For example, the lemma of the word "jumping" is "jump," and the lemma of the word "jumps" is also "jump." Lemmatization is a more sophisticated process than stemming, as it takes into account the word's part of speech and inflectional ending in order to determine the correct base form.

Stemming, on the other hand, is a simpler process that involves heuristically chopping off the ends of words to extract a base form. This base form, known as the stem, is not necessarily the dictionary form of the word, and is often just a rough approximation of the word's true lemma. For example, the stem of the word "jumping" might be "jump," but the stem of the word "jumps" might be just "jump." Stemming is a less accurate process than lemmatization, but it is faster and more efficient, and is often used in situations where speed is more important than accuracy and so for this project lemmatization will be used where we need more accuracy as it’s a sentiment analysis.

Text

Description automatically generated with medium confidence

Figure 29: Lemmatization of tweets

Graphical user interface, text

Description automatically generated

Figure 30: Stemming of tweets

Table

Description automatically generated with medium confidence

Figure 31: Comparison of Tokens, Stemming and Lemmatization

1. Stop words:

Stop words are common words in a language that do not convey significant meaning and are usually removed from texts before processing. In the context of sentiment analysis, stop words can be removed in order to focus on the more important content words that carry the sentiment of the text.

For example, consider the following sentence: "I am not particularly impressed by the movie." If we remove the stop words "I," "am," "not," and "by," we are left with the content words "particularly," "impressed," "movie." These words carry the sentiment of the text, with "particularly" and "impressed" indicating a negative sentiment, and "movie" providing the subject of the sentiment. By removing the stop words, we can more easily identify and extract the sentiment of the text.

However, it is important to note that stop words may still carry some sentiment and should not be completely ignored. For example, the stop word "not" in the above sentence negates the sentiment of the words "particularly" and "impressed," making the overall sentiment of the text more positive. Therefore, the decision to remove stop words should be carefully considered in the context of the specific sentiment analysis task at hand.

1. Normalisation:

Normalization only means cleaning the data, such as eliminating emoticons. If we wish to remove them or remove punctuation marks, hyperlinks, and other similar elements.

1. Part-of-speech (POS) tagging:

Part-of-speech (POS) tagging is the process of annotating each word in a text with its corresponding part of speech. This task is a fundamental component of natural language processing (NLP) and is frequently utilized to disambiguate word senses and identify the underlying grammatical structure of a sentence.

There are various methods for POS tagging, including rule-based approaches that utilize manually defined rules to tag words and machine learning-based approaches that utilize a trained model to predict the POS tags of words in a text.

A common machine learning approach to POS tagging involves the use of sequence labeling methods, such as hidden Markov models (HMMs) or conditional random fields (CRFs). These methods model the probability of a sequence of POS tags given a sequence of words and learn the probabilities of transitioning between different POS tags from the training data. During the prediction phase, the model uses these probabilities to predict the most likely POS tag for each word in the text. POS tagging is a crucial step in numerous NLP tasks, such as named entity recognition, information extraction, and machine translation.

## 7.3 Lexicon based approach

A lexicon-based approach to sentiment analysis identifies the sentiment of a given text by employing a predefined collection of terms, known as a lexicon. This method is based on the premise that certain words are connected with particular emotions, and that by counting the quantity of positive and negative terms in a text, we may estimate the text's overall mood.

A lexicon may contain a list of positive terms such as "good," "great," and "excellent," as well as a list of negative words such as "bad," "awful," and "dreadful." To assess the emotion of a particular text using this lexicon, we would simply count the amount of positive and negative terms and deduce the overall sentiment from this count. If there are more positive terms than negative ones, we may assume the emotion of the text is good, and vice versa.

A lexicon-based approach to sentiment analysis offers a number of benefits. One advantage is that it is simple and straightforward to apply. Another benefit is that it may be quite accurate, particularly if the lexicon is well-defined and exhaustive. Nonetheless, there are limits to this strategy. As words can have various meanings and implications, and the same word can occasionally communicate different emotions in different settings, it can be challenging to create an extensive and complete vocabulary. This method does not take into consideration the text's context or structure, therefore it may not be able to capture the mood of more complicated or nuanced texts effectively.

Textblob and VADER, two lexicon-based Python libraries, are utilised first.

Text

Description automatically generated

Figure 32: Example of TextBlob text sentiment classification

Text

Description automatically generated

Figure 33: Example of VADER text sentiment classification

### 7.3.1 Textblob and VADER Sentiment analysis

Using these pretrained models, sentiment analysis will be done on the tweets that we downloaded previously and placed in the 'tweets\_df' data frame which was utilised for data visualisation.

The tweets were initially preprocessed or cleaned using the methods which were discussed in Chapter 7.1 then a ‘tweet\_tokenizer’ object was created to tokenize the cleaned tweets by passing it through the built in ‘TweetTokenizer’ nltk function and stored them in the tweets\_df dataframe.

Graphical user interface

Description automatically generated

Figure 34: Clean Tweets

In TextBlob output the polarity value represents overall the sentiment of the tweet.

If polarity > 0, the sentiment is positive.

If polarity < 0, the sentiment is negative.

If polarity = 0, the sentiment is neutral.

In VADER output the compund score represents the sentiment of the tweet.

If compound score > 0.05 , the sentiment is positive.

If compound score < 0.05 , the sentiment is negative.

If compound score = 0.05 , the sentiment is neutral.

To find the difference in sentiment analysis of the both methods a new column ‘Different\_sent’ is created and a condition is written where if the sentiment output of both the method is same it will return 0 and if it’s different it’ll return 1.

