SENTIMENT ANALYSIS OF TWEETS

FINAL PROJECT REPORT

submitted by

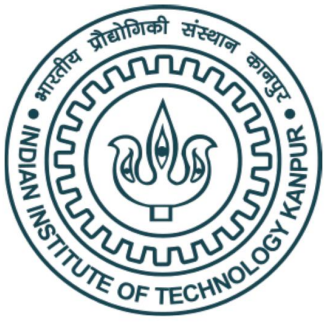
TEAM MATRIX

Team Members

AKKSHUNN VIJROY
ARAVIND UDAY
KAUMUDI KAMAL
SHIVANI DEO
SHREENATH BHARADWAJ
SIDDHANT SOLANKI (c)

to **Indian Institute of Technology Kanpur**

in partial fulfillment of the requirements for the award of certificate for the course in
Applied Machine Learning and Data Science - 2020



Department of Computer Science Engineering
Indian Institute of Technology Kanpur

July 2020

DECLARATION

We undersigned hereby declare that the project report "Sentiment Analysis of Tweets", submitted for partial fulfilment of the requirements for the award of the certificate on Applied Machine Learning and Data Science from IIT Kanpur is a bonafide work done by us. This submission represents our ideas in our own words and where ideas or words of others have been included, we have adequately and accurately cited and referenced the original sources. We also declare that we have adhered to ethics of academic honesty and integrity and have not misrepresented or fabricated any data or idea or fact or source in our submission. We understand that any violation of the above will be a cause for disciplinary action by the institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been obtained.

Date: 19th July, 2020

Akkshunn Vijroy
Aravind Uday
Kaumudi Kamal
Shivani Deo
Shreenath Bharadwaj
Siddhant Solanki (c)

CONTENTS

Sl No. Title

SENTIMENT ANALYSIS OF TWEETS

1. ABSTRACT

Social media is receiving a lot more attention nowadays than it did before. Public and private opinions about a wide variety of subjects are being expressed and spread continually via numerous social media. Twitter is an online micro-blogging and social-networking platform which allows users to write short status updates of maximum length 140 characters. It is a rapidly expanding service with over 200 million registered users out of which 100 million are active users and half of them log on twitter on a daily basis - generating nearly 250 million tweets per day. Twitter offers organizations a fast and effective way to analyze customers' perspectives on the critical success of the product in the marketplace. Developing a program for sentiment analysis helps to measure customers' perceptions computationally.

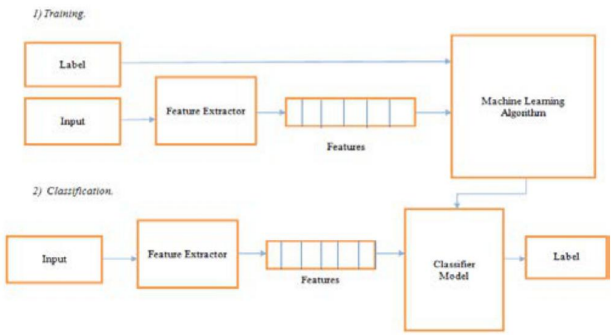
2. INTRODUCTION

This project reports on the design of sentiment analysis, extracting a vast number of tweets.
It addresses the problem of sentiment analysis in twitter; that is classifying tweets according to the sentiment expressed in them: positive, negative, or neutral. We hope to achieve a reflection of public opinion by analysing the sentiments expressed in the tweets. Analysing the general view is essential for many applications, including firms trying to find out the response of their products in the market, predicting political elections, and predicting socioeconomic phenomena like the stock exchange. This project aims to develop a functional classifier for accurate and automatic sentiment classification of an unknown tweet stream

Sentiment Analysis (opinion mining) is the automated process of identifying and extracting the subjective information that underlies a text. This can be either an opinion, a judgment, or a feeling about a particular topic or subject. The most common type of sentiment analysis is called ‘polarity detection’ and involves classifying a statement as ‘positive’, ‘negative’, or ‘neutral’.

Sentiment analysis tools use machine learning and natural language processing (NLP) to organize unstructured text data automatically. Sentiment analysis algorithms can learn from data samples to detect the polarity of Tweets in real-time.

