Detecting and Classifying Hate speech

By

Sachin Kose Paul

Abstract. An automated system to detect and classify hate speech is an urgent need to improve the security and safeguard the cyber world. In this paper we study different classification method to detect hate speech. This paper proposes to use logistic regression, Bi-LSTM and neural network with attention, to predict hate speeches in twitter. We try to create models which incorporate contexts of the tweets and compare traditional models to these untraditional models.

The Sentiment 140 dataset is used for the study**.** A random sample from the data is used for the entire study. Traditional models like Logistic regression, SVM, Naïve Bayes, Random Forest are built and compared with deep lea ring models like CNN, RNN, LSTM, Bi-LSTM. CNN model turned out to have an accuracy of 82%. Moreover, deep neural networks with attention were built using a BERT model. A further adjustment for the BERT model by including CNN layers to it gave as the highest accuracy of 93% among all the other models.

Contents

List of Figures

List of Table

Chapter 1

1.1 Introduction

Natural Language Processing is a subset of Artificial Intelligence where the human language is made understandable by a system in other words it is way to make human language intelligible to machines (Monkeylearn,2021). This technology uses several applications to breakdown the speech data to make it meaningful and thus to process accordingly (GeeksforGeeks, 2021). This paper is a detailed report of NLP in detecting positive and negative tweets in Sentiment 140 dataset.

Hate speech is defined by the Cambridge Dictionary as "public speech that expresses hate or encourages violence towards a person or group based on something such as race, religion, sex, or sexual orientation" (Wikipedia). Controlling and reducing hate speeches that insult or hurt certain group of people or individuals is a necessity especially in this era where social media can influence the decision making of people. Hate speeches are those which can potentially insult or degrade a certain group of people or individuals. These hate speeches are capable of leaving a long-lasting influence on the victims and the future society and hence a control over hate speech instances is on high demand as never.

The main challenge in dealing with hate speeches is that it is hard to classify, and this is because there is an absence of an exact legal binding of what a hate speech is (Crabb, 2019). A lot also depends upon the content and context of the text. Most of the text may seemingly appear as neutral or as not a hate speech when the context of the text is kept unknown. The inability to capture the context that makes a text to hate in a modern problem and as Mark Zuckerberg suggest, it might take the next 5-10 years for us to reach at a comfortable stage to deal with hate speeches (The Washington post, 10-04-2018). Moreover, an offensive text for a person may not be offensive for a liberal and broad-minded person. All these make hate speeches incredibly hard to handle with and not to mention in most of the cases these labels need to be labelled manually by the same people. The recent study that revealed that a minor changes to the languages can fail any algorithms (Meaker, 2019). All these make it incredibly hard to model a system to detect hate speeches.

Twitter is a main source of platform to express themselves. With around 1.3 billion accounts and 500 million tweets send per day (Ahlgren, 2021) it is a huge responsibility to ensure that these tweets are safe from any kind of discriminations. There are several previous studies that involving detection of hate speeches and its classifications. Attention in deep neural network is a new concept that helps to give prioritizing to certain words in a sentence (Kim, 2020). It helps to give attention to certain part when processing a big data of texts (Kim, 2020). This paper compares performances of different models like Support Vector Machine (SVM), Random Forest, Naïve Bayes, Logistic Regression and Neural networks. However, the main aim of the paper is to compare these widely used models with models that include context or the models including memory such as logistic regression, and neural networks with attention like BERT. The aim is to build a model that automatically classify a tweet as hate speech or not.

1.2 Natural Language Processing (NLP).

Natural language processing is a subfield of data science more specifically is a field of artificial intelligence or AI, computer science and linguistics (Wikipedia, 2022), where different techniques are used to extract a meaning to a text or a speech in the same way as humans understand a text or a speech. NLP aims to convert any text or

speech to a form that any system can extract meaningful information, and can be used for new tasks like classification, text generation and so on.

1.2.1 Main NLP tasks.

Human language is a combination of several linguistic properties like metaphors, grammars, sarcasms, and other variations in sentences which make any kind of language processing an extremely challenging problem. Understanding the context of the sentence or the entire texts itself is important as in most case scenarios all languages have same word for more than one meaning. An effective way of learning thus would be understanding the word in respect with the entire context. Another challenge involving any language processing is to convert the human tests into a format that a system could extract meaning from the data. In spite the extreme challenges the NLP field faces, it is currently one of the fastest and innovatively advancing field in the field of Artificial Intelligence. Some of the major tasks in natural language processing is:-

- **Sentiment Analysis** is the analysis done on text data to identify the sentiment behind the text (Shivanandhan, 2020). It is a powerful technique which has a lot of application in AI like preventing hate speeches in social media platforms. Tweets and Facebook comments are classic examples of texts for sentiment analysis. The main challenges in sentiment analysis are to identify the real meaning the of the users' texts.
- **Speech Recognition** is a technique to convert speech to texts. This technique required for applications followed by a spoken command (IBM Cloud,2020). The challenges of speech recognition is the different way people talk, which include the slangs the speed and so on.
- **Natural Language Generation** is the opposite of speech to text. This technique is to convert the structured data into natural form. The main challenge in this technique is to understand the unstructured way normal human language is.

1.2.2 NLP tools in Python.

The Python Language provides several libraries and tools to support natural language processing. The Natural Language Toolkit (NLTK)is an open source in Python that contain an enormous number of libraries and tools that help in extracting texts. Some of the few tools that NLTK offers are

1.2.2.1 **Lemmatization** is a technique to group variant form of a word into one.In other words it is the process of finding all similar words and grouping into one word. For example the words leafs and leaves will be converted into a single word leaf.

1.2.2.2 **Stemming** is a technique to convert the words to its root form. For example the words leafs and leaves will be converted to leaf and leav after stemming. 'PorterStemmer in NLTK can be used to perform stemming on a text input.

1.2.2.3 **Tokenization** is the technique in NLP of breaking down sentences into single words. For example, the sentence 'I love chocolates', can be tokenized into three tokens namely, 'I', 'love', and 'chocolates'. The NLTL library in python provides a package called 'word_tokenize' and 'sent_tokenize' to split the text input into sentences and words.

1.2.3 Applications

The applications and scope for NLP is endless. Some of the applications of NLP is discussed below: -

- **Spam Detection** technologies use NLP techniques to classify the scam mails or any kind of texts. Spam detection is one of the most advanced fields in NLP and is capable of correctly identify the words and phrase and the format of any spam messages.
- **Machine translation** is another field in NLP that has an increasingly high popularity. Google translator is an example of NLP in machine translation.
- **Chatbots** acts like virtual agents and need to understand the text data from the conversation to provide the response. The other application with chatbot is question- answering.
- **Sentiment Analysis in social media** is an essential application in NLP which makes sure the enormous amount of social media data is not corrupted towards any community or groups.

— **Text Summarization** helps to summarize huge data without losing much information.

1.3 Classification Mechanism and Deep Learning in NLP

Supervised learning algorithms are used to classify a categorical data in natural language processing. The first step is to divide the dataset into test and train datasets. The next part would be the data preprocessing. The cleaned and normalized dataset can be used to build machine learning models. These models are then tested on the test dataset.

Text Classification is the process of categorizing texts into classes or assigning each texts a pre-defined tag. Text Classification is one of the most used methods in natural language processing (NLP). Some of the examples in NLP where text classification is used are Sentiment Analysis, Topic Detection Spam Detection and so on (AssignmentExpertHelp, 2014).

Deep learning algorithms or unsupervised learning.

Deep learning includes all the unsupervised learning algorithms used in NLP. Deep learning is increasingly popular in NLP because these models are capable of learning by themselves. Another reason for increased popularity in deep learning on natural language processing is that the performance for these models continues to be better.

1.4 Overfitting of the model

The model is considered to be overfitted if the test set loss function decreases and train set loss function increases, or if the train accuracy increases over iteration whereas the test accuracy decreases. Overfitting occurs when the noise in the train data set greatly influence the learning algorithm in favor of certain weights.

There are few ways to deal with overfitting in deep learning. Increasing the data size is a good option when data is easily available. But in most cases accruing more data can be physically impossible or can be expensive. L2 regularization is another method to avoid overfitting. Regularization is any adjustments in the model to decrease the training error rate. L2 regularization is a mathematical adjustment in a way that the weights that are huge are adjusted such a way that they would not have a huge impact on the results (Palachy, 2018). L 2 regularization uses a ridge regression by adding a penalty term which restricts the model from getting itself too complicated (Tewari, 2021).

Dropout is another effective method to tackle overfitting model. Dropout is a phenomenon where certain layers are dropped randomly. This will result in a thinning the network and urging the network to learn with fewer nodes (Brownlee, 2018). The main idea behind Dropout is that some of the neurons are dropped in the hidden states using a Bernoulli selection criterion. This random dropping will avoid neurons to have extreme dependency between them and low the remaining neurons to learn the model effectively (Phaisangittisagul, 2016).

1.5 Attention in NLP

Attention in neural network was first introduced in the 2015 which introduces a method to include contextual information to a word. This is a huge milestone in Natural Language Processing (Bahdanau, 2015). One of the major challenges in NLP is that its human language can have different meaning with respect to context. For example, the word ‘right’ can mean the side right or that the answer is right. To tackle these kinds of confusions in NLP attention mechanism which can get hold of the previous states and predictions when considering a word was developed.

The basic idea of attention in NLP is to give a contextual meaning to the entire texts for applying learning algorithms. The disadvantage of using most of the algorithms is that they only use parts of the information from the data, most probably the words before and after. Attention mechanism help us to include the complete context of the sentence before concluding a meaning for them (Loginova, 2018).

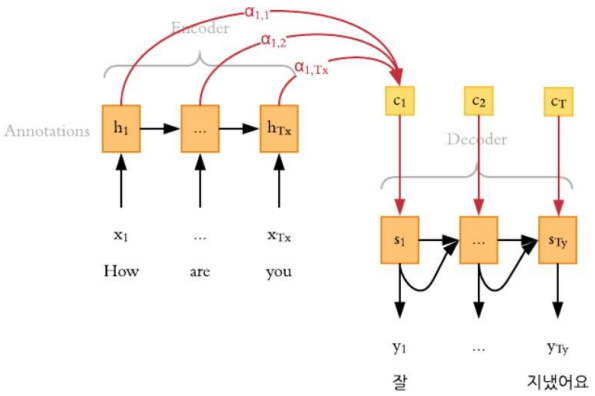

Fig. 36. Attention Mechanism (Bahdanau, 2014)

Figure 1 shows an attention algorithm. The encoder works as usual however in the decoder the difference is that the hidden state is computed using the previous output and the previous hidden state. In other words, the decoder in attention gets an input as a context vector rather than just the previous output. In that case each element will have a context vector passed through the decoder for prediction (Kulshrestha, 2020). This mechanism where both including the previous output and the hidden states give a model where the whole context of the text is considered. This is the basic idea of attention.

1.6 Literature Review

Research on hate speeches has a long history of thirty years (Tontodimamma, 2020). Different machine learning algorithms were tested on hate speeches over these periods. Attention-based recurrent neural network on Bangla language to detect hate speeches in Facebook (Das,2020) uses a dataset of approximately 7400 Bengali comments and these were then classified into different categories like hate speech, political comment etc. The author compares three different algorithms that uses attention, LSTM and Gated Recurrent Unit (GRU) based decoders and it showed an accuracy of 74% for LSTM and GRU. However, the algorithm with attention outperformed with an accuracy of 77%.

Different models were tested with English language for hate speech studies. Models like logistic regression, naïve Bayes, decision trees, random forests, and linear SVMs are used in a study tweet showed that linear SVM performed better than all other models (Davidson, 2017). The study used a logistic regression with L1 regularization to decide the required data and then used a L2 regularization for the final model. The entire dataset then was used to predict the labels of the tweet.

A study on Fox News User Comments by keeping the context information showed good performances (Gao, 2018). The authors created a new corpus from the Fox news comment section from 10 threads which were manually chosen and annotated by hand on whether a comment hurt any gender, ethnicity, or sexual orientation. They used the guidelines by Nobata et al. (2016) for annotation. They then used a Logistic regression and Neural network on the model. Logistic regression showed best result when both the features and context where combined and Neural network showed best result with bi-directional LSTM with attention.

A comparison study on different models and neural network models on Vietnamese language for hate speech detection shows that deep neural models give better results than traditional models (Luu, 2020). Another study uses a Bi-GRU-LSTM-CNN model was introduced for Vietnamese language and shows that it gives good performance of 70.5% (Huynh, 2019).

Hate speech detection on Vietnamese dataset using bi-directional-LSTM model (Nguyen, 2019), where used to predict the labels for the texts and gave a result of 71.43%.

Chapter 2

Methodology

2 Objective

The main aim of the study is to compare different traditional models with models including contexts or attention. Although there are several previous studies showing such untraditional models perform better compared to traditional ones, a study on Twitter tweet for English language is new. We aim to build different classification algorithms, neural network and neural network with attention, and compare their performances.

2.1 Data Analysis

2.1.1 Data Description.

The entire study is prepared on Sentiment -140 dataset. The dataset has 800,000 positive and 800,00 negative emoticons, for the total of 1.6 million training tweets. The dataset has only contained English Tweets. There are six variables included in the dataset. The variables are described below: -

1. Polarity – Polarity contains the sentiment of the tweet. Polarity 0 represents a negative emotion for the tweet, and polarity 1 represents a positive emotion.
2. Tweet – Tweet contains the actual Tweets.
3. Date – Date of the tweet
4. Id -Id is the user id of the account from which the tweet was made.
5. Unique_Number – represents the id of the tweet.
6. Query- Query contains the subject of the Tweet.

The figure below shows the data used for the study.

	Polarity	Unique_Number	Date	Query	Id	Tweet
0	0	1467810672	Mon Apr 06 22:19:49 PDT 2009	NO_QUERY	scotthamilton	is upset that he can't update his Facebook by ...
1	0	1467810917	Mon Apr 06 22:19:53 PDT 2009	NO_QUERY	mattycus	@Kenichan I dived many times for the ball. Man...
2	0	1467811184	Mon Apr 06 22:19:57 PDT 2009	NO_QUERY	ElleCTF	my whole body feels itchy and like its on fire
3	0	1467811193	Mon Apr 06 22:19:57 PDT 2009	NO_QUERY	Karoli	@nationwideclass no, it's not behaving at all....
4	0	1467811372	Mon Apr 06 22:20:00 PDT 2009	NO_QUERY	joy_wolf	@Kwesidei not the whole crew

Fig. 37. The Dataset.

2.2 Classification algorithms

Classification is a process of extracting information from a pre-existing training dataset to train the machine to identify different classes or categories for these training data points. The basic idea is to train a model so that it can assign any new datapoint into one of the classes (Wolff, 2020) that it has identified.