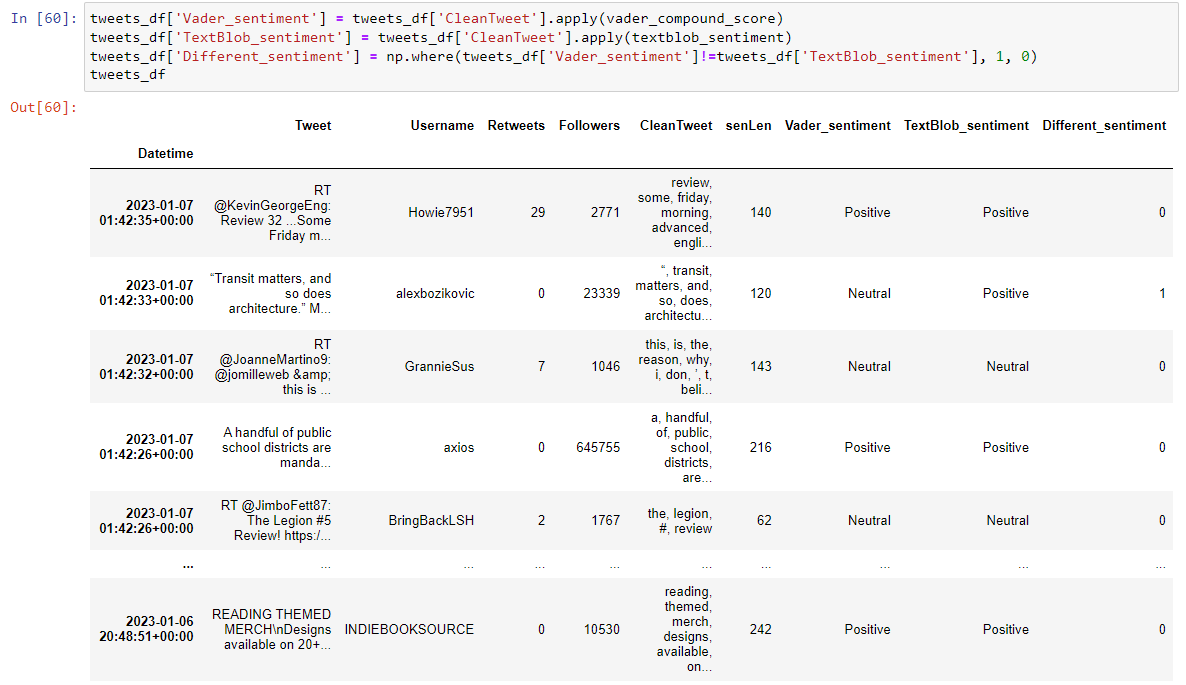


Figure 35: Screenshot of the tweets\_df dataframe after applying TextBlob and VADER sentiment analysis

The output is visualized using pie chart to find the number of times there’s a difference in the sentiment prediction and similarity in analysis of both the methods.

Chart, pie chart

Description automatically generated

Figure 36: Pie chart of VADER vs TextBlob Sentiment Analysis

## 7.3.2 roBERTa

The particular roBERTa model which is being used for this project is developed by Meta AI team and is trained on 58 million tweets.

The tokenizer and the model is downloaded from the hugging community website where the model is hosted using ‘pip install transformers’ command

A variable ‘tweet’ is created and a tweet is passed into it.



Figure 37: tweet text input:

Tweets are split and preprocessed to remove ‘@mentions’ which is how a tweet starts with the username tag and then url link is identified from the tweet if it starts with http and both of them are converted to strings and joined back together.

Text

Description automatically generated

Figure 38: Tweet preprocessed for roBERTa

Model and tokenizer is loaded from pretrained Roberta model and a variable ‘labels’ s created to store sentiments.

A screenshot of a computer

Description automatically generated with medium confidence

Figure 39: roBERTa model, tokenizer and labels

The tweet is tokenized and converted into tensors which is a multi-dimensional array of numerical values to encode the tweet which is the process of converting written or spoken language into a digital format that a computer can understand and process. It is a crucial step in natural language processing (NLP) and other text-based data analysis tasks, as it allows computers to analyze, understand, and generate human language.

Then sentiment analysis is performed by passing it to the model and then a tensor with gradient is obtained.

Text

Description automatically generated

Figure 40: Encoding and passing to model

The tensor needs to be detached from the current computing graph gradient and it is done by using ‘detach()’ function and then the three sentiment values are obtained which are three numbers with decimal places. It is converted into probabilities which is 0 and 1 using ‘softmax()’ function.

Text

Description automatically generated

Figure 41: Sentiment analysed by roBERTa model

The maximum value of the scores is taken which will be the corresponding sentiment and is printed.

Graphical user interface, text, application

Description automatically generated

Figure 42: Output of roBERTa model

## 7.4 Machine Learning based approach

There are several different categories of machine learning algorithms, which can be broadly classified into the following types:

1. Supervised learning: In this approach, a model is trained on labeled data, where the correct output is provided for each input example. The goal is to enable the model to generalize and make accurate predictions on new, unseen examples that are drawn from the same distribution as the training set.
2. Unsupervised learning: In this approach, the model is trained on unlabeled data, and the goal is to discover relationships or patterns in the data.
3. Reinforcement learning: This category of machine learning algorithms involves training an agent to make a series of decisions in an environment to maximize a reward.

Diagram

Description automatically generated

Figure 43: Machine learning approaches

Natural language processing and computational linguistic models for text comprehension are used for sentiment analysis. In the classification problem of sentiment analysis, datasets are separated into positive, negative, and neutral categories. Thus, we obtain categorical outputs.

Most common supervised classification machine learning algorithms are

* Random Forest
* Naive Bayes
* Support Vector Machines (SVMs)
* Decision Trees
* KNN

Random Forest, Naive Bayes and SVM are the methods used in this project in Machine Learning based approach.

### 7.4.1 Random Forest

**Training Data Preparation:**

For training the twitter sample files from nltk will be used which consists of three files which are positive, negative and unlabelled twtter sample ‘.json’ format files.

The positive and negative tweet file is tokenized and loaded into two separate arrays namely ‘positive\_tokens’ and ‘negative\_tokens’.