It gives an overview of wider public opinion behind certain topics. The applications of sentiment analysis are broad and powerful. A lot of companies today are using sentiment analysis to assess how well the products are being received by the customers. Not only that SA is being used to keep track of political views and for monitoring and analysing social phenomenon. It is also providing



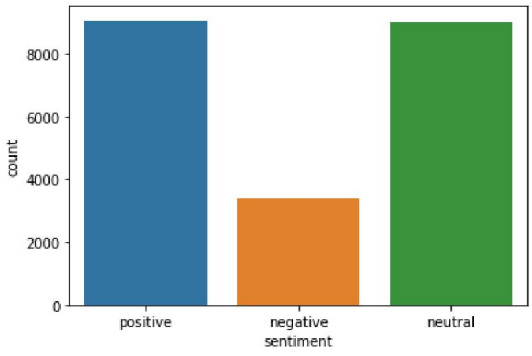
opportunities in the fields of market research.

The system architecture consists of the components such as Tweets extraction from twitter, pre-processing of data, feature extraction, Training set are defined for the given analysis. The training set is obtained by predefined set of positive, neutral or negative tweets and output obtained is positive, neutral and negative tweets. The Classifier will classify the tweets according the training set and regulates the polarity of the tweet as the output.

3. EDA (EXPERIMENTAL DATA ANALYSIS) ON DATASET

We used the dataset provided by Kaggle. There were 21,465 tweets on which we had to train the model and 5,398 tweets for testing. We had to predict the sentiment of a tweet, whether it is positive, negative or neutral. Below graph showing count on of sentiments in given training dataset: -

```
tweet_id,sentiment
264238274963451904,positive
```



Thus, from the above graph it is clear that the dataset which has been provided is an imbalanced dataset having positive and neutral tweets of about 42% and negative tweets of about 16%.

4. DATA PRE-PROCESSING

Raw data contains numerical value, punctuation, special character. These values can hamper the performance of model so before applying any text featurization first we need to do pre-processing of data

We used NLTK tool kit for pre-processing our dataset and imported tools which helped in the following ways:

4.1 Removing Noisy Data

We need to clean the tweets by reducing the noise as much as possible. There are various things which is unnecessary for a input in training text data mentions (@users), hashtags (#), punctuations, numbers and special characters. Also 'stop words' – common words (such as a, an, the) which do not add any meaning to the sentence and are hence bulking up the data.

4.2 Tokenization

In tokenization we convert groups of sentences into tokens. It is also called text segmentation or lexical analysis. It is basically splitting data into small chunks of words. We did this through NLTK library's word tokenize function

4.3 Normalization

In tokenization we came across various words such as punctuation, stop words (is, in, that, can etc), upper case words and lower-case words. After tokenization we are not focused on text level but on word level called Token Normalization. This is done by either stemming or lemmatization.

Stemming: This is a process of removing and replacing suffixes to get to the root form of the word, which is called stem. E.g.: Porter Stemming, Snowball Stemming

Lemmatization: returns the base or dictionary form of a word like WordNetLemmatizer.

5. METHODOLOGY

This section illustrates our implementations on traditional machine learning classifiers and neural network-based models in detail.

5.1 NAIVE BAYES CLASSIFIER

A Naive Bayes classifier is a probabilistic machine learning model that's basically based upon the Bayes theorem, which helps us compute the conditional probabilities of occurrence of two events based on the probabilities of occurrence of each individual event, encoding those probabilities is extremely useful. Naive Bayes models are called 'naive' algorithms because they make an assumption that the predictor variables are independent from each other. In other words, that the presence of a certain feature in a dataset is completely unrelated to the presence of any other feature.

$$P(y \mid x_1, \ldots, x_n) = \frac{P(y)P(x_1, \ldots, x_n \mid y)}{P(x_1, \ldots, x_n)}$$

Using the naive conditional independence assumption that

$$P(x_i|y, x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) = P(x_i|y),$$

$$P(y \mid x_1, \ldots, x_n) = \frac{P(y) \prod_{i=1}^{n} P(x_i \mid y)}{P(x_1, \ldots, x_n)}$$

Since $P(x_1,\ldots,x_n)$ is constant given the input, we can use the following classification rule, and we can use Maximum A Posteriori (MAP) estimation to estimate P(y) and $P(x_i|y)$; the former is then the relative frequency of class **y** in the training set.