Classification algorithm is a kind of supervised learning method that is used to identify new observations based on already available data. The already available data is taken as the training dataset and the algorithm is trained using the labels for a particular category. In a classification algorithm the output is a particular category. In this study we will be discussing four main classification algorithms.

1. Random Forest
2. Naïve Bayes
3. Support Vector Machine (SVM)
4. Logistic regression

2.2.1 Random Forest

Random Forest classifier is a supervised learning algorithm that can be used for regression and classification problems (Vadapalli, 2021). The random forest uses multiple decision trees performed on bootstrap samples on a random basis and get predictions on each tree. Random forest has three main hyperparameters, which include the node size, the number of trees, and the number of features sampled (IBM Cloud Education, 2020). The random forest classifier creates a set of decision trees for several subset of the dataset. These subsets are chosen randomly (Patel, 2017). The final class is decided by the majority votes from the different decision trees.

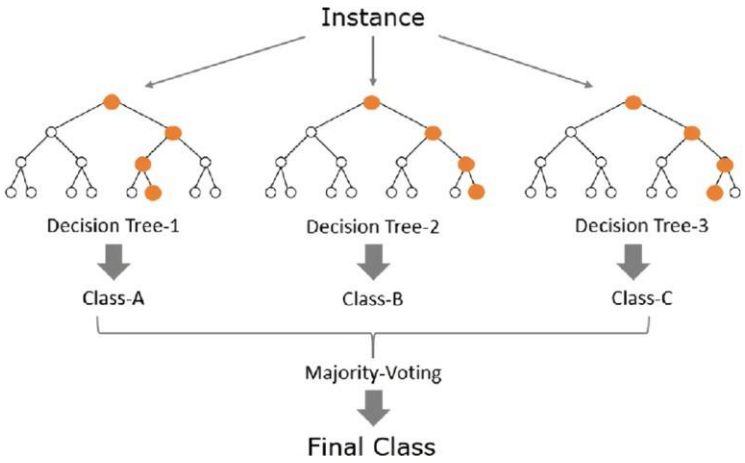

Fig. 38. Characteristic of Random Forest with three trees (Golze, 2020).

Figure 3 represents an algorithm chart for Random Forest. Each element

2.2.2 Naïve Bayes.

The naïve Bayes algorithm is a mostly used supervised learning algorithm that classifies the data points on conditional probability values. It is one of the important classification algorithms for text classification. It is a probabilistic classifier which predicts the probability of an object being in a class. The intuition of naïve Bayes is the Bayes theorem. (GeeksforGeeks, 2021). The Bayes theorem can be stated as:

$$P(A/B) = \mathrm{P(B/A)} * \mathrm{P(A)}\ P(B) \qquad (1)$$

Equation 1 gives the probability of an event given some information. The naïve Bayes algorithm equivalently uses a certain prior information of a data point to classify its class.

2.2.3 Support Vector Machine

Support Vector machines are supervised learning algorithms which learns from the train dataset to classify a new point according to a certain mathematical criterion. The train dataset is mapped into classes by maximizing the width of the gap between the classes. And a new point is assigned to a class based on which side of the gap it belongs to (Wikipedia, 2021).

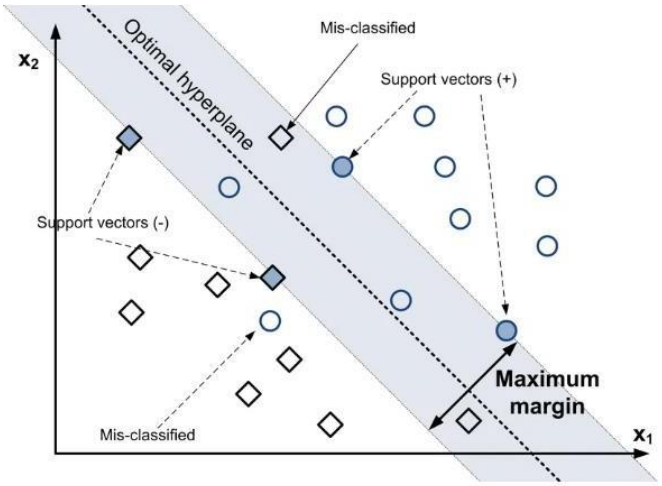

Fig. 39. Support Vector Machine.

Support Vector Machine uses hyperplanes to assigns or group the data points. It finds the optimal hyperplane that effectively divides the datapoints. The optimal hyperplane is decided on maximizing the length between the support vectors. Support vector are the elements from which the distance is maximized (PythonWife).

2.2.4 Logistic Regression

Logistic regression is also another member in the class of supervised machine learning algorithms. Logistic regression can be considered as a linear regression that is capable of classification (Edgar, 2017). The primary difference between a linear regression and logistic regression is that the logistic regression is bound between 0 and 1 by using the logistic function which is as follows.

$$\text{Logistic function} = \frac{1}{1+e^{-x}} \quad (2)$$ (Belyadi, 2021)

The x in equation 2 is the input variable. The logistic regression uses the loss function as the maximum likelihood estimate which is a conditional probability to decide which class a certain element would be classified into. If the probability is greater than 0.5 the prediction will be class 0 and otherwise as class 1.
The logistic regression is capable of classifying binary classes by predicting the probability of the outcome (Kanade, 2022).

2.3 Deep Learning

2.3.1 Word Embedding

Word embedding is the most important part in a natural language processing. The word embedding technique is a numerical representation of words such that a computer understands the text. It is a representation of words that can capture the meaning, context and find any relationship. The most important step in any NLP project is the text preprocessing converting the text into some vector format so that the algorithm can understand these texts for any future predictions. Word embedding helps us to convert the text into a vector which can be understood by a system. A model is built for the variable 'Tweet' using Word2Vec and each word is represented as a vector representation.

2.3.2 Word2Vec.

Word2Vec learns the relationship between the words in the corpus and embeds them into a vector form (Kleppen, 2021). This technique to represent the words will capture the meaning of the word at least to some extent. Word2Vec uses the position of each text to associate the meaning of the word (Vatsal, 2021).

```
array([-0.01049807,  0.00410215,  0.00018858, -0.00995583, -0.00889025,
       -0.00525256,  0.00476868,  0.0074109 , -0.00789437, -0.00778841,
       -0.00597767, -0.00540836, -0.00773599, -0.00851097, -0.00250181,
       -0.01050289, -0.00568486, -0.00851695, -0.00215727, -0.00518904,
       -0.00773622,  0.00032527,  0.00942404,  0.00736319, -0.00819614,
       -0.00389149,  0.00119055, -0.01123908, -0.00040483,  0.00606366,
        0.00777553, -0.008965  , -0.00427644, -0.00889313, -0.00026748,
        0.01147723,  0.005884  ,  0.00306376,  0.00392451, -0.00716358,
        0.00964162, -0.00858381, -0.00842138, -0.00242663,  0.00103623,
       -0.0038711 , -0.00175459,  0.00708834, -0.00018502, -0.00024343,
        0.00420483, -0.01045356,  0.0075152 ,  0.00857288, -0.01123747,
        0.00365532,  0.01094889, -0.00806221, -0.00983802, -0.00792899,
        0.00940648,  0.00069652,  0.0085388 , -0.00811357,  0.0013368 ,
        0.00441916,  0.00100892,  0.00365454, -0.00985507, -0.00691201,
        0.00087549,  0.00724403,  0.01019854,  0.00540864,  0.00145642,
        0.00912107, -0.00984853,  0.00297132, -0.00200128,  0.00511846,
        0.0068882 , -0.00719948,  0.00357565, -0.00669673, -0.00122882,
       -0.00835548, -0.00539962,  0.01123214,  0.00039555,  0.00388727,
        0.01195199,  0.00956463, -0.00818739, -0.00294598,  0.01391314,
        0.00713249, -0.00027705, -0.01050139,  0.00368793, -0.00684679],
      dtype=float32)
```

Fig. 40. Vector representation of a word

Figure 4 represent the vector representation of the word ‘love’.

This model can be used to find the most similar word. For example, figure 6 represents the most similar words of the word ‘love’.

```
[('haha', 0.3858332931995392),
 ('understand', 0.3849000930786133),
 ('ha', 0.37943321466445923),
 ('sorri', 0.3772396445274353),
 ('dream', 0.3750073313713074),
 ('bk', 0.32909688353538513),
 ('class', 0.3256164789199829),
 ('davidarchi', 0.3203093409538269),
 ('would', 0.3192213475704193),
 ('binncheol', 0.3189038038253784)]
```

Fig. 41. Similar words using Word2Vec

2.3.3 Convolutional Neural Network (CNN).

Neural networks are algorithms inspired by neurons in animal brain. An artificial neural network consists of many nodes that takes several inputs and produce an output based on a certain threshold value. This threshold value is usually determined by an activation function. An artificial neural network is trained by adjusting the wait associated with each connection. A typical artificial neural network in a deep learning setting consists of many layers of connected nodes which progressively represents data in increasing complexity.

CNN is a type of neural network which uses convolution which is a mathematical operation to identify patterns within the texts (Kumar, 2021). CNN has five different layers (Python Wife) as follows: -

1. Input layer
2. Convolution layer
3. ReLU layer
4. Pooling layer
5. Fully connected layer

Input Layer.

The input layer contains the input image or the width, height, and depth of the data.

Convolutional layer.

This layer computes the output volume using the convolutional filter.

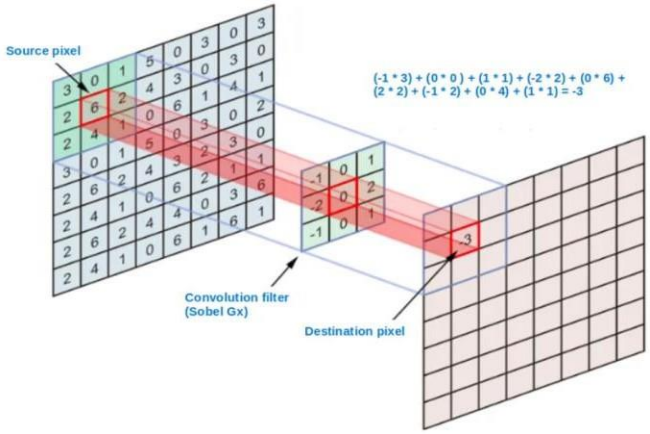

Fig. 42. A complete convolutional layer [Python Wife].

The source pixel can be taken as the input matrix. The convolutional filter also known as the kernel is the filter, we apply to the input matrix. An element wise dot product is done on each element to get the destination pixel matric.

ReLU layer.

ReLU stands for Rectified Linear Unit is an activation function. It is a nonlinear function that can easily back-propagate errors and activate multiple layers of neurons.

Pooling Layer.

The pooling layer reduces the size of the convolved features by extracting only the dominant features.

Fully connected layer.

The output pf the pooling layer is the input of the fully connected layer. The Fully connected layer then uses neural network to determine the output.

2.3.4 Recurrent Neural Network (RNN).

A recurrent neural network is a class of artificial neural network which has memory that is it remembers all the information that has been calculated. RNN allows to operate over vector sequences. RNN has a short-term memory that remembers all the information about what has been calculated.

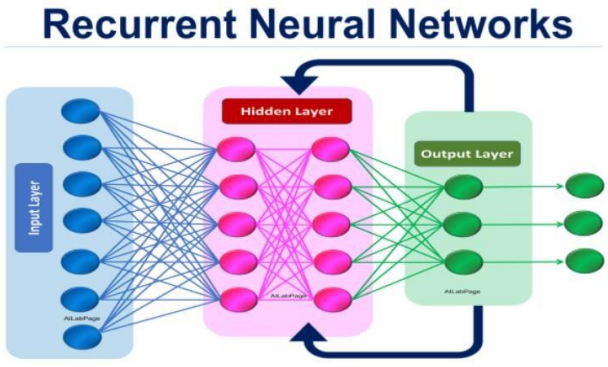

Fig. 43. Recurrent Neural Network (Boufeloussen, 2020).

The RNN when considering an input, it also considers the previous output hence incorporating the memory part (Pedamkar). The short-term memory in RNN can be a disadvantage in real time NLP problems. RNN with LSTM or RNN with GRU is a better solution to this.

2.3.5 LSTM

Long Short-Term Memory is a type of RNN which deals with sequential data. A major drawback to an RNN system is of vanishing and exploding gradient (Aditianu, 2021). Long Short-Term Memory helps in regulating the information for each neuron. The LSTM mechanism can decide which information to be considered as relevant and which not. The main idea of LSTM is that the output from any neuron is used as an input for the same neuron.

The LSTM cells consists of different gates. These gates keep the relevant information and give the network the relevant memory (Python Wife).

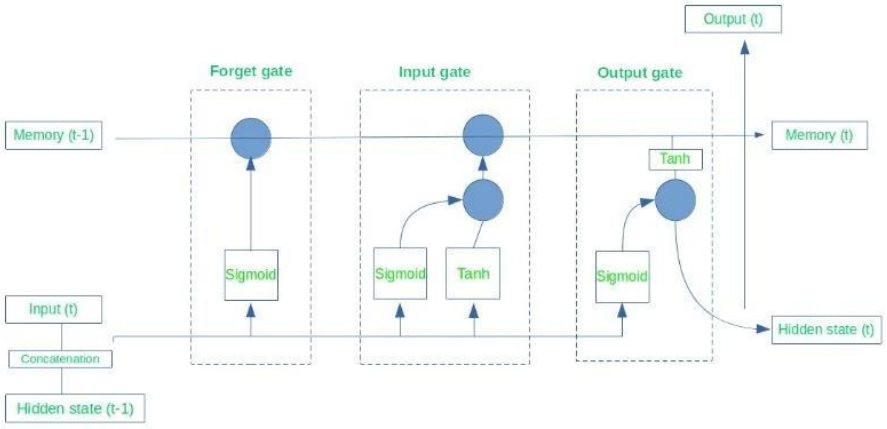

Fig. 44. LSTM Structure

The LSTM cell has three gates the forget gate, input gate and output gate and several activation functions. These gates have the ability to remove irrelevant information and keep the relevant information for future predictions (Colah, 2015).

The forget gate removes all the information that is no longer useful. The input at the particular time and the output of the previous cell is fed into the forget gate and the sigmoid function decides which information to be preserved (Chugh, 2021).

The input gate preserves all the information that is useful. Both sigmoid and tanh activation are used in input layer to store the required data (Chugh, 2021).

The output layer extracts the information from the cell to be presented as the output. An output layer also uses two activation functions namely tanh and sigmoid (Chugh, 2021).

2.3.6 Bidirectional LSTM (Bi-LSTM)

Bidirectional LSTM is the process of making the neural network learn both forward and backward directions. The regular LSTM takes the input in either forward or backward direction, whereas bi-LSTM can take both the directions to store the future and past information (Verma, 2021).

