

Master's Thesis

---

Classification of Data based on  
Sentiment of Text using Machine  
Learning & Deep Learning

---

by

Rozana Alam

Under supervision of

Prof. Dr. rer.nat. Matthias F. Wagner

Prof. Dr. Eicke Godehardt

A thesis submitted in the partial fulfilment for the degree of Master of Science  
in High Integrity Systems

Frankfurt University of Applied Sciences

Department of Computer Science and Engineering

September 23<sup>th</sup>, 2020



# Declaration

I, Rozana Alam, declare that this thesis titled, "Classification of data based on Sentiment of text using Machine Learning & Deep Learning" and the work presented in it are my own and I am the sole author and composer of my thesis and that no other sources or learning aids, other than those listed, have been used. Furthermore, I declare that I have acknowledged the work of others by providing detailed references of said work.

I hereby also declare, that my Thesis has not been prepared for another examination or assignment, either wholly or excerpts thereof.

Frankfurt, 22/09/2020

Place, Date

*Rozana Alam*

Signature

# Abstract

The main goal of this thesis project is the classification of data based on the sentiment of text by comparing the accuracies of different Machine Learning and Deep learning models and finding the best model that has the highest accuracy compared to models i.e. traditional machine learning and advanced level deep learning models. The data source is Twitter social platform which is in English language and we have collected it from Kaggle [1].

Exploratory data analysis(EDA) estimations show the 40.5% data is neutral, 31.2% is positive and 28.3% is negative. Different types of Machine Learning models have different accuracies i.e. Naïve Bayes has 78.31% , Logistic Regression has 78.41% and Decision Tree has 75.93% accuracy. In any case, the confusion matrix let us realize that choice of decision tree offers us more right response for our dataset.

Simple Neural network has achieved accuracy around 49.65%, Convolutional neural network has around 49% accuracy, while recursive neural network(LSTM) has only 32% accuracy for the sentiment analysis with Deep learning models. We also observed that models were not overfitted to our dataset. We used fine-tuning Bert\_base\_uncased model which has given us the best accuracy around 89% for our dataset, while all other models considered 27481 rows, we needed to set up a little chunks of dataset of 10000 rows for Bert model. For Bert model we considered only 7481 rows as the dataframe. Finally, it is concluded that the Bert\_base\_uncased model provides the best accuracy for our dataset though it has some limitations.

# Acknowledgments

I would like to express my gratitude to my supervisor Prof. Dr. rer.nat. Matthias F. Wagner for his useful comments, remarks and engagement through the learning process of this master thesis. My second reader, Prof. Dr. Eicke Godehardt, also made many helpful suggestions. Furthermore, I would like to thank to my friend Muhammad Adnan Khan for introducing me to the topic as well for the support on the way. Special thanks to my classmate Makhdoom Saleem Abbas. Also I would like to thank my loved ones who have supported me throughout the entire process, both by keeping me harmonious and helping me putting pieces together. I would like to thank to my husband Sayed Alam, my parents and all of my teachers, family members and all of my friends who have been an inspiration throughout my life. They have always supported my dreams and aspirations. I'd like to thank them for all they are, and all they have done for me and thank to my only one brother for his encouragement throughout the tenure. Special thanks to my international officer Jürgen Schwan who selected me for the DAAD scholarship for summer 2020 for this thesis purpose and always support me in Germany like my father in every difficult situation I face during my higher study in Germany. ...

# Contents

<b>Declaration</b>	<b>i</b>
<b>Abstract</b>	<b>ii</b>
<b>Acknowledgments</b>	<b>iii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Overview . . . . .	2
1.2 Motivation . . . . .	2
1.3 Related work . . . . .	3
1.3.1 Sentiment analysis on tweeter data using ML in python	4
1.3.2 Naïve Bayes and Artificial Neural Network model . . . .	4
1.3.3 Personalized Recommender System using ML . . . . .	5
1.3.4 Twitter Sentiment Analysis . . . . .	6
1.3.5 Deep Learning-based Classification . . . . .	6
1.3.6 Sentiment Analysis Based on Deep Learning . . . . .	8
1.3.7 Deep Learning based Multi-Label classification . . . . .	9
1.3.8 Analyzing Twitter for Social TV . . . . .	10
1.3.9 Sentiment Analysis of multi-class text data classification	11
1.4 Comparison of related work with our thesis project . . . . .	12
1.5 Project Plan . . . . .	13

1.6	Organization of the Thesis . . . . .	13
<b>2</b>	<b>Business Knowledge and proposed solution</b>	<b>15</b>
2.1	Natural Language Processing . . . . .	15
2.2	Sentiment Analysis . . . . .	15
2.2.1	Types of Sentiment Analysis . . . . .	15
2.3	Applications of Sentiment Analysis . . . . .	16
2.4	Problem Statement . . . . .	16
2.5	Proposed Solution . . . . .	17
2.6	Objectives . . . . .	17
2.7	Process model of sentiment analysis . . . . .	18
<b>3</b>	<b>Theoretical &amp; Technical Overview</b>	<b>21</b>
3.1	AI, Machine Learning and Deep Learning . . . . .	21
3.1.1	Artificial intelligence . . . . .	22
3.1.2	Machine Learning . . . . .	22
3.2	Difference to classical programming . . . . .	22
3.2.1	Categories of Machine Learning Models . . . . .	24
3.2.2	Deep Learning . . . . .	26
3.3	Selected Methods for the experiment on our Data . . . . .	27
3.3.1	Text classification with Machine Learning models . . . . .	27
3.3.1.1	Naïve Bayes classifier . . . . .	27
3.3.1.2	Logistic Regression . . . . .	28
3.3.1.3	Decision Tree . . . . .	30
3.3.2	Text classification with Deep Learning models . . . . .	31
3.3.2.1	Simple Neural Network . . . . .	31

3.3.2.2	Convolutional Neural Network . . . . .	32
3.3.2.3	Recurrent Neural Network(Long Sort-term Mem- ory (LSTM)) . . . . .	35
3.3.2.4	BERT model . . . . .	38
3.3.2.5	Generative Pre-trained Transformer 3 (GPT-3)	41
3.4	Forward Propagation . . . . .	42
3.5	Back Propagation . . . . .	42
3.6	Confusion Matrix . . . . .	43
3.7	Overview of Programming Languages . . . . .	43
3.7.1	R Programming Language . . . . .	43
3.7.2	Python . . . . .	43
3.7.3	Scala . . . . .	44
3.8	Anaconda . . . . .	44
3.9	Google Colab . . . . .	44
3.10	Libraries of Python . . . . .	45
3.10.1	NLTK . . . . .	45
3.10.2	Pandas . . . . .	45
3.10.3	NumPy . . . . .	46
3.10.4	Sklearn . . . . .	46
3.10.5	Regular Expression (RegEx) . . . . .	46
3.10.6	Keras TensorFlow . . . . .	47
3.11	Visualization Libraries . . . . .	47
<b>4</b>	<b>Data Collection And Data Investigation</b>	<b>49</b>
4.1	Unstructured Data and Structured data . . . . .	49



4.2	Data collection from source . . . . .	50
4.3	Loading the Data . . . . .	51
4.4	Data Understanding and Exploratory Data Analysis . . . . .	52
4.5	Visualization of dataset . . . . .	53
4.6	Analysis of 'text' and 'selected_text' Feature . . . . .	56
4.7	Exploring the selected_text column . . . . .	58
4.8	Exploring the 'text' column . . . . .	59
4.9	Summary . . . . .	61
<b>5</b>	<b>Implementation</b>	<b>62</b>
5.1	Sentiment Analysis with Machine Learning Models . . . . .	62
5.1.1	Data Preparation for Machine Learning . . . . .	62
5.1.1.1	Text Data Pre-processing . . . . .	64
5.1.1.2	Mapping of Response vector . . . . .	66
5.1.2	Splitting into Training and Test dataset . . . . .	66
5.1.3	Feature Engineering . . . . .	67
5.1.3.1	Import . . . . .	67
5.1.3.2	Instantiate . . . . .	68
5.1.3.3	Fit . . . . .	68
5.1.3.4	Transform . . . . .	69
5.1.4	Decision for the best Machine Learning model . . . . .	70
5.1.4.1	Naïve Bayes Model . . . . .	71
5.1.4.2	Logistic Regression Model . . . . .	73
5.1.4.3	Decision Tree Model . . . . .	74

5.2	Sentiment Analysis with Deep Learning Models . . . . .	75
5.2.1	Pre-process Data for Simple NN, CNN and RNN (LSTM)	75
5.2.2	The sentiment analysis with Simple Neural Network . . .	82
5.2.3	The sentiment analysis with Convolutional Neural Network	85
5.2.4	Sentiment Analysis with Recurrent Neural Network model	89
5.3	Sentiment Analysis with BERT Models . . . . .	92
5.3.1	Exploratory Data Analysis and Preprocessing . . . . .	93
5.3.2	Training/Validation Split . . . . .	96
5.3.3	Loading Tokenizer and Encoding our Data . . . . .	97
5.3.4	Setting up BERT Pre-trained Model . . . . .	99
5.3.5	Creating Data Loaders . . . . .	100
5.3.6	Setting Up Optimizer and Scheduler . . . . .	101
5.3.7	Defining our Performance Metrics . . . . .	102
5.3.8	Creating our Training Loop . . . . .	102
5.3.9	Results of Bert Model . . . . .	105
5.4	Result Analysis for all models . . . . .	107
<b>6</b>	<b>Conclusion</b>	<b>109</b>
6.0.1	Limitations . . . . .	109
6.0.2	Problem Faced . . . . .	109
6.0.3	Future Plan . . . . .	110
	<b>Bibliography</b>	<b>111</b>

## List of Figures

1	The ranking of different models[2] . . . . .	7
2	Project plan . . . . .	13
3	Process model . . . . .	19
4	AI, ML and Deep Learning relationship[3] . . . . .	21
5	ML Programming[3] . . . . .	23
6	Types of Machine Learning Algorithms . . . . .	23
7	Neural Networks . . . . .	26
8	Logistic Curve, The values of y cannot be less than 0 or greater than 1 . . . . .	29
9	Decision Tree . . . . .	30
10	Perceptron . . . . .	31
11	The architecture of a sample CNN model for text classification	34
12	Max Polling Layer . . . . .	35
13	Structure of the LSTM cell . . . . .	37
14	BERT model . . . . .	39
15	BERT input representation . . . . .	40
16	First 10 rows of S_df . . . . .	53
17	Data type of S_df . . . . .	53

18	S_df visualization . . . . .	54
19	Selected_text WordCloud Visulalization . . . . .	55
20	Value counts for 'text' column . . . . .	56
21	DataFrame value counts for 'selected_text' column . . . . .	57
22	Flow of Data Preparation for ML . . . . .	63
23	Check the dataset after pre-processing . . . . .	65
24	Flow of Defining Training and Test Data for ML . . . . .	66
25	Flow of Feature Engineering steps . . . . .	68
26	Feature matrix for X_train data . . . . .	70
27	Feature metrix for X_test . . . . .	70
28	Steps for best Machine Learning model . . . . .	71
29	Prediction by Naïve Bayes Model . . . . .	72
30	Accuracy for Naïve Bayes Model . . . . .	72
31	Confusion matrix for Naïve Bayes . . . . .	73
32	Prediction by Logistic Regression Model . . . . .	73
33	Accuracy for Logistic Regression . . . . .	74
34	Confusion matrix for Logistic Regression . . . . .	74
35	Prediction by Decision Tree Model . . . . .	74
36	Accuracy for Decision Tree Model . . . . .	75
37	Confusion matrix for Decision Tree . . . . .	75
38	Flow of preprocess Data for Simple NN, CNN and RNN (LSTM)	76
39	New dataframe df . . . . .	77
40	X list . . . . .	79
41	Embedding matrix . . . . .	82

42	Flow of Deep Learning best model . . . . .	82
43	Simple Neural Network model summery . . . . .	84
44	Output plot for loss and accuracy between the training and test sets of Simple neural network . . . . .	86
45	Summary of our CNN model . . . . .	87
46	The resulting plot of CNN model . . . . .	89
47	LSTM model summery . . . . .	90
48	Result Plot for the LSTM model . . . . .	91
49	Flowchart for BERT model . . . . .	93
50	'df' DataFrame . . . . .	94
51	'df' DataFrame details . . . . .	97

# List of Tables

1	Result analysis of Related works to compare with our Results	
	Table 2 . . . . .	12
2	Result analysis of all the model . . . . .	108

# Listings

4.1	Import libraries for Exploratory Data Analysis (EDA) . . . . .	52
4.2	Loading data in Jupyter notebook . . . . .	52
4.3	Code for S_df visualization . . . . .	54
4.4	Code for the WordCloud Visualization . . . . .	55
4.5	DataFrame value counts for 'text' column . . . . .	56
4.6	DataFrame value counts for 'selected_text' column . . . . .	57
4.7	Value counts for 'selected_text' . . . . .	58
4.8	Observe neutral sentiment of 'selected_text' column . . . . .	58
4.9	Observe negative 'sentiment' of 'selected_text' column . . . . .	58
4.10	Observe positive sentiment of 'selected_text' column . . . . .	59
4.11	DataFrame value counts for 'text' column . . . . .	59
4.12	Observed neutral sentiment for the 'text' column . . . . .	59
4.13	Observed positive sentiment for the 'text' column . . . . .	60
4.14	Observed negative sentiment for the 'text' column . . . . .	60
5.1	Import libraries and data loading . . . . .	63
5.2	Code for the missing value remove data . . . . .	63
5.3	Text preprocessing helper function . . . . .	64
5.4	Data preprocessing function code . . . . .	64
5.5	Applying cleaning function . . . . .	65

5.6	Mapping sentiment Data . . . . .	66
5.7	Define X and y . . . . .	66
5.8	Splitting into test and Training dataset . . . . .	67
5.9	Instantiating the Vectorizer . . . . .	68
5.10	Fitting for the training data . . . . .	68
5.11	The learned vocabulary . . . . .	69
5.12	Transform training data . . . . .	69
5.13	Transform test data . . . . .	70
5.14	Importing and Instantiating . . . . .	71
5.15	model training using training data . . . . .	72
5.16	Prediction by Naïve Bayes Model . . . . .	72
5.17	Confusion matrix for Naïve Bayes Model . . . . .	72
5.18	Importing and Instantiating Logistic Regression . . . . .	73
5.19	Model Training . . . . .	73
5.20	Prediction by Logistic Regression . . . . .	73
5.21	Accuracy for Logistic Regression . . . . .	73
5.22	Confusion matrix for Logistic Regression Model . . . . .	74
5.23	Prediction by the Decision Tree model . . . . .	74
5.24	Observed accuracy for the Decision Tree model . . . . .	74
5.25	Confusion matrix for Decision Tree Model . . . . .	75
5.26	Importing required libraries for Deep Learning model . . . . .	76
5.27	Create new dataframe 'df' . . . . .	77
5.28	Data cleaning function . . . . .	78
5.29	Function for remove tags from text . . . . .	78
5.30	'selected_text' converted to X list . . . . .	78



5.31	Define y . . . . .	79
5.32	Splitting training and test dataset . . . . .	79
5.33	Tokenize . . . . .	80
5.34	Padding process . . . . .	80
5.35	Code for GloVe embeddings . . . . .	81
5.36	Code for embedding matrix . . . . .	81
5.37	Code for Simple Neural Network model . . . . .	82
5.38	Code for Simple NN model summery . . . . .	83
5.39	Code for Simple NN model training . . . . .	84
5.40	Code for Simple NN model Evaluate . . . . .	84
5.41	Plot of the loss and accuracy differences for training and test sets	85
5.42	Code for create the convolutional model . . . . .	86
5.43	Code for fit and evaluate CNN methods . . . . .	88
5.44	The results of evaluate CNN methods . . . . .	88
5.45	The resulting plot code of CNN model . . . . .	88
5.46	The code of RNN(LSTM) model . . . . .	90
5.47	RNN(LSTM) model . . . . .	91
5.48	RNN(LSTM) model results . . . . .	91
5.49	Import libraries for Bert model . . . . .	93
5.50	Create small chunk from large data . . . . .	94
5.51	Code for DataFrame 'df' . . . . .	94
5.52	Number of observations . . . . .	94
5.53	Observations . . . . .	95
5.54	Check the unique labels . . . . .	95
5.55	Mapping data . . . . .	96

5.56	Label replace with numerical value . . . . .	96
5.57	Training/Validation Split . . . . .	96
5.58	DataFrame that all the data in the list of X_train will retrieval	96
5.59	Installing hugging face libraries . . . . .	97
5.60	Code for loading Tokenizer . . . . .	97
5.61	Code for Encoded the train dataset . . . . .	98
5.62	Code for converted the converted numerical data into Tensor data	99
5.63	Chunk size of DataFrame . . . . .	99
5.64	Code for setting up BERT Pretrained Model . . . . .	100
5.65	Code for creating Data Loaders . . . . .	100
5.66	code for setting Up Optimizer . . . . .	101
5.67	Code for setting Up Scheduler . . . . .	101
5.68	Code for define our performance rate . . . . .	102
5.69	Code for predict accuracy . . . . .	102
5.70	Code for setting random seeds . . . . .	102
5.71	Code to detect device . . . . .	103
5.72	Code for Loading and Evaluation of the model . . . . .	103
5.73	New model upload to achieve predicted result . . . . .	104
5.74	Results improvement per epoch . . . . .	105
5.75	Upload the pre trained_model . . . . .	106
5.76	Final result prediction . . . . .	106
5.77	Predicted accuracy for different sentiment . . . . .	106

# 1 Introduction

The main goal of this thesis is to experiment with Machine Learning and Deep learning models for the text classification of a dataset, which a machine can use to find a model that fits the data as best as possible. The challenge is to find a model which performs best for the given dataset.

People post their opinions for different topics, discuss current issues, complains and express their sentiments for products they use in daily life, etc. Micro blogging websites like twitter are one of the most important sources of information[4] detecting and analyzing the emotions expressed in social media, content benefits many applications in commerce, public health, social welfare, etc[5].

Each second tweets are publishing some opinions. It is difficult to say how much the sentiment of a particular tweet will influence a person, a brand for being viral, or destroy the profit of a company because of its negative sentiment[1].

"This novel is amazing." [sentiment: positive]. Manually tracking such information is a time-consuming task and requires a lot of resources as well. The process is repeated several times to avoid human error. Moreover, a specific skill set is needed like who has domain knowledge.

Natural Language Processing (NLP) allows computers to process and analyze large amounts of natural language data. Where text as input data, Word counts track the important words in a text. One of the most popular applications of NLP in the real world is tweet sentiment extraction.

## 1.1 Overview

The main goal of this thesis project is the classification of data based on the sentiment of text using machine learning or deep learning model that has the best accuracy compared to all of the models, i.e traditional machine learning and advanced level deep learning models.

The dataset is collected from Twitter social media platform which has a titled Sentiment Analysis. Sentiments in text tweets with existing sentiment labels, used here under creative commons attribution 4.0. international licence[1]. Collected data contains opinions in the form of reviews and comments. These opinions contain sentiments.

The data will be classified into three categories sentiment (i.e. positive, neutral, negative) using machine learning or deep Learning model. Higher accuracy can be achieved if the model is well trained by using a ‘labeled training data’. A well-trained model can be built by cleaning the dataset.

Different kinds of machine learning algorithms for analyzing the sentiments can be used i.e. Naïve Bayes, Logistic Regression, Random Forest, Decision Tree , and Deep Learning algorithms such as simple Neural network, Convolutional Neural network, Recursive neural network(LSTM), and Fine tuning Bert\_base\_uncased model.

In the proposed solution, the best machine learning model or deep learning model will be selected out of several models by comparing the accuracy of those models. For this purpose, the training dataset will be split into two parts i.e. training part and a validation part. Finally, the best model will be used on test data to get the results.

## 1.2 Motivation

Text Classification is a very diverse field of study. Sentiment analysis is an emerging field in the decision-making process and is developing fast and analyze the sentiments, on a topic which are extracted from the Twitter and determine

its nature i.e. positive, negative and neutral of the defined topics[4].

Sorting data manually at scale of thousands of tweets, reviews and customer support conversations is time-consuming which is not easy to handle and needs domain knowledge as well. Sentiment analysis took that responsibility of businesses process and take care of those unstructured data and by using a Machine Learning model we can check the sentiment of unstructured data and classify them easily.

Sentiment analysis mentions the use of NLP, text analysis, and biometrics to systematically identify, computational linguistics, extract, quantify etc. Day by day this field is opening new branches even the sentiment analysis is used by an abundance of social media monitoring tools.

3.8 billion people use social media now a days[6].They are sharing their opinions every second for every reason even for the simplest matter of their life. Companies are curious about the opinions of their customers about their products.

A lot of research work has already been done related to binary classification and multi-class classification. Sentiment analysis for more than two classes is crucial in current times.

### **1.3 Related work**

The purpose of this study is to inspect different convergence and methods in sentiment analysis that can be taken as a reference in future empirical studies. We have focused on key aspects of research, such as technical challenges, datasets, results of sentiment analysis, and the methods proposed in each study, and their application domains. The different datasets show different kinds of results on the same algorithm. Related works are based on Machine Learning and Deep Learning mainly.

### **1.3.1 Sentiment analysis on tweeter data using ML in python**

In this paper[4], text analysis techniques are used to dig into some of the data in a series of posts focusing on different trends of tweets languages, tweets volumes on twitter. Experimental evaluations show that the proposed machine learning classifiers are efficient and performed better in terms of accuracy and time. The proposed algorithm is implemented in Python.

Naïve Bayes methods and Neural Network are important tools for classification. There are 2 phases. Training and testing phase. In the training phase, the positive and negative comments are trained and assigned weights. The main purpose of the training phase is to create a dictionary of positive comments. In the next phase, testing is done based on the weighted dictionary. The artificial neural network is trained with labeled data to produce meaningful output. This process by which neural networks learn from labeled data is called as back propagation[4].

### **1.3.2 Naïve Bayes and Artificial Neural Network model**

In this paper[7], microblogging on twitter gains the interest of data researchers as there is an immense scope of mining and analyzing the huge amount of unstructured data in several ways.

The performance of these algorithms has been compared based on certain metrics. Since the machine learning algorithms have been performed on an unexplored dataset, language barriers to these algorithms have also been identified in terms of future scope and current feasibility of the algorithms. The analysis has been performed using classification algorithms – Naïve Bayes, Support Vector Machine, and Random Forest. This experimental work has been executed in Python and Excel has been used to further evaluate and plot some of the results. Since the sentiment of the tweets cannot be known, the test set has been manually prepared (i.e. not collected from any source) to prevent any errors in evaluating the accuracy and precision of the models[7].