As lemmatizer uses POS variable the tokens need to be mapped with their associated POS tag which tells what is the position of the words in a sentence whether it is verb, non-verb, adjective etc as per the following table.(*Building a Large Annotated Corpus of English: The Penn Treebank*, n.d.)

Graphical user interface, application

Description automatically generated with medium confidence

Figure 44: Alphabetical list of part-of-speech tags used in the Penn Treebank Project

So a variable ‘position\_tag’ is created to apply POS tagging to the tokens.

Text, letter

Description automatically generated

Figure 45: Tokens and their POS tags

These will be lemmatized to remove unnecessary symbols, retweets and links. A lemmatizer object namely ‘lemmatizer’ is created to import WordNetLemmatizer module. An array ‘lemma’ is created to append the lemmatized words.

A picture containing chart

Description automatically generated

Figure 46: Lemmatised tokens

The stopwords library from nltk is used to remove the stopwords from the lemmatized tokens.

Text

Description automatically generated

Figure 47: Stopwords library

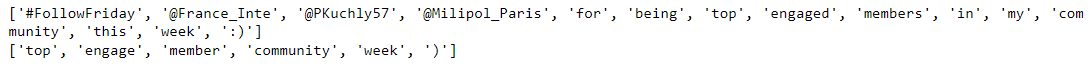
Text

Description automatically generated with medium confidence

Figure 48: Tokens after removing stop words

The size of the tokens array is reduced after removing the stop words.

Combining all these into a function called ‘clean\_tokens’ to clean the tokens, lemmatize and apply position tags as per the documentation of WordNetLemmatizer. This function will be used for the other Machine learning models as well.



*Fig: Array of positive tokens before and after passing it through clean\_token function*

As these tweets are currently unlabeled, it is necessary to provide a mechanism to add the status. For this, the 'data prepare' function is built, and then positive and negative tokens, together with their associated status/label, are passed to the function to generate positive and negative featureset arrays and they are combined together into one array.

Text, letter

Description automatically generated

Figure 49: Combined featureset array

A list of features and labels will be created and values from ‘featureset’ array will be appended to features and labels arrays for training.

Graphical user interface, text, application

Description automatically generated

Figure 50: Features and labels array

**Model Building:**

Tfidf vectorizer will be used for vectorization(representing a sequence of words or other text data as a numerical vector which the machine can understand). The parameters chosen here will be ngram\_range = (1,2), sublinear\_tf = True, max\_features = 3000 and preprocessor = ‘ ‘.join

In the field of natural language processing and information retrieval, an n-gram is a contiguous sequence of n items from a given sample of text or speech. The items can be words, letters, or other symbols. In the context of computing the term frequency-inverse document frequency (TF-IDF) weights for a sample of text, n-grams are used to represent the frequency of particular combinations of items in a document and the rarity of those combinations across a collection of documents. For example, if n is set to 2, the resulting 2-grams (also known as bigrams) capture the frequency and rarity of all possible combinations of two consecutive words in the text. Using n-grams in the computation of TF-IDF weights can be useful in cases where the meaning of a word depends on the words that come before or after it, as the n-gram representation captures this contextual information. However, using large values of n can result in a large number of unique n-grams and a correspondingly large feature space, which can be computationally expensive and may not always lead to improved results. So here unigram and bigram will be applied.

Sublinear term frequency (sublinear\_tf) scaling is a technique used to down-weight the importance of terms that occur very frequently in a document. Sublinear scaling is often used in information retrieval because it helps to reduce the importance of terms that are very common in a large collection of documents, such as stop words (e.g. "the", "a", "and", etc.), which can otherwise dominate the search results.

max\_features = 3000 as we got accuracy highest for this value which will be discussed later on.

preprocessor = ’ ‘.join signifies that the tokens are required to be joined and split again with Tfidf.

train\_test\_split function is created to split our data to split vectorized features and labels. 15% of the data will be used for testing and 85% for training. Shuffle parameter is set true so that the dataset will be shuffled for training and not train only on positive or negative tweets alone.

‘RandomForestClassifier()’ function is called and stored to an object ‘rf\_classifier’ with n\_estimators parameter as 500 (number of trees to build before taking in the final predictions). This is the optimal value we got for best accuracy for this particular model.

This classifier is then fitted or trained with X\_train and y\_train. For prediction on the model X\_test is used.

Classification report and accuracy for the model was obtained which will be discussed in Evaluation part of the Chapter.

The Random Forest classifier model and Vectorizer were saved on the disc using ‘pickle’ module so that it can be called later for analysing tweets.

### 7.4.2 Multinomial Naïve Bayes

Multinomial Naïve Bayes implementation is similar to Random Forest where ‘naive\_bayes.MultinomialNB()’ function is called and stored to an object ‘nb\_classifier’ and Tfidf vectorizer is used by calling the function to ‘nb\_vectorizer’ object variable with parameters chosen same as before which will be ngram\_range = (1,2), sublinear\_tf = True, max\_features = 3000 and preprocessor = ‘ ‘.join

As Multinomial Naïve Bayes is found to be best for text classification when compared with other naïve Bayes model, it will be used here for sentiment analysis. In the field of natural language processing, the occurrence counts of words in a text document serve as a frequent illustration of the multinomial distribution. (*Discrete Probability Distributions for Machine Learning - MachineLearningMastery.Com*, n.d.)

The Multinomial Naïve Bayes classifier model and Vectorizer were saved on the disc using ‘pickle’ module so that it can be called later for analysing tweets.

### 7.4.3 Support Vector Machines

SVM is similar to Random Forest where ‘svm.SVC()’ function is called and stored to an object ‘svm\_classifier’ and Tfidf vectorizer is used by calling the function and storing to ‘svm\_vectorizer’ object variable with parameters chosen same as before which will be ngram\_range = (1,2), sublinear\_tf = True, max\_features = 3000 and preprocessor = ‘ ‘.join

The SVM classifier model and Vectorizer were saved on to the disc using ‘pickle’ module so that it can be called later for analysing tweets.