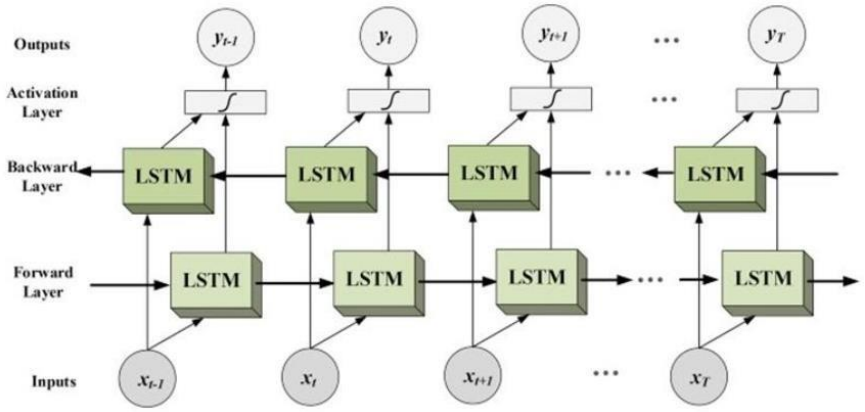

Fig. 45. Bi-LSTM architecture.

In bi-LSTM the output of both forward and backward is given to the activation layer which is a neural network to get to the output as shown in the figure.

2.4 Neural Network with attention

2.4.1 Bidirectional Encoder Representations from Transformers (BERT)

Bidirectional Encoder Representations from Transformers (BERT) is a research paper by Google published in 2018 (Vaswani, 2017). BERT is based on transformers, a deep learning model where every output variable is connected to every input variable. In other words, BERT can read the input text sequence in both directions simultaneously (Lutkevich, 2020). BERT has been pre-trained on the entire Wikipedia and the Book Corpus and can be fine-tuned to use for several tasks of NLP.

BERT is the first innovation in NLP that uses self- attention mechanism. This is possible by introducing the bi-directional transformers. The transformers consider any word in context of the full context rather than considering just one word. The transformer has an encoder-decoder network (GeeksforGeeks, 2020). The encoder reads the input text and the decoder do the prediction part (Horev, 2018). BERT uses this transformer encoder architecture to create a bi-directional self-attention for the input sequence (Towards A1, 2020). The addition of this attention layer to the BERT architecture makes it one of the powerful algorithms in NLP.

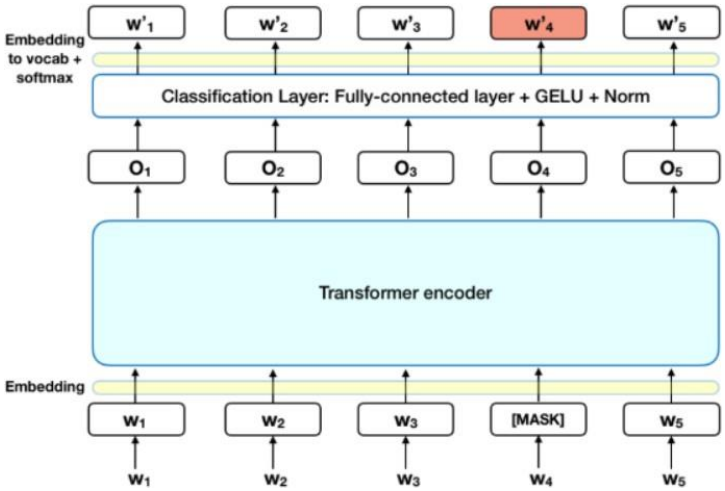

Fig. 46. BERT Algorithm

BERT algorithms give attention weights to each input sentences. The model is pre-trained on a large dataset and the trained model is used for new tasks. Hence, instead of obtaining new weights for a separate dataset the same weights are used. These weights are fine-tuned and modified according to the new task.

BERT's architecture in the original paper published in 2018 comes with two model forms: -

1. BERT Base- This model contains 12 transformer blocks, 12 attention heads, 768 hidden layer size (bforblack, 2021).
2. BERT Large- This model 24 transformer blocks, 16 attention heads, 1024 hidden layer size (bforblack, 2021)

2.4.2 BERT with CNN

In BERT with CNN model both the algorithms are used to train the Sentiment 140 dataset. The preprocessed data is pass through the BERT model to which returns a sequence output, and this is passed through a CNN layer.

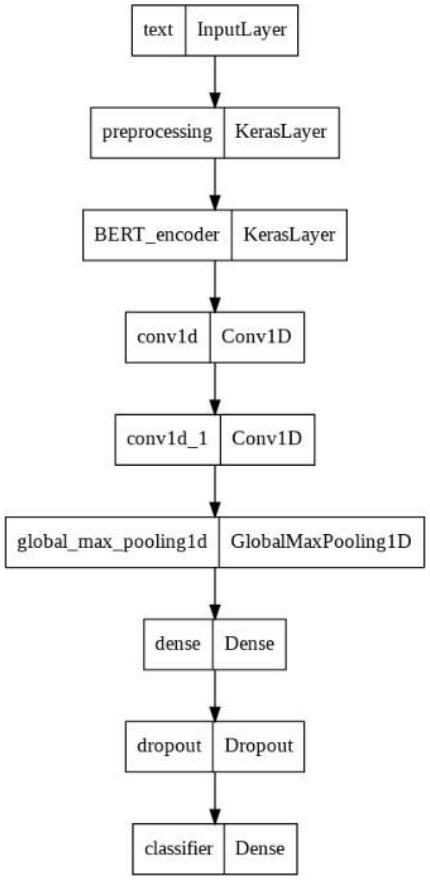

Fig. 47. Algorithm for the BERT with CNN model.

Figure 12 explains the working algorithm of BERT with CNN model. The input text is preprocessed and passed to a BERT encoder. The output is passed to a CNN algorithm. A dropout layer is added in the convolutional neural network (CNN) to decrease the complexity of the model, and the CNN classifies the text input by passing it through a linear layer.

3 Chapter 3

Experiment

This chapter we built different model techniques on the Sentiment 140 tweet dataset and compare the results. The data contains 1.6 million data entries, and we would be only considering a random sample for most of the studies. This is because most of the deep learning algorithms uses a large computational power which can be quite expensive.

3.1 Exploratory Data Analysis

Exploratory data analysis (EDA) is a technique to investigate the large dataset for the general distribution, for any underlying patterns or to summarize the data in regards on certain characteristics. EDA usually use visual aids to explain the dataset. Visual representation is a better and less time assuming technique to understand any anomalies and trends in the data. The general insight to the data will help a data analyst to conclude which advance techniques can be used on the data for analysis.

3.2 Visualizing the Sentiment 140 dataset

The main aim of the whole study is to predict the polarity of a given tweet. The positive polarity is denoted by '4' and the negative tweets are represented by a polarity '0'. The distribution of the polarity over the dataset is as the figure below.

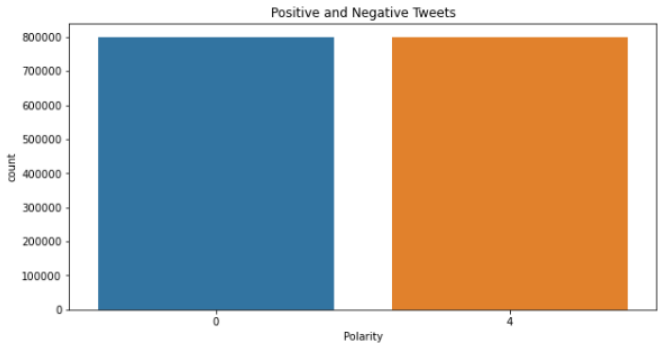

Fig. 48. Distribution of the Tweets

The figure 12 shows that the polarity is equally distributed in the data and hence no further adjustments are needed. An equally distributed global set is required to effectively built analysis techniques.

3.3 World cloud

Word cloud can be used in NLP to visualize the most common words in a certain class. In this study world cloud is used to visualize the most occurring words in the class of group where the polarity is '4' (positive tweets), and for the class where the polarity is '0' (negative tweets).

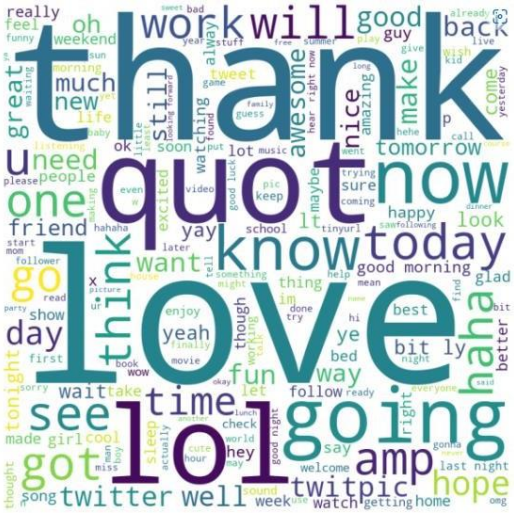

Fig. 49. World Cloud in positive tweets

The figure 13 above shows the most common words among the positive tweets. The words like ‘thank’, ‘love’ are the ones that are most occurring in positive tweets.

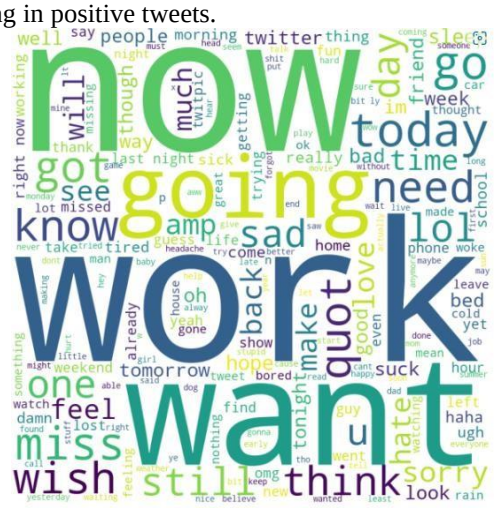

Fig. 50. World cloud in negative tweets

The figure above shows the words that are most occurring in negative tweets, that is the group that have polarity as ‘0’. Figure 14 shows the words like ‘work’, ‘want’ are frequently used in negative tweets.

3.4 Data Pre-processing for NLP

Prepping the dataset for NLP is an important step before moving on to build any models. The variable of interest here is ‘Tweet’ in the dataset. A random sample of 10000 data entries are used for this. These texts were converted by removing all other variables from the texts and then converting every word to lower case letters and then removing the stop words. The resulted text data is used for data analytics part.

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 10000 entries, 411663 to 1245490
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Polarity       10000 non-null  int64
 1   Unique_Number  10000 non-null  int64
 2   Date           10000 non-null  object
 3   Query          10000 non-null  object
 4   Id             10000 non-null  object
 5   Tweet          10000 non-null  object
dtypes: int64(2), object(4)
memory usage: 546.9+ KB
```

Fig. 51. Sample data

3.5 Classification analysis

Classification is a process of extracting information from a pre-existing training dataset to train the machine to identify different classes or categories for these training data points. The basic idea is to train a model so that it can assign any new datapoint into one of the classes [Wolff, 2020] that it has identified. Classification algorithm is a kind of supervised learning method that is used to identify new observations based on already available data. The already available data is taken as the training dataset and the algorithm is trained using the labels for a particular category. In a classification algorithm the output is a particular category.

3.5.1 Random Forest.

A random sample of 10000 is used for creating a random forest classifier. A random forest classifier is used to build a model for the classifying positive and negative tweets. The dataset is divided into test and train in a ratio of 90:10. The random forest classifier built has an accuracy of 0.69.

	precision	recall	f1-score	support
0	0.69	0.72	0.70	1532
1	0.69	0.66	0.68	1468
accuracy			0.69	3000
macro avg	0.69	0.69	0.69	3000
weighted avg	0.69	0.69	0.69	3000

Fig. 52. Random Forest for Sentiment 140 data set.

3.5.2 Support Vector Machine (SVM)

Support Vector machines are supervised learning algorithms which learns from the train dataset to classify a new point according to a certain mathematical criterion. The train dataset is mapped into classes by maximizing the width of the gap between the classes. And a new point is assigned to a class based on which side of the gap it belongs to [Wikipedia, 2021]. A support vector machine algorithm is built for the Sentiment 140 dataset and got an accuracy of 0.48.

	precision	recall	f1-score	support
0	1.00	0.00	0.00	1532
1	0.49	1.00	0.66	1468
accuracy			0.49	3000
macro avg	0.74	0.50	0.33	3000
weighted avg	0.75	0.49	0.32	3000

Fig. 53. SVM for Sentiment 140 data set.

3.5.3 Naïve Bayes

A naïve Bayes algorithm to classify a tweet as positive or negative. The technique of a naïve Bayes algorithm is to use the words in the given text to predict the class it belongs to. The idea of bag of words representation is to represent the text as a set of words and its counts- the number of times the word occurred in the text. Other information like the order of the words, the style of sentences is ignored.

The Naive Bayes algorithm had an accuracy of 0.59.

	precision	recall	f1-score	support
0	0.69	0.38	0.49	1532
1	0.56	0.82	0.67	1468
accuracy			0.60	3000
macro avg	0.63	0.60	0.58	3000
weighted avg	0.63	0.60	0.58	3000

Fig. 54. Naïve Baes for Sentiment 140 data set.

3.5.4 Logistic Regression

A logistic regression classifier is modeled on Sentiment 140 dataset. The logistic regression uses a random sample of 10000 from the original data set and got an accuracy of 0.73.

	precision	recall	f1-score	support
0	0.74	0.72	0.73	1511
1	0.72	0.74	0.73	1489
accuracy			0.73	3000
macro avg	0.73	0.73	0.73	3000
weighted avg	0.73	0.73	0.73	3000

Fig. 55. Logistic Regression for Sentiment 140 data set.

3.6 Deep Learning Algorithms

3.6.1 Word Embedding

Word embedding is the most important part in a natural language processing. The word embedding technique is a numerical representation of words such that a computer understands the text. It is a representation of words that can capture the meaning, context and find any relationship. The most important step in any NLP project is the text preprocessing converting the text into some vector format so that the algorithm can understand these texts for any future predictions. Word embedding helps us to convert the text into a vector which can be understood by a system. A model is built for the variable ‘description’ using Word2Vec and each word is represented as a vector representation. Figure 21 represent the vector representation of the word ‘love’.