K-nearest neighbor classifier classes not necessarily required to be linearly

distinguishable but in this classifier, it is time-consuming to find the nearest neighbors if the dataset is huge.

In the SVM classifier, the accuracy of results can be high, but it is complex and requires more space and time in both training and testing.

In the ANN classifier, it works very well with only a few parameters to adjust, but the processing time can be high if the neural network is large. In this paper, probabilistic Naïve Bayes has best accuracy 88% which has simple implementation, excellent efficiency and classification rate. All the algorithms have given false results majorly on nonEnglish (Hindi, Urdu) languages[7].

### **1.3.3 Personalized Recommender System using ML**

A need for personalized Recommender system, in this paper [8], they propose a social framework, which extracts user's reviews, comments of restaurants and points of interest such as occasion and places, to find-out the rank suggestions based on customers favor. For the optimizing search query results they used the Machine Learning and Sentiment Analysis based techniques which helps customer to find-out easily their expected data.

For the rule-based sentiment analysis, the python Natural Language Toolkit package was used for natural language processing. In-built tokenizers were used to convert the sentences to tokens. The two approaches of rule-based sentiment analysis do not require a training set and can directly be tested.

For machine learning-based techniques i.e. Naïve Bayes classifier, Support Vector Machines and Random Forest classifier are used to classify text dataset. The Cornell movie reviews dataset is used which contains a thousand positive reviews and thousand negative reviews. About seven hundred positives and seven hundred negative reviews were used as the training set. Sentiment Analysis using Machine Learning techniques for training samples dataset is 1400.

In this project[8], a framework was implemented which enabled the user to choose from only those data, which he likes and not waste time on the data

irrelevant to his needs. This enables faster services, better results and also the removal of irrelevant data and calculated the binary class sentiment analysis with 72% accuracy for rule-based model and machine learning model .

#### **1.3.4 Twitter Sentiment Analysis**

This paper[9] reports on the design of sentiment analysis, extracting a vast amount of tweets which is a prototyping. Developing a program for sentiment analysis is an approach to be used to computationally measure customers' perceptions. Results classify customers' perspective via tweets into positive and negative, which is represented in a pie chart which is representing each percentage positive, negative, and null sentiment hashtags in a different color and HTML page. However, the program had planned to develop on a web application system, but due to limitations, they were not able to do that. Two approaches for extracting sentiment automatically which are the lexicon-based approach and machine-learning-based approach i.e. supervised classification approaches used where sentiment detection is framed as a binary which is positive and negative.

NLP, Case-Based Reasoning (CBR), Artificial Neural Network (ANN), Support Vector Machine(SVM) methods are used. Finally, Twitter sentiment analysis is developed to analyze customers' perspectives toward the critical to success in the marketplace[9].

#### **1.3.5 Deep Learning-based Classification**

In this paper[2], it is shown that the Deep learning based models has better performance compared to classical machine learning-based approaches in various text classification tasks, including sentiment analysis, news categorization, question answering and natural language inference. A detailed review of more than 150 deep learning-based models for text classification developed in recent years has been shown and discuss their technical contributions, similarities, and strengths. The summary of more than 40 popular datasets widely used for text



classification. Analysis of the performance of different deep learning models on popular benchmarks, and future research directions are clearly explained in this newly realized paper in April 2020.

Method	IMDB	SST-2	Amazon-2	Amazon-5	Yelp-2	Yelp-5
<i>Naive Bayes</i>	-	81.8	-	-	-	-
<i>LDA</i>	67.4	-	-	-	-	-
<i>BoW+SVM</i>	87.8	-	-	-	-	-
<i>tf.<math>\Delta</math> idf</i>	88.1	-	-	-	-	-
Char-level CNN	-	-	94.49	59.46	95.12	62.05
Deep Pyramid CNN	-	84.46	96.68	65.82	97.36	69.40
ULMFiT	95.4	-	-	-	97.84	70.02
BLSTM-2DCNN	-	89.5	-	-	-	-
Neural Semantic Encoder	-	89.7	-	-	-	-
BCN+Char+CoVe	91.8	90.3	-	-	-	-
GLUE ELMo baseline	-	90.4	-	-	-	-
BERT ELMo baseline	-	90.4	-	-	-	-
CCCapsNet	-	-	94.96	60.95	96.48	65.85
Virtual adversarial training	94.1	-	-	-	-	-
Block-sparse LSTM	94.99	93.2	-	-	96.73	-
BERT-base	95.63	93.5	96.04	61.6	98.08	70.58
BERT-large	95.79	94.9	96.07	62.2	98.19	71.38
ALBERT	-	95.2	-	-	-	-
Multi-Task DNN	83.2	95.6	-	-	-	-
Snorkel MeTaL	-	96.2	-	-	-	-
BERT Finetune + UDA	95.8	-	96.5	62.88	97.95	62.92
RoBERTa (+additional data)	-	96.4	-	-	-	-
XLNet-Large (ensemble)	96.21	96.8	97.6	67.74	98.45	72.2

**Figure 1:** The ranking of different models[2]

The automatic text classification process can be divided into three types is Rule-based methods, machine learning (data-driven) based methods, and hybrid methods. Based on their neural network architectures, such as models based on recurrent neural networks (RNNs), convolutional neural networks (CNNs), Bert model, attention, transformer , Capsule Nets, and more. The contributions of [2] this paper can be summarized as follows:

- Overview of more than 150 deep learning models proposed for text classification.

- Review on 40 plus popular text classification datasets.
- Analysis of the performance of a selected set of deep learning models on 16 popular benchmarks and there is a discussion on challenges and future directions.

Deep Learning Models like different CNN, RNN, DNN and different Bert models achieved results for text classification in detail discussed as shown in fig.1. WordCloud presentation has shown of Sentiment Analysis the datasets. Two classification tasks are defined on this dataset. Test samples for negative and positive classes i.e. binary class classification.

Finally, they also review modeling technologies that used supervised learning and unsupervised learning. Depending on different data sets they used binary and multi-class sentiment analysis. As shown in fig.1[2].

### **1.3.6 Sentiment Analysis Based on Deep Learning**

In this [10] paper the latest studies are shown the deep learning model to solve sentiment analysis problems, such as sentiment polarity. Models using term frequency-inverse document frequency (TF-IDF) and word embedding have been applied to a series of datasets.

The tweets were already labeled with the polarity of the sentiment conveyed by the person writing them negatively, positive for different datasets.

They showed word embedding after training the dataset with a minimum count of 5000 and Wordcloud package view of the topics of the dataset.

To conduct the tests, they used a GeForce GTX2070 GPU card, the Keras, and Tensorflow libraries.

DNN, CNN and RNN models were applied to perform experiments with the different datasets, to analyze the performance of those algorithms using both word embedding and TF-IDF feature extraction. Accuracy values of deep-learning models with TF-IDF and word embedding for different numbers of tweets (percentage of the dataset).

At the end, there was a huge related work has been discussed in this [10] paper and the experimental results achieved for the different models and input features.

### 1.3.7 Deep Learning based Multi-Label classification

Most previous work on sentiment and emotion analysis has only focused on single-label classification but this paper[11] works with multi-label emotion classification task, which aims to develop an automatic system to determine the existence in a text of none, one, or more out of eleven emotions: the eight categories (joy, sadness, anger, fear, trust, disgust, surprise, and anticipation) plus optimism, pessimism, and love.

The system performs better than others and showed an accuracy score of 59% on SemEval2018 Task 1, which has done for the multi-label emotion classification problem[11].

The total work is done by 4 step[11]:

- Transformation mechanism for the multi-label classification problem into a single binary classification problem.
- Developed a new transformation method i.e. attentive deep learning system, called Binary Neural Network (BNet) to solve the transformed problem.
- Evaluate the proposed system on the challenging multi-label emotion classification dataset of SemEval-2018 Task1: Affect in Tweets.
- The experimental results show that the system outperforms the state-of-the-art systems.

One possible solution is to take the attention mechanism to model the relationships between different n-gram tokens and labels[11]. Attention mechanism based on a common-sensical intuition that “attend to” a certain part when processing a large amount of information[12].

### 1.3.8 Analyzing Twitter for Social TV

This paper [5] is Sentiment Extraction for Sports. When someone watches TV they also share their opinion on social media most of the time and that is important information about the TV program and also they express how they hang on to the specific TV program. This paper said that they built a web service, SportSense, which recognizes major events in the US National Football League (NFL) games within 40 seconds after an event takes place by analyzing data retrieved from Twitter in real-time.

In this paper [5] the report support to extend SportSense to extract TV watchers sentimental reaction to major events in live broadcast sports games in real time and present the ongoing work that leverages SportSense for a social TV system. It enables TV watchers to better select interesting programs in real time and to produce personalized program summaries and enables advertisers to customize ads based on recognized events and extracted audience sentiments.

They had shown a nice Visualized results of real-time event recognition and sentiment extraction at SportSense.com site.

This paper collected tweets during the game time using the Streaming API and game keywords identified from the 2010 Super Bowl. They also collected the tweets and their metadata such as tweet source, created time, location, and device.

In a Pre-Processing step they remove the noise of tweet data to extract sentiments because the tweets are short, informal, and ungrammatical by its nature.

Finally, after necessary steps, they produced a list of sentimental words using adjectives and verbs from the list of most frequent words as seed.

In this [5] paper described the ongoing effort in extracting real-time audience sentiments by analyzing Twitter and showed that the limited vocabulary of sports games makes lexicon-based analysis methods highly effective. For several major sports games, SportSense cannot only recognize major game events in

real-time but also capture the sentiment toward each side of a game during the game.

### **1.3.9 Sentiment Analysis of multi-class text data classification**

This paper [13] describes the fifth year of the Sentiment Analysis in the Twitter dataset of multi-class text data classification Task. SemEval-2017 Task 4 continues with a next step of the subtasks of SemEval-2016 Task 4. Compared to 2016, they made two new changes which are introduced a new language, Arabic, for all subtasks and they made available information from the profiles of the Twitter users who posted the target tweets. The task continues to be very popular, with a total of 48 teams participating this year.

Task Definition[13] has 4 different categories of work have five subtasks, each has done in Arabic and English.

- They identified a tweet that has positive, negative, or neutral sentiment or a tweet and a topic, classify the sentiment that topic on a two-point scale positive and negative.
- This paper also has shown how a tweet and a topic, classify the sentiment five-point classes which are strongly positive, weakly positive, neutral, weak negative, and strong negative. They have first mapped the categorical labels to the integer values - 2, - 1, 0, 1, 2.
- A set of tweets of a topic calculate the distribution of tweets across the positive and negative classes.

20 teams were used for deep learning and neural network methods such as CNN and LSTM networks. Supervised SVM and Liblinear are also a very popular method and other teams used classifiers such as Maximum Entropy, Logistic Regression, Random Forest, Naïve Bayes classifier, and Conditional Random Fields. The topics included many entities such as named entities, geopolitical entities and many other entities.

Result analysis of Related works		
Model Type	Model name	Accuracy
Machine Learning model	Naïve Base Algorithm[7]	88%
	Support Vector machine [7]	81%
	Random Forest[7]	63%
Deep Learning Model	Deep Pyramid CNN (Amazon 2)[2]	94.68%
	BLSTM-(SST2)[2]	89.5%
Bert model	Bert base(IMBD) [2]	95.63%
Multiclass Classification[11]	eight category	59%

**Table 1:** Result analysis of Related works to compare with our Results Table 2

## 1.4 Comparison of related work with our thesis project

The WordCloud data visualization for our dataset will be shown in Chapter 4, as like as the paper [2].

Compared to this paper[11], eight categories sentiments, our dataset have three categories i.e. positive, negative and neutral sentiments. So we have to do multiclass classification based on three sentiment analysis as shown in Chapter 2.

The paper[13] has first shown, how to map labels to five categorical and we also going to map labels like that but as because we have three sentiments we will label our data into 0,1,2 integer values and will shown in Chapter5 the multiclass classifications and compare the models result based on accuracy and we will shown the visual representation of dataset to understand the data as shown in the paper[2].

Similar to this paper[10], we will also use frequency-inverse document frequency (TF-IDF) for the traditional classification machine learning model and word embedding for the deep learning models for our datasets in the data pre-processing steps in chapter 5.

In chapter 5, Table.1 shows some of the ML and Deep learning models results which will compare with Table.2 which is our achieved result.

## 1.5 Project Plan



Figure 2: project plan

The project starting date was in 31/4/2020 and the final submission date is 1/10/2020. The overall project plan is discussed in the Gantt chart Project plan as shown in fig.2. After each phase of work, there was a continuous report submission and presentation online because of COVID -19 pandemic.

## 1.6 Organization of the Thesis

This thesis has been organized into 6 chapters.

**Chapter 1** highlights the motivation, objectives discussion of the related work. This chapter covers the domain of the thesis goals it is trying to achieve and the direction it will take to pursue these objectives. The readers will also be able to identify the problem area and possible solutions.

**Chapter 2** will highlight the problems and solutions of related work. The reader will be able to get a detailed discussion of business knowledge of sentiment analysis. A solution will be proposed by the end of this chapter.

**Chapter 3** will cover the discussion about some programming languages

used for Machine Learning and Deep Learning models. Moreover, it will also describe all the techniques used to solve the sentiment analysis problem. This chapter will also precisely describe the machine learning algorithms i.e. Naïve Bayse, Logistic Regression, Decision Tree and deep learning models i.e. NN, CNN, LSTM and finally Bert models.

**Chapter 4** will discuss the data collection process and visual Exploratory Data Analysis (EDA) of Dataset.

**Chapter 5** will describe the development of NLP model that will be used for classification of text data based on the sentiment of text. Finally achieved results will be analyzed.

**Chapter 6** will conclude our thesis by analyzing our achievements and will discuss the limitations, problems faced and possibility for further research and development.



## 2 Business Knowledge and proposed solution

### 2.1 Natural Language Processing

Natural language processing (NLP) is field of artificial intelligence. NLP is the study of understanding the human language in a smart and useful way.

### 2.2 Sentiment Analysis

Sentiment analysis is a popular branch of text classification, which aims to analyze people's opinions in textual data (such as product reviews, movie reviews, and tweets) and extract their polarity and viewpoint[2].

Sentiment Analysis (also called 'Opinion Mining') is a field within the Natural Language Processing (NLP) that builds systems that try to identify and extract opinions within the text. Following are the useful terms in sentiment analysis:

- **Polarity:** if the speaker expresses a positive, negative or neutral opinion,
- **Subject:** the thing that is being talked about
- **Opinion holder:** the person or entity that expresses the opinion.

#### 2.2.1 Types of Sentiment Analysis

Sentiment analysis can be either a binary class classification or a multi-class classification problem[2].

- **Binary classification** is the classification of texts into two categories i.e. 1 or 0 .
- **Multi-class classification** focuses on classifying data into more than two categories

## 2.3 Applications of Sentiment Analysis

Sentiment Analysis in Business can prove a breakthrough for the complete brand revitalization.

Example: automated systems in customer support to prioritize tickets, find-out customer insights, customer trends, etc.

The applications of sentiment analysis in business cannot be overlooked. Sentiment analysis in businesses can be very helpful in predicting customer trends. Once we get acquainted with the current customer trends, strategies can easily be developed to capitalize on them. And eventually, gain a leading edge in the competition. Purchasing product or service, quality improvement in product or service, market research, recommendation systems, spam detection and policy making by government.[4]

## 2.4 Problem Statement

Following are the most commonly faced problems and risks involved in this manual work

- It would consider a huge amount of time to classify those text data manually based on sentiment analysis
- The possibility of human error cannot be neglected
- In order to avoid human error, repetition is always needed
- It needs to have domain knowledge of every tweet

- A Tweet can be contain anger, jealous or other emotions. Depend on our dataset we will classify the sentiments in three categories positive, negative or neutral.
- Once we will create some model which model will have the best accuracy for the dataset.
- Words in tweets support a positive, negative, or neutral sentiment or not that need to be decide based on Machine Learning, and Deep learning model.

## 2.5 Proposed Solution

After having a clear understanding of the problem statement, I proposed a solution depending on the dataset.

The proposed solution is based on the following strategies for sentiment analysis and mainly consists of two steps: In the first step, A dataset already collected from Kaggle. The necessary steps will be performed to make it ready for analysis on it. The dataset needs to be ready in every aspect before starting analyzing it. In the second step a solution for automating the whole procedure of will be applied which is through Natural Language Processing, Machine Learning and Deep Learning methods.

## 2.6 Objectives

This project has been divided into 2 phases. The first phase is a literature study is conducted. A literature study involves conducting studies on various sentiment analysis techniques and methods currently in use. In the second phase, application requirements and functionalities are defined before its development. Two methodologies for the work:

1. Traditional machine learning algorithms

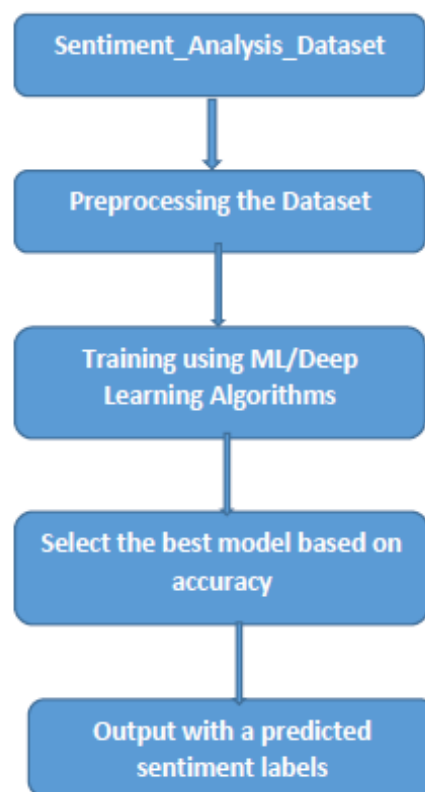
## 2. Deep learning with neural network

The main focus will be on the following objectives.

- To prepare an NLP model in python
- To build a ‘dataset’ to train the model
- To select the best machine learning or Deep Learning model to predict the results
- Compare accuracy and confusion Matrix of the results
- Give decision for the best model

### **2.7 Process model of sentiment analysis**

Figure 3 is showing the process model of sentiment analysis step by step. There will be detailed discussion in next chapters each steps of our proposed process model. Data processing steps are different for different machine learning and deep learning algorithm.



**Figure 3:** Process model

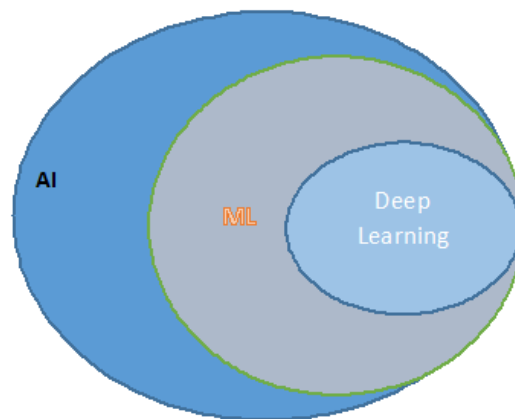


## 3 Theoretical & Technical Overview

In this chapter, machine learning and deep learning models are explained in detail and programming languages such as 'Python', 'R' and 'Scala' are also discussed.

### 3.1 AI, Machine Learning and Deep Learning

Machine Learning(ML) and Deep learning come under the roof of Artificial intelligence(AI). Fig.4 shows a relationship of Machine Learning(ML) and Deep learning with Artificial Intelligence(AI).



**Figure 4:** AI, ML, and Deep Learning relationship[3]

### 3.1.1 Artificial intelligence

Artificial intelligence(AI) is a modern concept in which computers have the ability to learn human behaviour and mimic it[14].

### 3.1.2 Machine Learning

*"Machine Learning is the field of study that gives computers the ability to learn without being explicitly programmed. - Arthur Samuel, 1959" [14]*

*"A computer program is said to learn from experience  $E$  with respect to some task  $T$  and some performance measure  $P$ , if its performance on  $T$ , as measured by  $P$ , improves with experience  $E$ . — Tom Mitchell, 1997" [14]*

Machine learning applications are as follows :

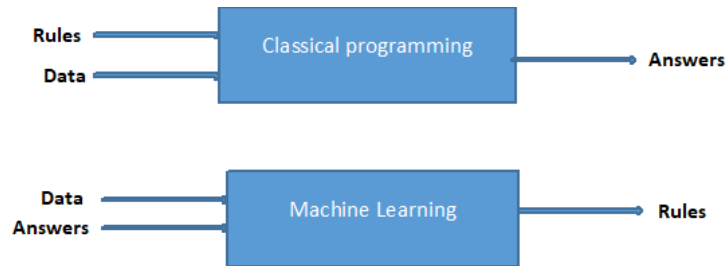
- Make predictions
- Classify data
- Give recommendations on their own etc.
- Sentiment analysis of text data
- Pattern recognition
- Face recognition: Pose, lighting, occlusion (glasses, beard), make-up, hair style
- Medical diagnosis: from symptoms to illnesses
- Web advertising: Predict if a user clicks on an ad on the Internet.
- Recommendation systems etc

## 3.2 Difference to classical programming

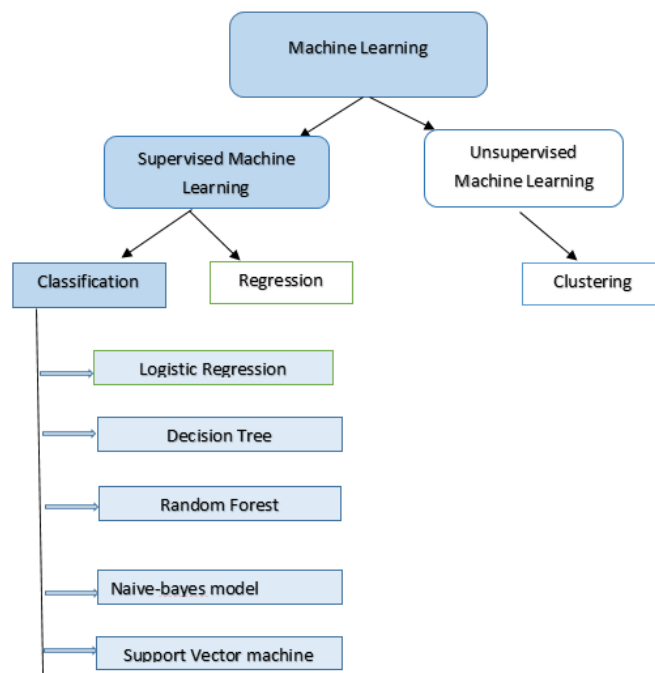
In classical programming, the example of type AI, humans input rules as a program and data to be processed according to these rules and achieved output



answers. With machine learning, humans input data as well as the answers expected from the data and out come the rules. The rules we get as output are generally used later to new data to produce expected answers.[3]



**Figure 5:** ML Programming [3]



**Figure 6:** Types of Machine Learning Algorithms

The fig.6 shows the classification of supervised and unsupervised machine learning models.