Text

Description automatically generated

Figure 51: ML models functions

All these functions were called and the tweets were passed through them and sentiement analysis was done.

Text

Description automatically generated

Figure : Model function call and sentiment analysis

## 7.5 Deep Learning approach

## 7.5.1 Bi directional LSTM

1. **Creating Dataset**

An implementation of Bi- LSTM will have the following basic steps:

Diagram, text

Description automatically generated

Figure 53: Steps to Implement BiLSTM

The dataset is made using the negative and positive tweet files from the nltk package as before and is stored in a dataframe for deep learning model implementation. The ‘.json’ format files contains a lot of values which we don’t need for sentiment analysis so the tweets alone were extracted using ‘strings’ function from nltk as the tweets text were in string format.

Text

Description automatically generated

Figure 54: Extracting tweets from .json file

They were used to create ‘featureset’ arrays which was did in 7.4.1. The features which are tweets in this case and labels which are sentiment here are separately stored in two arrays and stored in a dataframe ‘df1’

A picture containing chart

Description automatically generated

Figure 55: Dataframe creation for Bi-LSTM model

This dataset has equal sentiment distribution which is 5000 positive and negative sentiment tweets.

1. **Cleaning Dataset:**

The similar method which was adopted in Chapter 7.1 will be used here as well which involves:

* Convert all text to lower case
* Removing numbers, hashtags, user mentions, and other noise
* Removing special characters, retweets, spaces
* Removing URLs
* Remove Stop words

Text

Description automatically generated

Figure 56: Cleaning Dataset for Bi-LSTM

A picture containing graphical user interface

Description automatically generated

Figure 57: Function to remove repeating words

1. **Transform data into numeric tensors or Vectorization:**

As with all other neural networks, deep-learning models cannot accept unprocessed text as input; they can only process numeric tensors; hence, this step is non-negotiable. There are numerous methods to do this; for instance, like for the Machine learning models which were discussed earlier TFIDF Vectorizer were used or the simple but subpar Bag-Of-Words method. As Deep Learning method is being employed, the ideal technique is to convert your text data (which may be regarded as word or character sequences) into low-dimensional floating-point vectors.

All tokenized sentences will be detokenized using ‘TreebankWordDetokenizer’

Graphical user interface, text

Description automatically generated

Figure 58: Detokenize words

The array of texts need to be transformed into 2D numeric arrays before converting sentences to floating point tensors.

As per official Keras documentation (*Timeseries Data Preprocessing*, n.d.) The ‘pad\_sequences’ function turns a list (of length ‘num\_samples’) of sequences (lists of integers) into a 2D Numpy array of shape (‘num\_samples’, ‘num\_timesteps’). ‘num\_timesteps’ is either the ‘maxlen’ parameter if supplied, or the length of the longest sequence in the list.

Sequences that are shorter than num timesteps are padded to num timesteps length using value. Sequences greater than num timesteps are shortened such that they suit the required duration

A picture containing text

Description automatically generated

Figure 59: Converting text array to 2D numeric arrays (padding)

Word embeddings, a type of natural language text representation, encode words in a continuous numerical vector space. This permits mathematical operations such as addition and subtraction to be performed on vectors, thereby capturing the meaning and relationships between words. There are a number of approaches for learning word embeddings, such as word2vec and GloVe, which have both proved successful in various natural language processing applications. Specifically, word embeddings have proven essential in achieving cutting-edge outcomes in language translation and text classification.

Two methods exist for acquiring word embeddings:

* Using a pre-trained collection of word vectors as an additional layer in your model, similar to how you may use a pre-trained neural network layer, is one approach for leveraging word embeddings.
* Optionally, word embeddings can be learned from scratch by beginning with randomly initialised word vectors and subsequently modifying them to capture significant associations between words. This may be accomplished by adding a layer to your TensorFlow or Keras neural network model. This strategy may be more suitable when learning a fresh embedding space for each unique job.

1. **Embedding Layer**



Figure 60: Embedding layer

The embedding layer is often put in a neural network model above a layer that takes 2D integer tensors with a shape of (samples, sequence length). This layer requires two arguments: the token count and the dimension of the embeddings. In this instance, there are 1000 tokens, and the embedding dimension is 64. The tensor is changed after going through the embedding layer into a 3D floating-point tensor of shape (samples, sequence length, embedding dimensions), which the neural network may then evaluate.

1. **Single LSTM Layer**

Recurrent neural networks (RNNs), such as long short-term memory (LSTM) layers, may handle sequences of input by iterating over the sequence's constituents and preserving information about what has already been processed. This is in contrast to other neural network types, such as highly linked or convolutional networks, which process each input independently without considering the context of the other inputs.

A picture containing text

Description automatically generated

Figure 61: Single LSTM layer model with respective embedding layer

As a regularisation strategy, the LSTM model's layer has 15 hidden units and a dropout rate to randomly disable part of the hidden units during training. The model also consists of a final dense layer with three output units, which correspond to the three possible classes in the dataset, and a softmax activation (*Softmax Function Definition | DeepAI*, n.d.) function for producing probabilistic outputs. The model is trained with the categorical crossentropy loss function and optimised with the RMSprop optimizer. Additionally, checkpoints are utilised to retain the optimal model acquired during training. This enables the user to receive the model with the highest performance based on the selected evaluation metric. Finally, the conventional design. fit is utilised to train the model for a predetermined number of epochs

1. **Bidirectional Layers**

A bidirectional layer (*Bidirectional Layer*, n.d.) is a type of layer that is used in recurrent neural networks (RNNs) and is designed to improve the order sensitivity of the RNN by combining the representations of two RNNs that process the input sequence in opposite directions. This allows the model to detect more complex patterns than a single RNN layer could detect. In this case, one layer processes the sequence in chronological order, while the other processes it in anti-chronological order. Bidirectional RNNs are commonly used because they can outperform traditional RNNs. In this example, a basic model is used without extensive hyperparameter tuning, though the model's design and regularization can be modified in order to improve performance and reduce the risk of overfitting. This particular architecture, although not always the best for text classification, is known to produce good results when working with text datasets.