```
Out[35]: array([-0.08387534,  0.66543734, -0.00599339,  0.17322096,  0.02218527,
                -1.1934026 ,  0.48949146,  1.4693251 , -0.38839564, -0.6753641 ,
                -0.3012839 , -1.1203067 , -0.09924171,  0.3142377 ,  0.29586715,
                -0.51331913,  0.05291342, -0.80142576, -0.1210779 , -1.3895481 ,
                 0.17941055,  0.33546448,  0.3498646 , -0.21609864, -0.31819224,
                -0.05009368, -0.5254263 , -0.51870626, -0.7132249 ,  0.13408172,
                 0.88100874,  0.1716633 ,  0.28607652, -0.5706506 , -0.3148269 ,
                 0.85282683,  0.1558161 , -0.46932542, -0.42872804, -1.3920364 ,
                 0.28050146, -0.6190191 , -0.19599861,  0.00366832,  0.63509786,
                -0.471051  , -0.6626999 , -0.3055358 ,  0.3474735 ,  0.47551054,
                 0.4466207 , -0.48760635, -0.26262274, -0.14071515, -0.29495594,
                 0.49333745,  0.26566952, -0.03448988, -0.8202079 ,  0.2300319 ,
                 0.24778353,  0.13257828, -0.01733883, -0.08865828, -0.89479005,
                 0.6527379 ,  0.16673425,  0.4097938 , -1.0635567 ,  0.729206  ,
                -0.50886065,  0.5185068 ,  0.75738716, -0.1370122 ,  0.94662756,
                 0.35055608, -0.0517937 , -0.16456702, -0.46194422,  0.2825629 ,
                -0.31139213, -0.13070339, -0.79377216,  1.0509937 , -0.1127357 ,
                -0.10906634,  0.17532608,  0.739964  ,  0.9809793 ,  0.40689415,
                 0.91650045,  0.5318926 ,  0.01071415,  0.3398684 ,  1.153184  ,
                 0.6719718 ,  0.2677654 , -0.71527255,  0.3047992 , -0.04050438],
               dtype=float32)

Out[36]: [('lol', 0.999865710735321),
          ('even', 0.9998559951782227),
          ('make', 0.9998529553413391),
          ('amp', 0.9998525381088257),
          ('still', 0.9998456239700317),
          ('im', 0.9998381733894348),
          ('look', 0.9998348951339722),
          ('play', 0.9998337626457214),
          ('come', 0.9998281598091125),
          ('oh', 0.999826192855835)]
```

Fig. 56. Word representation of the word ‘Love’ and Similar words using Word2Vec

3.6.2 Convolutional Neural Network (CNN).

A convolutional neural network is built for predicting the Sentiment 140 dataset.

The first step is to use embedding technique to the text variable. Each variable is given a unique index in the complete vocabulary. The whole text is tokenized using the Tokenizer from the Tensorflow package. The tokenized test and train datasets are converted to sequences using the bag of word method. Each word is converted into an array of equal length using pad sequences.

A ten-layer CNN is built and used for prediction. For the given model ‘epoch’ is defined as five, since from number greater than five the validation accuracy starts decreasing that means the weights will be changed ten times. After ten iterations the model has a training accuracy of 85% and the validation accuracy is 57%.

```
Epoch 1/10
282/282 [==============================] - 5s 9ms/step - loss: 0.6881 - accuracy: 0.5219 - val_loss: 0.6698 - val_accuracy: 0.5
730
Epoch 2/10
282/282 [==============================] - 2s 6ms/step - loss: 0.6153 - accuracy: 0.6432 - val_loss: 0.6458 - val_accuracy: 0.6
030
Epoch 3/10
282/282 [==============================] - 2s 6ms/step - loss: 0.5413 - accuracy: 0.6987 - val_loss: 0.6915 - val_accuracy: 0.5
870
Epoch 4/10
282/282 [==============================] - 2s 8ms/step - loss: 0.4755 - accuracy: 0.7431 - val_loss: 0.7302 - val_accuracy: 0.5
970
Epoch 5/10
282/282 [==============================] - 2s 8ms/step - loss: 0.4066 - accuracy: 0.7882 - val_loss: 0.7783 - val_accuracy: 0.5
910
Epoch 6/10
282/282 [==============================] - 2s 8ms/step - loss: 0.3439 - accuracy: 0.8126 - val_loss: 0.8787 - val_accuracy: 0.5
920
Epoch 7/10
282/282 [==============================] - 2s 8ms/step - loss: 0.2929 - accuracy: 0.8416 - val_loss: 0.9914 - val_accuracy: 0.5
850
Epoch 8/10
282/282 [==============================] - 2s 7ms/step - loss: 0.2630 - accuracy: 0.8434 - val_loss: 1.0611 - val_accuracy: 0.5
780
Epoch 9/10
282/282 [==============================] - 2s 7ms/step - loss: 0.2419 - accuracy: 0.8538 - val_loss: 1.1777 - val_accuracy: 0.5
620
Epoch 10/10
282/282 [==============================] - 2s 6ms/step - loss: 0.2264 - accuracy: 0.8554 - val_loss: 1.4465 - val_accuracy: 0.5
720
```

Fig. 57. CNN for Sentiment 140 data set

The training dataset loss and validation dataset loss is given as below. Although the training dataset seems to have a decreasing trend of accuracy, the validation dataset

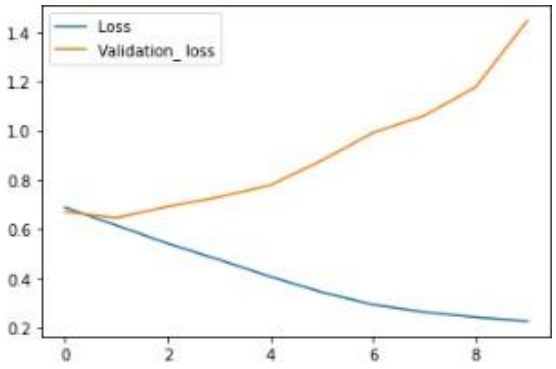

Fig. 58. Loss function for training and validation dataset for CNN.

The training accuracy and the validation accuracy is given as below. Although training accuracy has an increasing trend the figure shows that the training accuracy shows a constant trend.

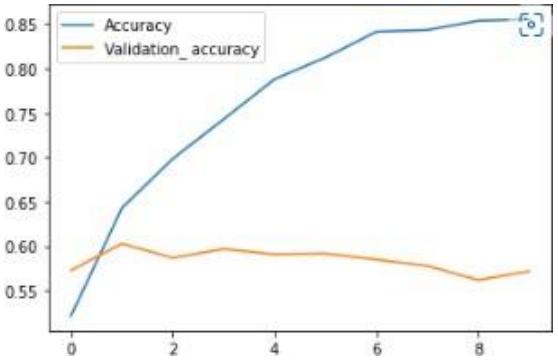

Fig. 59. Accuracy for train and test dataset for CNN

The above example where the validation lose increases and validation accuracy decreases or remains constant indicate that the neural network is overfitted. The increase in the accuracy of the training set but the inability of the network to work on a new test data set is due to this overfitting phenomenon.

3.6.3 Convolutional Neural Network with L2 Regularization and Dropout

The same data set is used to build a convolutional neural network but with L2 regularization and a dropout layer to avoid overfitting. The new neural network has an accuracy of 0.74 and a validation accuracy 0.67.

```
Epoch 1/4
282/282 [==============================] - 3s 7ms/step - loss: 0.7089 - accuracy: 0.5512 - val_loss:
0.6766 - val_accuracy: 0.5980
Epoch 2/4
282/282 [==============================] - 1s 5ms/step - loss: 0.6410 - accuracy: 0.6593 - val_loss:
0.6512 - val_accuracy: 0.6230
Epoch 3/4
282/282 [==============================] - 2s 6ms/step - loss: 0.5914 - accuracy: 0.7137 - val_loss:
0.6467 - val_accuracy: 0.6600
Epoch 4/4
282/282 [==============================] - 1s 5ms/step - loss: 0.5497 - accuracy: 0.7423 - val_loss:
0.6497 - val_accuracy: 0.6700
```

Fig. 60. CNN with L2 Regularization and Dropout for CNN

The new neural networks loss function for both training and test dataset shows a decreasing trend. The Regularization and Dropout layer has rectified the overfitting problem of the original network.

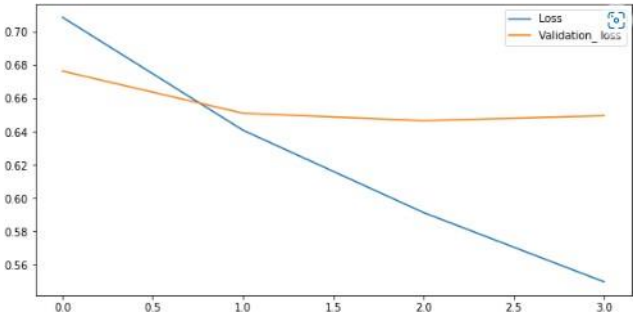

Fig. 61. Loss function for CNN with L2 regularization and dropout.

The training accuracy and the validation accuracy is given as below. Both training accuracy the test accuracy has an increasing trend now. The model is now good for future references.

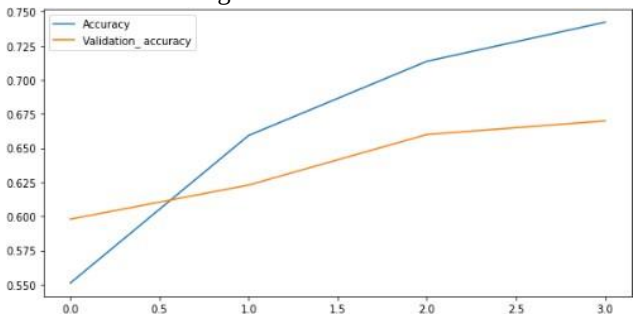

Fig. 62. Accuracy for CNN with L2 Regularization and dropout

3.6.4 Recurrent Neural Network

A recurrent neural network with an LSTM layer is created on the dataset. The tokenized dataset after removing the stop words is used for the creating an RNN model. The model has an accuracy of 0.62 for the training dataset and has a validation accuracy of 0.54.

```
Epoch 1/10
282/282 [==============================] - 9s 15ms/step - loss: 0.6875 - accuracy: 0.5199 - val_loss:
0.6787 - val_accuracy: 0.5538
Epoch 2/10
282/282 [==============================] - 4s 14ms/step - loss: 0.6652 - accuracy: 0.5607 - val_loss:
0.6770 - val_accuracy: 0.5356
Epoch 3/10
282/282 [==============================] - 4s 15ms/step - loss: 0.6458 - accuracy: 0.5773 - val_loss:
0.7008 - val_accuracy: 0.5349
Epoch 4/10
282/282 [==============================] - 4s 15ms/step - loss: 0.6299 - accuracy: 0.5932 - val_loss:
0.6726 - val_accuracy: 0.5582
Epoch 5/10
282/282 [==============================] - 4s 15ms/step - loss: 0.6164 - accuracy: 0.6030 - val_loss:
0.6781 - val_accuracy: 0.5504
Epoch 6/10
282/282 [==============================] - 4s 14ms/step - loss: 0.6054 - accuracy: 0.6066 - val_loss:
0.6868 - val_accuracy: 0.5611
Epoch 7/10
282/282 [==============================] - 4s 14ms/step - loss: 0.5941 - accuracy: 0.6131 - val_loss:
0.6910 - val_accuracy: 0.5633
Epoch 8/10
282/282 [==============================] - 4s 14ms/step - loss: 0.5830 - accuracy: 0.6226 - val_loss:
0.7120 - val_accuracy: 0.5398
Epoch 9/10
282/282 [==============================] - 4s 14ms/step - loss: 0.5721 - accuracy: 0.6288 - val_loss:
0.7036 - val_accuracy: 0.5602
Epoch 10/10
282/282 [==============================] - 4s 14ms/step - loss: 0.5629 - accuracy: 0.6280 - val_loss:
0.7251 - val_accuracy: 0.5412
```

Fig. 63. RNN for Sentiment 140 dataset

The figure shows the loss function for both training and validation dataset. The training dataset has a decreasing trend for the loss whereas the validation dataset has an increasing fluctuated trend.

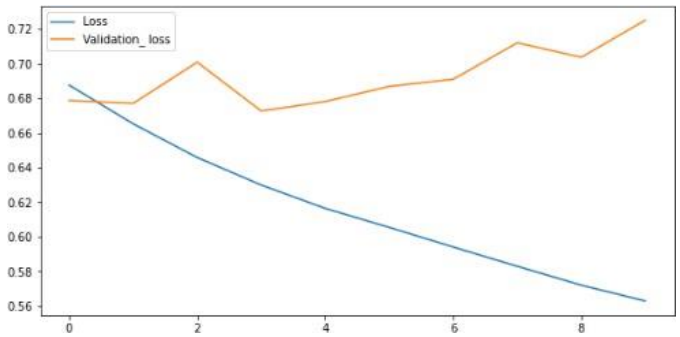

Fig. 64. Loss function for RNN

The figure shows the accuracy for both training and validation dataset over the ten iterations. The validation dataset shows a fluctuating trend which seemingly fluctuate over a constant.

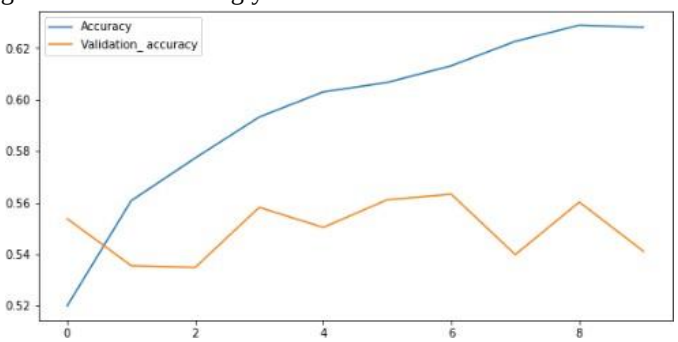

Fig. 65. Accuracy for RNN

A similar trend of overfitting needs to be addressed for the RNN.

3.6.5 RNN with L2 Regularization and Dropout.

The RNN with regularization and a dropout layer to avoid overfitting and has an accuracy of 0.55 and a validation accuracy of 0.52.