### 3.2.1 Categories of Machine Learning Models

Machine Learning problems can be classified mainly into four types. The categories are the following:

- Supervised Learning
- Unsupervised Learning
- Semi-Supervised Learning
- Reinforcement Learning

#### Supervised Machine Learning

*"Supervised learning algorithms try to model relationships and dependencies between the target output and the input features such that we can predict the output values for new data based on those relationships which it learned from the training data sets,"[15].*

Most of the machine learning models use supervised learning algorithms i.e. qualitative(Classification) and quantitative(Regression).

**Regression Analysis** has the goal is to predict continuous values i.e. employee salary, cost of house [15].

**Classification of Machine Learning Models** are used for classification problems, as shown in fig. 6. List of common classification algorithms are as follows:

- **Logistic Regression**
  - Used to estimate discrete values (True/False, Yes/No) based on given set of independent variable(s)[16].
- **Decision tree**
  - Used for classification problems. It works for both categorical and continuous dependent variables[17].

- **Random Forest**

- "Random forests function by building multiple decision trees and yielding a class which is basically, the mode of all classes." [8].

- **Naïve Bayes classifier**

- It is a classification technique based on Bayes theorem and probabilistic model [8].

- **Support Vector machine**

- "Support vector machines are binary linear classifiers. However, nonlinear classification is also possible, by mapping the inputs to higher dimensional feature spaces. These mappings are known as kernel functions. The amount of fitting on the training set can be increased by using higher order kernel functions." [8].

## **Unsupervised Machine Learning**

Unsupervised learning is a descriptive Model [15]. The goal for unsupervised learning is to model the underlying structure or distribution in the data in order to learn more about the data. Algorithms are left to their own devices to discover and present the interesting structure in the data [18]. Unsupervised learning, as we might guess, the training data is unlabeled and the system tries to learn without a teacher [14].

### **Semi supervised learning**

Semi supervised learning tasks is mixture of the above two, where we use the raw data (unlabeled data) and small amount of validated data. Here the group memberships of the unlabeled data are unknown, this data carries important information about the group parameters. [15]

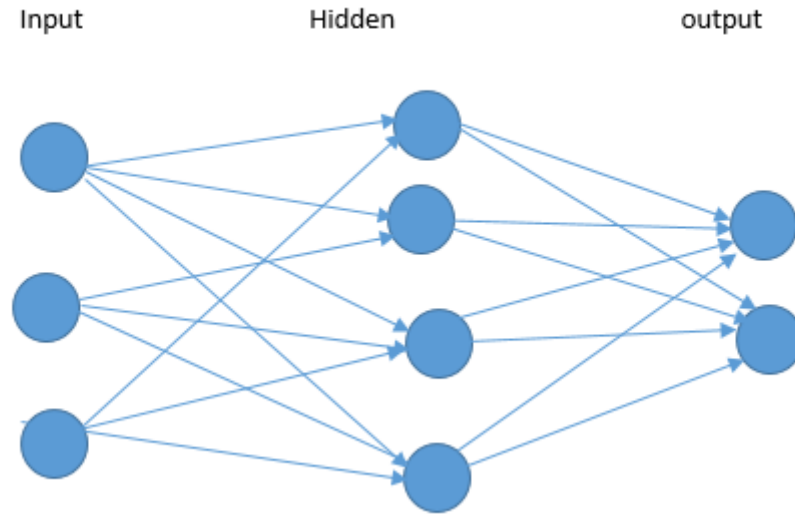
### **Reinforcement Learning**

*"Reinforcement Learning is a type of machine learning and thereby also a branch of Artificial Intelligence. It allows machines and software agents to automatically determine the ideal behavior within a specific context, in order to maximize its performance."* [15]

### 3.2.2 Deep Learning

The idea of Artificial Neural Network comes from the human biological neurons. Warren McCulloch and Walter Pitts proposed a very simple model of the biological neuron which later called as an artificial neuron. It has one or more binary (on/off) inputs and one binary output. Artificial Neural Network is called a Deep Neural Network when ANN has two or more hidden layers[14]. Hidden layers reside in-between input and output layers which perform non-linear transformations of the inputs entered into the network[19].

A Neural Network classifier with the appropriate network structure can handle the correlation or dependence between the input variables. Neural Networks learn from labeled is what is called the back propagation process.[4]



**Figure 7:** Neural Networks[4]

As shown in figure 7[4] feed forward layer is made up of nodes and edges. Each node is a part of a layer and each node in a layer points to every node in the next layer. Input is fed into the first layer called the input layer, in which the input follows the edges to the nodes in the next layer until it reaches the output layer[4].

"Deep learning is the part of a large family of machine learning methods based on artificial neural networks with representation learning where Learning can be supervised, semi-supervised or unsupervised"[6].

"There is now a huge quantity of data available to train neural networks, and Artificial Neural Networks frequently outperform other Machine Learning techniques on very large and complex problems"[14].

### **3.3 Selected Methods for the experiment on our Data**

Following machine learning and deep learning models will be experiment on our dataset to achieve the best model:

#### **Text classification with Machine Learning models**

- Naïve Bayes classifier
- Logistic Regression
- Decision Tree

#### **Text classification with Deep Learning models**

- general Neural Network
- Convolutional Neural Network
- Long Sort-term Memory(LSTM)
- Bert base uncased model

#### **3.3.1 Text classification with Machine Learning models**

##### **3.3.1.1 Naïve Bayes classifier**

Naïve Bayes algorithms are one of the best method for text classification specially in spam filtering, recommendation systems and sentiment analysis etc [4, 7, 14]. Bayes theorem estimates the probability of an event occurring

given the probability of the occurrence of another event that has already been occurred[4, 8].

Mathematically,

$$P(A/B) = (P(B/A)P(A))/(P(B)) \quad (1)$$

Where  $P(A/B)$  and  $P(B/A)$  are conditional probabilities.

For text classification, multinomial used for document classification problem and this approach of NB model is generally used for probabilistic learning method.

The first supervised learning model is the multinomial Naïve Bayes or multinomial NB model, a probabilistic learning method. The probability of a document  $d$  being in class  $c$  is computed as shown in eq.2 for multinomial NB[20].

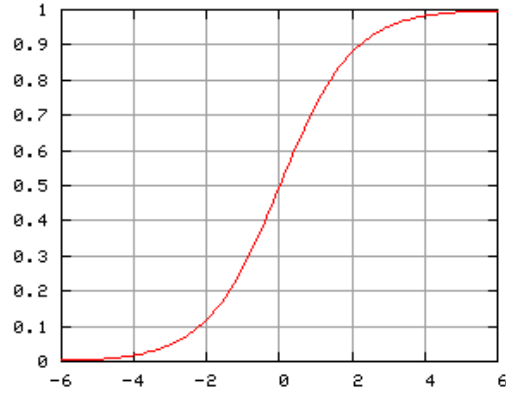
$$P(c/d) \propto \prod_{1 \leq k \leq nd} P(c)P(tk/c) \quad (2)$$

Where,

- $P(tk/c)$  conditional probability of  $tk$  occurred in a document of class  $c$ .
- $P(tk/c)$  interpret as a measure of how much evidence  $tk$  contributes that  $c$  is the correct class.
- $P(c)$  is the prior probability of a document occurring in class  $c$ .
- $t1, t2, \dots, tnd$  are the tokens in 'd' that are part of the vocabulary we use for classification and  $nd$  is the number of such tokens in  $d$ .

### 3.3.1.2 Logistic Regression

"Logistic Regression, also known as Logit Regression or Logit Model, is a mathematical model used in statistics to estimate (guess) the probability of an event occurring having been given some previous data. Logistic Regression works with binary data, where either the event happens (1) or the event does not happen (0)"[16] as shown in eq. 3.



**Figure 8:** Logistic Curve, The values of  $y$  cannot be less than 0 or greater than 1[16]

"Continuous or categorical predictors are used to predict the probabilities of the two different outcomes of the response variable"[21].

Mathematically Logistic Regression has shown in eq. 3,

$$\log(P_i/1 - P_i) = \beta_0 + \beta_1 * x_1 + \beta_2 * x_2 \quad (3)$$

- i- Observation number
- Probability of even occurring in the i-th case

$$P_i$$

- log - logarithm to base e

$$\beta_0, \beta_1 \text{ and } \beta_2$$

are unknown regression coefficients to be estimated.

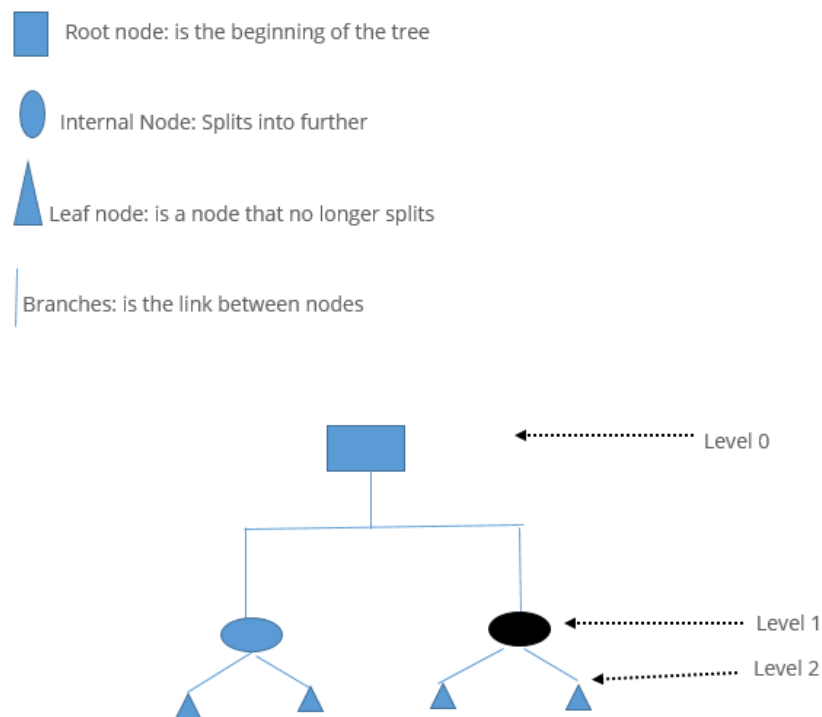
The estimated probabilities can be guaranteed between 0 and 1 if a transformation is applied to the probabilities. This transformation is called logit transformation.

$$\text{logit}(P_i) = \log(P_i/1 - P_i) \quad (4)$$

Logistic regression develops an S-shaped curve shown in fig.8, takes any Real-valued input and model it to the value in the range of 0 to 1.

### 3.3.1.3 Decision Tree

A Decision Tree plays an important rule in the wide area of machine learning. It is covering both classification and regression. Decision tree splits the training data into more parts in order to separate the data so that we can achieve the possible accurate output. Decision tree classifier is a supervised machine learning classifier. All the labelled input data is used as input data and decision tree classifier algorithm try to build a decision tree. This decision tree is used for predicting the category/label for the new data. This algorithm is also known as 'greedy algorithm' because of its recursive nature, the groups formed can be sub-divided using same procedure[22].



**Figure 9:** Decision Tree[17]



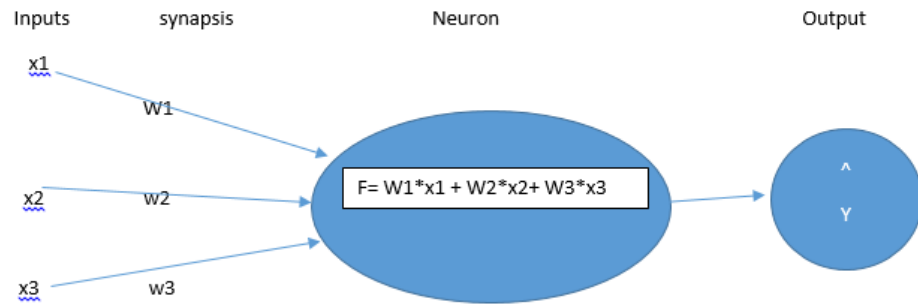
As shown in fig.9 Decision Trees are composed of nodes, branches, and leaves. Each node represents an attribute, each branch produce a decision, and each leaf gives us an predicted result. The depth of a Tree is defined by the number of levels, not including the root node[17].

The splitting can be binary or multiway. Multiway splits each node into multiple sub-groups. This sub-group parts as existing distinct values. Multiway splits put an end all information in a nominal attribute then finally in any path from the root to the leaf an attribute comes just once[17].

### 3.3.2 Text classification with Deep Learning models

#### 3.3.2.1 Simple Neural Network

Simple Neural Network's each edge has a weight and when the input travels it is multiplied by the weight associated with the edge and evaluate the network performance as shown in fig. 10 [4].



**Figure 10:** Perceptron[4, 14]

Simple Neural Network[4, 14] has the following mathematical eq.5

$$\sum_{i=1}^3 x_i w_i = w_1 * x_1 + w_2 * x_2 + w_3 * x_3 \quad (5)$$

Each edge has a weight and when the input  $x_i$  travels it is multiplied by the weight associated with the edge. Evaluate the network performance[4]. Then it

calculate finally the sentiment is positive, negative or neutral and that is our expected result and predicted value  $\hat{Y}$ .

Neural networks can be defined as: "A neural network is an interconnected assembly of simple processing elements ,units or nodes, whose functionality is loosely based on the animal neuron. The processing ability of the networks is stored in inter unit connection strengths or weights, obtained by a process of adaptation to or learning from a set of training patterns." [23]

The artificial neuron prototype and weight indicates to synapses. So in this , each input multiplied with weights before being sent to the cell body. The weighted signals are added together by simple arithmetic function to supply a node activation. Threshold logic unit - the activation is compared with the threshold, and if the activation exceeds the value of threshold then the unit produces a high valued output, else we receive the outputs is zero [23].

The connections between inter neuron, and electro chemical junctions is called synapses. This synapses placed on branches of cell are so to speak as dendrites. Each single neuron perceives instructions of many thousands of other neurons, and therefore constantly receiving thousands of incoming signals, which destined to the cell body. If the resulting signal reaches up to the threshold and exceeds then eventually neuron generate voltage impulse in response. This is transmitted to other neurons via fibre known as axon [23].

The term networks refers to the system of artificial neurons. This means the connection of single node to a collection of nodes and their communication is known as network , in which each node is connected to the another node in the net. One type of network can be seen in fig.10 [23].

### **3.3.2.2 Convolutional Neural Network**

" Convolutional Neural Network is a specialized kind of neural network for processing data that has a known grid-like topology. The name 'convolutional neural network' indicates that the network employs a mathematical operation called convolution. Convolution is a specialized kind of linear operation. Con-

volutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers." [24]

**The Convolution Operation:** The general concept of convolution is a mathematical operation of two functions that generates a new function. That new generated function mirrored to which extent the main original functions match if their graphs are aligned with each other. The convolution theorem says that under some special conditions the Fourier transform of a convolution is a point-wise product of Fourier transforms [25].

In one dimension, the convolution between two functions can be described as follows:

$$g(x) = f(x) * h(x) = \int_{-\infty}^{\infty} f(s)h(x-s)ds \quad (6)$$

In this eq.6  $f(x)$  and  $g(x)$  both are functions :

$s$  is a dummy variable of integration (having values of 0 or 1).

In two dimensions, the convolution between both functions can be defined as :

$$\begin{aligned} g(x, y) &= f(x, y) * h(x, y) = \\ &= \iint_{-\infty}^{\infty} f(s, t)h(x-s, y-t)dsdt \end{aligned} \quad (7)$$

The convolution operation is generally represented with an asterisk  $*$  and could not be confused with multiplication [26]. In one-dimensional applications there is a signal time domain,  $x(t)$  and a frequency domain,  $w(a)$ , based on the convolutional theorem, the convolution operation is

$$s(t) = (x * w)(t) \quad (8)$$

Where,

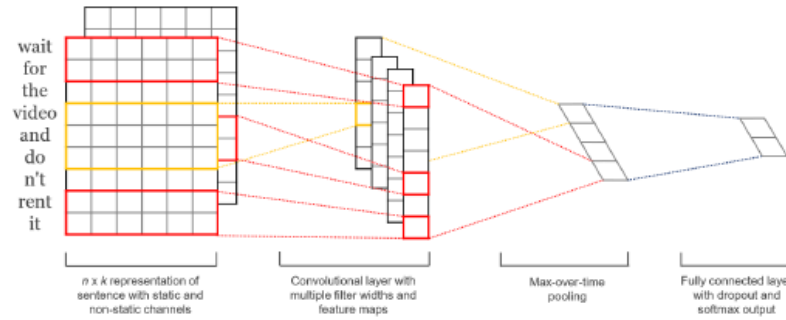
- $x$ , the first argument is represented as the input.

- $w$ , the second argument is shown to be the kernel,
- $s(t)$  output is referred to as the feature map or kernel map [26].

In ML applications, the input is normally a multidimensional array of data and the kernel is usually a multidimensional array of parameters those are adapted by the learning algorithm [27]. These multidimensional arrays are referred to be as tensors[26].

### Structure and Layers of CNN

The overall structure of CNN can be seen in fig.11 a sample CNN model for text classification[28] as follows:



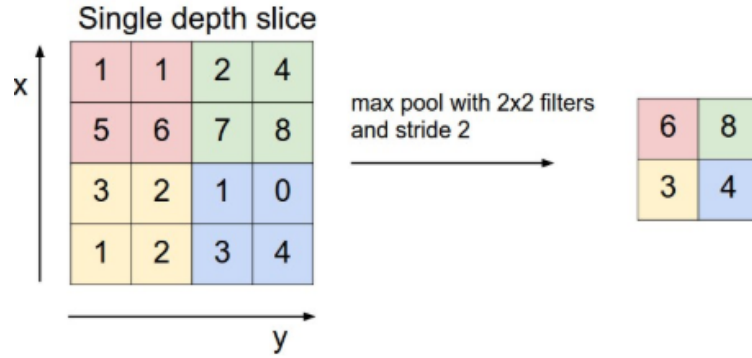
**Figure 11:** The architecture of a sample CNN model for text classification[28]

Different types of CNN layers[29] processing as follows:

- **Convolutional layer:** It is like a “filter” which passes through an image, scans few pixels simultaneously and creating a feature map that forecast the class to which each feature belongs to [29].
- **Pooling layer (down sampling):** This layer reduced the amount of information in each feature acquired in the convolutional layer while maintaining the crucial information. There are many iterations of convolution and pooling[29].
- **Fully connected input layer (flatten):** This layer takes the output

of all the previous layers, “flattens” to output and convert them to a single vector. Single vector can be an input for the next corresponding step [29].

- **First fully connected layer:** First fully connected layer is responsible for having the inputs from the feature analysis. It also applies weights to predict the correct label[29].



**Figure 12:** Max Pooling Layer[30]

- **Fully connected output layer:** It gives the final probabilities for each label[29]. Max pooling layer is a tool for that as shown in figure 12.

CNN is simple because it only does the multiplication and sum operation. We can start convolution from the first channel or from the last channel and split the computation with in one channel in a parallel fashion. This is the reason GPU performs better because they can employ all the code and its computation in a parallel way.

### 3.3.2.3 Recurrent Neural Network(Long Sort-term Memory (LSTM))

It is the nature of humans that they don't begin thinking from scratch after every second. Our every word actually depends on previous words concluding that human thoughts have persistence[31]. Conventional neural networks do not have the ability to do this and it looks to be prominent shortcoming. It is not clear how a classical or traditional neural network could use its reasoning about previous events in the film to inform later ones. Recurrent neural networks

provide the solution of this problem. They are networks with iterations in them, allowing information to persist[31].

Recurrent Neural Network (RNN) is the network that comprises of at least one feed-back connection, so the activation can flow round in an iterative form. That make the networks able to do temporal processing and learn sequences[32].

The main function of RNN is the processing of sequential information on the basis of the internal memory captured by the directed cycles. RNN can remember the previous computation of information and can reuse it by applying it to the next element in the sequence of inputs. A special type of RNN is long short-term memory (LSTM), which is capable of using long memory as the input of activation functions in the hidden layer[2].

### Structure of LSTM

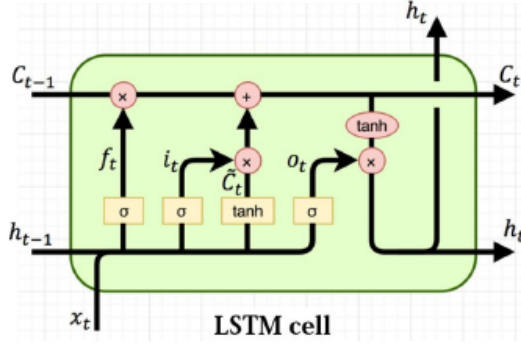
Long Short-Term Memory(LSTM) architecture for Recurrent Neural Networks have been proposed since its inception in 1995. Recurrent neural networks with LSTM have emerged as an effective and scalable model for several learning problems related to sequential data.[33]

The structure of extended LSTM with forget gate can be seen in figure 13. Structure of the LSTM cell and equations that describe the gates of an LSTM cell has the following feature as shown in fig.13 where we have following equations[34]:

$$\textbf{Input gate output}, i_t = \sigma (x_t U^t + h_{t-1} W^t) \quad (9)$$

$$\textbf{Forget gate output}, f_t = \sigma (x_t U^f + h_{t-1} W^f) \quad (10)$$

$$\textbf{Output gate output}, o_t = \sigma (x_t U^o + h_{t-1} W^o) \quad (11)$$



**Figure 13:** Structure of the LSTM cell[34]

$$\text{Candidate, } \tilde{C}_t = \tanh(x_t U^g + h_{t-1} W^g) \quad (12)$$

$$\text{New cell state, } C_t = \sigma(f_t * C_{t-1} + i_t * \tilde{C}_t) \quad (13)$$

$$\text{Hidden state, } h_t = \tanh(C_t) * o_t \quad (14)$$

- An LSTM network is a type of RNN.
- LSTM is used to handling long-term dependencies in data.
- LSTM is made up of three gates and one cell state where three gates are input gate, forget gate, output gate and these gates decide which signals are going to be forwarded to another node.
- W is the recurrent connection between the previous hidden layer and current hidden layer.
- U is the weight matrix that connects the inputs to the hidden layer.
- $\tilde{C}$  is a candidate hidden state that is computed based on the current input and the previous hidden state. C is the internal memory of the unit, which is a combination of the previous memory, multiplied by the forget gate, and the newly computed hidden state, multiplied by the input gate.