Text

Description automatically generated with medium confidence

Figure 62: Bidirectional layer LSTM model

In this instance, the model is trained for 70 epochs (*Epoch Definition | DeepAI*, n.d.). An epoch is a single pass through the entire training dataset, during which the model is presented with the training data and the model's weights and biases are updated based on the loss function and optimizer being used. The number of epochs is a hyperparameter that can be adjusted. It is important to select the appropriate number of epochs for the model, as training for too few epochs can result in an underfit model that has not learned enough from the training data, while training for too many epochs can lead to an overfit model that does not generalize well to new, unseen data. In practice, deep learning models are often trained for hundreds or thousands of epochs, using techniques such as early stopping to avoid overfitting.

# **Chapter 8: Evaluation**

## 8.1 Classification Results

### 8.1.1 Random Forest

Table

Description automatically generated

Figure 63: Classification result of Random Forest

.

The classification report shows that the RF model has 0.9315 or 93.15% overall accuracy and has a high level of certainty when it predicts a text to be negative or positive, with precision of 0.92 for negative and 0.95 for positive sentiment. The model also has a good ability to find all the positive and negative instances in the dataset, with recall of 0.95 for negative and 0.91 for positive sentiment. Furthermore, the F1-score values of 0.91 for negative and 0.93 for positive sentiment indicate that the model has a good balance between precision and recall, making it a good model for the classification task.

### 8.1.2 Multinomial Naïve Bayes

Table

Description automatically generated

Figure 64: Classification result of Multinomial Naïve Bayes

The classification report shows that the model has a 0.806 or 80.6% overall accuracy has a precision of 0.79 for negative sentiment and 0.83 for positive sentiment, indicating that the model has a moderate level of certainty when it predicts a text to be negative or positive. The model also has a recall of 0.82 for negative sentiment and 0.79 for positive sentiment, indicating that the model has a moderate ability to find all the positive and negative instances in the dataset. Furthermore, the F1-score values of 0.81 for negative and 0.81 for positive sentiment indicate that the model has a moderate balance between precision and recall. Overall, the performance of the model is moderate in classifying the text into positive or negative sentiment.

### 8.1.3 Support Vector Machine

Table

Description automatically generated

Figure 65: Classification result of SVM

The classification report shows that the model has an accuracy score of 0.814 or 81.4%, indicating that the model is correctly classifying text into positive or negative sentiment with a reasonable degree of accuracy. The model also has a precision of 0.77 for negative sentiment and 0.86 for positive sentiment, indicating a moderate level of certainty when it predicts a text to be negative or positive. The model has a recall of 0.87 for negative sentiment and 0.76 for positive sentiment, indicating a moderate ability to find all the positive and negative instances in the dataset. Furthermore, the F1-score values of 0.82 for negative and 0.81 for positive sentiment indicate that the model has a moderate balance between precision and recall, making it a moderate model for the classification task.

### 8.1.4 Bi LSTM

Table

Description automatically generated with low confidence

Figure 66: Classification result of Bi-LSTM

Here 0 denotes ‘Negative’ and 1 denotes ‘Positive’. The classification report shows that the model has an accuracy score of 0.7648 or 76.8%, indicating that the model is correctly classifying text into positive or negative sentiment but with a moderate-low degree of accuracy. The model also has a precision of 0.75 for negative sentiment and 0.78 for positive sentiment, indicating a moderate level of certainty when it predicts a text to be negative or positive. The model has a recall of 0.78 for negative sentiment and 0.75 for positive sentiment, indicating a moderate ability to find all the positive and negative instances in the dataset. Furthermore, the F1-score values of 0.77 for negative and 0.76 for positive sentiment indicate that the model has a moderate balance between precision and recall, making it a moderate model for the classification task.

Chart, line chart

Description automatically generated

Figure 67: Train and test accuracy for BiLSTM

It was found that the testing accuracy was not so varying and was around 75% where as training accuracy went upto 92.3%

Graphical user interface, text, application, Word, website

Description automatically generated

Figure 68: Highest Train accuracy

### 8.1.5 Accuracy of TextBlob and VADER

As TextBlob and VADER are pretrained models their accuracy were tested using the same twitter positive and negative tweets from nltk corpura which was used for training the ML and Deep learning Bi-LSTM model. A dataset is created first using the tweets.

Text

Description automatically generated with low confidence

Figure 69: Dataset created using labelled tweets

RF, TextBlob and VADER sentiment analysis were performed on the data and added to dataframe

Table

Description automatically generated

Figure 70: RF, TextBlob and Vader sentiment

If the predicted sentiment matches with real sentiment a value one is assigned and stored in separate accuracy columns for each model and stored in dataframe.

Text

Description automatically generated with medium confidence

Figure 71: Accuracy columns creation for the models

Graphical user interface, application

Description automatically generated

Figure 72: Data frame with Accuracy columns

Then the accuracy of the models were calculated based on the number of time the accuracy columns of respective models had the value ‘1’ in them.

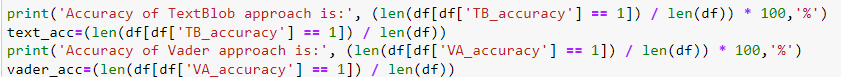


Figure : Accuracy calculation of pretrained models

The accuracy of the models were finally obtained.



Figure : Accuracy of TextBlob and VADER

## 8.2 Confusion Matrix

### 8.2.1 Random Forest

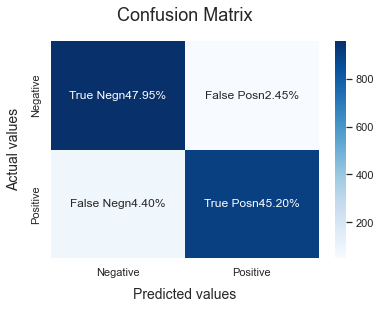


Figure : Confusion matrix of Random Forest

The confusion matrix of a Random Forest Sentiment analysis model shows that the model was able to correctly classify the negative sentiments (True negatives) 47.95% of the time and the positive sentiments (True positives) 45.20% of the time. It also indicates that the model made incorrect predictions for negative sentiments 2.45% of the time and for positive sentiments 4.4% of the time. Overall, the confusion matrix gives a quick summary of how well the model performed in correctly identifying negative and positive sentiments with the results of this model indicating that the model has a very high performance.