```
Epoch 1/4
282/282 [==============================] - 6s 12ms/step - loss: 0.7035 - accuracy: 0.5201 - val_loss:
0.6927 - val_accuracy: 0.5190
Epoch 2/4
282/282 [==============================] - 3s 10ms/step - loss: 0.6827 - accuracy: 0.5415 - val_loss:
0.6852 - val_accuracy: 0.5223
Epoch 3/4
282/282 [==============================] - 3s 11ms/step - loss: 0.6707 - accuracy: 0.5486 - val_loss:
0.6861 - val_accuracy: 0.5233
Epoch 4/4
282/282 [==============================] - 3s 12ms/step - loss: 0.6639 - accuracy: 0.5591 - val_loss:
0.6893 - val_accuracy: 0.5239
```

Fig. 66. RNN with L2 Regularization and dropout.

The L2 regularization and dropout layer clears the overfitting problem. Figures confirms that the model is ready to use for future references.

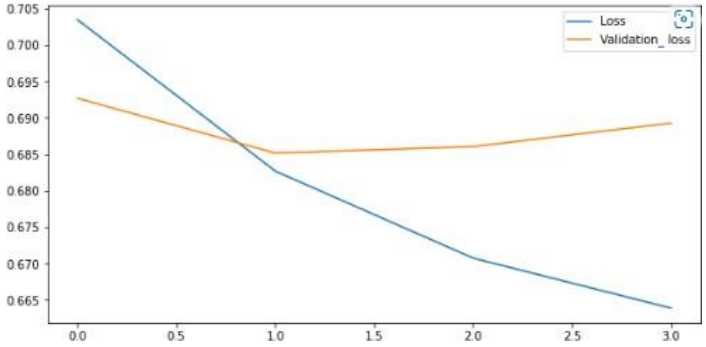

Fig. 67. Accuracy for RNN with L2 Regularization and dropout.

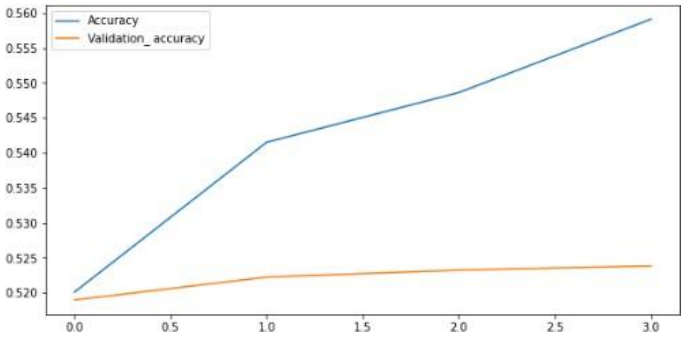

Fig. 68. Accuracy for RNN with L2 Regularization and Dropout.

3.6.6 Long Short-Term Memory (LSTM) Model with L2 Regularization and dropout layer.

The LSTM model is built for the Sentiment 140 dataset, to perform a classification algorithm.

```
Epoch 1/4
282/282 [==============================] - 16s 42ms/step - loss: 4.6738 - accuracy: 0.4952 - val_los
s: 0.7944 - val_accuracy: 0.4820
Epoch 2/4
282/282 [==============================] - 10s 36ms/step - loss: 4.0968 - accuracy: 0.5720 - val_los
s: 0.8164 - val_accuracy: 0.5220
Epoch 3/4
282/282 [==============================] - 10s 36ms/step - loss: 4.1926 - accuracy: 0.6063 - val_los
s: 0.7442 - val_accuracy: 0.5490
Epoch 4/4
282/282 [==============================] - 10s 35ms/step - loss: 4.1647 - accuracy: 0.6231 - val_los
s: 0.7071 - val_accuracy: 0.5630
```

Fig. 69. LSTM for Sentiment 140 data set.

The LSTM has an accuracy of 0.62 and a validation accuracy of 0.56.

3.6.7 Bidirectional LSTM with L2 Regularization and Dropout layer.

A bidirectional LSTM is built for the dataset, to perform classification problem.

```
Epoch 1/2
29/29 [==============================] - 10s 57ms/step - loss: 0.7427 - accuracy: 0.5404 - val_loss:
0.7090 - val_accuracy: 0.4400
Epoch 2/2
29/29 [==============================] - 0s 11ms/step - loss: 0.6171 - accuracy: 0.6524 - val_loss:
0.9112 - val_accuracy: 0.6030
```

Fig. 70. Bi-LSTM for Sentiment 140 dataset.

The Bi-LSTM has an accuracy of 0.65 and validation accuracy of 0.60.

3.7 Sentiment Classification Using Bidirectional Encoder Representations from Transformers (Attention Models)

A very small sample from the Sentiment 140 dataset is used to create a Bidirectional Encoder Representations from Transformers (BERT) model. A random sample of 100 is used.

```
begin training using onecycle policy with max lr of 2e-05...
8/8 [==============================] - 2498s 311s/step - loss: 0.7027 - accuracy: 0.5240 - val_loss:
0.7150 - val_accuracy: 0.4960
```

Fig. 71. BERT for Sentiment 140 data set.

An accuracy of 0.52 is observed in BERT model.

3.8 Bidirectional Encoder Representations from Transformers with Convolutional Neural Network (Attention Models)

A very small sample is used for creating a BERT with CNN model. The output from the BERT model is passed as input for the Convolutional Neural Network layer.

```
Epoch 1/20
3/3 [==============================] - 17s 6s/step - loss: 1.1261 - accuracy: 0.2716 - val_loss: 1.0964 - val_accuracy: 0.3333
Epoch 2/20
3/3 [==============================] - 16s 5s/step - loss: 1.1177 - accuracy: 0.3210 - val_loss: 1.0962 - val_accuracy: 0.3333
Epoch 3/20
3/3 [==============================] - 16s 5s/step - loss: 1.1148 - accuracy: 0.3210 - val_loss: 1.0953 - val_accuracy: 0.3333
Epoch 4/20
3/3 [==============================] - 20s 7s/step - loss: 1.1156 - accuracy: 0.3086 - val_loss: 1.0931 - val_accuracy: 0.4444
Epoch 5/20
3/3 [==============================] - 16s 5s/step - loss: 1.1006 - accuracy: 0.4074 - val_loss: 1.0915 - val_accuracy: 0.3333
Epoch 6/20
3/3 [==============================] - 15s 5s/step - loss: 1.0945 - accuracy: 0.3704 - val_loss: 1.0900 - val_accuracy: 0.3333
Epoch 7/20
3/3 [==============================] - 16s 5s/step - loss: 1.0851 - accuracy: 0.4444 - val_loss: 1.0883 - val_accuracy: 0.3333
Epoch 8/20
3/3 [==============================] - 15s 5s/step - loss: 1.0784 - accuracy: 0.4815 - val_loss: 1.0877 - val_accuracy: 0.4444
Epoch 9/20
3/3 [==============================] - 16s 5s/step - loss: 1.0668 - accuracy: 0.5309 - val_loss: 1.0845 - val_accuracy: 0.4444
Epoch 10/20
3/3 [==============================] - 15s 5s/step - loss: 1.0536 - accuracy: 0.5062 - val_loss: 1.0820 - val_accuracy: 0.5556
Epoch 11/20
3/3 [==============================] - 16s 5s/step - loss: 1.0578 - accuracy: 0.5926 - val_loss: 1.0777 - val_accuracy: 0.5556
Epoch 12/20
3/3 [==============================] - 17s 6s/step - loss: 1.0354 - accuracy: 0.6296 - val_loss: 1.0737 - val_accuracy: 0.5556
Epoch 13/20
3/3 [==============================] - 15s 5s/step - loss: 1.0293 - accuracy: 0.6543 - val_loss: 1.0696 - val_accuracy: 0.5556
Epoch 14/20
3/3 [==============================] - 16s 5s/step - loss: 1.0111 - accuracy: 0.7160 - val_loss: 1.0646 - val_accuracy: 0.5556
Epoch 15/20
3/3 [==============================] - 15s 5s/step - loss: 0.9908 - accuracy: 0.7160 - val_loss: 1.0597 - val_accuracy: 0.5556
Epoch 16/20
3/3 [==============================] - 16s 5s/step - loss: 0.9679 - accuracy: 0.8148 - val_loss: 1.0523 - val_accuracy: 0.5556
Epoch 17/20
3/3 [==============================] - 15s 5s/step - loss: 0.9614 - accuracy: 0.8148 - val_loss: 1.0456 - val_accuracy: 0.5556
Epoch 18/20
3/3 [==============================] - 15s 5s/step - loss: 0.9276 - accuracy: 0.8889 - val_loss: 1.0383 - val_accuracy: 0.5556
Epoch 19/20
3/3 [==============================] - 15s 5s/step - loss: 0.9035 - accuracy: 0.9259 - val_loss: 1.0300 - val_accuracy: 0.5556
Epoch 20/20
3/3 [==============================] - 17s 6s/step - loss: 0.8676 - accuracy: 0.9383 - val_loss: 1.0209 - val_accuracy: 0.6667
```

Fig. 72. BERT with CNN for Sentiment 140 data set.

The BERT with CNN model is created over 20 iterations and got an accuracy of 0.93 is observed with this model. This is the highest accuracy we have obtained in this paper.

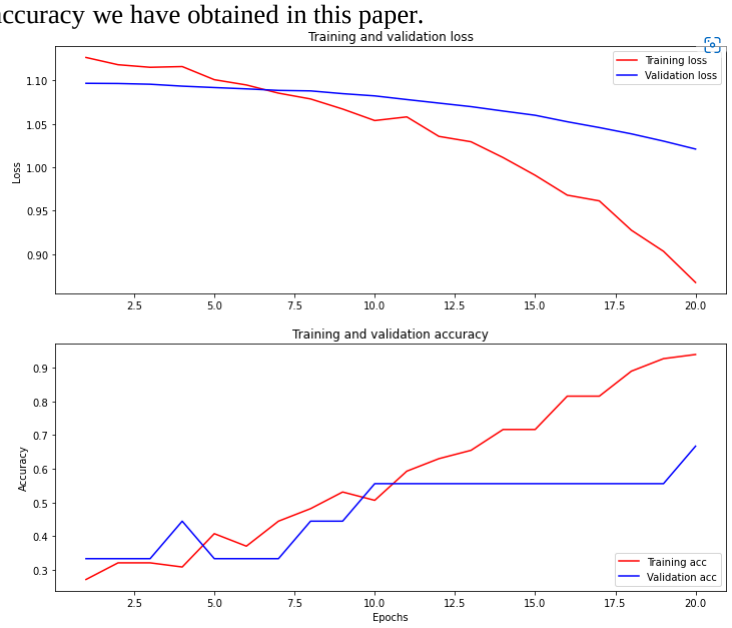

Fig. 73. Loss function and accuracy for BERT with CNN model.

Figure 38 shows the training and validation datasets loss function and accuracy. The loss function shows a decreasing trend for both validation and training data set and the accuracy graph shows an increasing trend over the iterations.

Chapter 4

Results

In this chapter the results of all the models are compared. This chapter is divided into three sets to compare and assess the obtained results.

1. Classification Algorithms
2. Deep Learning Algorithms
3. Complex Deep Learning Algorithms with Attention

4.1 Classification Algorithms

Different techniques for classifying the sentiment of the tweet on Sentiment 140 dataset was used. The dataset used was a random subset of the sentiment 140 dataset which contains 10000 entries. The first step is the NLP data preprocessing, where all the necessary transformations are carried out. Four models are built to classify the Tweet data set. A random forest classifier, Support Vector Machine, Naïve Bayes Classifier and Logistic regression are built on the data set.

The accuracy for each of these data is given below.

Classification Algorithm	Accuracy
Random Forest	0.70
Support Vector Machine	0.51
Naïve Bayes	0.63
Logistic Regression	0.73

Table 6. Comparison study for classification algorithms

Logistic regression and Random Forest have better accuracy compared to Support Vector Machine and Naïve Bayes classifiers. Logistic regression has an accuracy of 73% and Support Vector Machine has an accuracy of 72%.

4.2 Deep Learning Algorithms

Several Deep learning algorithms are developed for tweet classification. A convolutional Neural Network is trained the for the dataset. The CNN with and without regularization is given below.

Classification Algorithm	Accuracy
CNN	0.79
CNN with L2 Regularization and Dropout	0.50

Table 7. CNN Accuracy

The ordinary CNN has a greater accuracy, but the model was likely overfitted. L2 Regularization and a dropout layer is introduced to rectify the overfitting of the data.

A Recurrent Neural Network is trained, and the accuracy is as follows.

Classification Algorithm	Accuracy
RNN	0.57
RNN with L2 Regularization and Dropout	0.52

Table 8. RNN Accuracy

The ordinary RNN showed trends of overfitting. The second result however is after adjusting the overfitting phenomenon.

Two more models, Long Short-Term Memory (LSTM) and Bidirectional Long Short-Term Memory (Bi-LSTM) are built in this section and the table below shows the accuracy comparison of all four of the deep learning mechanisms.

Deep Learning Algorithm	Accuracy
CNN with L2 Regularization and Dropout	0.82
RNN with L2 Regularization and Dropout	0.56
LSTM	0.79
Bi-LSTM	0.79

Table 9. Deep Learning algorithm accuracy.

The CNN with L2 Regularization has comparatively higher accuracy than any other model. Both Long Short Term Model and Bidirectional Long Short Term Memory model has almost equal accuracy.

4.3 Complex Deep Learning Algorithms with Attention

A random sample of 100 is used for this study. This is because of the original data is huge for the local system to carry out these kinds of complex algorithms. A Bidirectional Encoder Representations from Transformers (BERT) model and a BERT model with CNN is carried out and the results can be summarized as below.

Attention Algorithms	Accuracy
BERT	0.52
BERT with CNN	0.93

Table 10. Deep learning with attention algorithms.

BERT with CNN has a higher accuracy than any other model. The BERT with CNN has an accuracy of 93% which outperforms any other model in this study.

4 Conclusion

Hate speech in general is an expression of hate towards an individual or a group pf people (Borkelo, 2014). With the development of public social medias, it had become a platform for anyone to talk about anything without any constraints. This is all good to an extend but the downside is that there is no proper way to monitor these platform speeches effectively. Hate speeches can have a deeper effect on the society in a way that it corrupts the way people's opinions.

In this study a detailed analysis method is used to detect and classify hate speeches in twitter using the Sentiment 140 data set. The main aim was to develop an effective algorithm to detect hate speeches in social media platform.

We created four classification algorithms namely, Random Forest, Support Vector Machine (SVM), Naïve Bayes, and Logistic Regression. Out of the four classification algorithms Random Forest has an accuracy of 72% and the logistic regression has an accuracy of 73%.

The next study was to develop deep learning algorithms for the study. A Convolutional Neural Network, A recurrent neural network with LSTM, Long Short-Term Memory (LSTM) and a Bidirectional LSTM was created for the data set. A convolutional Neural Network (CNN) with L2 regularization and a dropout layer was developed, and the model has an accuracy of 82%. Both LSTM and Bidirectional LSTM has an accuracy of 79%.

Furthermore, deep learning algorithms with attention is developed for the Sentiment 140 data set. A Bidirectional Encoder Representations from Transformers (BERT) is developed and has an accuracy of 52%.

Finally, a new algorithm BERT with CNN, in which the BERT output is used as an input for the CNN layer is used to classify the twitter data. This algorithm has an accuracy of 93% outperforming all other algorithms.

4.1 Limitations of the Study

The data analysis done on the entire paper is a random subset of the original Sentiment 140 data set which contains 1.6 million data entries. However, the huge amount of the data set is nearly impossible for the local computers to process when in case of working on deep learning algorithms like BERT. Since only a part of the data is considered, the models were forced to learn from a limited resource. This in fact would have a huge impact on the precision of the models.

Additional limitations on the study are that the data set used does not contain any suggestion on the topic of the tweet, which in fact restricts the study into a more general classification, and not allowing us to dive deep into a particular topic such us 'women's rights'.

The sentiment of the data was assigned before the study, the credibility of the method for deciding the polarity, or to know if there is any underlying corruption in the data set itself need to be addressed.

The entire data set used contain only two sentiments the negative tweets and the positive tweets and lacked any tweets that are neutral tweets.

4.2 Future research ideas

There is an enormous scope for future study with this dataset. Some of the interesting scope that can be added to this study are as follows:

- To redevelop the entire study using the entire dataset on a more powerful system and to compare the results.
- To extract more information on the negative tweets by doing a separate study on them.
- In general, develop other models that perform with a greater precision that the models created in this paper.

5 Appendix

5.1 Appendix A-Classification Algorithms.