#### 3.3.2.4 BERT model

"BERT (Bidirectional Encoder Representations from Transformers) is a natural language model which is designed to pre-train deep bidirectional representations from unlabeled text by jointly conditioning on both left and right context in all layers. As a result, the pre-trained BERT model can be fine-tuned with just one additional output layer to create state-of-the-art models for a wide range of tasks, such as question answering and language inference, without substantial task-specific architecture modifications." [35]

BERT is a technique for NLP pre-trained developed by Google that was published in 2018 by Jacob Devlin and his colleague from Google. Bert is an algorithm that helps Google better understand search queries. Bert uses 'Transformers' and 'Masked language Modeling'. [35]

**Bidirectional** means it will predict the next word of text in both directions.

**Transformers** Bert uses transformers model which reads entire sequence of tokens at once.

#### **BERT training task [35]**

- Masked Language model
- Next Sentence Prediction.

BERT will help Google to better understand human language. As shows in fig.14. BERT is a two-step framework: pre-training and fine-tuning. 'Bertology' is the study of why BERT does things with Hugging Face library which is a transformer.

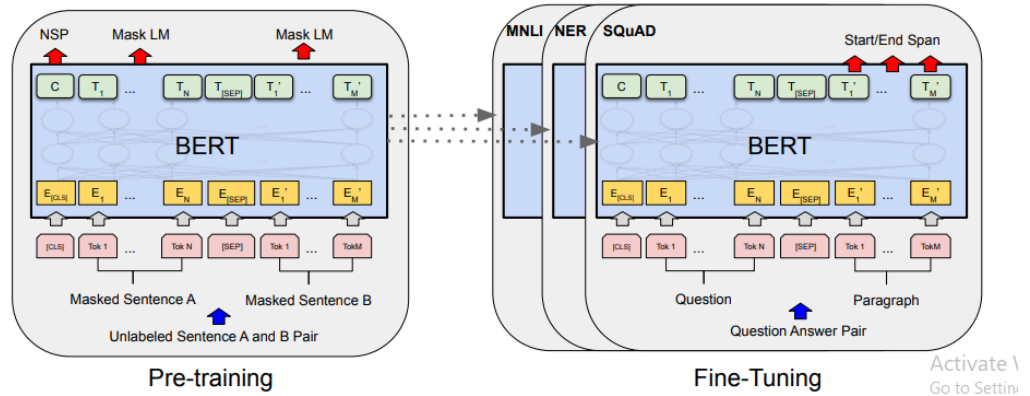
BERT architecture is a multi-layer bidirectional Transformer encoder. BERT has two models

1. BERT-base: 12 Encoders with 12 bidirectional self-attention heads and 110 million parameters
2. BERT-large: 24 Encoders with 24 bidirectional self-attention heads and 340 million parameters



Fig.14 shows apart from output layers, the same architectures are used in both pre-training and fine-tuning. The same pre-trained model parameters are used to initialize models for different down-stream tasks. During fine-tuning, all parameters are fine-tuned. [CLS] is a special symbol added in front of every input example, [SEP] is a special separator token which is inserted at the end of each sentence, i.e. separating questions and answers.[35]

There is also Next Sentence Prediction (NSP), a Masked Language Modeling (MLM) that is used in Bert model. The pre-trained BERT model can be fine-tuned with an additional output layer to create state-of-the-art models for a wide range of NLP tasks[36]. BERT consists of a stack of transformers and broadly speaking, transformers do not encode the sequential nature of their inputs[37].



**Figure 14:** BERT model[35]

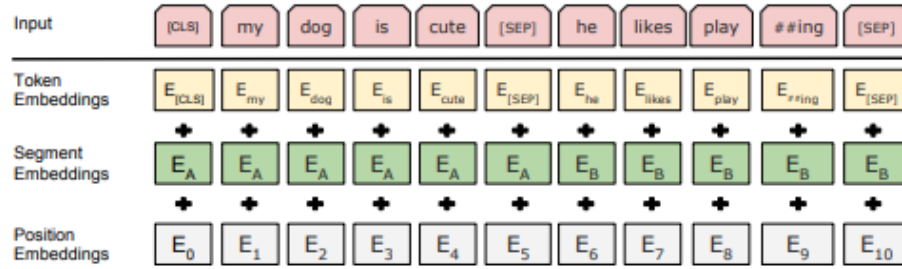
**Bert Attention** is the most important part of Bert model. It makes whether or not a dimension in the input vector is text or padding[35].

**Random seeds** are use to for the shake of productivity[36].

**BERT base model (uncased)** "Pre-trained model on English language using a masked language modeling (MLM) objective. This model is uncased: it does

not make a difference between english and English."[35]

BERT passes the words in the input text through a Token Embedding layer. Each token is transformed then into a vector representation. Unlike other deep learning models, BERT has additional embedding layers in the form of Segment Embeddings and Position Embeddings.



**Figure 15:** BERT input representation[35]

Fig.15 shows the input embeddings are the sum of the token embeddings, the segmentation embeddings and the position embeddings. If we want to predict the second sentence is joint to the first sentence or not, the following steps are performed:

- The entire input sequence goes through the Transformer model.
- The output of the [CLS] token is transformed into a  $2 \times 1$  shaped vector that is using a simple classification layer and learned matrices of weights and biases.
- Calculating the probability of IsNextSequence[35] with softmax.

Classification tasks such as sentiment analysis are done by next sentence classification, by adding a classification layer on top of the transformer output for the [CLS] token. Types of Natural Language Processing tasks with the BERT can help us

- Named entity determination
- Next Sentence Prediction.
- Question answering
- Word sense disambiguation
- Automatic summarizing

### 3.3.2.5 Generative Pre-trained Transformer 3 (GPT-3)

Generative Pre-trained Transformer 3 (GPT-3) developed by OpenAI, is also a pre-trained transformer model which is very large language model With 175 billion parameters that uses deep learning to produce human-like text. In this model, input text taken then probabilistically predicts what tokens from a known vocabulary will come next .

The main difference between GPT3 and other earliest model is its largest size of trainable parameters and weights are updated during training. If a model has more parameter than it need to train it with more data. GPT3 is trained on large dataset such as Common Crawl(410 billion)dataset, WebText2(19 billion data), Wikipedia(3 billion) data etc

In terms of architecture GPT3 is similar to the original structure of Transformer except the larger size of dataset it can handle. Compared to Bert and GPT3 model then GPT3 has more than millions of parameters while Bert large model has nothing compared to GPT3. It is also little different from Bert model.[38]

Bert model is designed to take the raw text and then produce the embeddings which can be used in other machine learning applications and in comparison to GPT1, GPT2 and GPT3 all use just the decoder so they take embeddings and then they produce text[38]. Some of the proposed task of GPT3 are as follows[39]:

- Addition

- Subtraction
- Generating new article
- Reserving a word etc

Though GPT3 is more fluent-sounding for text but it is only good for English language(93% of training data). It is also very expensive where training cost is \$12 million. There are also many errors like misconceptions about trained model which has happened[38, 39].

### 3.4 Forward Propagation

"A feed forward neural network to accept an input  $x$  and produce an output  $\hat{y}$ , information flows forward through the network. The input  $X$  gives the initial information that then propagates up to the hidden units at each layer, and finally produces  $\hat{y}$ . This is called forward propagation. During training, forward propagation can continue onward until it produces a scalar cost."[24]

### 3.5 Back Propagation

In machine learning, back propagation is a widely used algorithm in training feed forward neural networks for supervised learning[40] and back propagation allows the information from the cost to the flow backward through the network in order to compute the gradient. Back propagation refers only to the method for computing the gradient, while another algorithm, such as stochastic gradient descent is used to perform learning using this gradient[24].

The back propagation algorithm works by computing the gradient of the loss function with respect to each weight by the chain rule, computing the gradient one layer at a time, iterating backward from the last layer to avoid redundant calculations of intermediate terms in the chain rule[40, 24].

## 3.6 Confusion Matrix

The summary of prediction results on classification problems shown by a confusion matrix, i.e. the number of correct and incorrect predictions are summarized[41].

**Steps for the calculation a Confusion Matrix[41]:**

- Need a test dataset with expected outcome values.
- Make a prediction for each row in test dataset.
- From the expected outcomes and predictions count:
  - The number of correct predictions for each class.
  - The number of incorrect predictions for each class, organized by the class that was predicted.

## 3.7 Overview of Programming Languages

There are three common languages R, Python and Scala which are being used for processing text. In this section, we are going to overview each language briefly.

### 3.7.1 R Programming Language

Statistical computing and the graphical representation of data can be done in R language. It also provides infinitive support for its community and data scientists from all over the world. R language offers a collection of modules and tools for performing data analysis[42]. The most common packages supported by R language includes CRAN, Gmodel, Apart, E1071, TM, dplyr, lattice.

### 3.7.2 Python

Python language is very simple to understand, user-friendly, and highly productive language. Python has gained a lot of popularity in the past few years in the

Data Science field. Python language has a very robust collection of libraries and all libraries are open-sourced. The most commonly used libraries are NLTK, Pandas, Numpy, Sklearn, and RegEx. Python libraries are very efficient and it has the simpler syntax for machine learning and text processing. It has gained importance across the globe as computer giant Google has made it one of its official programming languages[43].

### **3.7.3 Scala**

Scala is a high-level programming language. Functional, static programming and high-performance systems can be done by Scala. It runs on the standard Java platform and inter-operates seamlessly with all Java libraries. This language is a blend of object-oriented and functional programming concepts in a statically typed language. Some of the core libraries are Breeze, Saddle, Scalalab, ScalaNLP, Smile[44].

## **3.8 Anaconda**

Anaconda is a python distribution which comes with several different tools. It is world's most popular data science platform. Anaconda has some of the built-in libraries i.e. Numpy, Pandas, Matplotlib, Seaborn, Sklearn etc.

One of the popular Interactive developed Environment(IDE)s for python is Jupyter notebook. We will use Jupyter notebooks for executing Machine Learning models codes.

## **3.9 Google Colab**

Colaboratory or "Colab" for short, allows us to write and execute Python in your browser, with zero configuration required and free access to GPU easy sharing. Whether a student, a data scientist or an AI researcher, Colab can make work easier. It is an interactive environment called a "Colab notebook"

that allows us to write and execute code[45]. We will use Google colab for executing our Simple Neural Network,CNN, LSTM and BERT model for our data set.

## **3.10 Libraries of Python**

### **3.10.1 NLTK**

Natural Language ToolKit (NLTK) library performs the task of symbolic and statistical Natural Language Processing very efficiently. NLP natural language processing provides an interface between humans and computers. NLP tool for python is the Natural language possessing toolkit NLTK used here. NLTK underpins tokenization, stemming, labeling, parsing and semantic thinking functionalities[7].

NLTK is one of the most important platforms for building Python programs to work with human language data. NLTK interface is easy-to-use[46]. We can get many resources such as WordNet. Text processing libraries for tagging, parsing, classification, tokenization, stemming and semantic reasoning, wrappers for industrial-strength NLP libraries we can get through NLTK.

The main aim of NLTK development is to facilitate teaching and research in the field of Natural language processing. NLTK plays a very important role in the development of complex research models for complex operations such as sentiment analytics, automatic summarizing, etc.

### **3.10.2 Pandas**

Pandas is a Python package. It is a nice data structure that can be used in “relational” or “labeled” data both and easy to manage by users. Pandas is just a list of lists. It is a tool for creating DataFrame[47]. It can efficiently perform the major tasks like data manipulation, aggregation and data visualization in the form of graphs and plots.

Some of the operations that Pandas library can perform are insertion and

deletion of data from the dataset, converting data structures to data frame objects, handling missing data from a dataset and also provides efficient and robust grouping by functionality.

### **3.10.3 NumPy**

Numpy is a python package that we can use to solve an advanced mathematical problem like sine, cosine, exponential, square root calculations. This package can be used to create an array and for special data type like vector(one-dimensional array), matrices(two-dimensional array) and also tensors( multidimensional array). NumPy[48] supports an object-oriented approach. The vectorization of mathematical operations day by day performing better and also speeds up the execution.

### **3.10.4 Sklearn**

Scikit-learn also known as sklearn can support different supervised and unsupervised machine learning algorithms. It is provided under a permissive BSD license. Sklearn is built on the top of the SciPy (Scientific Python). It supports various regression, classification and clustering algorithms which play a very vital role in the development of various kinds of machine learning models.[49]

### **3.10.5 Regular Expression (RegEx)**

RegEx library in python is supported by re module. Regular expression library can help us to find-out a particular pattern exists in a given sequence of strings or the set of characters or not. It is used to handle the text data. The real-life applications of Regular expression includes validating email addresses or the password, parsing text files, replacing or deleting strings.[50]



### 3.10.6 Keras TensorFlow

Keras is TensorFlow's high-level API for building and training deep learning models. It is used for fast prototyping, state-of-the-art research and production[51].

Three main advantages of Keras are as follows:

- User-friendly,
- Modular and Composable,
- Easy to extend.

For advanced text classification we can use keras. We need Tensorflow version 1.11 or higher for the BERT model to work.

## 3.11 Visualization Libraries

Visualization is an important part in development of machine learning and deep learning models. The different types of visualization supported by Python language are Matplotlib, seaborn, Plotly, wordcloud etc.

- **Matplotlib**

- It is used to graphical representation of data such as Pie chart, bar diagram and Line graphs etc.

- **Plotly**

- "Plotly's Python graphing library makes interactive, publication-quality graphs. Examples of how to make line plots, scatter plots, area charts, bar charts, error bars, box plots, histograms, heat-maps, subplots, multiple-axes, polar charts and bubble charts." [52].

- **Seaborn**

- Focused on the visualization

- **Wordcloud**

- "A Wordcloud (or Tag cloud) is a visual representation of text data. It displays a list of words, the importance of each being shown with font size or color. This format is useful for quickly perceiving the most prominent terms."[53].

Once we will apply each of the visualisation library and the Machine Learning algorithms in this project there will be a related discussion in the later chapters.

## 4 Data Collection And Data Investigation

This chapter will discuss data collection process and visual representation of dataset by Exploratory Data Analysis (EDA) and finally based on EDA build a feature extraction matrix on selected columns. Data collection is the most important step in solving any machine learning problem. Text classifier can only be as good as the dataset it is built from[51]. After building the dataset, data is investigated and column is selected for the feature matrix very carefully, while dependent variable is already available in the dataset. Below are the details of the data collection from source and data preprocessing:

### 4.1 Unstructured Data and Structured data

**Unstructured data** is also called qualitative data. It is not possible to fit unstructured data directly into a spreadsheet or database. It can be textual or non-textual. Sometimes it is machine-generated and sometimes people can create the unstructured data by their own[54].

Examples -

- **Social Media:** Data from social networking sites like facebook, Twitter and Linked-In
- **Media:** Audio and video files, images
- **Text files:** Word docs, PowerPoint presentations, email

- **Email:** There is some internal metadata structure, so it is sometimes called semi-structured but the message field is unstructured and difficult to analyze with traditional tools
- **Mobile data:** Text messages, locations
- **Communications:** Chat, call recordings

**Structured data** is also called as quantitative data which can be structured into a relational database. It is already organised in a systematic way so people use it regularly for further research. We can say structured data nicely fits in a spreadsheet[54].

Examples- Text data, Dates, Phone numbers, ZIP codes, Customer names etc.

Structured Data can be also categorised into the following two types:

***Labelled data*** it means there is a dependent variable, and that is response vector for the feature Matrix.

***Unlabelled data*** means it does not have any response value and we need to build the ML model based the feature matrix.

## 4.2 Data collection from source

"A collection of tweets is used as the primary corpus for sentiment analysis, which refers to the use of opinion mining or natural language processing" [9]. Twitter data can be collected via Twitter API. We need to connect to Twitter and query latest tweet and create an account on twitter and define an application. Users need to go to the [apps.twitter.com/app/new](https://apps.twitter.com/app/new) and generate the API keys and finally we will receive data from Twitter[4, 55, 7].

Data preprocessing is a very important step before feeding data into machine learning models and this affects the performance and accuracy. Higher accuracy can be achieved if the model is well trained by using a 'labeled training data'.

Kaggle is subsidiary of Google LLC. People participate via an online community in Kaggle, most of them are interested in the field of data science and machine learning. People can use data set from Kaggle and can also publish data sets, explore and create models in a web-based data-science environment, work with other data scientists and machine learning engineers and enter competitions to solve data science challenges[56].

That's why it was feasible to collect dataset from Kaggle which is already labelled [1]. Because of well structure and already there is a declared sentiment for each tweet. So for that purpose an account needed to create at "www.kaggle.com" and then from "https://www.kaggle.com/c/tweet-sentiment-extraction/data" collected the data which is titled as "Tweet Sentiment Extraction". There are three CSV files i.e. train.csv, test.csv and sample-submission.csv but only the 'train.csv' file is selected for the thesis project. Emotions in text tweets with existing sentiment labels used here under creative commons attribution 4.0. international licence[1]. Collected data contains opinions in the form of reviews and comments. These opinions have sentiments i.e. Positive, Neutral or Negative. This data may also contain tweets. The test data will be classified into 3 categories (i.e. Positive, Neutral, Negative) using Machine Learning and Deep Learning models.

### **4.3 Loading the Data**

First of all, I need to import the necessary libraries in Jupyter notebook in anaconda environment. The file "train.csv" should be downloaded from Kaggle and also need to upload in Jupyter Notebook and 'train.csv' file will be in S\_df DataFrame.

As shown in the following listing 4.1 imported libraries are:

```
1 #Importing Libraries
2 import pandas as pd
3 import numpy as np
4 import matplotlib.pyplot as plt
5 import seaborn as sns
6 from wordcloud import WordCloud
7 import matplotlib.pyplot as plt
```

**Listing 4.1:** Import libraries for Exploratory Data Analysis (EDA)

The following listing 4.2 is showing the dataset uploaded in S\_df DataFrame :

```
1 S_df= pd.read_csv("train.csv") # DataFrame create based on rows and
   columns # dataset is loaded
```

**Listing 4.2:** Loading data in Jupyter notebook

## 4.4 Data Understanding and Exploratory Data Analysis

After loading the dataset, it is necessary to understand the data and need to observe the relationship between the independent variable and dependent variable.

Fig.16 shows that 'S\_df' DataFrame has 'textID', 'text', 'selected\_text' and 'sentiment' columns. 'text' and 'selected\_text' are independent variable and both columns containing text that can be used for the feature matrix where the proper decision can be taken after the exploratory analysis of data. Column "sentiment" is dependent variable which shows the sentiment of 'text' and 'selected\_text'. Fig. 16 is showing the first 10 rows of DataFrame. 'S\_df' is pandas.core.frame DataFrame.

Fig.17 shows that 'S\_df' is a DataFrame and each column of 'S\_df' is object type and there are 27481 rows and 4 columns in 'S\_df' DataFrame.

```
S_df.head(10) # first 5 series
```

	textID	text	selected_text	sentiment
0	cb774db0d1	I'd have responded, if I were going	I'd have responded, if I were going	neutral
1	549e992a42	Sooo SAD I will miss you here in San Diego!!!	Sooo SAD	negative
2	088c60f138	my boss is bullying me...	bullying me	negative
3	9642c003ef	what interview! leave me alone	leave me alone	negative
4	358bd9e861	Sons of ****, why couldn't they put them on t...	Sons of ****,	negative
5	28b57f3990	http://www.dothebouncy.com/smf - some shameles...	http://www.dothebouncy.com/smf - some shameles...	neutral
6	6e0c6d75b1	2am feedings for the baby are fun when he is a...	fun	positive
7	50e14c0bb8	Soooo high	Soooo high	neutral
8	e050245fbd	Both of you	Both of you	neutral
9	fc2cbefa9d	Journey!? Wow... u just became cooler. hehe....	Wow... u just became cooler.	positive

Figure 16: First 10 rows of S\_df

```
: type(S_df)
: pandas.core.frame.DataFrame

: S_df.dtypes
: textID          object
: text            object
: selected_text    object
: sentiment        object
: dtype: object

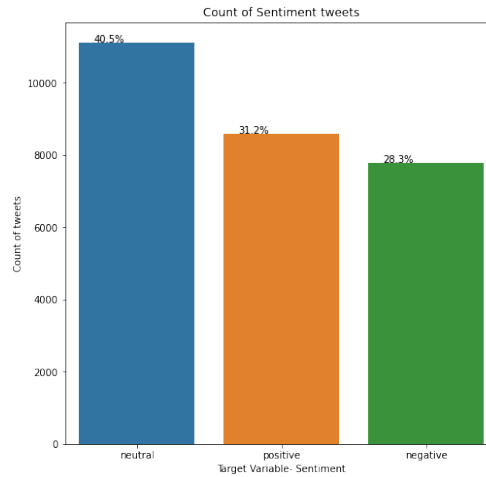
: S_df.shape # no, of rows and columns in Data frame
: (27481, 4)
```

Figure 17: Data type of S\_df

## 4.5 Visualization of dataset

For the data visualization we already imported the libraries in previous listing 4.1. There are many research papers with visual representation of just two sentiments which is basically a binary classification i.e. positive and negative only[1, 9]. In this thesis project, this classification has been extended to multiclass classification i.e. positive, negative and neutral. After loading the 'S\_df' DataFrame fig.18 shows visual representation of all sentiments

and following code for the fig.18, where the bar diagram shows that 40.5% observations are neutral, 31.2% are positive and 28.3% are negative sentiments. The following listing 4.3 shows the code for 'S\_df' data visualization:



**Figure 18:** S\_df visualization

```

1 plt.figure(figsize=(8,8))
2 ax=sns.countplot(data=S_df,x=S_df['sentiment'],order=S_df['sentiment'
   ].value_counts().index)
3 plt.xlabel('Target Variable- Sentiment')
4 plt.ylabel('Count of tweets')
5 plt.title('Count of Sentiment tweets')
6 total = len(S_df)
7 for p in ax.patches:
8     ax.annotate('{:.1f}%'.format(100*p.get_height()/total), (p.get_x()
   +0.1, p.get_height()+5))

```

**Listing 4.3:** Code for S\_df visualization





There is a nice visualization done by using WordCloud library as shown in fig.19 for the visualization of the 'Selected\_text' which shows positive, negative and neutral words. The above listing 4.4 shows the code for the WorldCloud visualization[1]:

```
1 wordcloud = WordCloud(  
2     background_color='green',  
3     max_words=100,  
4     max_font_size=50,  
5     random_state=30,  
6     collocations=False,  
7     colormap="spring").generate(' '.join(S_df['selected_text'].  
8     value_counts()[:100].index.tolist()))  
9  
10 plt.figure(figsize=(10,6))  
11 plt.title('Major words', fontsize=12)  
12 plt.imshow(wordcloud)  
13 plt.axis('off')  
14 plt.show()
```