### 8.2.2 Multinomial Naïve Bayes

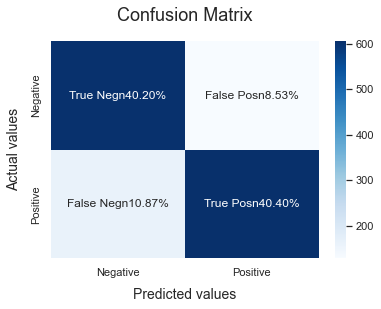


Figure : Confusion matrix of Multinomial Naive Bayes

The confusion matrix of a Multinomial Naive Bayes Sentiment analysis model shows that the model correctly classified negative sentiments (True negatives) 40.20% of the time, and correctly classified positive sentiments (True positives) 40.40% of the time. It also indicates that the model made incorrect predictions for negative sentiments 8.53% of the time and for positive sentiments 10.87% of the time. Overall, the confusion matrix here shows the performance of this model indicating that the model has a moderate performance.

### 8.2.2 Support Vector Machine



Figure : Confusion matrix of Support Vector Machine

The confusion matrix of a SVM Sentiment analysis model shows that the model correctly classified negative sentiments (True negatives) 42.6% of the time, and correctly classified positive sentiments (True positives) 38.8% of the time. It also indicates that the model made incorrect predictions for negative sentiments 6.13% of the time and for positive sentiments 12.47% of the time Overall, the confusion matrix here shows the performance of this model indicating that the model has a moderate-low performance.

### 8.2.2 Bidirectional LSTM

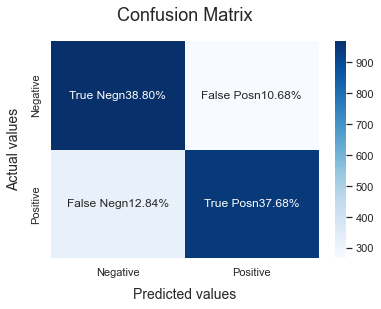


Figure : Confusion matrix of BiLSTM

The confusion matrix of a BiLSTM Sentiment analysis model shows that the model correctly classified negative sentiments (True negatives) 38.80% of the time, and correctly classified positive sentiments (True positives) 37.68% of the time. It also indicates that the model made incorrect predictions for negative sentiments 10.68% of the time and for positive sentiments 12.84% of the time. Overall, the confusion matrix gives a quick summary of how well the model performed in correctly identifying negative and positive sentiments, with the results of this model indicating that the model has a moderate-low performance.

## 8.4 Model Accuracy

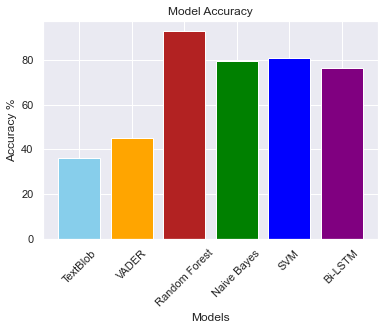


Figure : Model Accuracy graph

The graph displaying the accuracy of the models clearly illustrates that the Random Forest model had the best performance. The SVM model came in second in terms of accuracy, closely followed by Multinomial Naive Bayes. The BiLSTM model had the fourth highest accuracy among the models tested.

## 8.3 Sentiment analysis on downloaded tweets

Text

Description automatically generated with low confidence

This was the final output ‘tweets\_df’ where 9000 tweets were pulled from twitter and then the sentiment of the tweets were analysed using Random Forest, MNB, SVM and BiLSTM models.

# **Chapter 9. Conclusion**

In conclusion, this study aimed to investigate the effectiveness of various machine learning and deep learning models for sentiment analysis on Twitter data. The models tested included Random Forest, Multinomial Naive Bayes, SVM, Bi-directional LSTM, Textblob, Vader, and Roberta.

The results of the analysis showed that Random Forest and Multinomial Naive Bayes performed well in terms of accuracy, with SVM and Bi-directional LSTM also producing satisfactory results. However, the results for the Roberta model were not as successful, indicating the need for further optimization or alternative approaches.

Additionally, the analysis revealed that the training dataset, which was compiled using positive and negative tweets from the NLTK package, did not include neutral tweets. As a result, all of the tweets were classified as either positive or negative by the models trained in this project. On the other hand, the pretrained models, Textblob and Vader, were able to identify neutral tweets from the real-time tweets pulled from Twitter, as they were trained on a larger dataset containing all sentiments.

Overall, this study highlights the potential of using machine learning and deep learning techniques for sentiment analysis on social media data. However, it also underscores the importance of a balanced and diverse training dataset in order to achieve more comprehensive and accurate results.

# **Chapter 10. Future Works**

As a future direction for this project, it would be beneficial to address several limitations that were identified during the current study. One such limitation is the inability to identify emoticons, abbreviations, and other non-standard language forms present in the Twitter data. Implementing techniques for handling these types of language features could improve the accuracy of the sentiment analysis.

Another important consideration for future work is the detection and filtering of abusive words, spam, and hate speech. These types of content can significantly impact the overall sentiment of the data and should be accounted for in order to obtain more accurate results.

In addition to these technical considerations, it would also be valuable to address the issue of sarcasm and humor in the data. These language forms can often be difficult to interpret and may be misinterpreted as a different sentiment. Incorporating methods for detecting and properly classifying these types of language usage could improve the robustness of the model.

Finally, as mentioned in the previous analysis, the inclusion of a neutral sentiment class in the training dataset could also significantly improve the model's ability to accurately classify tweets with a neutral sentiment.