```
#imorting the packages
import numpy as np
import keras as ks
import tensorflow as tf
import pandas as pd
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
import seaborn as sns
from wordcloud import WordCloud, STOPWORDS
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
from keras.preprocessing.text import Tokenizer
import nltk
import pandas as pd
import gensim
from gensim.models import Word2Vec, KeyedVectors
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.layers import Dense,Input, GlobalMaxPooling1D, Dropout
from tensorflow.keras.layers import Conv1D, MaxPooling1D,Embedding
from tensorflow.keras.layers import LSTM,Embedding
from tensorflow.keras.models import Model
from keras.optimizers import Adam
from sklearn.metrics import roc_auc_score
from keras.layers import LSTM, Bidirectional, GlobalMaxPool1D, Dropout
from keras.preprocessing import sequence
from keras.models import Sequential


#Loading the dataset
headerList = ['Polarity','Unique_Number', 'Date', 'Query','Id','Tweet']
data=pd.read_csv("traindata.csv",encoding="ISO-8859-1")
data.to_csv("d1.csv", header=headerList, index=False)

# display modified csv file
train = pd.read_csv("d1.csv")
print('\nModified file:')
train.head()
```

```
#Exploratory Data Analysis
plt.rcParams['figure.figsize']=[10,5]
plt.rcParams['figure.dpi']
sns.countplot('Polarity',data=train)
plt.title('Positive and Negative Tweets')

#Random sample
df=pd.read_csv("d1.csv")
df=train.sample(10000)



#data exploration and preprocessing
p_data=df[df['Polarity']==4]
n_data=df[df['Polarity']==0]


#World Cloud for positive tweets
comment_words = ''
stopwords = set(STOPWORDS)
# iterate through the csv file
for val in p_data.Tweet:

    # typecaste each val to string
    val = str(val)

    # split the value
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    comment_words += " ".join(tokens)+" "

wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(comment_words)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()


#wordcloud for negative tweets
comment_words = ''
stopwords = set(STOPWORDS)
```

```
# iterate through the csv file
for val in n_data.Tweet:

    # typecaste each val to string
    val = str(val)

    # split the value
    tokens = val.split()

    # Converts each token into lowercase
    for i in range(len(tokens)):
        tokens[i] = tokens[i].lower()

    comment_words += " ".join(tokens)+" "

wordcloud = WordCloud(width = 800, height = 800,
                background_color ='white',
                stopwords = stopwords,
                min_font_size = 10).generate(comment_words)

# plot the WordCloud image
plt.figure(figsize = (8, 8), facecolor = None)
plt.imshow(wordcloud)
plt.axis("off")
plt.tight_layout(pad = 0)

plt.show()


# replacing the value 4 by 1
df['Polarity'] = df['Polarity'].replace({4: 1})
df.info()


#data preprocessing for NLP
c=[]
from nltk.stem.porter import PorterStemmer
import nltk
import re
from nltk.corpus import stopwords
ps=PorterStemmer()
for i in df.index:
    n=re.sub('[^a-zA-Z]',' ', df['Tweet'][i])
    n=n.lower()
    n=n.split()
    n=[ps.stem(word) for word in n if word not in stopwords.words('english') ]
    n=" ".join(n)

    c.append(n)

df.Tweet=c
df.head()
```

```
# Model Building
#model 1 ---- Random Forest
from sklearn.metrics import classification_report, accuracy_score, confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.ensemble import RandomForestClassifier
from sklearn.pipeline import Pipeline
classifier=Pipeline([("tfidf", TfidfVectorizer()) , ("classifier", RandomForestClassifier(n_estimators=100))])
X_train, X_test,y_train, y_test=train_test_split( df['Tweet'], df['Polarity'], test_size=0.30, random_state=0, shuf-
fle=True)
y=df.iloc[:, 1].values
classifier.fit(X_train, y_train)
y_pred=classifier.predict(X_test)
print(accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))

#Model 2----- Support Vector Machine
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
X_train, X_test,y_train, y_test=train_test_split(df['Tweet'], df['Polarity'], test_size=0.30, random_state=0, shuf-
fle=True)
svm=Pipeline([("tfidf",TfidfVectorizer()),("classifier",SVC(C=100,gamma='auto'))])
svm.fit(X_train,y_train)
y_pred=svm.predict(X_test)
print(accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))

#Model 3---- Naive Bayes
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import CountVectorizer
cv=CountVectorizer(max_features=1500)
x=cv.fit_transform(c).toarray()
y=df.iloc[:, 0].values
X_train, X_test,y_train, y_test=train_test_split(x,y, test_size=0.30, random_state=0, shuffle=True)
classi=GaussianNB()
classi.fit(X_train, y_train)
y_pred=classi.predict(X_test)
print(accuracy_score(y_test, y_pred))
print(classification_report(y_test, y_pred))

#Model-- Logistic Regression
x=df['Tweet']
y=df['Polarity']
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.30)
count_vect=CountVectorizer()
x_train_counts=count_vect.fit_transform(x_train)
tfidf_transformer=TfidfTransformer()
x_train_tfidf=tfidf_transformer.fit_transform(x_train_counts)
vectorizer=TfidfVectorizer()
x_train_vect=vectorizer.fit_transform(x_train)
clf=LogisticRegression(solver='lbfgs')
clf.fit(x_train_tfidf,y_train)
```

```
text_clf=Pipeline([('tfidf', TfidfVectorizer()),
                   ('clf',LogisticRegression()),])
text_clf.fit(x_train,y_train)
predictions=text_clf.predict(x_test)
print(metrics.classification_report(y_test,predictions))
print(metrics.accuracy_score(y_test,predictions))

# Neural Networks
des=df['Tweet'].values
#Tokenize the description
des_tok=[nltk.word_tokenize(description) for description in des]
#model building
model=Word2Vec(des_tok, min_count=1)
#Predict the Word2Vec
model.wv['love']


#Finding the similar word
model.wv.most_similar('love')

#Convolutional Neural Network
x_train, x_test,y_train, y_test=train_test_split(df['Tweet'],df['Polarity'], test_size=0.10)
#converting into sequence
max_vocab_size=20000
tokenizer=Tokenizer(num_words=max_vocab_size)
tokenizer.fit_on_texts(x_train)
sequences_train=tokenizer.texts_to_sequences(x_train)
sequence_test=tokenizer.texts_to_sequences(x_test)
sequences_train
#unique tokens
word2idx=tokenizer.word_index
v=len(word2idx)
#pad sequences for equal sequences
data_train=pad_sequences(sequences_train)
t=data_train.shape[1]
#pad the test datset
data_test=pad_sequences(sequence_test,maxlen=t)
#Building the model
D=20
i=Input(shape=(t,) )
x=Embedding(v+1, D)(i)
x=Conv1D(32,3,activation='relu', padding='same')(x)
x=MaxPooling1D(3)(x)
x=Conv1D(64,3,activation='relu', padding='same')(x)
x=MaxPooling1D(3)(x)
x=Conv1D(128,3,activation='relu', padding='same')(x)
x=GlobalMaxPooling1D()(x)
x=Dense(1,activation='sigmoid')(x)
model=Model(i,x)
model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accurancy'])
model.compile(loss='binary_crossentropy', metrics=['accuracy'])
#train the model
r=model.fit(x=data_train, y=y_train,epochs=4,validation_data=(data_test, y_test))
```

```
#The loss function
import matplotlib.pyplot as plt
plt.plot(r.history['loss'],label='Loss')
plt.plot(r.history['val_loss'],label='Validation_ loss')
plt.legend()
plt.show()

#The accuracy
import matplotlib.pyplot as plt
plt.plot(r.history['accuracy'],label='Accuracy')
plt.plot(r.history['val_accuracy'],label='Validation_ accuracy')
plt.legend()
plt.show()


# Convolutional Neural Network with L2 Regularization and Dropout
x_train, x_test,y_train, y_test=train_test_split(df['Tweet'],df['Polarity'], test_size=0.30)
#converting into sequence
max_vocab_size=20000
tokenizer=Tokenizer(num_words=max_vocab_size)
tokenizer.fit_on_texts(x_train)
sequences_train=tokenizer.texts_to_sequences(x_train)
sequence_test=tokenizer.texts_to_sequences(x_test)
sequences_train
#unique tokens
word2idx=tokenizer.word_index
v=len(word2idx)
#pad sequences for equal sequences
data_train=pad_sequences(sequences_train)
t=data_train.shape[1]
#pad the test datset
data_test=pad_sequences(sequence_test,maxlen=t)
#Building the model with L2 Regularization and dropout
D=20
i=Input(shape=(t,) )
x=Embedding(v+1, D)(i)
x=Conv1D(32,3,kernel_regularizer=ks.regularizers.l2(0.001),activation='relu', padding='same')(x)
x=ks.layers.Dropout(0.5)(x)
x=MaxPooling1D(3)(x)
x=Conv1D(64,3,kernel_regularizer=ks.regularizers.l2(0.001),activation='relu', padding='same')(x)
x=ks.layers.Dropout(0.5)(x)
x=MaxPooling1D(3)(x)
x=Conv1D(128,3,kernel_regularizer=ks.regularizers.l2(0.001),activation='relu', padding='same')(x)
x=ks.layers.Dropout(0.5)(x)
x=GlobalMaxPooling1D()(x)
x=Dense(1,activation='sigmoid')(x)
model=Model(i,x)
model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accurancy'])
model.compile(loss='binary_crossentropy', metrics=['accuracy'])
#train the model
r=model.fit(x=data_train, y=y_train,epochs=4,validation_data=(data_test, y_test))
```

```
#Recurrent Neural Network
#Building the model
x_train, x_test,y_train, y_test=train_test_split(df['Tweet'],df['Polarity'], test_size=0.30)
D=30
M=15
i=Input(shape=(t,) )
x=Embedding(v+1, D)(i)
x=LSTM(M,return_sequences=True)(x)
x=MaxPooling1D()(x)

x=Dense(1,activation='sigmoid')(x)
model=Model(i,x)



model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accurancy'])
model.compile(loss='binary_crossentropy', metrics=['accuracy'])

#train the model
r=model.fit(x=data_train, y=y_train,epochs=4,validation_data=(data_test, y_test))

#The loss function
import matplotlib.pyplot as plt
plt.plot(r.history['loss'],label='Loss')
plt.plot(r.history['val_loss'],label='Validation_ loss')
plt.legend()
plt.show()

#The accuracy
import matplotlib.pyplot as plt
plt.plot(r.history['accuracy'],label='Accuracy')
plt.plot(r.history['val_accuracy'],label='Validation_ accuracy')
plt.legend()
plt.show()

#Recurrent Neural Network
#Building the model with L2 Regularization and dropout
x_train, x_test,y_train, y_test=train_test_split(df['Tweet'],df['Polarity'], test_size=0.30)
D=30
M=15
i=Input(shape=(t,) )
x=Embedding(v+1, D)(i)
x=LSTM(M,kernel_regularizer=ks.regularizers.l2(0.001),return_sequences=True)(x)
x=ks.layers.Dropout(0.5)(x)
x=MaxPooling1D()(x)

x=Dense(1,activation='sigmoid')(x)
model=Model(i,x)
model.compile(optimizer='adam',loss='binary_crossentropy',metrics=['accurancy'])
model.compile(loss='binary_crossentropy', metrics=['accuracy'])
#train the model
r=model.fit(x=data_train, y=y_train,epochs=3,validation_data=(data_test, y_test))
```

```
#The loss function
import matplotlib.pyplot as plt
plt.plot(r.history['loss'],label='Loss')
plt.plot(r.history['val_loss'],label='Validation_ loss')
plt.legend()
plt.show()

#The accuracy
import matplotlib.pyplot as plt
plt.plot(r.history['accuracy'],label='Accuracy')
plt.plot(r.history['val_accuracy'],label='Validation_ accuracy')
plt.legend()
plt.show()

#LSTM
max_review_length = 80
x_train, x_test,y_train, y_test=train_test_split(df['Tweet'],df['Polarity'], test_size=0.10)
tokenizer=Tokenizer(num_words=20000, lower=True, oov_token='<UNK>')
tokenizer.fit_on_texts(x_train)
word_index=tokenizer.word_index
tokenizer=Tokenizer(num_words=20000, lower=True, oov_token='<UNK>')
tokenizer.fit_on_texts(x_test)
word_index=tokenizer.word_index
d_seq_train=tokenizer.texts_to_sequences(x_train)
d_seq_test=tokenizer.texts_to_sequences(x_test)
x_train = sequence.pad_sequences(d_seq_train, truncating='pre', padding='pre', maxlen=max_review_length)
x_test = sequence.pad_sequences(d_seq_test, truncating='pre', padding='pre', maxlen=max_review_length)

model=Sequential()
model.add(Embedding(input_dim=50000, output_dim=10, mask_zero=True))
model.add(LSTM(units=50))
model.add(Dense(units=1,activation='sigmoid'))
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
batch_size=32
r=model.fit(x_train, y_train, batch_size=batch_size,epochs=4, validation_data=(x_test,y_test))


#The loss function
import matplotlib.pyplot as plt
plt.plot(r.history['loss'],label='Loss')
plt.plot(r.history['val_loss'],label='Validation_ loss')
plt.legend()
plt.show()

#The accuracy
import matplotlib.pyplot as plt
plt.plot(r.history['accuracy'],label='Accuracy')
plt.plot(r.history['val_accuracy'],label='Validation_ accuracy')
plt.legend()
plt.show()


#Bi directional LSTM
```

```
x_train, x_test,y_train, y_test=train_test_split(df['Tweet'],df['Polarity'], test_size=0.50)
max_review_length = 80
x_train, x_test,y_train, y_test=train_test_split(df['Tweet'],df['Polarity'], test_size=0.10)
tokenizer=Tokenizer(num_words=20000, lower=True, oov_token='<UNK>')
tokenizer.fit_on_texts(x_train)
word_index=tokenizer.word_index
tokenizer=Tokenizer(num_words=20000, lower=True, oov_token='<UNK>')
tokenizer.fit_on_texts(x_test)
word_index=tokenizer.word_index
d_seq_train=tokenizer.texts_to_sequences(x_train)
d_seq_test=tokenizer.texts_to_sequences(x_test)
x_train = sequence.pad_sequences(d_seq_train, truncating='pre', padding='pre', maxlen=max_review_length)
x_test = sequence.pad_sequences(d_seq_test, truncating='pre', padding='pre', maxlen=max_review_length)

model=Sequential()
model.add(Embedding(input_dim=50000, output_dim=10, mask_zero=True))
model.add(Bidirectional(LSTM(units=50)))
model.add(Dense(units=1,activation='sigmoid'))
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
batch_size=32
r=model.fit(x_train, y_train, batch_size=batch_size,epochs=4, validation_data=(x_test,y_test))


#The loss function
import matplotlib.pyplot as plt
plt.plot(r.history['loss'],label='Loss')
plt.plot(r.history['val_loss'],label='Validation_ loss')
plt.legend()
plt.show()

#The accuracy
import matplotlib.pyplot as plt
plt.plot(r.history['accuracy'],label='Accuracy')
plt.plot(r.history['val_accuracy'],label='Validation_ accuracy')
plt.legend()
plt.show()
```

5.2 Appendix B-Complex Neural Network with Attention.