**Listing 4.4:** Code for the WordCloud Visualization

## 4.6 Analysis of 'text' and 'selected\_text' Feature

DataFrame 'S\_df' has two text columns 'text' and 'selected\_text'. It is necessary to choose any one of the columns for the further procedure. But after the analysis it is explored that 'selected\_text' column has the selected filtered text from the 'text' column and 'text' column has more data then the 'selected\_text' column and it is also observed that 'selected\_text' has more clean data. The suitable column for feature extraction which is labelled with sentiments of text can be selected by doing the following steps:

Fig.20 shows the tweets length based on sentiment of the 'text' column where sentiment of text is positive, negative and neutral. Code for the visual representation of 'text' column is shown in listing 4.5 as follows:

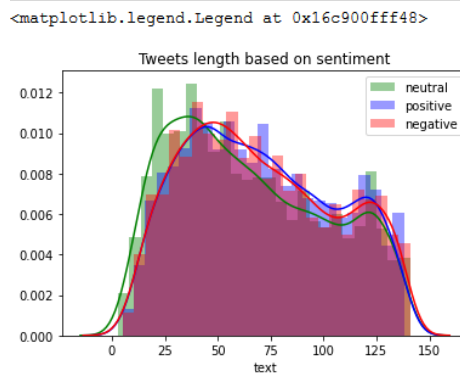


Figure 20: Value counts for 'text' column

```
1
2 neutral_tweets = S_df[S_df['sentiment']=='neutral']
3 positive_tweets = S_df[S_df['sentiment']=='positive']
4 negative_tweets = S_df[S_df['sentiment']=='negative']
5 neutral_length = neutral_tweets['text'].apply(lambda x: len(str(x)))
6 positive_length = positive_tweets['text'].apply(lambda x: len(str(x)))
7 negative_length = negative_tweets['text'].apply(lambda x: len(str(x)))
8 fig, ax = plt.subplots()
9 sns.distplot(neutral_length, ax=ax, color = 'green')
10 sns.distplot(positive_length, ax=ax, color = 'blue')
11 sns.distplot(negative_length, ax=ax, color = 'red')
```

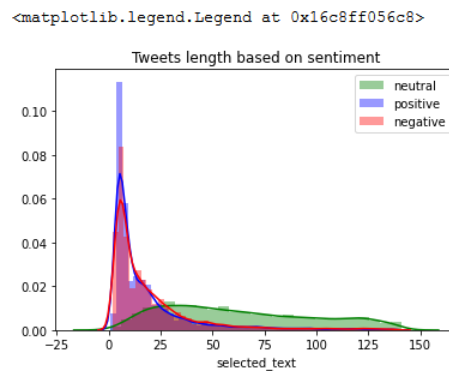
```

12 plt.title('Tweets length based on sentiment')
13 plt.legend(['neutral', 'positive', 'negative'])

```

**Listing 4.5:** DataFrame value counts for 'text' column

Fig.21 shows the value count of the 'selected\_text' where the normal distribution curve is observable that 'selected\_text' column contains less data with more clean text than the 'text' column. The curve also shows the sentiment of text which is positive, negative and neutral. The code for the visual representation of 'selected\_text' column is shown in listing 4.6 as follows:



**Figure 21:** DataFrame value counts for 'selected\_text' column

```

1 neutral_tweets = S_df[S_df['sentiment']=='neutral']
2 positive_tweets = S_df[S_df['sentiment']=='positive']
3 negative_tweets = S_df[S_df['sentiment']=='negative']
4 neutral_length = neutral_tweets['selected_text'].apply(lambda x: len(
    str(x)))
5 positive_length = positive_tweets['selected_text'].apply(lambda x: len(
    str(x)))
6 negative_length = negative_tweets['selected_text'].apply(lambda x: len(
    str(x)))
7 fig, ax = plt.subplots()
8 sns.distplot(neutral_length, ax=ax, color = 'green')
9 sns.distplot(positive_length, ax=ax, color = 'blue')
10 sns.distplot(negative_length, ax=ax, color = 'red')
11 plt.title('Tweets length based on sentiment')

```

```

12 plt.legend(['neutral', 'positive', 'negative'])
13 #standard normal distribution

```

**Listing 4.6:** DataFrame value counts for 'selected\_text' column

## 4.7 Exploring the selected\_text column

This section explores the data of each column's five rows of the index of 'S\_df' DataFrame i.e. n=5. The following analysis of 'selected\_text' and 'sentiment' columns relationship. As shown in listing 4.7 is value counting for 'selected\_text' column where output has the list of 5 texts.

```

1 n = 5
2 S_df['selected_text'].value_counts()[:n].index.tolist()
3 Output:
4 ['good', 'love', 'Happy', 'miss', 'happy']

```

**Listing 4.7:** Value counts for 'selected\_text'

The following listing 4.7 shows the 5 neutral rows of the 'sentiment' column:

```

1 n = 5
2 S_df[S_df['sentiment']=='neutral']['selected_text'].value_counts()[:n]
   ].index.tolist()
3 Output:
4 ['I'm at work', 'aw', 'I see', 'morning', 'salt and vinegar']

```

**Listing 4.8:** Observe neutral sentiment of 'selected\_text' column

The following listing 4.8 shows the 5 negative rows of the 'sentiment' column:

```

1 n = 5
2 S_df[S_df['sentiment']=='negative']['selected_text'].value_counts()[:n]
   ].index.tolist()
3 Output:
4 ['miss', 'sad', 'sorry', 'hate', '****']

```

**Listing 4.9:** Observe negative 'sentiment' of 'selected\_text' column

The following listing 4.9 shows the 5 positive rows of the 'sentiment' column.

```
1 n = 5
2 S_df[S_df['sentiment']=='positive']['selected_text'].value_counts()[:n
   ].index.tolist()
3 Output:
4 ['good', 'love', 'Happy', 'happy', 'thanks']
```

**Listing 4.10:** Observe positive sentiment of 'selected\_text' column

## 4.8 Exploring the 'text' column

In this section the 'text' column is explored and observed its positive, negative, neutral sentiments step by step like previous section. Following listing 4.11 is value counting for 'text' column which output has more texts than 'selected\_text' and contains list of 5 texts:

```
1 n = 5
2 S_df['text'].value_counts()[:n].index.tolist()
3
4 Output:
5 [' Sigh. It is a sad, lonely profession.',
6  'Perhaps the answers to all my questions, lie in the deep waves of
   the ocean. Beach trip anyone!?',
7  ' ohh i would def give it 2 u but tht was kenyatta's cam not mine
   sry but i want tht pic 2',
8  ' That's not how we share links on Twitter Yours is http://bit.ly/
   QljYb and you can view your stats at http://bit.ly/info/QljYb',
9  ' i like']
```

**Listing 4.11:** DataFrame value counts for 'text' column

```
1 n = 5
2 S_df[S_df['sentiment']=='neutral']['text'].value_counts()[:n].index.
   tolist()
3 Output:
4 ['gonna head to bed then work, then moncton to see Jill',
```

```

5 'http://twitpic.com/4j95z - I don't know why, but I LOLd.',
6 '_85 Aaaargh help -it's a conspiracy LOL! How are you today Lizzi?',
7 ' what r u doing tonight bro?I wanna go out',
8 '_Kay planted in the garden last week, ive got to check on it']

```

**Listing 4.12:** Observed neutral sentiment for the 'text' column

```

1 n = 5
2 S_df[S_df['sentiment']=='positive']['text'].value_counts()[:n].index.
   tolist()
3 Output:
4 [' thanks neemah. I'm gonna be soooo close to you and izzy, yet so far
   ',
5  'Listening to the simply awesome Ratatat on a bank holiday monday
   before a BBQ later http://bit.ly/gJqSh',
6  "_Coleman Looks a little too 'fried' for me, but you have fun with
   that, Paul!",
7  ' Yea its nice! But im only here for a weekend!',
8  ' Discrimination is not a bad thing. I've learned to say no. My
   children would say I mastered that years ago']

```

**Listing 4.13:** Observed positive sentiment for the 'text' column

```

1 n = 5
2 S_df[S_df['sentiment']=='negative']['text'].value_counts()[:n].index.
   tolist()
3 Output:
4 ['Just started feeling bad again ugh. I hate it when I don't feel
   good!!',
5  '_Beckiie_x omg are you going to see it?! another reason to not live
   in devon none of the f***in cinemas have the film im so annoyed',
6  'i just fell going into my house.. not fun',
7  'Aw man, Half Term isn't long enough lol!',
8  'i'm sick of waking up and feeling exhausted']

```

**Listing 4.14:** Observed negative sentiment for the 'text' column

## 4.9 Summary

From the analysis of previous sections about the 'text' and 'selected\_text' column's feature it has been observed that 'selected\_column' contains more cleaned text than the 'text' column of S\_df DataFrame and the decision is taken depending on that the "selected\_text" column is chosen for the next step for data preprocessing. Finally, it can be clearly mentioned that the 'selected\_text' column will be best selection for feature extraction because it will give the following :

- Less time consumption
- Resource constraint
- Less consumption of power

The next chapter will explain the implementation process for different models.

## 5 Implementation

In this chapter, the implementation process is explained step by step in which different kinds of models are used and finally analysed the achieved results. The comparison of each model's accuracy will help to find out a best model suitable for the dataset. This chapter is divided into three parts, part one will contain the implementation of text classification based on sentiment analysis with Machine Learning models, second part will include the text classification based on sentiment analysis with Deep Learning models and finally at the end of this chapter, analysis of each model will be discussed.

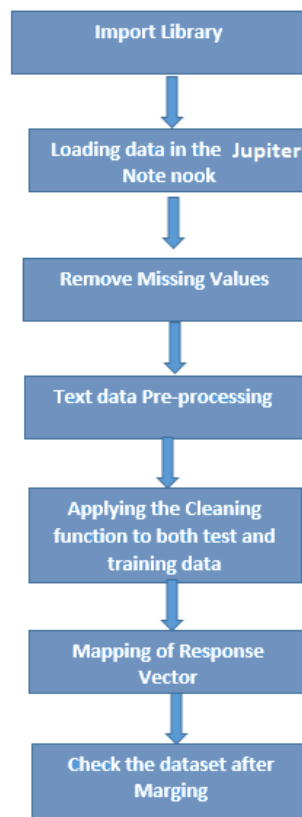
### 5.1 Sentiment Analysis with Machine Learning Models

Sentiment analysis with Machine Learning Models can be explained in four task. As shows in fig.22 the flow of data preparation for machine learning models, fig.24 flow of training and test data for ML, fig.25 flow of feature engineering steps and fig.28 steps for find out best machine learning model. Each step is discussed sequentially as follows:

#### 5.1.1 Data Preparation for Machine Learning

First of all, we need to import the necessary libraries such as pandas and numpy as shown in the listing 5.1. After that we need to load the dataset in to the Jupyter notebook. Like the previous chapter we keep the name of DataFrame as 'S\_df' and load the data "train.csv" into 'S\_df'. The step of data preprocessing is shown in fig.22.





**Figure 22:** Flow of Data Preparation for ML

```
1 import pandas as pd
2 import numpy as np
3 S_df= pd.read_csv("train.csv")
```

**Listing 5.1:** Import libraries and data loading

It is very important to remove the missing values from the 'S\_df' because missing values will interrupt to implement the models with the dataset. There is only one missing value and after applying the `dropna()` function there will be no more missing values as shown in listing 5.2 code.

```
1 print(S_df.isnull().sum())
2 S_df.dropna(inplace=True)
```

---

**Listing 5.2:** Code for the missing value remove data

#### 5.1.1.1 Text Data Pre-processing

In this section the cleaning function will be applied to both test and training dataset for the text data preprocessing. Before starting any NLP project, it is necessary to pre-process the data to get it all in a consistent format. In this step after cleaning and tokenization, the dataset will be converted into a numerical matrix. For that purpose we create a function which will perform the following tasks on the 'selected\_text' column. The function name is "clean\_text()" which will make text lowercase, removes hyperlinks, remove punctuation, removes numbers and "text\_preprocessing()" will do the tokenization and remove stopwords by returning text parameter, it will apply into 'selected\_text' column. As shown in the listing 5.3 text pre-processing helper function.

```
1 import re
2 def clean_text(text):
3     '''Make text lowercase, remove text in square brackets,remove
4     links,remove punctuation
5     and remove words containing numbers.'''
6     text = text.lower()
7     text = re.sub('\[.*?\]', '', text)
8     text = re.sub('https?://\S+|www.\S+', '', text)
9     text = re.sub('<.*?>+', '', text)
10    text = re.sub('\n', '', text)
11    text = re.sub('\w*\d\w*', '', text)
12    return text
```

**Listing 5.3:** Text preprocessing helper function

```
1 import nltk
2 from nltk.corpus import stopwords
3 def text_preprocessing(text):
```

```

4     tokenizer = nltk.tokenize.RegexpTokenizer(r'\w+') # r stands for
read and w stands for w
5     nopunctuation = clean_text(text)# no punctuation and now its
punctuation free text now
6     tokenized_text = tokenizer.tokenize(nopunctuation)
7     remove_stopwords = [w for w in tokenized_text if w not in
stopwords.words('english')]
8     combined_text = ' '.join(tokenized_text)
9     return combined_text

```

**Listing 5.4:** Data preprocessing function code

After applying the cleaning function on 'selected\_text' column we added a new column 'clean\_text' and this column will contain the new clean texts. News 'S\_df' DataFrame in fig.23 after merging the 'clean\_text' column.

```

1 S_df['clean_text'] = S_df['selected_text'].apply(str).apply(lambda x:
text_preprocessing(x))

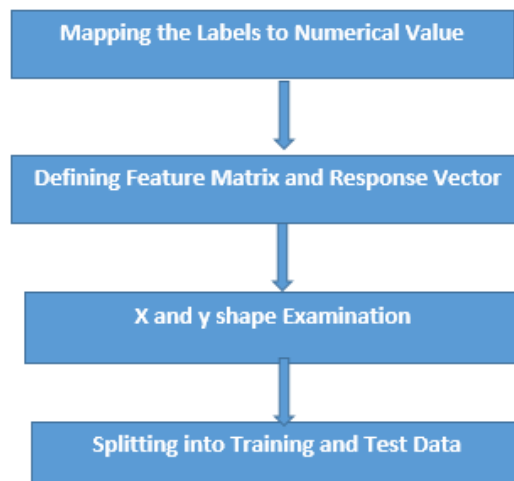
```

**Listing 5.5:** Applying cleaning function

S\_df.head(n=15)

	textID	text	selected_text	sentiment	clean_text
0	cb774db0d1	I'd have responded, if I were going	I'd have responded, if I were going	neutral	i d have responded if i were going
1	549e992a42	Sooo SAD I will miss you here in San Diego!!!	Sooo SAD	negative	sooo sad
2	088c60f138	my boss is bullying me...	bullying me	negative	bullying me
3	9642c003ef	what interview! leave me alone	leave me alone	negative	leave me alone
4	358bd9e861	Sons of ****, why couldn't they put them on t...	Sons of ****,	negative	sons of
5	28b57f3990	<a href="http://www.dothebouncy.com/smf - some shameles...">http://www.dothebouncy.com/smf - some shameles...</a>	<a href="http://www.dothebouncy.com/smf - some shameles...">http://www.dothebouncy.com/smf - some shameles...</a>	neutral	some shameless plugging for the best rangers f...
6	6e0c6d75b1	2am feedings for the baby are fun when he is a...	fun	positive	fun
7	50e14c0bb8	Soooo high	Soooo high	neutral	soooo high
8	e050245fd	Both of you	Both of you	neutral	both of you
9	fc2cbefa9d	Journey! Wow... u just became cooler. hehe....	Wow... u just became cooler.	positive	wow u just became cooler
10	2339a9b08b	as much as i love to be hopeful, i reckon the...	as much as i love to be hopeful, i reckon the ...	neutral	as much as i love to be hopeful i reckon the c...
11	16fab9f95b	I really really like the song Love Story by Ta...	like	positive	like
12	74a76f6e0a	My Sharple is running DANGERously low on ink	DANGERously	negative	danqerously

**Figure 23:** Check the dataset after pre-processing



**Figure 24:** Flow of Training and Test Data for ML

#### 5.1.1.2 Mapping of Response vector

Usually in machine learning, the text values need to be converted to the numeric form which is also known as mapping. DataFrame 'S\_df' will add a new column with the new equal value of sentiment column.

A column with a name 'sentiment\_num' will be added which displays the numerical value of column "Sentiment". I created a dictionary where positive = 1, negative = 2, neutral= 0 value assign to the positive, negative, neutral keys by using following command in listing 5.6:

```
1 S_df['sentiment_num'] = S_df.sentiment.map({'positive':1, 'neutral':0, 'negative':2})
```

**Listing 5.6:** Mapping sentiment Data

#### 5.1.2 Splitting into Training and Test dataset

X denotes feature of a dataset and y denotes the label i.e. response vector.

```
1 X = S_df.clean_text
2 y=S_df.sentiment_num
```

---

**Listing 5.7:** Define X and y

After defining X and y, we have taken some necessary steps to split training and test dataset. I define here four variables which consist of randomly selected statements and corresponding sentiment values. For this purpose, I use 'train\_test\_split()' function from the 'model\_selection' module in Sklearn. In the above listing, the parameter 'train size' has the value 0.75 which means that 75% of the data will be used to train the model and the remaining 25% will be the test data by using the following code:

```
1 from sklearn import model_selection
2 X_train,X_test,y_train,y_test=model_selection.train_test_split(X,y,
    test_size=0.25,random_state=100)
```

**Listing 5.8:** Splitting into test and Training dataset

By splitting the data randomly and assigning to X train, X test, y train and y test variables, the model will be trained and tested for evaluating the most suitable machine learning algorithm.

### 5.1.3 Feature Engineering

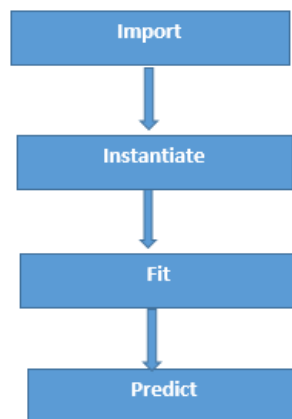
Term frequency (Tf) provides the frequency and Inverse Document frequency(Idf) provides the number of times the particular word appears in entire set of documents[57].

I will take Tf-Idf Vectorizer for feature extraction where Tf-Idf Vectorizer scales down the impact of tokens that occur very frequently in a given corpus. Feature extraction steps are shown in fig.25

#### 5.1.3.1 Import

'TfidfVectorizer' will be imported from 'feature\_extraction.text' module in Scikit-Learn library.

```
1 from sklearn.feature_extraction.text import TfidfVectorizer
```



**Figure 25:** Flow of Feature Engineering steps

#### 5.1.3.2 Instantiate

When the import step is finished, next step is instantiated by creating an object 'vect' and Input parameter 'stop words', is a list of all the unused words which will be removed from the text as shown in the listing 5.9.

```
1 vect = TfidfVectorizer(stop_words = 'english')
2 #vect is here to define a variable for vectorizer
3 #Instantiating the Vectorizer
4 print(vect)
```

**Listing 5.9:** Instantiating the Vectorizer

#### 5.1.3.3 Fit

The 'fit' function is used to learn the vocabulary of the training data which is also called the training of data.

```
1 vect.fit(X_train)
```

**Listing 5.10:** Fitting for the training data

The input parameter passed to fit function 'X\_train' is the training data defined on the bases of which the vectorizer will learn the vocabulary.

```
1 vocabulary = vect.get_feature_names()
```

**Listing 5.11:** The learned vocabulary

The 'fit' function will change the text as follows:

- All the articles will be removed by this 'fit' function.
- 'lower case' parameter will convert all the letters of any text to lower case.
- The punctuation from the text will be removed.
- Punctuation marks will be removed from the text.
- Duplicated words will be also removed.

#### 5.1.3.4 Transform

In the transformation vocabulary will be transformed to the numerical features.

**For training data** the transformation will be applied on both training and test dataset and the document term matrix `train_d_t_m` for the training dataset which has 20610 rows, and 13640 columns. As shown in the following listing 5.12:

```
1 train_tf_idf = vect.transform(X_train)
2 train_d_t_m = train_tf_idf.toarray()
3 train_d_t_m #feature matrix for X_train
```

**Listing 5.12:** Transform training data

**For test data** Fig.27 the document term matrix `test_d_t_m` for the test dataset which has 6870 rows and 13640 columns. For the test dataset matrix the following listing 5.13 is shown:

```

train_d_t_m
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])

train_d_t_m.shape
(20610, 13640)

```

**Figure 26:** Feature matrix for X\_train data

```

1 test_tf_idf=vect.transform(X_test)
2 test_d_t_m= test_tf_idf.toarray()
3 test_d_t_m #feature metric for X_test

```

**Listing 5.13:** Transform test data

```

array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       ...,
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.]])

test_d_t_m.shape
(6870, 13640)

```

**Figure 27:** Feature metrix for X\_test

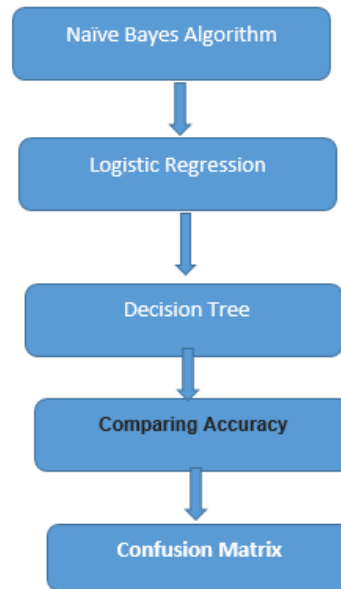
Finally, 'tfidf' vectorizer used to convert the text data in to numerical form and we have document term matrices for both training and test data which are 'train\_d\_t\_m' and 'test\_d\_t\_m'.

#### 5.1.4 Decision for the best Machine Learning model

For our NLP model, we implemented the machine learning model in this section which could be the best model to predict the sentiments of our text data. In this section, we test the three models i.e. 'Naïve Bayes', 'Logistic Regression' and 'Decision Tree' which we mentioned in previous chapters. The final decision



will be taken by comparing the accuracy of all the models. The flow of this section is shown in the fig.28.