Overall, there are many potential avenues for future work in the area of Twitter sentiment analysis. By addressing the limitations of the current study and incorporating additional techniques and approaches, it is possible to create even more effective and reliable models for this task.

# **Chapter 11: References**

1. *A Gentle Introduction To Calculating The TF-IDF Values | by Ann Sebastian | Towards Data Science*. (n.d.). Retrieved January 10, 2023, from https://towardsdatascience.com/a-gentle-introduction-to-calculating-the-tf-idf-values-9e391f8a13e5
2. Asghar, M. Z., Khan, A., Ahmad, S., Qasim, M., & Khan, I. A. (2017). Lexicon-enhanced sentiment analysis framework using rule-based classification scheme. *PLoS ONE*, *12*(2). https://doi.org/10.1371/JOURNAL.PONE.0171649
3. Bengio, Y., Lamblin, P., Popovici, D., & Larochelle, H. (n.d.). *Greedy Layer-Wise Training of Deep Networks*.
4. *Bidirectional layer*. (n.d.). Retrieved January 9, 2023, from https://keras.io/api/layers/recurrent\_layers/bidirectional/
5. Breiman, L. (2001). Random forests. *Machine Learning*, *45*(1), 5–32. https://doi.org/10.1023/A:1010933404324/METRICS
6. *Building a large annotated corpus of English: the Penn Treebank*. (n.d.). Retrieved January 10, 2023, from https://catalog.ldc.upenn.edu/docs/LDC95T7/cl93.html
7. Burnap, P., & Williams, M. L. (2015). Cyber hate speech on twitter: An application of machine classification and statistical modeling for policy and decision making. *Policy and Internet*, *7*(2), 223–242. https://doi.org/10.1002/POI3.85
8. Dai, J., Pan, F., Shou, Z., & Zhang, H. (2021). RoBERTa-IAN for aspect-level sentiment analysis of product reviews. *Journal of Physics: Conference Series*, *1827*(1), 012079. https://doi.org/10.1088/1742-6596/1827/1/012079
9. Daniel, J., & Martin, J. H. (2023). *Speech and Language Processing Question Answering and In-formation Retrieval*.
10. *Discrete Probability Distributions for Machine Learning - MachineLearningMastery.com*. (n.d.). Retrieved January 8, 2023, from https://machinelearningmastery.com/discrete-probability-distributions-for-machine-learning/
11. Domingos, P., & Pazzani, M. (1997). On the Optimality of the Simple Bayesian Classifier under Zero-One Loss. *Machine Learning*, *29*(2–3), 103–130. https://doi.org/10.1023/A:1007413511361/METRICS
12. Dr. G. S. N. Murthy, Shanmukha Rao Allu, Bhargavi Andhavarapu, & Mounika Bagadi, Mounika Belusonti. (2020). Text based Sentiment Analysis using LSTM. *International Journal of Engineering Research And*, *V9*(05). https://doi.org/10.17577/IJERTV9IS050290
13. *Epoch Definition | DeepAI*. (n.d.). Retrieved January 9, 2023, from https://deepai.org/machine-learning-glossary-and-terms/epoch
14. Evgeniou, T., & Pontil, M. (2001). Support vector machines: Theory and applications. *Lecture Notes in Computer Science (Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *2049 LNAI*, 249–257. https://doi.org/10.1007/3-540-44673-7\_12
15. Feder Cooper, A., Levy, K., & de Sa, C. (2021). *Accuracy-Efficiency Trade-Offs and Accountability in Distributed ML Systems*. https://doi.org/10.1145/3465416.3483289
16. Gers, F. A., Schmidhuber, J., & Cummins, F. (2000). Learning to forget: Continual prediction with LSTM. *Neural Computation*, *12*(10), 2451–2471. https://doi.org/10.1162/089976600300015015
17. Graves, A., & Schmidhuber, J. (n.d.). *Framewise Phoneme Classification with Bidirectional LSTM Networks*.
18. Haddi, E., Liu, X., & Shi, Y. (2013). The Role of Text Pre-processing in Sentiment Analysis. *Procedia Computer Science*, *17*, 26–32. https://doi.org/10.1016/J.PROCS.2013.05.005
19. Hazarika, D., Konwar, G., Deb, S., & Bora, D. J. (2020). Sentiment Analysis on Twitter by Using TextBlob for Natural Language Processing. *Proceedings of the International Conference on Research in Management & Technovation 2020*, *24*, 63–67. https://doi.org/10.15439/2020KM20
20. Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, *9*(8), 1735–1780. https://doi.org/10.1162/NECO.1997.9.8.1735
21. *How to process textual data using TF-IDF in Python*. (n.d.). Retrieved January 10, 2023, from https://www.freecodecamp.org/news/how-to-process-textual-data-using-tf-idf-in-python-cd2bbc0a94a3/
22. Hutto, C. J., & Gilbert, E. (2014). VADER: A Parsimonious Rule-Based Model for Sentiment Analysis of Social Media Text. *Proceedings of the International AAAI Conference on Web and Social Media*, *8*(1), 216–225. https://doi.org/10.1609/ICWSM.V8I1.14550
23. Kim, Y. (2014). Convolutional Neural Networks for Sentence Classification. *EMNLP 2014 - 2014 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference*, 1746–1751. https://doi.org/10.48550/arxiv.1408.5882
24. Kiritchenko, S., Zhu, X., & Mohammad, S. M. (2014). Sentiment analysis of short informal texts. *Journal of Artificial Intelligence Research*, *50*, 723–762. https://doi.org/10.1613/JAIR.4272
25. Kwok, I., & Wang, Y. (2013). Locate the Hate: Detecting Tweets against Blacks. *Proceedings of the AAAI Conference on Artificial Intelligence*, 1621–1622. https://doi.org/10.1609/AAAI.V27I1.8539