```
#Sentiment classification using BERT
#Random sample
import pandas as pd
df=pd.read_csv("d1.csv")
df=train.sample(100)
# replacing the value 4 by 1
df['Polarity'] = df['Polarity'].replace({4: 1})
df.info()
#data preprocessing for NLP
c=[]
from nltk.stem.porter import PorterStemmer
import nltk
import re
```

```
from nltk.corpus import stopwords
ps=PorterStemmer()
for i in df.index:
    n=re.sub('[^a-zA-Z]',' ', df['Tweet'][i])
    n=n.lower()
    n=n.split()
    n=[ps.stem(word) for word in n if word not in stopwords.words('english') ]
    n=" ".join(n)

    c.append(n)
df.Tweet=c
df.head()
#values of original dataframe
d1_train = df.sample(frac = 0.5)

# Creating dataframe with
# rest of the 50% values
d2_test = df.drop(d1_train.index)
(X_train, y_train),(X_test,y_test),preproc=text.texts_from_df(train_df=d1_train,
                                          text_column='Tweet',
                                          label_columns='Polarity',
                                          val_df=d2_test,
                                          maxlen=50,
                                          preprocess_mode='bert')
model=text.text_classifier(name='bert',
                      train_data=(X_train, y_train),
                      preproc=preproc)
learner=ktrain.get_learner(model=model,train_data=(X_train,y_train),
                val_data=(X_test,y_test),
                batch_size=64)
learner.fit_onecycle(lr=2e-5,epochs=1)

#BERT +CNN
import torch
import torch.nn as nn
class MixModel(nn.Module):
    def __init_(self,pre_trained='bert-base-uncased'):
        super()._init_()
        self.bert = AutoModel.from_pretrained('distilbert-base-uncased')
        self.hidden_size = self.bert.config.hidden_size
        self.conv = nn.Conv1d(in_channels=768, out_channels=256, kernel_size=5, padding='valid', stride=1)
        self.relu = nn.ReLU()
        self.pool = nn.MaxPool1d(kernel_size= 64- 5 + 1)
        print(11)
        self.dropout = nn.Dropout(0.3)
        print(12)
        self.clf = nn.Linear(self.hidden_size*2,6)
        print(13)



    def forward(self,inputs, mask , labels):
```

```
        cls_hs = self.bert(input_ids=inputs,attention_mask=mask, return_dict= False)
        x=cls_hs[0]
        print(cls_hs[0])
        print(len(cls_hs[0]))
        print(cls_hs[0].size())
        #x = torch.cat(cls_hs,0) # x= [416, 64, 768]
        x = x.permute(0, 2, 1)
        x = self.conv(x)
        x = self.relu(x)
        x = self.pool(x)
        x = self.dropout(x)
        x = self.clf(x)
        return x
!pip install tensorflow-text
import pandas as pd
import numpy as np
import tensorflow as tf
import tensorflow_text as text
from sklearn.model_selection import train_test_split
data=pd.read_csv('/content/testdata.csv')
headerList = ['Polarity','Unique_Number', 'Date', 'Query','Id','Tweet']
data.to_csv("d1.csv", header=headerList, index=False)
# display modified csv file
train_data = pd.read_csv("d1.csv", encoding= 'utf-8')
print('\nModified file:')
#print(train_data)
#data_test.to_csv("d2.csv", header=headerList, index=False)
train_data
# display modified csv file
#test_data = pd.read_csv("d2.csv")
#print('\nModified file:')
#print(test_data)
train_data['Polarity'] = train_data['Polarity'].replace({4: 1})
train_data.info()

train_data['data_type'] = ['not_set']*train_data.shape[0]

train_data.tail()
train_data['data_type'].where(~(train_data.Polarity==2), other='neutral', inplace=True)
train_data['data_type'].where(~(train_data.Polarity==4), other='positive', inplace=True)
train_data['data_type'].where(~(train_data.Polarity==0), other='negative', inplace=True)
train_data
import pandas as pd
#df=pd.read_csv("train_data.csv")
df=train_data.sample(100)
import nltk
nltk.download('stopwords')
c=[]
from nltk.stem.porter import PorterStemmer
import nltk
import re
from nltk.corpus import stopwords
ps=PorterStemmer()
```

```
    for i in df.index:
        n=re.sub('[^a-zA-Z]',' ', df['Tweet'][i])
        n=n.lower()
        n=n.split()
        n=[ps.stem(word) for word in n if word not in stopwords.words('english') ]
        n=" ".join(n)

        c.append(n)
    df.Tweet=c
    df.head()
    x_train, x_test,y_train, y_test=train_test_split(df['Tweet'],df['Polarity'], test_size=0.10)
    def build_CNN_classifier_model():
        text_input = tf.keras.layers.Input(shape=(), dtype=tf.string, name='text')
        preprocessing_layer = hub.KerasLayer(tfhub_handle_preprocess, name='preprocessing')
        encoder_inputs = preprocessing_layer(text_input)
        encoder = hub.KerasLayer(tfhub_handle_encoder, trainable=True, name='BERT_encoder')
        outputs = encoder(encoder_inputs)
        #net = outputs['pooled_output'] # [batch_size, 768].
        net = sequence_output = outputs["sequence_output"] # [batch_size, seq_length, 768]


        net = tf.keras.layers.Conv1D(32, (2), activation='relu')(net)
        #net = tf.keras.layers.MaxPooling1D(2)(net)

        net = tf.keras.layers.Conv1D(64, (2), activation='relu')(net)
        #net = tf.keras.layers.MaxPooling1D(2)(net)
        net = tf.keras.layers.GlobalMaxPool1D()(net)

    #    net = tf.keras.layers.Flatten()(net)

        net = tf.keras.layers.Dense(512, activation="relu")(net)

        net = tf.keras.layers.Dropout(0.1)(net)
    #  net = tf.keras.layers.Dense(1, activation=None, name='classifier')(net)
        net = tf.keras.layers.Dense(3, activation="softmax", name='classifier')(net)

        return tf.keras.Model(text_input, net)
    bert_model_name = 'small_bert/bert_en_uncased_L-4_H-512_A-8'
    #bert_model_name = 'bert_en_uncased_L-12_H-768_A-12'
    map_name_to_handle = {
        'bert_en_uncased_L-12_H-768_A-12':
            'https://tfhub.dev/tensorflow/bert_en_uncased_L-12_H-768_A-12/3',
        'bert_en_cased_L-12_H-768_A-12':
            'https://tfhub.dev/tensorflow/bert_en_cased_L-12_H-768_A-12/3',
        'bert_multi_cased_L-12_H-768_A-12':
            'https://tfhub.dev/tensorflow/bert_multi_cased_L-12_H-768_A-12/3',
        'small_bert/bert_en_uncased_L-2_H-128_A-2':
            'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-2_H-128_A-2/1',
        'small_bert/bert_en_uncased_L-2_H-256_A-4':
            'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-2_H-256_A-4/1',
        'small_bert/bert_en_uncased_L-2_H-512_A-8':
            'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-2_H-512_A-8/1',
        'small_bert/bert_en_uncased_L-2_H-768_A-12':
```

```
    'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-2_H-768_A-12/1',
'small_bert/bert_en_uncased_L-4_H-128_A-2':
    'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-4_H-128_A-2/1',
'small_bert/bert_en_uncased_L-4_H-256_A-4':
    'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-4_H-256_A-4/1',
'small_bert/bert_en_uncased_L-4_H-512_A-8':
    'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-4_H-512_A-8/1',
'small_bert/bert_en_uncased_L-4_H-768_A-12':
    'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-4_H-768_A-12/1',
'small_bert/bert_en_uncased_L-6_H-128_A-2':
    'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-6_H-128_A-2/1',
'small_bert/bert_en_uncased_L-6_H-256_A-4':
    'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-6_H-256_A-4/1',
'small_bert/bert_en_uncased_L-6_H-512_A-8':
    'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-6_H-512_A-8/1',
'small_bert/bert_en_uncased_L-6_H-768_A-12':
    'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-6_H-768_A-12/1',
'small_bert/bert_en_uncased_L-8_H-128_A-2':
    'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-8_H-128_A-2/1',
'small_bert/bert_en_uncased_L-8_H-256_A-4':
    'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-8_H-256_A-4/1',
'small_bert/bert_en_uncased_L-8_H-512_A-8':
    'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-8_H-512_A-8/1',
'small_bert/bert_en_uncased_L-8_H-768_A-12':
    'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-8_H-768_A-12/1',
'small_bert/bert_en_uncased_L-10_H-128_A-2':
    'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-10_H-128_A-2/1',
'small_bert/bert_en_uncased_L-10_H-256_A-4':
    'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-10_H-256_A-4/1',
'small_bert/bert_en_uncased_L-10_H-512_A-8':
    'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-10_H-512_A-8/1',
'small_bert/bert_en_uncased_L-10_H-768_A-12':
    'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-10_H-768_A-12/1',
'small_bert/bert_en_uncased_L-12_H-128_A-2':
    'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-12_H-128_A-2/1',
'small_bert/bert_en_uncased_L-12_H-256_A-4':
    'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-12_H-256_A-4/1',
'small_bert/bert_en_uncased_L-12_H-512_A-8':
    'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-12_H-512_A-8/1',
'small_bert/bert_en_uncased_L-12_H-768_A-12':
    'https://tfhub.dev/tensorflow/small_bert/bert_en_uncased_L-12_H-768_A-12/1',
'albert_en_base':
    'https://tfhub.dev/tensorflow/albert_en_base/2',
'electra_small':
    'https://tfhub.dev/google/electra_small/2',
'electra_base':
    'https://tfhub.dev/google/electra_base/2',
'experts_pubmed':
    'https://tfhub.dev/google/experts/bert/pubmed/2',
'experts_wiki_books':
    'https://tfhub.dev/google/experts/bert/wiki_books/2',
'talking-heads_base':
    'https://tfhub.dev/tensorflow/talkheads_ggelu_bert_en_base/1',
```

```
}

map_model_to_preprocess = {
    'bert_en_uncased_L-12_H-768_A-12':
        'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/1',
    'bert_en_cased_L-12_H-768_A-12':
        'https://tfhub.dev/tensorflow/bert_en_cased_preprocess/1',
    'small_bert/bert_en_uncased_L-2_H-128_A-2':
        'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/1',
    'small_bert/bert_en_uncased_L-2_H-256_A-4':
        'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/1',
    'small_bert/bert_en_uncased_L-2_H-512_A-8':
        'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/1',
    'small_bert/bert_en_uncased_L-2_H-768_A-12':
        'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/1',
    'small_bert/bert_en_uncased_L-4_H-128_A-2':
        'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/1',
    'small_bert/bert_en_uncased_L-4_H-256_A-4':
        'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/1',
    'small_bert/bert_en_uncased_L-4_H-512_A-8':
        'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/1',
    'small_bert/bert_en_uncased_L-4_H-768_A-12':
        'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/1',
    'small_bert/bert_en_uncased_L-6_H-128_A-2':
        'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/1',
    'small_bert/bert_en_uncased_L-6_H-256_A-4':
        'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/1',
    'small_bert/bert_en_uncased_L-6_H-512_A-8':
        'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/1',
    'small_bert/bert_en_uncased_L-6_H-768_A-12':
        'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/1',
    'small_bert/bert_en_uncased_L-8_H-128_A-2':
        'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/1',
    'small_bert/bert_en_uncased_L-8_H-256_A-4':
        'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/1',
    'small_bert/bert_en_uncased_L-8_H-512_A-8':
        'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/1',
    'small_bert/bert_en_uncased_L-8_H-768_A-12':
        'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/1',
    'small_bert/bert_en_uncased_L-10_H-128_A-2':
        'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/1',
    'small_bert/bert_en_uncased_L-10_H-256_A-4':
        'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/1',
    'small_bert/bert_en_uncased_L-10_H-512_A-8':
        'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/1',
    'small_bert/bert_en_uncased_L-10_H-768_A-12':
        'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/1',
    'small_bert/bert_en_uncased_L-12_H-128_A-2':
        'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/1',
    'small_bert/bert_en_uncased_L-12_H-256_A-4':
        'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/1',
    'small_bert/bert_en_uncased_L-12_H-512_A-8':
        'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/1',
```

```
    'small_bert/bert_en_uncased_L-12_H-768_A-12':
        'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/1',
    'bert_multi_cased_L-12_H-768_A-12':
        'https://tfhub.dev/tensorflow/bert_multi_cased_preprocess/1',
    'albert_en_base':
        'https://tfhub.dev/tensorflow/albert_en_preprocess/1',
    'electra_small':
        'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/1',
    'electra_base':
        'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/1',
    'experts_pubmed':
        'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/1',
    'experts_wiki_books':
        'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/1',
    'talking-heads_base':
        'https://tfhub.dev/tensorflow/bert_en_uncased_preprocess/1',
}

tfhub_handle_encoder = map_name_to_handle[bert_model_name]
tfhub_handle_preprocess = map_model_to_preprocess[bert_model_name]

print(f'BERT model selected           : {tfhub_handle_encoder}')
print(f'Preprocess model auto-selected: {tfhub_handle_preprocess}')

import tensorflow_hub as hub
cnn_classifier_model = build_CNN_classifier_model()
bert_raw_result = cnn_classifier_model(tf.constant(x_train))
print(tf.sigmoid(bert_raw_result))
cnn_classifier_model.summary()
tf.keras.utils.plot_model(cnn_classifier_model)
loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
#metrics = tf.metrics.CategoricalCrossentropy()
#metrics = tf.metrics.Accuracy()
X_train_, X_test, y_train_, y_test = train_test_split(
    df.index.values,
    df.Polarity.values,
    test_size=0.10,
    random_state=42,
    stratify=df.Polarity.values,
)
X_train_, X_test, y_train_, y_test = train_test_split(
    df.index.values,
    df.Polarity.values,
    test_size=0.10,
    random_state=42,
    stratify=df.Polarity.values,
)
df['data_type'] = ['not_set']*df.shape[0]
df.loc[X_train, 'data_type'] = 'train'
df.loc[X_val, 'data_type'] = 'val'
df.loc[X_test, 'data_type'] = 'test'
df_train = df.loc[df["data_type"]=="train"]
df_train.head(5)
```