$$P(y \mid x_1, \ldots, x_n) \propto P(y) \prod_{i=1}^{n} P(x_i \mid y)$$

$$\Downarrow$$

$$\hat{y} = \arg\max_y P(y) \prod_{i=1}^{n} P(x_i \mid y),$$

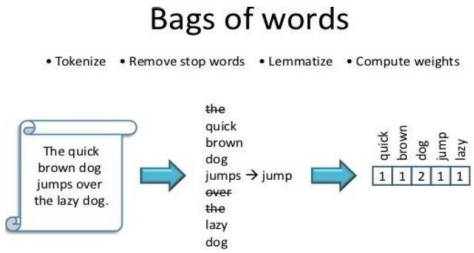
5.1.1 MODEL ARCHITECTURE

In our case we had a classification problem at hand which is sentiment analysis. The goal of classification is to take a single observation, extract some useful features, and thereby classify the observation into one of a set of discrete classes. We started it by doing these steps.

5.1.2 DATA VECTORIZATION

5.1.2.1 BAG OF WORDS (BoW)

BOW model is used for feature extraction in text data. It is a term used to specify the problems that have a collection of text data that needs to be processed. So, for this we imported count vectorizer and stored the frequency of words occurring in the tweet,



but this ends up in ignoring rare words which could have helped is in processing our data more efficiently.

COUNT VECTORIZER

Count Vectorizer works on Terms Frequency, i.e. counting the occurrences of tokens and building a sparse matrix of documents x tokens.

TF-IDF VECTORIZER

TF-IDF stands for term frequency-inverse document frequency. TF-IDF weight is a statistical measure used to evaluate how important a word is to a document in a collection or corpus. The importance increases proportionally to the number of times a word appears in the document but is offset by the frequency of the word in the corpus.

Term Frequency (TF):

$$TF(t) = \frac{Number\ of\ times\ term\ t\ appears\ in\ a\ document}{Total\ number\ of\ terms\ in\ the\ document}$$

Inverse Document Frequency (IDF):

$$IDF(t) = log_e(\frac{Total\ number\ of\ documents}{Number\ of\ documents\ with\ term\ t\ in\ it})$$

Thus,

$$TF - IDFscore = TF * IDF$$

5.1.3 TRAINING

From sci kit learn, we implemented multinomial Naive Bayes classifier which is suitable for classification with discrete features. We set the Additive (Laplace/Lidstone) smoothing parameter to 1 which is the default value. We have fitted the Naive Bayes classifier according to training data.

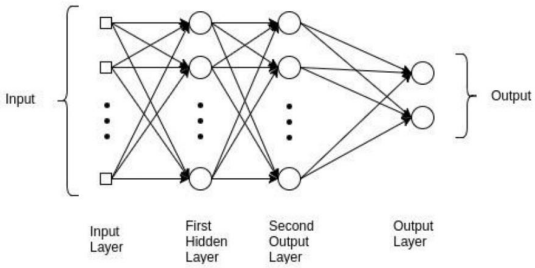
5.1.4 EVALUATION

Although it is a fairly good classifier, naive Bayes is known to be poor estimator. The naive assumption of independence is very unlikely to match real-world data. We tried both the count vectorizer and TF-IDF one on the tweets to obtain an accuracy of 59.124% and 60.0074% on test data.

5.2 MULTILAYER PERCEPTRON ALGORITHM

A multilayer perceptron (MLP) is a class of feedforward artificial neural network (ANN). The term **MLP** is used ambiguously, sometimes loosely to any feedforward **ANN**, sometimes strictly to refer to networks composed of multiple layers of perceptron (with threshold activation). Multilayer perceptron's are sometimes colloquially referred to as "vanilla" neural network especially if they have a single hidden layer.

An **MLP** consists of at least three layers of node: an input layer, a hidden layer and an output layer. Except for the input nodes, each node is a neuron that uses a nonlinear activation function. Each node in one layer connects with a certain weight $\mathbf{W_{ij}}$ to every node in the following layer. **MLP** utilizes a supervised learning technique called backpropagation for training. Its multiple layers and non-linear activation distinguish **MLP** from a linear perceptron. It can distinguish data that is not linearly separable.



5.2.1 MODEL ARCHITECTURE

Here in sentiment analysis also we have used MLP i.e. multilayer perceptron as one of the methods of classification.