**Figure 28:** Steps for best Machine Learning model

#### 5.1.4.1 Naïve Bayes Model

Multinomial Naïve Bayes model implementation will be shown here. By using the model 'Naïve Bayes' in sklearn, I import the classifier 'MultinomialNB' and then instantiate it by creating an object 'classifier'. It takes 2.14s in Jupyter notebook to train the model by using the following python code:

```
1 from sklearn.naive_bayes import MultinomialNB
2 classifier = MultinomialNB()
```

**Listing 5.14:** Importing and Instantiating

By using the 'fit' function the model will be trained. The input parameters will be 'train\_d\_t\_m' and 'y\_train'. As it can be observed in the figure 26 which is a matrix also called 'document term matrix' calculated from training

data. Training data cannot be passed directly because model building requires numerical data. `y_train` is a vector contains all those labels which make a sentiment analysis for a text is a 'positive' or a 'negative' or a 'neutral' comments on text data.

```
1 %time classifier.fit(train_d_t_m,y_train)
```

**Listing 5.15:** model training using training data

The model has learned from the '`train_d_t_m`' and '`y_train`' two parameters and '`fit`' function will train the model. Now the model will predict for some test data. '`predict`' function will be used and the input parameter will be the data for which the prediction is needed. As we have discussed that it must be numerical data which can be observed in fig.29 as follows:

```
1 predicNBML = classifier.predict(test_d_t_m )
2 predicNBML
```

**Listing 5.16:** Prediction by Naïve Bayes Model

```
Out[88]: array([0, 0, 1, ..., 0, 2, 0], dtype=int64)
```

**Figure 29:** Prediction by Naïve Bayes Model

### Accuracy for Naïve Bayes Model

```
1 from sklearn import metrics
2 metrics.accuracy_score(y_test,predicNBML)
```

After executing the code we achieved the following result:

```
Out[90]: 0.7831149927219796
```

**Figure 30:** Accuracy for Naïve Bayes Model

### Confusion matrix for Naïve Bayes Model

```
1 metrics.confusion_matrix(y_test,predicNBML)
```

**Listing 5.17:** Confusion matrix for Naïve Bayes Model

```
Out[91]: array([[2541, 146, 105],
               [ 407, 1667, 46],
               [ 687, 99, 1172]], dtype=int64)
```

**Figure 31:** Confusion matrix for Naïve Bayes

#### 5.1.4.2 Logistic Regression Model

In this section we will implement the Logistic Regression model for our dataset.

```
1 from sklearn.linear_model import LogisticRegression
2 clf_logistic= LogisticRegression()
```

**Listing 5.18:** Importing and Instantiating Logistic Regression

Naïve Bayes model uses the same input parameters which is 'train\_d\_t\_m' and 'y\_train' and the model train itself measures the total 1min 2s time for the process to complete in Jupyter note book.

```
1 %time clf_logistic.fit(train_d_t_m, y_train)
```

**Listing 5.19:** Model Training

The model will predict values for the input test data.

```
1 Predict_m_log= clf_logistic.predict(test_d_t_m)
2 Predict_m_log
```

**Listing 5.20:** Prediction by Logistic Regression

```
Out[96]: array([0, 0, 1, ..., 0, 2, 0], dtype=int64)
```

**Figure 32:** Prediction by Logistic Regression Model

#### Accuracy for Logistic Regression

```
1 metrics.accuracy_score(y_test, Predict_m_log)
```

**Listing 5.21:** Accuracy for Logistic Regression

#### Confusion matrix for Logistic Regression

```
Out[97]: 0.7885007278020378
```

**Figure 33:** Accuracy for Logistic Regression

```
1 metrics.confusion_matrix(y_test, Predict_m_log)
```

**Listing 5.22:** Confusion matrix for Logistic Regression Model

```
Out[98]: array([[2428, 177, 187],
               [ 343, 1703,  74],
               [ 573,   99, 1286]], dtype=int64)
```

**Figure 34:** Confusion matrix for Logistic Regression

#### 5.1.4.3 Decision Tree Model

In this section Decision tree model will be applied to dataset like previous section. In this case wall time: 16min 14s measured.

```
1 from sklearn.tree import DecisionTreeClassifier
2 Dt_Clf_model = DecisionTreeClassifier()
3 %time Dt_Clf_model.fit(train_d_t_m, y_train)
4 Dt_predict= classifier.predict(test_d_t_m )
5 Dt_predict
```

**Listing 5.23:** Prediction by the Decision Tree model

Following is the predicted output for decision tree:

```
Out[103]: array([1, 0, 1, ..., 2, 2, 2], dtype=int64)
```

**Figure 35:** Prediction by Decision Tree Model

#### Accuracy for Decision Tree Model

```
1 metrics.accuracy_score(y_test, Dt_predict)
```

**Listing 5.24:** Observed accuracy for the Decision Tree model

```
Out[57]: 0.7831149927219796
```

**Figure 36:** Accuracy for the Decision Tree Model

### Confusion matrix for Decision Tree Model

```
1 metrics.confusion_matrix(y_test,Dt_predict)
```

**Listing 5.25:** Confusion matrix for Decision Tree Model

```
Out[105]: array([[2065,  245,  482],
                 [ 220, 1686,  214],
                 [ 359,  131, 1468]], dtype=int64)
```

**Figure 37:** Confusion matrix for Decision Tree

## 5.2 Sentiment Analysis with Deep Learning Models

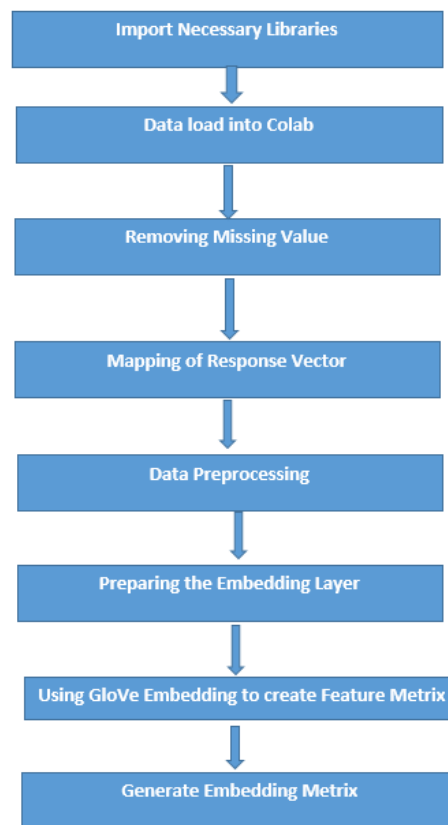
In this section, Simple Neural Network, Convolution Neural network and Recurrent Neural Network will be implemented. The deep learning models have a bit different data preprocessing steps as compared to the machine learning model.

In this project, the Deep Learning classification algorithms for our training and testing is done on the same dataset like before. Here SimpleNN, CNN and RNN(LSTM) models will be built step by step[58]. Google Colab notebook environment is used for the code execution as follows:

### 5.2.1 Pre-process Data for Simple NN, CNN and RNN (LSTM)

In this section, the data pre-processing steps for Deep learning models. The steps for data preprocessing remain same for the simple NN, CNN and RNN (LSTM) as shown in fig.38.

**Importing Required Libraries for Deep learning model:** Here I used Keras Embedding Layer and GloVe word embeddings to convert text data to



**Figure 38:** Flow of preprocess Data for Simple NN, CNN and RNN (LSTM)

numeric form. So for that purpose we need to import necessary libraries as shown in the listing 5.26:

```
1 import pandas as pd
2 import numpy as np
3 import re
4 import nltk
5 from nltk.corpus import stopwords
6 from numpy import array
7 from keras.preprocessing.text import one_hot
8 from keras.preprocessing.sequence import pad_sequences
9 from keras.models import Sequential
10 from keras.layers.core import Activation, Dropout, Dense
11 from keras.layers import Flatten
```

```

12 from keras.layers import GlobalMaxPooling1D
13 from keras.layers.embeddings import Embedding
14 from sklearn.model_selection import train_test_split
15 from keras.preprocessing.text import Tokenizer

```

**Listing 5.26:** Importing required libraries for Deep Learning model

**Create new dataframe 'df':** After importing the libraries we load the dataset into Google Colab notebook in S\_df DataFrame. Here again we remove the missing values from the dataset. We create a new column 'sentiment\_num' for the mapping of response vector by creating a dictionary like previous section. Next we create a new dataframe 'df' as shown in fig.39 which contain only the 'selected\_text' and the 'sentiment\_num' columns and then we define 'y' by using the following listing 5.27:

```

1 S_df['sentiment_num']= S_df.sentiment.map({'positive':1,'neutral':0, '
   negative':2 })
2 df=pd.DataFrame()
3 df['selected_text']=S_df['selected_text']
4 df['sentiment_num']=S_df['sentiment_num']

```

**Listing 5.27:** Create new dataframe 'df'



	selected_text	sentiment_num
0	I'd have responded, if I were going	0
1	Sooo SAD	1
2	bullying me	1
3	leave me alone	1
4	Sons of ****,	1
...	...	...
27476	d lost	1
27477	, don't force	1
27478	Yay good for both of you.	2
27479	But it was worth it ****.	2
27480	All this flirting going on - The ATG smiles. Y...	0

27480 rows x 2 columns

**Figure 39:** New dataframe df

**Data Preprocessing:** Here we create a function called `preprocess_text()`, a user defined function which takes the input argument and process, finally returns the resulting text based on the methods we call inside the function. The following listing 5.28 shows the cleaning function:

```

1 def preprocess_text(sen):
2     sentence = remove_tags(sen) # Removing html tags
3     sentence = re.sub('[^a-zA-Z]', ' ', sentence) # Remove
    punctuations and numbers
4     sentence = re.sub(r"\s+[a-zA-Z]\s+", ' ', sentence) # Single
    character removal.second input argument= ' ' by blank symbol.
5     sentence = re.sub(r'\s+', ' ', sentence)#Removing multiple spaces
6     sentence = sentence.lower() #will make all the text lower case
7     sentence = re.sub('[.*?\\]', '', sentence) #'[.*?\\]' removed
8     sentence = re.sub('https?://\S+|www\.\S+', '', sentence) #'https
    ?://\S+|www\.\S+' will be removed
9     sentence = re.sub('<.*?>+', '', sentence) # '<.*?>+' will be
    removed and sub means substance for substitute
10    sentence = re.sub('\n', '', sentence)
11    sentence = re.sub('\w*\d\w*', '', sentence) #W* is raw text
    removed
12    return sentence

```

**Listing 5.28:** Data cleaning function

To remove tags from the input text we need to use the following listing 5.29:

```

1 TAG_RE = re.compile(r'<[^>]+>')
2 def remove_tags(text):
3     return TAG_RE.sub('', text)

```

**Listing 5.29:** Function for remove tags from text

After using the `preprocess_text()` function on 'selected\_text' column, it is free from html tags, punctuation, numbers, single character removal, special character, raw text and whole text will be converted to lower case. Finally, it will be stored them in a new list `X[]` as shown below:



```

1 X = []
2 sentences = list(df['selected_text'])
3 for sen in sentences:
4     X.append(preprocess_text(sen))

```

**Listing 5.30:** 'selected\_text' converted to X list

```

X
'blood everywhere ',
'pulled up walmart aunt got out went in fell asleep hours later were you lol',
'searching my home for few things to cook them for dinner this evening it mothers day so guess who im eating with',
'very very cute',
'woke up at then fell back to sleep woke up at and back to sleep again woke up at and m staying awake morning ',
'great',
'hurt my arm ',
'',
'tiiiiiiiiired going to bed ',
'just been in that kind of mood not reason at all lol but ll try not to be too mushy around you can behave ',
'i hope',
'keeps getting such delayed responses why is my internet so messed up ',
'and dont tell burnsy but no comparison between the rocky mountains and mountains in enlgland',
'sad '

```

**Figure 40:** X list

```

1 y=df['sentiment_num']

```

**Listing 5.31:** Define y

**Training test splitting:** In this step, we spilt X, y dataset into train and test datasets, like before 75% training data set and 25% test data set. The training dataset will be used to train deep learning models while the test dataset will be used to evaluate how well the model performs. We can use train\_test\_split method from the sklearn.model.selection module like before as shown below:

```

1 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
    =0.25, random_state=42)

```

**Listing 5.32:** Splitting training and test dataset

**Preparing the Embedding Layer:** In this section [58], we use the Tokenizer class from the keras.texts\_to\_sequences . Here the text module to create a word-to-index dictionary. In the word-to-index dictionary, each word in the corpus is used as a key, while a corresponding unique index is used as the value for the key. The following listing 5.33 is showing the Tokenize process:

```

1 tokenizer = Tokenizer(num_words=5000)
2 tokenizer.fit_on_texts(X_train)
3 X_train = tokenizer.texts_to_sequences(X_train)
4 X_test = tokenizer.texts_to_sequences(X_test)

```

**Listing 5.33:** Tokenize

The 'X\_train' variable has 20610 lists where each list contains integers now. Each list corresponds to each sentence in the training dataset and in test dataset has X\_test. The size of each list is different because each sentence has different length.

**Padding Process[58]:** In the padding process we use the maximum size of each list to 100. The lists with size greater than 100 will be truncated to 100. For the lists that have length less than 100, we will add 0 at the end of the list until it reaches the max length. We will also add 1 because of reserved 0 index. The following listing 5.34 script finds the vocabulary size and then perform padding on both training and test dataset:

```

1 #vocab_size is the vocabulary size.
2 vocab_size = len(tokenizer.word_index) + 1
3 maxlen = 100
4 X_train = pad_sequences(X_train, padding='post', maxlen=maxlen)
5 X_test = pad_sequences(X_test, padding='post', maxlen=maxlen)

```

**Listing 5.34:** Padding process

Now X\_train or X\_test will have all the lists have same length of 100. Also, the vocab\_size variable now contains a value 14132 i.e. the corpus has 14132 unique words.

### **GloVe embeddings loaded:**

In this section, shows Global Vectors for Word Representation(Glove)[59] and we install 'Glove\_python' and load the 'glove.6B.100d.txt' file in Google Colab. GloVe embeddings are used to create feature matrix. For that we need to load the GloVe word embeddings and create a dictionary which contains

words as keys and their corresponding embedding lists as values as shown in the following listing 5.35:

```
1 from numpy import array
2 from numpy import asarray
3 from numpy import zeros
4
5 embeddings_dictionary = dict()
6 glove_file = open('/content/glove.6B.100d.txt', encoding="utf8")
7
8 for line in glove_file:
9     records = line.split()
10    word = records[0]
11    vector_dimensions = asarray(records[1:], dtype='float32')
12    embeddings_dictionary[word] = vector_dimensions
13 glove_file.close()
```

**Listing 5.35:** Code for GloVe embeddings

**Embedding matrix creation[58]:** In the embedding matrix, each row number is equal to the index of the word in the corpus and the matrix has 100 columns and each column will contain the GloVe word embeddings for the words in corpus. Here 'embedding\_matrix' contains 14945 rows and 100 columns as shown in the following listing 5.36:

```
1 embedding_matrix = zeros((vocab_size, 100))
2 for word, index in tokenizer.word_index.items():
3     embedding_vector = embeddings_dictionary.get(word)
4     if embedding_vector is not None:
5         embedding_matrix[index] = embedding_vector
```

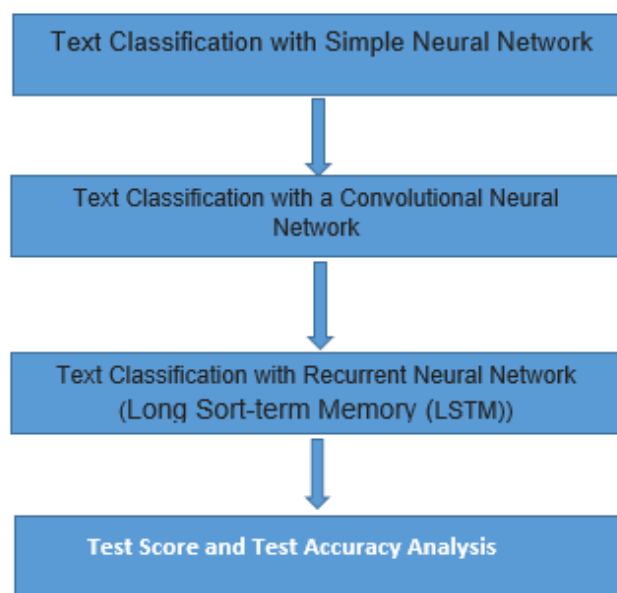
**Listing 5.36:** Code for embedding matrix

After generating the embedding matrix we can now build simple NN, CNN, and RNN deep learning models.

```
[ ] embedding_matrix.shape
(14945, 100)

▶ embedding_matrix
array([[ 0., 0., 0., ..., 0.,
        0., 0.],
       [-0.046539, 0.61966002, 0.56647003, ..., -0.37616,
        -0.032502, 0.80620003],
       [-0.18970001, 0.050024, 0.19084001, ..., -0.39804,
        0.47646999, -0.15983],
       ...,
       [ 0., 0., 0., ..., 0.,
        0., 0.],
       [-0.68193001, -0.28771001, 0.77758002, ..., 1.06729996,
        1.07920003, -1.10609996],
       [ 0., 0., 0., ..., 0.,
        0., 0.]])
```

**Figure 41:** Embedding matrix



**Figure 42:** Flow of Deep Learning Best model

### 5.2.2 The sentiment analysis with Simple Neural Network

The simple neural network[58] is developed in this section.

```

1 model = Sequential()
2 embedding_layer = Embedding(vocab_size, 100, weights=[embedding_matrix
    ], input_length=maxlen, trainable=False)
3 model.add(embedding_layer)
4 model.add(Flatten())
5 model.add(Dense(1, activation='sigmoid'))

```

**Listing 5.37:** Code for Simple Neural Network model

**Important steps define as follows:**

1. As shown in listing 5.37, a Sequential model is built
2. Create embedding layer. This embedding layer has an input length of 100, the output vector dimension will also be 100
3. Used the GloVe embedding
4. Set trainable to false and in the weights attribute we pass our own embedding matrix. The embedding layer is then added to our model
5. We are directly connecting our embedding layer to densely connected layer so we flatten the embedding layer
6. Finally, we add a dense layer with a sigmoid activation function.

**The adam optimizer :** Adam optimizer is used to compile our model, 'binary\_crossentropy' as our loss function and accuracy as metrics as shown in the following listing 5.38. Next, we will receive the summary of our model:

```

1 model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['
    acc'])
2 print(model.summary())

```

**Listing 5.38:** Code for Simple NN model summery

As shown in the following fig.43:

Since there are 1413200 words in the corpus and to the index of each word is represented as a 100-dimensional vector, the number of the trainable parameters

Model: "sequential"

Layer (type)	Output Shape	Param #
embedding (Embedding)	(None, 100, 100)	1413200
flatten (Flatten)	(None, 10000)	0
dense (Dense)	(None, 1)	10001
Total params: 1,423,201		
Trainable params: 10,001		
Non-trainable params: 1,413,200		
None		

**Figure 43:** Simple Neural Network model summery

will be 1413200x100 in the embedding layer. In the flattening layer, we simply multiply rows and columns. We have, in the dense layer the number of parameters is 10000 (from the flattening layer) and 1 for the bias parameter, for a total of 10001.

```
1 history = model.fit(X_train, y_train, batch_size=128, epochs=6,
    verbose=1, validation_split=0.25)
```

**Listing 5.39:** Code for Simple NN model training

**'fit' function:** To train neural network model, the 'fit' function is used. The 'validation\_split' of 0.25 means that 25% of the dataset is used to find the training accuracy of the algorithm like before.

To evaluate the performance of the model, we can simply pass the test set to the evaluate method of our model and to check the test accuracy and loss, executed the following script:

```
1 score = model.evaluate(X_test, y_test, verbose=1)
2 print("Test Score:", score[0])
3 print("Test Accuracy:", score[1])
4
5 Output :
6 Test Score: -0.850335955619812
```

```
7 Test Accuracy: 0.48660844564437866
```

**Listing 5.40:** Code for Simple NN model Evaluate

Once executed, the above script achieved 48.66% test accuracy and 48.21% training accuracy. This means that this model is not overfitting on the training dataset. 48.21%. We can see the plots for loss and accuracy differences for training and test datasets by executing the following script:

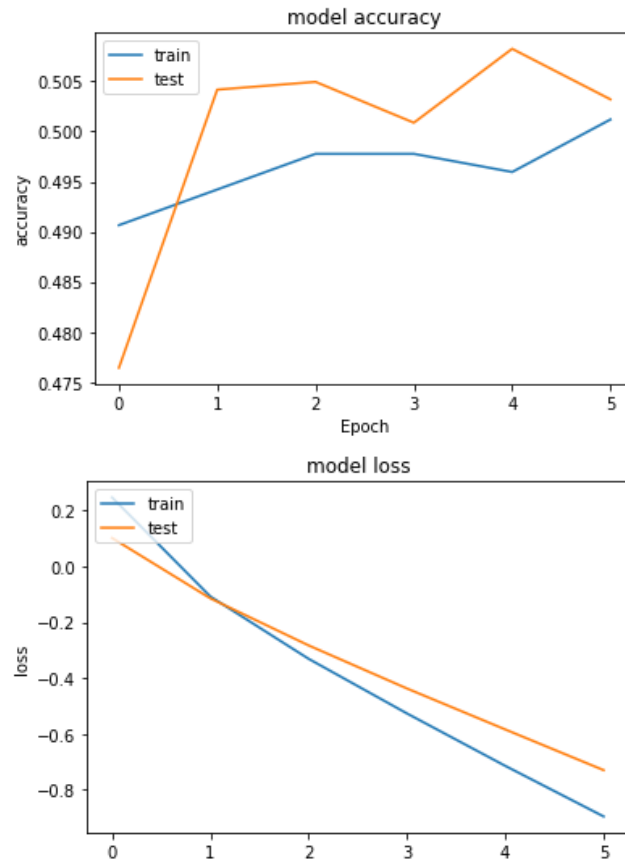
```
1 import matplotlib.pyplot as plt
2 plt.plot(history.history['acc'])
3 plt.plot(history.history['val_acc'])
4 plt.title('model accuracy')
5 plt.ylabel('accuracy')
6 plt.xlabel('Epoch')
7 plt.legend(['train', 'test'], loc='upper left')
8 plt.show()
9 plt.plot(history.history['loss'])
10 plt.plot(history.history['val_loss'])
11 plt.title('model loss')
12 plt.ylabel('loss')
13 plt.xlabel('epoch')
14 plt.legend(['train', 'test'], loc='upper left')
15 plt.show()
```

**Listing 5.41:** Plot of the loss and accuracy differences for training and test sets

We can see the differences for loss and accuracy between the training and test datasets which is not so high as shown in the following fig.44:

### 5.2.3 The sentiment analysis with Convolutional Neural Network

In the previous chapter, we already discussed the Convolutional Neural Network. Here the implementation of a simple convolutional neural network with 1 convolutional layer and 1 pooling layer has shown [58]. The code up to the creation of the embedding layer will remain the same, to create of the embedding layer we use the following listing 5.42:



**Figure 44:** Output plot for loss and accuracy between the training and test sets of Simple neural network

```

1 import numpy
2 from keras.models import Sequential
3 from keras.layers.convolutional import Conv1D
4 numpy.random.seed(7)
5 model = Sequential()
6 embedding_layer = Embedding(vocab_size, 100, weights=[embedding_matrix
7                             ], input_length=maxlen , trainable=False)
8 model.add(embedding_layer)
9 model.add(Conv1D(128, 5, activation='relu'))
10 model.add(GlobalMaxPooling1D())
11 model.add(Dense(1, activation='sigmoid'))
12 model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['

```



```
acc']])
```

**Listing 5.42:** Code for create the convolutional model

Here we created a sequential model, followed by an embedding layer after that we created one-dimensional convolutional layer with 128 features or kernels. The kernel size is 5 and the activation function used is sigmoid. To reduce feature size finally we add a global max-pooling layer. Next a dense layer is added with sigmoid activation. The compilation process is the same as it was in the previous section.