```
df_val = df.loc[df["data_type"]=="val"]
df_val.head(5)
df_test = df.loc[df["data_type"]=="test"]
df_test.head(5)
train_ds = tf.data.Dataset.from_tensor_slices((df_train.Tweet.values, df_train.Polarity.values))
val_ds = tf.data.Dataset.from_tensor_slices((df_val.Tweet.values, df_val.Polarity.values))
test_ds = tf.data.Dataset.from_tensor_slices((df_test.Tweet.values, df_test.Polarity.values))
train_ds = train_ds.shuffle(len(df_train)).batch(32, drop_remainder=False)
train_ds
val_ds = val_ds.shuffle(len(df_val)).batch(32, drop_remainder=False)
val_ds
test_ds = test_ds.shuffle(len(df_test)).batch(32, drop_remainder=False)
test_ds
from scipy import optimize, special
from official.nlp import optimization
epochs = 20
steps_per_epoch = tf.data.experimental.cardinality(train_ds).numpy()
num_train_steps = steps_per_epoch * epochs
num_warmup_steps = int(0.1*num_train_steps)

init_lr = 3e-5
optimizer = optimization.create_optimizer(init_lr=init_lr,
                                          num_train_steps=num_train_steps,
                                          num_warmup_steps=num_warmup_steps,
                                          optimizer_type='adamw')

cnn_classifier_model.compile(optimizer=optimizer,
                     loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
                     metrics=tf.keras.metrics.SparseCategoricalAccuracy('accuracy'))
bert_preprocess_model = hub.KerasLayer(tfhub_handle_preprocess)
for text_batch, label_batch in train_ds.take(1):
  for i in range(1):
    tweet = text_batch.numpy()[i]
    print(f'Tweet: {text_batch.numpy()[i]}')
    label = label_batch.numpy()[i]
    print(f'Label : {label}')

text_test = ['this is such an amazing movie!']
text_test = [tweet]

text_preprocessed = bert_preprocess_model(text_test)

print(f'Keys       : {list(text_preprocessed.keys())}')
print(f'Shape      : {text_preprocessed["input_word_ids"].shape}')
print(f'Word Ids : {text_preprocessed["input_word_ids"][0, :12]}')
print(f'Input Mask : {text_preprocessed["input_mask"][0, :12]}')
print(f'Type Ids : {text_preprocessed["input_type_ids"][0, :12]}')
bert_model = hub.KerasLayer(tfhub_handle_encoder)
bert_results = bert_model(text_preprocessed)

print(f'Loaded BERT: {tfhub_handle_encoder}')
print(f'Pooled Outputs Shape:{bert_results["pooled_output"].shape}')
print(f'Pooled Outputs Values:{bert_results["pooled_output"][0, :12]}')
```

```
print(f'Sequence Outputs Shape:{bert_results["sequence_output"].shape}')
print(f'Sequence Outputs Values:{bert_results["sequence_output"][0, :12]}')
print(f'Training model with {tfhub_handle_encoder}')
cnn_history = cnn_classifier_model.fit(x=train_ds,
                                       validation_data=val_ds,
                                       epochs=20,

                                      )
loss, accuracy = cnn_classifier_model.evaluate(test_ds)

print(f'Loss: {loss}')
print(f'Accuracy: {accuracy}')
import matplotlib.pyplot as plt
history_dict = cnn_history.history
print(history_dict.keys())

acc = history_dict['accuracy']
val_acc = history_dict['val_accuracy']
# acc = history_dict['binary_accuracy']
# val_acc = history_dict['val_binary_accuracy']
loss = history_dict['loss']
val_loss = history_dict['val_loss']

epochs = range(1, len(acc) + 1)
fig = plt.figure(figsize=(12, 10))
fig.tight_layout()

plt.subplot(2, 1, 1)
# "bo" is for "blue dot"
plt.plot(epochs, loss, 'r', label='Training loss')
# b is for "solid blue line"
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
# plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.legend()

plt.subplot(2, 1, 2)
plt.plot(epochs, acc, 'r', label='Training acc')
plt.plot(epochs, val_acc, 'b', label='Validation acc')
plt.title('Training and validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
```

6 References

1. Crabb, Jacob., Yang, Sherry., Zubova, Anna., (2019), 'Classifying Hate Speech: an overview', Available at:https://towardsdatascience.com/classifying-hate-speech-an-overview-d307356b9eba
2. Kim, Buomsoo., (2020),' Attention in Neural Networks - 1. Introduction to attention mechanism', Available at: Attention in Neural Networks - 1. Introduction to attention mechanism · Buomsoo Kim (buomsoo-kim.github.io)
3. Hore, Prodip., Chatterjee, Syam.,(2019) 'A Comprehensive Guide to Attention Mechanism in Deep Learning for Everyone' Available at: Attention Mechanism In Deep Learning | Attention Model Keras (analyticsvidhya.com)
4. Dataset:-Sentiment140, Available at: For Academics - Sentiment140 - A Twitter Sentiment Analysis Tool
5. Nguyen, Kiet., (2019), 'Hate Speech Detection on Social Media Text using the Bidirectional-LSTM Model', available at: (99+) (PDF) Hate Speech Detection on Social Media Text using the Bidirectional-LSTM Model | Kiet Nguyen - Academia.edu
6. Meaker, Morgan., (2019), ''HATE IS WAY MORE INTERESTING THAN THAT': WHY ALGORITHMS CAN'T STOP TOXIC SPEECH ONLINE', Available at: How Neo-Nazis Are Outsmarting A.I. (and Human) Hate Speech Detectors - Pacific Standard (psmag.com)
7. The Washington Post (10-04-2018), 'Transcript of Mark Zuckerberg's Senate hearing', Available at: Transcript of Mark Zuckerberg's Senate hearing - The Washington Post
8. Davidson, Thomas., Warmsley, Dana., Macy, Michael., Weber, Ingmar., (2017) ,' Automated Hate Speech Detection and the Problem of Offensive Language', Available at: Automated Hate Speech Detection and the Problem of Offensive Language (arxiv.org)
9. Gao, Lei., Huang, Ruihong., (2018), 'Detecting Online Hate Speech Using Context Aware Models', Available at: 1710.07395.pdf (arxiv.org)
10. Al Asif, Abdullah., (2020),' Bangla hate speech detection on social media using attention-based recurrent neural network', Available at: (99+) (PDF) Bangla hate speech detection on social media using attention-based recurrent neural network | Abdullah Al Asif - Academia.edu
11. Tontodimamma, Alice., Nissi, Eugenia., Sarra, Annalina., (2020), 'Thirty years of research into hate speech: topics of interest and their evolution', Available at : Thirty years of research into hate speech: topics of interest and their evolution | SpringerLink
12. Nguyen, Kiet., (2019),' Hate Speech Detection on Social Media Text using the Bi-GRU-LSTM-CNN Model', Available at: (99+) (PDF) Hate Speech Detection on Social Media Text using the Bi-GRU-LSTM-CNN Model | Kiet Nguyen - Academia.edu
13. Kargin, Kerem., (2021), ' NLP :Tokenization,Stemming, Lemmatization and Part of Speech Tagging', Available at :- https://medium.com/mlearning-ai/nlp-tokenization-stemming-lemmatization-and-part-of-speech-tagging-9088ac068768, (Accessed on 1 June 2022).
14. Pereira-Kohatsu, Juan., Quijano-Sanchez, Lar., Liberatore, Federico., Cmacho-Collados, Miguel., (2019), 'Detecting and Monitoring Hate Speech in Twitter', Available at :- https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6864473/, (Accessed on 1 June 2022).
15. Bjorkelo, Kristian., (2014),' What hate speech is, and isn't', Available at:- https://www.hate-speech.org/what- hate-speech-is-and-isnt/, (Accessed on 1 June 2022).
16. Phaisangittisagul, Ekachai., (2016), 'An Analysis of the Regularization between L2 and Dropout in Single Hidden Layer Neural Network', Available at :- https://uksim.info/isms2016/CD/data/0665a174.pdf, (Accessed on 20 March 2022).
17. Tewari, Ujwal., (2021), 'Regularization — Understanding L1 and L2 regularization for Deep Learning', Available at :- https://medium.com/analytics-vidhya/regularization-understanding-l1-and-l2-regularization-for-deep-learning-a7b9e4a409bf, (Accessed on 20 March 2022).
18. Python Wife, 'The Support Vector Machines (SVM) algorithm for NLP', Available at :- https://python-wife.com/the-support-vector-machines-svm-algorithm-for-nlp/, (Accessed on 20 March 2022).

19. Palachy, Shay., (2018), 'Understanding the scaling of L^2 regularization in the context of neural networks', Available at :- https://towardsdatascience.com/understanding-the-scaling-of-l%C2%B2-regularization-in-the-context-of-neural-networks-e3d25f8b50db (Accessed on 20 Feb 2022).
20. Brownlee, Jason., (2018), 'A Gentle Introduction to Dropout for Regularizing Deep Neural Networks', Available at :- https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/#:~:text=Dropout%20is%20implemented%20per-layer%20in%20a%20neural%20network.,as%20well%20as%20the%20visible%20or%20input%20layer. (Accessed on 20 Feb 2022).
21. Bahdanau, Dzmitry., Cho, Kyunghyun., Bengio, Yoshua., (2014), 'Neural Machine Translation by Jointly Learning to Align and Translate', Available at :- https://arxiv.org/abs/1409.0473 (Accessed on 3 Feb 2022).
22. Loginova, Kate., (2018), 'Attention in NLP', Available at :- https://medium.com/@edloginova/attention-in-nlp-734c6fa9d983, (Accessed on 2 Feb 2022).
23. Kulshrestha, Ria., (2020), ' Understanding Attentipn in Deep Learning', Available at :- https://towardsdatascience.com/attaining-attention-in-deep-learning-a712f93bdb1e, (Accessed on 1 Feb 2022).
24. Srivastava, Tushar., (2019), 'NLP: A quick guide to Stemming', Available at :- https://medium.com/@tusharsri/nlp-a-quick-guide-to-stemming-60f1ca5db49e (Accessed on 17 Feb 2022).
25. MonkeyLearn , 'Sentiment Analysis: A Definitive Guide', Available at :- https://monkeylearn.com/sentiment-analysis/ (Accessed on 17 Feb 2022).
26. GeekforGeeks, (2021), 'Natural Language Processing – Overview', Available at:- https://www.geeksforgeeks.org/natural-language-processing-overview/, (Accessed on 20 March 2022).
27. IBM Cloud Education, (2021), 'Natural Language Processing (NLP)', Available at:- What is Natural Language Processing? | IBM (Available at 20 March 2022).
28. Vaswani, Ashish., Shazeer, Noam., Parmar, Niki., Uszkoreit, Jakob., Jones, Llion., Gomez, Aidan., Kaiser, Lukasz., Polosukhin, Illia., (2017), 'Attention Is All You Need', Available at :- 1706.03762.pdf (arxiv.org), (Accessed on 15 March 2022).
29. Towards AI., (2021), 'Understanding BERT', Available at :- Understanding BERT – Towards AI, (Accessed on 25 March 2022).
30. Anand, Mrinal., (2019), 'Explainability Of BERT Through Attention', Available at :- Explainability Of BERT Through Attention | by Mrinal Anand | Analytics Vidhya | Medium, *(Accessed on 20 March 2022).
31. bforblack, (2021), 'Understanding the BERT Model', Available at:- Understanding the BERT Model. Bert is one the most popularly used… | by bforblack | Analytics Vidhya | Medium, (Accessed on 20 March 2022).
32. Lutkevich, Ben., 'DEFINITION BERT language model', Available at :- https://www.techtarget.com/searchenterpriseai/definition/BERT-language-model, (Accessed on 20 March 2022).
33. GeeksforGeeks, 'Explanation of BERT Model – NLP', Available at :- https://www.geeksforgeeks.org/explanation-of-bert-model-nlp/, (Accessed on 20 March 2022).
34. Horev, Rani., (2018), 'BERT Explained: State of the art language model for NLP', Available at :- https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270, Accessed on 20 March 2022).
35. 11. Verma, Yugesh., (2021), ' Complete Guide To Bidirectional LSTM (With Python Codes)', Available at :- https://analyticsindiamag.com/complete-guide-to-bidirectional-lstm-with-python-codes/, (accessed on 1 march 2022).
36. 12. Mungalpara, Jaimin., (2021), 'What does it mean by Bidirectional LSTM?', Available at https://medium.com/analytics-vidhya/what-does-it-mean-by-bidirectional-lstm-63d6838e34d9, (Accessed on 9 March 2022).
37. Belyadi Hoss., Haghighat, Alireza., (2021), ' Machine learning guide for oil and gas using python', Available at https://www.sciencedirect.com/book/9780128219294/machine-learning-guide-for-oil-and-gas-using-python, (Accessed on :- 20 Feb 2022).
38. Edgar, Thomas., Manz, David., (2017), ' Research methods for Cyber Security', Available at :- https://www.sciencedirect.com/book/9780128053492/research-methods-for-cyber-security, (Accessed on 21 Feb 2022).
39. 3. Vatsal(2021), 'Word2Vec Explained', Available at:- https://towardsdatascience.com/word2vec-explained-49c52b4ccb71, (Accessed on 22 Feb 2022).
40. 4. Kleppen, Eric., (2021), 'Hoe to Practice Word2Vec for NLP Using Python', Available at :- https://builtin.com/machine-learning/nlp-word2vec-python, (Accessed on 22 Feb, 2022).

41. 5. Python Wife, ‘Convolutional Neural Networks (CNNs) for NLP’, Available at :- https://python-wife.com/convolutional-neural-networks-cnns-for-nlp/, (Accessed on 21 Feb 2022).
42. 6. Baufeloussen, Omar., (2020), ‘Simple Explanation of Recurrent Neural Network (RNN)’, Available at :- https://medium.com/swlh/simple-explanation-of-recurrent-neural-network-rnn-1285749cc363, (Accessed on 22 Feb 2022).
43. 7. Pedamkar, Priya., ‘Reccurent Neural Network’, Available at :- https://www.educba.com/recurrent-neural-networks-rnn/, (Accessed on 20 Feb 2022).
44. 8. Chugh, Akshara, (2021), ‘Deep Learning | Introduction to Long Short Term Memory’, Available at:- https://www.geeksforgeeks.org/deep-learning-introduction-to-long-short-term-memory/, (Accessed on 20 Feb 2022).
45. 9. Aditianu, (2021), ‘Understanding of LSTM Networks’, Available at :- https://www.geeksforgeeks.org/understanding-of-lstm-networks/, Available at 20 Feb 2022.
46. 10. Python Wife, ‘Long Short-Term Memory (LSTMs) for NLP’, Available at :- https://python-wife.com/long-short-term-memory-lstms-for-nlp/ (Accessed on 21 Feb 2022).