26. Lewis, D. D. (n.d.). *Naive (Bayes) at Forty: The Independence Assumption in Information Retrieval*.
27. Li, F., Zhang, M., Fu, G., Qian, T., & Ji, D. (2016). *A Bi-LSTM-RNN Model for Relation Classification Using Low-Cost Sequence Features*. https://doi.org/10.48550/arxiv.1608.07720
28. Liaw, A., & Wiener, M. (2002). *Classification and Regression by randomForest*. *2*(3). http://www.stat.berkeley.edu/
29. Linder, F. (2017). Improved Data Collection from Online Sources Using Query Expansion and Active Learning. *SSRN Electronic Journal*. https://doi.org/10.2139/SSRN.3026393)
30. Liu, B. (2012). Sentiment Analysis and Opinion Mining. *Http://Dx.Doi.Org/ 10.2200/S00416ED1V01Y201204HLT016* , *5*(1), 1–184. https://doi.org/10.2200/S00416ED1V01Y201204HLT016
31. Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., Stoyanov, V., & Allen, P. G. (2019). *RoBERTa: A Robustly Optimized BERT Pretraining Approach*. https://doi.org/10.48550/arxiv.1907.11692
32. Nguyen, T. T. T., Nguyen, T. T., Pham, X. C., Liew, A. W. C., Hu, Y., Liang, T., & Li, C. T. (2016). A Novel Online Bayes Classifier. *2016 International Conference on Digital Image Computing: Techniques and Applications, DICTA 2016*. https://doi.org/10.1109/DICTA.2016.7796993
33. Nigam, S., Das, A. K., & Chandra, R. (2018). Machine Learning Based Approach to Sentiment Analysis. *Proceedings - IEEE 2018 International Conference on Advances in Computing, Communication Control and Networking, ICACCCN 2018*, 157–161. https://doi.org/10.1109/ICACCCN.2018.8748848
34. Oram, P. (2001). WordNet: An electronic lexical database. Christiane Fellbaum (Ed.). Cambridge,    MA: MIT Press, 1998. Pp. 423. *Applied Psycholinguistics*, *22*(1), 131–134. https://doi.org/10.1017/S0142716401221079
35. Pang, B., & Lee, L. (2008). Opinion mining and sentiment analysis. *Foundations and Trends in Information Retrieval*, *2*(1–2), 1–135. https://doi.org/10.1561/1500000011
36. Pang, B., Lee, L., & Vaithyanathan, S. (n.d.). *Thumbs up? Sentiment Classification using Machine Learning Techniques*. 79–86. Retrieved January 9, 2023, from http://reviews.imdb.com/Reviews/
37. *(PDF) SENTIMENT ANALYSIS: USES IN BUSINESS*. (n.d.). Retrieved January 9, 2023, from https://www.researchgate.net/publication/320959115\_SENTIMENT\_ANALYSIS\_USES\_IN\_BUSINESS
38. Ranco, G., Aleksovski, D., Caldarelli, G., Grčar, M., & Mozetič, I. (2015). The Effects of Twitter Sentiment on Stock Price Returns. *PLOS ONE*, *10*(9), e0138441. https://doi.org/10.1371/JOURNAL.PONE.0138441
39. Robertson, S. (n.d.). *Understanding Inverse Document Frequency: On theoretical arguments for IDF*. *5*, 503–520.
40. Salloum, S. A., Al-Emran, M., Monem, A. A., & Shaalan, K. (2018). *Using Text Mining Techniques for Extracting Information from Research Articles* (pp. 373–397). https://doi.org/10.1007/978-3-319-67056-0\_18
41. Sarker, I. H. (2021). Machine Learning: Algorithms, Real-World Applications and Research Directions. *SN Computer Science*, *2*(3), 1–21. https://doi.org/10.1007/S42979-021-00592-X/FIGURES/11
42. Schuster, M., & Paliwal, K. K. (1997). Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, *45*(11), 2673–2681. https://doi.org/10.1109/78.650093
43. *SENTIMENTAL ANALYSIS USING VADER. interpretation and classification of… | by Aditya Beri | Towards Data Science*. (n.d.). Retrieved January 10, 2023, from https://towardsdatascience.com/sentimental-analysis-using-vader-a3415fef7664
44. Shah, S., Kumar, K., & Sarvananguru, Ra. K. (2016). Sentimental Analysis of Twitter Data using Classifier Algorithms. *International Journal of Electrical and Computer Engineering (IJECE)*, *6*(1), 357. https://doi.org/10.11591/IJECE.V6I1.PP357-366
45. *Softmax Function Definition | DeepAI*. (n.d.). Retrieved January 9, 2023, from https://deepai.org/machine-learning-glossary-and-terms/softmax-layer
46. Soliman, H. (2020). Random Forest Based Searching Approach for RDF. *IEEE Access*, *8*, 50367–50376. https://doi.org/10.1109/ACCESS.2020.2980155
47. *TextBlob: Simplified Text Processing — TextBlob 0.16.0 documentation*. (n.d.). Retrieved January 10, 2023, from https://textblob.readthedocs.io/en/dev/
48. *Timeseries data preprocessing*. (n.d.). Retrieved January 8, 2023, from https://keras.io/api/preprocessing/timeseries/#pad\_sequences
49. *Understanding TF-IDF for Machine Learning | Capital One*. (n.d.). Retrieved January 10, 2023, from https://www.capitalone.com/tech/machine-learning/understanding-tf-idf/
50. Yeturu, K. (2020). Machine learning algorithms, applications, and practices in data science. *Handbook of Statistics*, *43*, 81–206. https://doi.org/10.1016/bs.host.2020.01.002
51. Zhang, D., Liu, Z. C., Hu, X. D., & Ying, X. (2019). *An Overview of Overfitting and its Solutions You may also like Traffic management of intersection with more than four road segments R Malia, N Fadhly and S Sugiarto-An improved multi-layer stopper in a GaN-based laser diode An Overview of Overfitting and its Solutions*. 22022. https://doi.org/10.1088/1742-6596/1168/2/022022