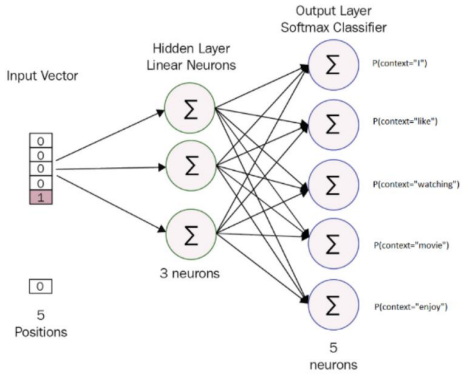
5.2.2 DATA VECTORISATION

In this method of classification, we have used Adam for weight optimization.

For the encoding in the **MLP** classifier, we have used the Word2vec encoding.

5.2.2.1 Word2Vec

Word2vec is an approach that helps us to achieve similar vectors for similar words.



Words that are related to each other are mapped to points that are closer to each other in a high dimensional space.

It is a three-layer neural network, in which first is the input layer and the lasts are the output layers. The middle layer builds a latent representation so the input words are transformed into the output vector representation.

5.2.3 TRAINING

We imported MLP classifier from sklearn. neural_network and trained our model on it. Class MLP Classifier implements a multi-layer perceptron (MLP) algorithm that trains using Backpropagation. We used hidden layer of size (100,100,100) on which we used Adam and stochastic gradient descent as optimizers. We also set some of the parameters such as maximum number of iterations to 500 and L2 penalty parameter(alpha) as 0.0001.

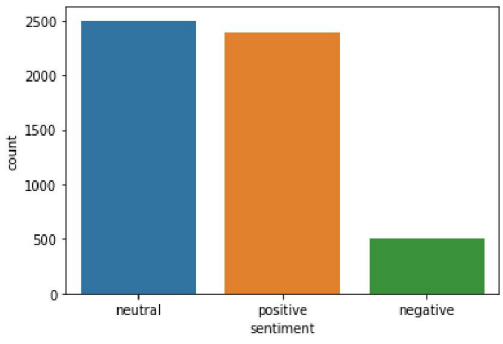
5.2.4 HYPERPARAMETER TUNING

The hyperparameters which were involved in training this model are hidden layer size, number of iterations, choice of optimizer, learning rate. We started with a single hidden layer with 100 units and gradually increased the layers and observed that its performance was getting good at around 4 hidden layers with 100 number of units in each layer. We gave it around 100 number of iterations but it was highly overfitting so we provided it with an early stopping with a tolerance limit of around 1e-9, and it took around 15 iterations to reach the tolerance limit. Again, we had the same choice of optimizer and Adam was performing just a little better than others. The activation used in each layer was ReLU.

5.2.5 MODEL EVALUATION

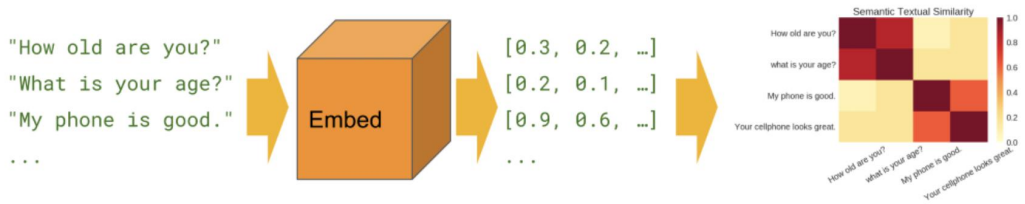
After applying and finding the accuracy on the separate test data the accuracy we obtained was 62.783%.

The graph obtained on the test data using **MLP** classifier on the test data is as follows.



5.3 SENTIMENT ANALYSIS USING TENSORFLOW

To use the text as input for a model, we first need to convert the tweet's text into vectors.



5.3.1 DATA VECTORIZATION

5.3.1.1 WHY DO WE CONVERT TEXTS INTO VECTORS?

A vector is an array of numbers of a particular dimension. A vector of size 5×1 contains 5 numbers and we can think of it as a point in 5D space. If there are two vectors each of dimension 5, they can be thought of two points in a 5D space. Thus, we can calculate how close or distant those two vectors are, depending on the distance measure between them.

Hence, lots of efforts in machine learning research are put to converting data into a vector as once data is converted into a vector, we can say two data points are similar or not by calculating their distance. But while embedding a sentence, along with words the context of the whole sentence needs to be captured in that vector. This is where the "Universal Sentence Encoder" comes into the picture.