The summary of our CNN model is as follows:

```
[41] print(model.summary())
```

Model: "sequential\_1"

Layer (type)	Output Shape	Param #
embedding_1 (Embedding)	(None, 100, 100)	1413200
conv1d (Conv1D)	(None, 96, 128)	64128
global_max_pooling1d (Global	(None, 128)	0
dense_1 (Dense)	(None, 1)	129

=====  
Total params: 1,477,457  
Trainable params: 64,257  
Non-trainable params: 1,413,200  
=====  
None

**Figure 45:** Summary of our CNN model

In Convolutional neural network we do not need to flatten our embedding layer and the feature size is now reduced using the pooling layer. To train our model and evaluate it on the training set the process remains the same. For that, we can use the fit and evaluate methods, respectively.

```

1 history = model.fit(X_train, y_train, batch_size=128, epochs=6,
    verbose=1, validation_split=0.2)
2 score = model.evaluate(X_test, y_test, verbose=1)

```

**Listing 5.43:** Code for fit and evaluate CNN methods

The following listing 5.44 prints the results and fig.46 shows the plots:

```

1 print("Test Score:", score[0])
2 print("Test Accuracy:", score[1])
3 Output:
4 Test Score: -189.70225524902344
5 Test Accuracy: 0.48515284061431885

```

**Listing 5.44:** The results of evaluate CNN methods

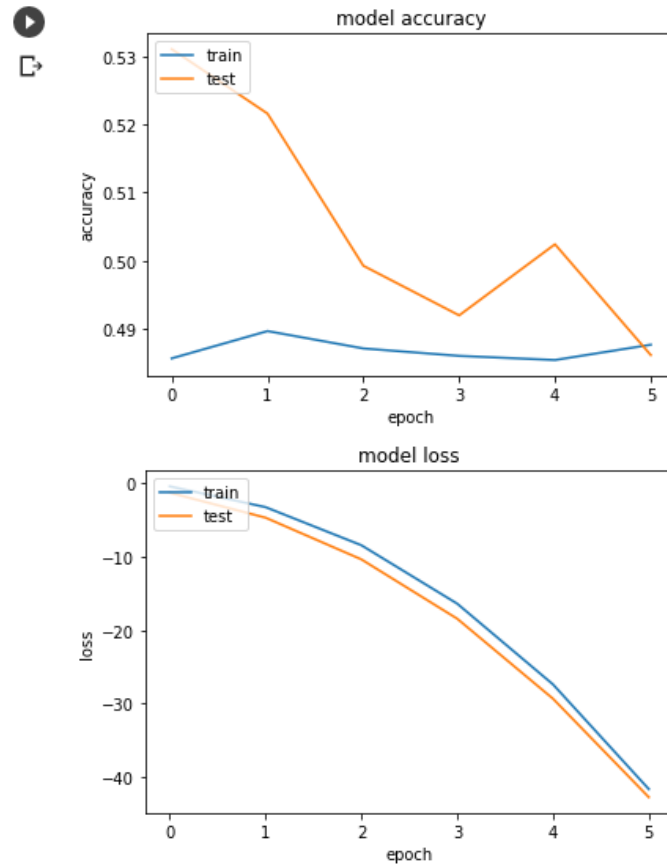
```

1 import matplotlib.pyplot as plt
2 plt.plot(history.history['acc'])
3 plt.plot(history.history['val_acc'])
4 plt.title('model accuracy')
5 plt.ylabel('accuracy')
6 plt.xlabel('epoch')
7 plt.legend(['train', 'test'], loc = 'upper left')
8 plt.show()
9 plt.plot(history.history['loss'])
10 plt.plot(history.history['val_loss'])
11 plt.title('model loss')
12 plt.ylabel('loss')
13 plt.xlabel('epoch')
14 plt.legend(['train', 'test'], loc = 'upper left')
15 plt.show()

```

**Listing 5.45:** The resulting plot code of CNN model

If we compare the training and test accuracy, we will see that the training accuracy for CNN will be around 49%, which is little greater than the training accuracy of the simple neural network. The test accuracy is around 48% for



**Figure 46:** The resulting code of CNN model

the CNN, which is also little less than the test accuracy for the simple neural network, which was around 48%. Our CNN model has not a big difference between the training and test accuracy. So the CNN model is also not overfitting. As shown in fig.46 plots the loss and accuracy and there is less difference between the training and test dataset.

#### 5.2.4 Sentiment Analysis with Recurrent Neural Network model

In this section, LSTM (Long Short Term Memory network) model designed[58], which is a variant of RNN, to solve sentiment classification problem. The batch size is 128, whereas the number of epochs is 6. We need to execute code until the word embedding section and then we can create the model of RNN(LSTM).

As shown in following listing 5.46:

```

1  from keras.layers.core import Dense, Dropout
2  from keras.layers.recurrent import LSTM
3  model = Sequential()
4  embedding_layer = Embedding(vocab_size, 100, weights=[embedding_matrix
5                             ], input_length=maxlen, trainable=False)
6  model.add(embedding_layer)
7  model.add(LSTM(128))
8  model.add(Dense(1, activation='sigmoid'))
9  model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['
    acc'])

```

**Listing 5.46:** The code of RNN(LSTM) model

We initiate a sequential model which is followed by the creation of the embedding layer and a LSTM layer creation with 128 neurons. The rest of the code is same as it was for the CNN model.

As shown in fig.47 the LSTM model summary looks like this:

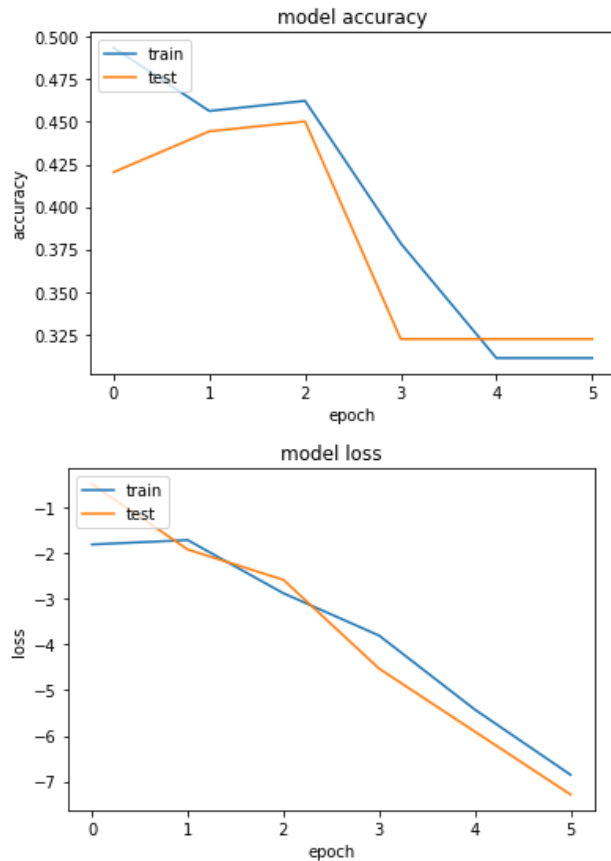
```
[90] print(model.summary())
```

Model: "sequential\_3"

Layer (type)	Output Shape	Param #
embedding_3 (Embedding)	(None, 100, 100)	1413200
lstm (LSTM)	(None, 128)	117248
dense_3 (Dense)	(None, 1)	129
Total params: 1,530,577		
Trainable params: 117,377		
Non-trainable params: 1,413,200		
None		

**Figure 47:** LSTM model summery

The next step is to train the model on the training dataset and evaluate its performance on the test dataset. Once the model is trained, we can see the LSTM model's results on the test dataset. As show in the listing 5.47 as



**Figure 48:** Result Plot for the LSTM model

follows:

```
1 history = model.fit(X_train, y_train, batch_size=128, epochs=6,
2                     verbose=1, validation_split=0.2)
3 score = model.evaluate(X_test, y_test, verbose=1)
```

**Listing 5.47:** RNN(LSTM) model

```
1 print("Test Score:", score[0])
2 print("Test Accuracy:", score[1])
3 Output:
4 Test Score: -7.136790752410889
5 Test Accuracy: 0.30960699915885925
```

**Listing 5.48:** RNN(LSTM) model results

The above script prints the results of LSTM:

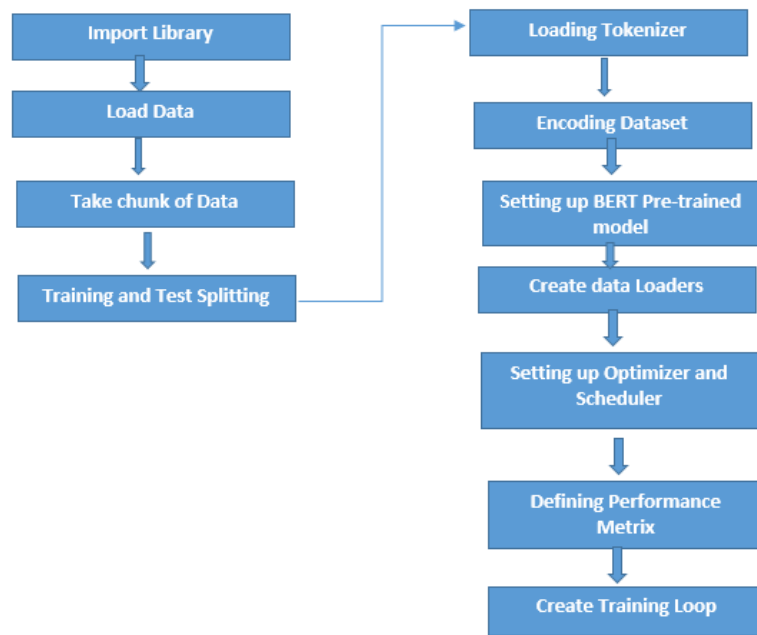
We found the training accuracy of LSTM around 32% and test accuracy around 30.09%, as shown in fig. 48. We observed that the test accuracy is around 30.09%. The test accuracy is not better than both the CNN and densely connected neural network. There is a very small difference between the training accuracy and test accuracy which means that our model is not overfitting.

### 5.3 Sentiment Analysis with BERT Models

In the previous chapter 3 we already discussed the BERT model. 'BERT\_Base\_uncased' is a large-scale transformer-based language model that can be fine-tuned for a variety of tasks. Because of large dataset 27481 rows and 4 columns, the execution was failed because of the limited capacity of system i.e. RAM 8GB, processor Intel(R) Core(TM)i3-6100U CPU 2.3GHz used with Jupyter notebook for training steps with BERT model in personal laptop. So I decided to work with a small chunk from the original data in Google Colab, GPU services which is free of cost and in the same personal laptop it produced better results. Following steps are done step by step as shown in fig.49:

#### **Steps for the Bert Model:**

1. Data Preprocessing
2. Training/Validation Split
3. Loading Tokenizer and Encoding our Data
4. Setting up BERT Pretrained Model
5. Creating Data Loaders
6. Setting Up Optimizer and Scheduler
7. Defining our Performance Metrics
8. Creating our Training Loop



**Figure 49:** Flowchart of BERT model

### 5.3.1 Exploratory Data Analysis and Preprocessing

The following listing 5.49 shows us the list of library which is imported for the BERT model:

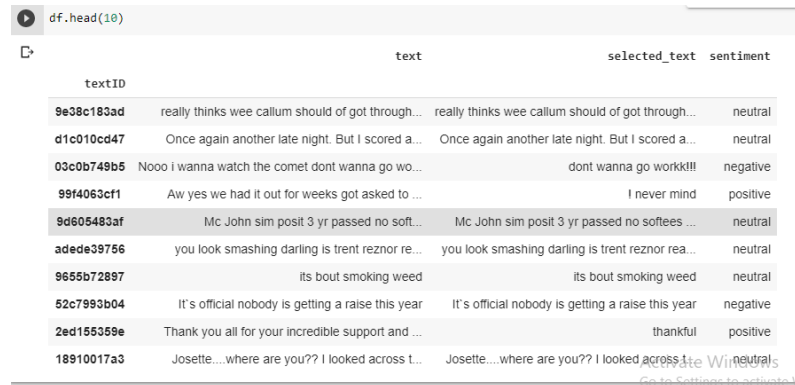
```

1 import pandas as pd
2 import torch
3 import pandas as pd
4 from tqdm.notebook import tqdm
5 from sklearn.model_selection import train_test_split
6 from transformers import BertTokenizer
7 from torch.utils.data import TensorDataset
8 from transformers import BertForSequenceClassification
9 from torch.utils.data import DataLoader, RandomSampler,
   SequentialSampler
10 from transformers import AdamW, get_linear_schedule_with_warmup
11 import numpy as np
12 from sklearn.metrics import f1_score
13 import random

```

### Listing 5.49: Import libraries for Bert model

The following listing 5.50 is showing the loaded data 'train1' into Google Colab notebook as a new data frame name 'df':



	textID	text	selected_text	sentiment
	9e38c183ad	really thinks wee callum should of got through...	really thinks wee callum should of got through...	neutral
	d1c010cd47	Once again another late night. But I scored a...	Once again another late night. But I scored a...	neutral
	03c0b749b5	Nooo i wanna watch the comet dont wanna go wo...	dont wanna go workd!!!	negative
	99f4063cf1	Aw yes we had it out for weeks got asked to ...	I never mind	positive
	9d605493af	Mc John sim posit 3 yr passed no soft...	Mc John sim posit 3 yr passed no softees ...	neutral
	adee39756	you look smashing darling is trent reznor re...	you look smashing darling is trent reznor rea...	neutral
	9655b72897	its bout smoking weed	its bout smoking weed	neutral
	52c7993b04	It's official nobody is getting a raise this year	It's official nobody is getting a raise this year	negative
	2ed155359e	Thank you all for your incredible support and ...	thankful	positive
	18910017a3	Josette....where are you?? I looked across t...	Josette....where are you?? I looked across the Win...	neutral

Figure 50: 'df' DataFrame

Here I created a chunk size of 10000 records where our total data size is 27480 records after removing one missing value. The listing 5.51 shows us the dataset which has 7481 rows and the name of the data chunk is 'train1'. This 'train1' dataset is splitted into the training dataset and the test dataset.

```
1 chunk_size = 10000
2 batch_num = 1
3 for chunk in pd.read_csv('/content/train.csv', chunksize=chunk_size):
4     chunk.to_csv('train' + str(batch_num) + '.csv', index=False)
5     batch_num += 1
```

Listing 5.50: Create small chunk from large data

```
1 df = pd.read_csv('/content/train1.csv')
2 df.set_index('textID', inplace=True)
```

Listing 5.51: Code for DataFrame 'df'

```
1 len(df) Output: 7481
```

Listing 5.52: Number of observations



The new DataFrame 'df' which is created by 'train1' chunk is containing the text data in each index and there are 3051 neutral , 2268 positive and 2162 negative comments in the whole dataset of 7481 records. The number of observations has 7481 rows as shown in the following listing 5.52:

```
1 df.selected_text.iloc[0]
2 Output:
3 'really thinks wee callum should of got through last night'
4 df.sentiment.value_counts()
5 Output:
6 neutral      3051
7 positive     2268
8 negative     2162
9 Name: sentiment, dtype: int64
```

**Listing 5.53:** Observations

After removing the missing value from the DataFrame 'df' by using the previous approach the above list shows the unique label of dataset and we observed here again three classes of sentiment i.e. multiclass. As we already observed before that 'sentiment' column is a dependent variable and our independent variable is 'selected\_text' column. Here the listing 5.54 converted the 'sentiment' column's text values to numerical values neutral= 0, positive = 2, negative= 1 automatically as shown in fig.51 and keep all the data inside DataFrame in a new column named 'label\_dict'.

```
1 possible_labels = df.sentiment.unique()
2 possible_labels
3 Output:
4 array(['neutral', 'negative', 'positive'], dtype=object)
```

**Listing 5.54:** Check the unique labels

Enumerate loops which returns the index that we are currently on. The first tuple receive, will be the first possible -labels , first index is zero and second is 1 and so on. We created the dictionary dict[], set to the index and add this

data to the label dict[]. Now add this value to the column label. As shown in listing 5.55 the mapping of data as Follows:

```
1 label_dict = {}
2 for index, possible_label in enumerate(possible_labels):
3     label_dict[possible_label] = index
4
```

**Listing 5.55:** Mapping data

The following listing 5.56 is showing how the label replaced with numerical value:

```
1 df['label'] = df.sentiment.replace(label_dict)
```

**Listing 5.56:** Label replace with numerical value

### 5.3.2 Training/Validation Split

In this section, we split our DataFrame 'df' in four parts X\_train, X\_val, y\_train, y\_val where we have 75% training dataset and 25% of test dataset as shown in following listing 5.57:

```
1 X_train, X_val, y_train, y_val = train_test_split(df.index.values, df
    .label.values, test_size=0.25, random_state=17, stratify=df.label.
    values)
```

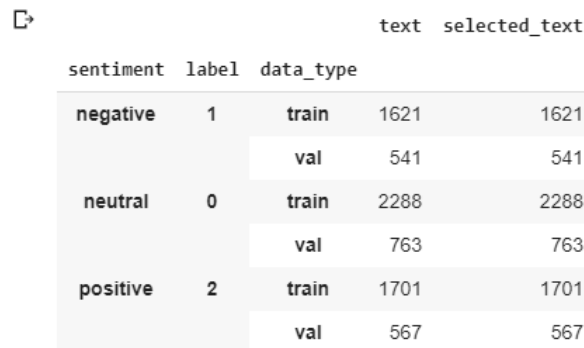
**Listing 5.57:** Training/Validation Split

Here the training and the test datasets are distributed in a way, that both of them must contain positive, negative, and neutral labels. This is called the 'Stratify' approach.

```
1 df['data_type'] = ['not_set']*df.shape[0]
2 df.loc[X_train, 'data_type'] = 'train'
3 df.loc[X_val, 'data_type'] = 'val'
4 df.groupby(['sentiment', 'label', 'data_type']).count()
```

**Listing 5.58:** DataFrame that all the data in the list of X\_train will retrieval

The fig.51 is showing the DataFrame, which has the grouping based on count using the 'groupby()' method.



			text	selected_text
sentiment	label	data_type		
negative	1	train	1621	1621
		val	541	541
neutral	0	train	2288	2288
		val	763	763
positive	2	train	1701	1701
		val	567	567

**Figure 51:** 'df' DataFrame details

### 5.3.3 Loading Tokenizer and Encoding our Data

In this section, we installed the 'transformers' for 'the hugging face' library by using the list command in Google Colab notebook as shown in the following listing 5.59:

```
1 !pip install transformers
```

**Listing 5.59:** Installing hugging face libraries

Tokenizer takes raw texts from 'train' dataset and splits it into tokens. A token is a numerical value which represents a certain word. We used 'Bert base uncased' method. The following listing 5.60 is used to create data tokens:

```
1 tokenizer = BertTokenizer.from_pretrained('bert-base-uncased',
    do_lower_case=True)
```

**Listing 5.60:** Code for loading Tokenizer

The listing 5.61 shows the code, which converted all the data from sentence to encoded form and we used batch encode. Batch can take multiple strings and convert them as we need them for the next step.

The first parameter is the actual data that we have 'train' inside 'data\_type'. Here 'add\_special\_tokens' is used to identify when a sentence starts and when it ends.

Next, we need to return the attention mask. 'attention\_mask' tells us that we are using a fixed length of input. For example if we are using a sentence with 5 words and another sentence has 15 words then it will create a situation where we do not have same dimension of sentences. The solution is 'max\_length' which is fixed length of 256 for all sentences. Here 'max\_length' can contain a large value and attention mask just tell us what length of actual value of a sentence and keep zeroes on the rest of the places of a sentence. We have built the same structure for the training dataset. Here 'encoded\_data\_val' actually returns the 'input\_ids' which returns the numbers for each word.

```
1 encoded_data_train = tokenizer.batch_encode_plus(  
2     df[df.data_type=='train'].selected_text.values,  
3     add_special_tokens=True,  
4     return_attention_mask=True,  
5     pad_to_max_length=True,  
6     max_length=256,  
7     return_tensors='pt'  
8 )  
9 encoded_data_val = tokenizer.batch_encode_plus(  
10    df[df.data_type=='val'].selected_text.values,  
11    add_special_tokens=True,  
12    return_attention_mask=True,  
13    pad_to_max_length=True,  
14    max_length=256,  
15    return_tensors='pt'  
16 )  
17 input_ids_train = encoded_data_train['input_ids']  
18 attention_masks_train = encoded_data_train['attention_mask']  
19 labels_train = torch.tensor(df[df.data_type=='train'].label.values)  
20 input_ids_val = encoded_data_val['input_ids']  
21 attention_masks_val = encoded_data_val['attention_mask']  
22 labels_val = torch.tensor(df[df.data_type=='val'].label.values)
```

---

**Listing 5.61:** Code for Encoded the train dataset

We get two datasets 'dataset\_train' and dataset\_val' for training and test, the model which is converted into tensors data format by using the following listing 5.61:

```
1 dataset_train = TensorDataset(input_ids_train, attention_masks_train,
    labels_train)
2 dataset_val = TensorDataset(input_ids_val, attention_masks_val,
    labels_val)
```

**Listing 5.62:** Code for converted the converted numerical data into Tensor data

The following listing 5.62 shows that we have now training dataset 'dataset\_train' which has 5610 rows and test data 'dataset\_val' has 1871 rows where the total DataFrame has 7481 rows in total.

```
1 len(dataset_train)
2 Output: 5610
3 len(dataset_val)
4 Output: 1871
5 len(df)
6 Output: 7481
```

**Listing 5.63:** Chunk size of DataFrame

### 5.3.4 Setting up BERT Pre-trained Model

To set up the Bert pre-trained model as shown in the following list we need to call the BertForSequenceClassificaton[35] package from transformers which is Hugging Face module. We just need to run the Google Colab cell with the listing 5.64 and we treat each task as its own unique sequence. So one sequence is classified into one of three classes which we have already seen in previous section and this is our fine-Tuning step. This is the point where we redefined the architecture to include the part that we need.