5.3.1.2 UNIVERSAL SENTENCE ENCODER

The Universal Sentence Encoder is able to embed not only words but phrases and sentences. That is, it takes variable length English text as input and outputs a 512-dimensional vector. And since the model was trained at the word level, it will likely find typos and difficult words challenging to process.

5.3.2 MODEL ARCHITECTURE

We are using a universal sentence encoder model on tweets of given dataset to convert text data of tweets into vectors:

module URL = https://tfhub.dev/google/universal-sentence-encoder/4

Getting input shape of size (1, 512) of each tweet text in the dataset.

In a neural network, we are using 2 hidden layers with nodes of 256 and 128. The formula from one layer to the next is this short equation:

$$o_j = f\left(\sum_i w_{i,j} a_i + b_i\right)$$

The layer with nodes serves as input for the layer with nodes o. In order to calculate the values for each output node, we have to multiply each input node by a weight w and add a bias b.

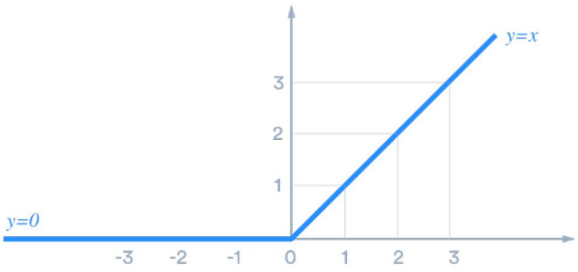
All of those have to be then summed and passed to a function f. This function is considered the activation function and there are various different functions that can be used depending on the layer or the problem.

We are using a rectified linear unit **(ReLU)** for hidden layers, a **softmax** function for the output layer of multi-class classification problems.

5.3.3 ACTIVATION FUNCTION

5.3.3.1 RECTIFIED LINEAR UNIT (ReLU)

ReLU stands for rectified linear unit, and is a type of activation function. Mathematically, it is defined as $y = max\ (0, x)$.



5.3.3.2 SOFTMAX FUNCTION

It reports back the "confidence score" for each class which deals with probabilities for multi-classification labels, the scores returned by the SoftMax function will add up to 1. The predicted class is, therefore, the item in the list where confidence score is the highest.

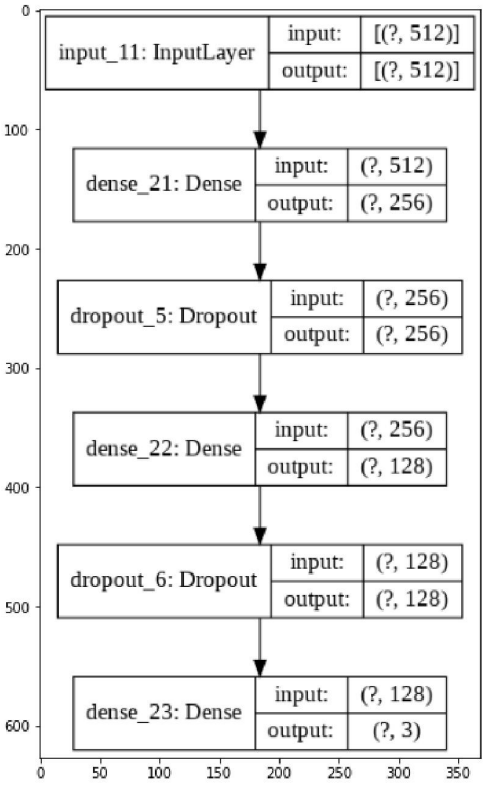
$$\frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

The error is determined by a ***cross entropy loss function*** whose loss we want to minimize with the optimizer.

$$-\sum c=1 Myo,c \log(po,c)$$

The most common optimizer we are using is called Adam which has a good performance in various problems.