Here 'num\_labels' says how many output labels must this final layer of Bert model have, will be able to classify and 'output\_hidden\_states' is false because we do not really want to have any empty unnecessary input.

The basic idea of Attention is that I predicted the words based on the five words related to it in my input and the other words it might have. The 'output\_hidden\_states' works just before of the 'Attention' to the actually predicted value.

```
1 model = BertForSequenceClassification.from_pretrained("bert-base-uncased", num_labels=len(label_dict), output_attentions=False, output_hidden_states=False)
```

**Listing 5.64:** Code for setting up BERT Pretrained Model

The fine-tuning step is the addition of a layer on the top of the size, we have three different classes. We need to be able to predict and it is just going to be a classifier that will predict, in one of the three values. We have to pick the correct one. So we set up our model and now it is ready to be trained essentially.

### 5.3.5 Creating Data Loaders

In this section, we create a simple data loader, which offers a way to iterate through our dataset in batches. Here I import the batch are RandomSampler, SequentialSampler. So we are just using RandomSampler for training. It just randomized how our model is training and what data is being exposed to. SequentialSampler is used for our validation dataset we do not really care about how data is and it randomly sorting our dataset. Because of limited memory space capacity have taken as 'batch\_size' is 10 but it be taken more also, 'Random\_Sampler()' randomize our 'training\_dataset' and it gives the output of the dataset, which is ready to go for the next step. As shown in the listing 5.65 as follows:

```
1 batch_size = 10
```

```

2 dataloader_train = DataLoader(dataset_train, sampler=RandomSampler(
    dataset_train),
3 batch_size=batch_size)
4 dataloader_validation = DataLoader(dataset_val, sampler=
    SequentialSampler(dataset_val), batch_size=batch_size)

```

**Listing 5.65:** Code for creating Data Loaders

### 5.3.6 Setting Up Optimizer and Scheduler

In this section, we setting up an optimizer and scheduler. An optimizer is to find our learning rate and how it changes through each epoch very simply and 'AdamW' is a way of optimizing our weight, so this is a casting optimization approach. Finally, our scheduler which controls the learning rate as shown in listing 5.66 as follows:

```

1 optimizer = AdamW(model.parameters(), lr=1e-5, eps=1e-8)

```

**Listing 5.66:** code for setting Up Optimizer

We have taken five epochs to train our dataset. Here have the length of 'dataloader\_train' which is multiplied with the epochs number i.e. how many training steps we will used and how many learning steps is actually need to change. Here we finalize what optimizer 'Adamw' is used to solve our data sentiment to find a approximation our dataset, in terms of optimizing in a way to give optimal output. As shown in the following listing 5.67:

```

1 epochs = 5
2 scheduler = get_linear_schedule_with_warmup(optimizer,
    num_warmup_steps=0, num_training_steps=len(dataloader_train)*
    epochs)

```

**Listing 5.67:** Code for setting Up Scheduler

The scheduler is here just to learning rate as training goes on based on how is it is going on.

### 5.3.7 Defining our Performance Metrics

Here 'preds\_flat' contains the fractional number such as 0.9, 0.5, 0.1 and so on. Here we have to convert this fractional numbers in to binary numbers. 'f1\_score()' method is used to just to receive 'preds' is a prediction and labels being the true labels. As the number of epochs is rises the accuracy of the model is improved and suddenly it comes to a threshold point of accuracy.

```
1 def f1_score_func(preds, labels):
2     preds_flat = np.argmax(preds, axis=1).flatten() #a single array
3     labels_flat = labels.flatten()
4     return f1_score(labels_flat, preds_flat, average='weighted')
```

**Listing 5.68:** Code for define our performance rate

```
1 def accuracy_per_class(preds, labels):
2     label_dict_inverse = {v: k for k, v in label_dict.items()}
3     preds_flat = np.argmax(preds, axis=1).flatten()
4     labels_flat = labels.flatten()
5     for label in np.unique(labels_flat):
6         y_preds = preds_flat[labels_flat==label]
7         y_true = labels_flat[labels_flat==label]
8         print(f'Class: {label_dict_inverse[label]}')
9         print(f'Accuracy: {len(y_preds[y_preds==label])}/{len(y_true)}\n')
```

**Listing 5.69:** Code for predict accuracy

Here we have a single array. The recommended random seed is 'seed\_val = 17' and it could be hard to train data if it goes to CPU and with more seed\_val. As shown in the following listing 5.70:

### 5.3.8 Creating our Training Loop

```
1 seed_val = 17
2 random.seed(seed_val)
3 np.random.seed(seed_val)
```



```

4 torch.manual_seed(seed_val)
5 torch.cuda.manual_seed_all(seed_val)

```

**Listing 5.70:** Code for setting random seeds

As we know the performance of the model depends on the capacity of the 'CPU' or 'GPU' the following sequence of code is search for the type of device we are using for a notebook to execute our code. In our case we use 'cuda' in Google Colab which is free of charge service.

```

1 device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
2 model.to(device)
3 print(device)

```

**Listing 5.71:** Code to detect device

This part of code we have a function 'evaluate()' which evaluate the model based on training dataset as given the input is 'input\_ids', 'attention\_mask' and 'labels' dataset as shown in the following listing 5.72:

```

1 def evaluate(dataloader_val):
2     model.eval()
3     loss_val_total = 0
4     predictions, true_vals = [], []
5     for batch in dataloader_val:
6         batch = tuple(b.to(device) for b in batch)
7         inputs = {'input_ids':      batch[0],
8                  'attention_mask': batch[1],
9                  'labels':         batch[2],
10                 }
11         with torch.no_grad():
12             outputs = model(**inputs)
13             loss = outputs[0]
14             logits = outputs[1]
15             loss_val_total += loss.item()
16             logits = logits.detach().cpu().numpy()
17             label_ids = inputs['labels'].cpu().numpy()
18             predictions.append(logits)

```

```

19     true_vals.append(label_ids)
20     loss_val_avg = loss_val_total/len(dataloader_val)
21     predictions = np.concatenate(predictions, axis=0)
22     true_vals = np.concatenate(true_vals, axis=0)
23     return loss_val_avg, predictions, true_vals

```

**Listing 5.72:** Code for Loading and Evaluation of the model

Next, predict the output what is the sentiment of each sentence in an array as 'label\_ids' depend on the value for the inputs 'labels'. Output will also calculate the loss, logits and loss\_value\_total. As shown in the following listing 5.73:

```

1  for epoch in tqdm(range(1, epochs+1)):
2      model.train()
3      loss_train_total = 0
4      progress_bar = tqdm(dataloader_train, desc='Epoch {:1d}'.format(
epoch), leave=False, disable=False)
5      for batch in progress_bar:
6          model.zero_grad()
7          batch = tuple(b.to(device) for b in batch)
8          inputs = {'input_ids':      batch[0],
9                    'attention_mask': batch[1],
10                   'labels':         batch[2],
11                   }
12          outputs = model(**inputs)
13          loss = outputs[0]
14          loss_train_total += loss.item()
15          loss.backward()
16          torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)
17          optimizer.step()
18          scheduler.step()
19          progress_bar.set_postfix({'training_loss': '{:.3f}'.format(
loss.item()/len(batch))})
20          torch.save(model.state_dict(), f'finetuned_BERT_epoch_{epoch}.
model')
21          tqdm.write(f'\nEpoch {epoch}')

```

```

22     loss_train_avg = loss_train_total/len(dataloader_train)
23     tqdm.write(f'Training loss: {loss_train_avg}')
24     val_loss, predictions, true_vals = evaluate(dataloader_validation)
25     val_f1 = f1_score_func(predictions, true_vals)
26     tqdm.write(f'Validation loss: {val_loss}')
27     tqdm.write(f'F1 Score (Weighted): {val_f1}')

```

**Listing 5.73:** New model upload to achieve predicted result

Predicted output is the labels of the text in a numerical form, which is actually the sentiment of the text value.

### 5.3.9 Results of Bert Model

We can observed the accuracy of epoch 1 to epoch 4 and then it stable with 88.5% accuracy which is better result than all other models as shows in the following listing 5.74:

```

1 Epoch 1
2 Training loss: 0.5332776628129198
3 Validation loss: 0.34617412270621417
4 F1 Score (Weighted): 0.8694751523343955
5
6 Epoch 2
7 Training loss: 0.2894711494943834
8 Validation loss: 0.3903133689990624
9 F1 Score (Weighted): 0.8806266194550348
10
11 Epoch 3
12 Training loss: 0.19923969274370162
13 Validation loss: 0.44232756839668813
14 F1 Score (Weighted): 0.8801663818224016
15
16 Epoch 4
17 Training loss: 0.14858646519189955
18 Validation loss: 0.4809418695513159
19 F1 Score (Weighted): 0.8895254889132972

```

```

20
21 Epoch 5
22 Training loss: 0.103441213461597
23 Validation loss: 0.5122659648954373
24 F1 Score (Weighted): 0.8856181176739232

```

**Listing 5.74:** Results improvement per epoch

Load the trained model that we achieved in the last section in Google Colab , where we trained the model file with the number of epochs. As we have five epoch we have five files. Here I took the 'epoch 5' file which is 'finetuned\_BERT\_epoch\_5.model'. As shows in the following listing 5.75:

```

1 model.load_state_dict(torch.load('/content/finetuned_BERT_epoch_5.
    model', map_location=torch.device('cuda')))
2
3 Output:
4 <All keys matched successfully>

```

**Listing 5.75:** Upload the pre trained\_model

Once the every keys matched successfully we can run the lising 5.76 to predict the result and neutral class has 773 data but it gives 665 outputs which is 86.02%, Negative class has 541 data but it gives output 480 data which is 88.724% and we have positive 567 data but we achieved 512 which is 90.299%.

```

1 _, predictions, true_vals = evaluate(dataloader_validation)

```

**Listing 5.76:** Final result prediction

```

1 accuracy_per_class(predictions, true_vals)
2
3 Output:
4 Class: neutral
5 Accuracy: 665/763
6 Class: negative
7 Accuracy: 480/541
8 Class: positive

```

```
9 Accuracy: 512/567
```

**Listing 5.77:** Predicted accuracy for different sentiment

## 5.4 Result Analysis for all models

As we can see in the following table 2, which compared to the all machine learning models and the best result is achieved by the Logistic Regression which has 78% accuracy but when we analyze the confusion matrix we have seen that the Decision Tree model have accurate result than Naïve Bayes and Logistic Regression for our dataset. Finally, we can say these models are well executed with our local personal computer power. We do not need to make a small chunk of dataset.

On the other hand, we have observed that compared to Simple NN, CNN, and LSTM model, CNN has around 49% accuracy which is better than simple NN and LSTM. We all observed that these three model simple NN, CNN, LSTM are not overfitted to our training dataset though they do not produce the best accuracy for our dataset. At the same time the crucial point is for the 'Bert\_Base \_Fine\_Tuning' model has a very good result compared to other models, which is 88.56% accuracy overall sentiments but the biggest problem is we were not able to upload the full dataset of 27481 data for the model. We were creating a small chunk size of 10000 of data.

Result analysis of all the model		
Model Type	Model name	Accuracy
Machine Learning model	Naïve Base Algorithm	78.31%
	Logistic Regression	78.85%
	Decision Tree	75.96%
Deep Learning Model	Simple Neural Network	48.66%
	CNN	49%
	LSTM	32.50%
Bert Base Fine Tuning model	Bert model	88.56%
Individual texts sentiment for Bert model	Neutral	86.028%
	negative	88.724%
	positive	90.29%

**Table 2:** Result analysis of all the model

## 6 Conclusion

This masters thesis project is based on NLP text classification, i.e sentiment analysis of multiclass text data. Different machine learning and deep learning models have used in the training dataset to find out the best model, based on accuracy. In this case, Bert model attains best results with 88.65% accuracy. Even though there were limitations, we still thrived to find the best model for the dataset.

Finally, in comparison of table1 with table 2 from related work, based on server capacity or processing power model gives better results. Depending on the available capacity of Google Colab and personal laptop the 'Bert\_Base\_uncased' model is best for our dataset.

### 6.0.1 Limitations

Bert model gives us best accuracy but it has some strong limitations too, such as, due to technical difficulties, the whole dataset was fragmented for analysis. In this thesis project, different models where used in the dataset but nothing in particular was applied for techniques to improve the model or error correcting process, since Bert model yields better results. Moreover NN, CNN and RNN(LSTM) was not overfitted to the dataset but they had low accuracy. There was no GPT3 model done in this thesis project.

### 6.0.2 Problem Faced

Most of all, during this whole period of thesis all the libraries and university labs were closed due to COVID-19 pandemic. As a result only online equipment's

and personal laptops were used for the thesis project. Using personal laptop with 8GB RAM had lots of technical difficulties with 'CPU' while implementing CNN, RNN, NN and Bert model.

Finally, after long research, I started working with Google Colab. It was easy to install Keras, Tensorflow and HuggingFace libraries. In Google colab used 'GPU' and small chunks of dataset for Bert model. But still, experimenting with 'TPU' power was not possible since it is a paid service.

### **6.0.3 Future Plan**

This thesis project is done in English Language text classification. In future, we would like to extend for Multi-class classification based on sentiment analysis on different languages such as Bengoli, German and French language. If necessary GPT3 model can be used to handle the huge dataset.



# Bibliography

- [1] Kaggle, “Tweet sentiment extraction.” <https://www.kaggle.com/c/tweet-sentiment-extraction/overview>, 2020. (Accessed on 05/29/2020).
- [2] E. C. N. N. J. G. Shervin Minaee, Nal Kalchbrenner, “Deep learning based text classification: A comprehensive review,” *Publication rights licensed to ACM*, April 2020.
- [3] F. Chollet, “Deep learning with python.”
- [4] D. S. S. Siddharth, R. Darsini, “Sentiment analysis on tweeter data using machine learning algorithm in python.,” *Review of Scientific Instruments*, February 2018.
- [5] J. W. V. V. Siqi Zhao, Lin Zhong, “Analyzing twitter for social tv: Sentiment extraction for sports.”
- [6] “Wikipedia.” [https://en.wikipedia.org/wiki/Deep\\_learning](https://en.wikipedia.org/wiki/Deep_learning). (Accessed on 07/25/2020).
- [7] S. S. Medha Khurana, Anurag Gulati, “Sentiment analysis framework of twitter data using classification,” *IEEE*, 2018.
- [8] S. K. S. P. V. J. Meghana Ashok, Swathi Rajanna, “A personalized recommender system using machine learning based sentiment analysis over social data,” *IEEE*, 2016.
- [9] S. B. Aliza Sarlan, Chayanit Nadam, “Twitter sentiment analysis,” *IEEE*, 2014.

- [10] F. D. I. P. Nhan Cach Dang, María N. Moreno-García, “Sentiment analysis based on deep learning: A comparative study,” *electronics*, 14 March 2020.
- [11] A. M. Mohammed Jabreel, “A deep learning-based approach for multi-label emotion classification in tweets,” *Applied Science*, March 2019.
- [12] B. Kim, “Attention in neural networks - 1. introduction to attention mechanism.” <https://buomsoo-kim.github.io/attention/2020/01/01/Attention-mechanism-1.md/>. (Accessed on 01/09/2020).
- [13] P. N. Sara Rosenthal, Noura Farra, “Semeval-2017 task 4: Sentiment analysis in twitter,” December 2019.
- [14] A. Géron, *Hands-On Machine Learning with Scikit-Learn TensorFlow*. O’Reilly Media Inc, 2017.
- [15] D. Fumo, “Types of machine learning algorithms you should know.” <https://towardsdatascience.com/types-of-machine-learning-algorithms-you-should-know-953a08248861>. (Accessed on 02/09/2020).
- [16] Wikipedia, “Logistic regression.” [https://simple.wikipedia.org/wiki/Logistic\\_Regression](https://simple.wikipedia.org/wiki/Logistic_Regression).
- [17] D. L. Yse, “The complete guide to decision trees.” <https://opendatascience.com/the-complete-guide-to-decision-trees-part-1/>. (Accessed on 07/27/2020).
- [18] “unsupervised machine learning.” <https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/>. (Accessed on 07/25/2020).
- [19] F. Malik, “What are hidden layers?.” <https://medium.com/fintechexplained/what-are-hidden-layers-4f54f7328263>. (Accessed on 02/09/2020).

- [20] C. U. Press, “Text classification and naive bayes.” <https://nlp.stanford.edu/IR-book/pdf/13bayes.pdf>. (Accessed on 07/26/2020).
- [21] P. D. C. Andersson, “Logistic regression, lecture notes.” fb2, University of Applied Sciences, Frankfurt/M, Germany., 2014.
- [22] P. Gupta, “Decision trees in machine learning.” <http://www.statisticssolutions.com/assumptions-of-logistic-regression/>, 17 May 2017. (Accessed on 07/27/2020).
- [23] K. Gurney, *An introduction to neural networks*. CRC press, 1997.
- [24] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. <http://www.deeplearningbook.org>.
- [25] I. Namatēvs, “Deep convolutional neural networks: Structure, feature extraction and training,” *Information Technology and Management Science*, vol. 20, no. 1, pp. 40–47, 2017.
- [26] S. Albawi, T. A. Mohammed, and S. Al-Zawi, “Understanding of a convolutional neural network,” in *2017 International Conference on Engineering and Technology (ICET)*, pp. 1–6, IEEE, 2017.
- [27] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.
- [28] Y. Kim, “Convolutional neural networks for sentence classification,” 2014.
- [29] “Fully connected layers in convolutional neural networks: The complete guide - missinglink.ai.” <https://missinglink.ai/guides/convolutional-neural-networks/fully-connected-layers-convolutional-neural-networks-complete-guide/>. (Accessed on 06/07/2020).
- [30] “Cs231n: Convolutional neural networks for visual recognition.” <http://cs231n.stanford.edu/>.

- [31] “Understanding lstm networks – colah’s blog.” <https://colah.github.io/posts/2015-08-Understanding-LSTMs/>. (Accessed on 05/29/2020).
- [32] “Microsoft word - l12.doc.” <https://www.cs.bham.ac.uk/~jxb/INC/l12.pdf>. (Accessed on 06/06/2020).
- [33] K. Greff, R. K. Srivastava, J. Koutník, B. R. Steunebrink, and J. Schmidhuber, “Lstm: A search space odyssey,” *IEEE transactions on neural networks and learning systems*, vol. 28, no. 10, pp. 2222–2232, 2016.
- [34] C. G. Savvas Varsamopoulos, Koen Bertels, “Designing neural network based decoders for surface codes,” 2018.
- [35] J. Devlin, M. Chang, K. Lee, and K. Toutanova, “BERT: pre-training of deep bidirectional transformers for language understanding,” *CoRR*, vol. abs/1810.04805, 2018.
- [36] R. Khandelwal, “Intuitive explanation of bert- bidirectional transformers for nlp.” <https://towardsdatascience.com/intuitive-explanation-of-bert-bidirectional-transformers-for-nlp-cdc1efc69> (Accessed on 08/2/2020).
- [37] N. P.-J. U. Ashish Vaswani, Noam Shazeer, “Attention is all you need,” December 6 2017.
- [38] “Language models are few-shot learners,” July 2020.
- [39] V. Warmerdam, “Gpt-3: Careful first impressions.” <https://blog.rasa.com/gpt-3-careful-first-impressions/>, July 22 2020. (Accessed on 14/09/2020).
- [40] Wikipedia, “Back propagation.” <https://en.wikipedia.org/wiki/Backpropagation>. (Accessed on 08/06/2020).
- [41] J. Brownlee, “What is confusion matrix in machine learning?.” <https://machinelearningmastery.com/confusion-matrix-machine-learning/>. (Accessed on 21/09/2020).

- [42] “R.” <https://www.r-project.org/about.html>. (Accessed on 06/01/2020).
- [43] “Advantages and disadvantages of python programming language.” <https://medium.com/@mindfiresolutions.usa/advantages-and-disadvantages-of-python-programming-language-fd0b394f2121>, 2017. (Accessed on 06/21/2020).
- [44] “Scala.” <https://www.scala-lang.org/>. (Accessed on 06/21/2020).
- [45] Google, “Google colab.” [https://colab.research.google.com/notebooks/intro.ipynb#scrollTo=GJBs\\_flRovLc](https://colab.research.google.com/notebooks/intro.ipynb#scrollTo=GJBs_flRovLc).
- [46] “Nltk 3.5 documentation.” <https://www.nltk.org/>. (Accessed on 06/21/2020).
- [47] “Pandas package overview.” [https://pandas.pydata.org/pandas-docs/stable/getting\\_started/overview.html](https://pandas.pydata.org/pandas-docs/stable/getting_started/overview.html), 2020.
- [48] P. community, “Numpy.” <https://numpy.org/devdocs/user/whatisnumpy.html>, 18 Oct 2015. (Accessed on 06/21/2020).
- [49] “Sklearn.” <https://scikit-learn.org/stable/>. (Accessed on 12/09/2020).
- [50] Goyvaerts, “Regular expressions quick start,” December 8 2016.
- [51] “keras.tensorflow.” <https://www.tensorflow.org/guide/keras>. (Accessed on 04/29/2020).
- [52] “plotly.” <https://plotly.com/python/>. (Accessed on 04/29/2020).
- [53] “wordcloud.” <https://python-graph-gallery.com/wordcloud/>. (Accessed on 04/01/2020).
- [54] G. Hollander, “What is structured data vs. unstructured data?,artificial intelligence, information management,” September 26 2019.

- [55] R. M. R. G. Monisha Kanakaraj, "Nlp based sentiment analysis on twitter data using ensemble classifiers," *IEEE, (ICSCN)*, 2015.
- [56] "Wikipedia." <https://en.wikipedia.org/wiki/Kaggle>. (Accessed on 05/29/2020).
- [57] "Sklearn." [https://scikit-learn.org/stable/modules/feature\\_extraction.html](https://scikit-learn.org/stable/modules/feature_extraction.html). (Accessed on 1/09/2020).
- [58] U. Malik, "Python for nlp: Movie sentiment analysis using deep learning in keras." <https://stackabuse.com/python-for-nlp-movie-sentiment-analysis-using-deep-learning-in-keras/>. (Accessed on 06/06/2020).
- [59] C. D. M. Jeffrey Pennington, Richard Socher, "Global vectors for word representation." <https://nlp.stanford.edu/projects/glove/>. (Accessed on 21/09/2020).