5.3.4 FINAL ARCHITECTURE

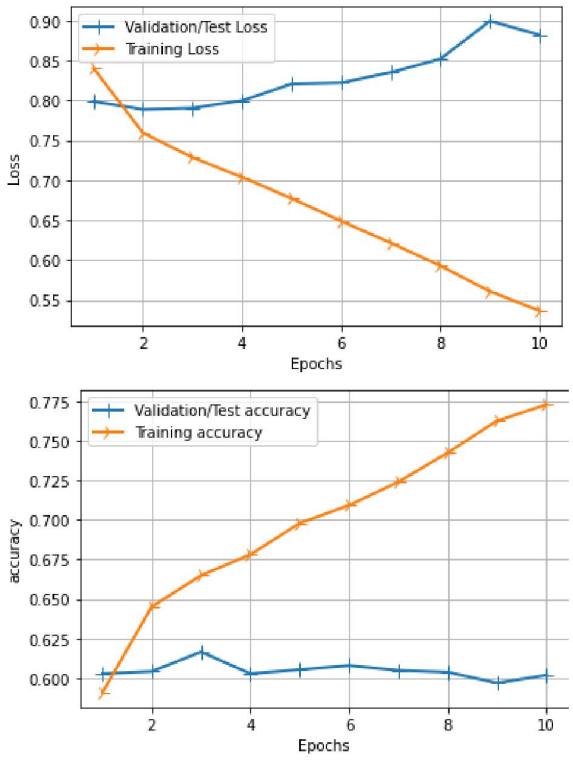


5.3.5 HYPER-PARAMETER TUNING

We have tried various Hyperparameters to tune this neural network model like **Dropouts**, **Kernel Initialization**, **L2 Regularization**, increasing and decreasing the number **of layers and nodes, batch-size** etc. At the end after training various models, to avoid overfitting we have found two hidden layers with nodes 256 and 128 with dropouts (0.5) and batch-size 32 have some good effect to overcome overfitting to some epochs.

5.3.6 MODEL EVALUATION

After applying and finding the accuracy on the separate test data the accuracy we obtained was 66.59%.



5.4 BERT CLASSIFIER

5.4.1 PyTorch

PyTorch is an optimized open source machine learning tensor library based on the Torch which is written in Python, C++, CUDA used for deep learning. It uses GPUs and CPUs. It is used for implementing network architectures like RNN, CNN, LSTM, etc. and other high-level algorithms used for applications such as computer vision and natural language processing like, primarily developed by Facebook's AI Research lab (FAIR). It is a free and open-source software released under the Modified BSD license. It is known for providing two of the most high-level features; namely, tensor computations with strong GPU acceleration support and building deep neural networks on a tape-based auto grad system.

5.4.2 TRANSFORMER

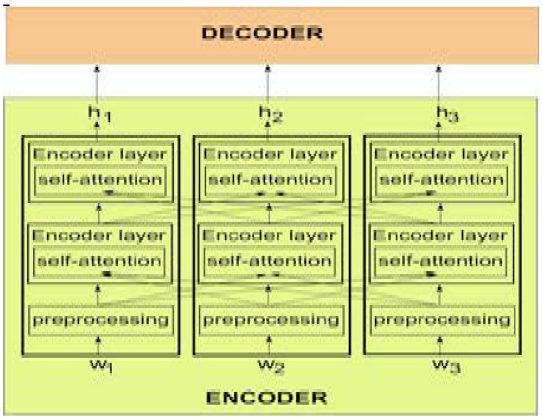
The **Transformer** architecture was introduced in the paper whose title is worthy of that of a self-help book: *Attention is All You Need*. Again, another self-descriptive heading: the authors literally takes the RNN Encoder-Decoder model with Attention, and throw away the RNN. Attention is all you need! Well, it ends up being quite a bit more complicated than that in practice, but that is the basic premise.

How does this work? To start with, each pre-processed (more on that later) element of the input sequence $\boldsymbol{w_i}$ gets fed as input to the Encoder network - this is done in parallel, unlike the RNNs. The Encoder has multiple layers (e.g. in the original Transformer paper their number is six). Let us use $\boldsymbol{h_i}$ to label the final hidden state of the last Encoder layer for each $\boldsymbol{w_i}$. The Decoder also contains multiple layers - typically, the number is equal to that of the Encoder. All of the hidden states h_i will now be fed as inputs to *each* of the six layers of the Decoder. If this looks familiar to you, it is for a good reason: this is the Transformer's **Encoder-Decoder Attention**, which is rather similar in spirit to the Attention mechanism that we discussed above. Before we move on to how the Transformer's Attention is implemented, let's discuss the *pre-processing* layers.

There are two parts to pre-processing: first, there is the familiar word embedding, a staple in most modern NLP models. These word embeddings could be learned during training, or one could use one of the existing pre-trained embeddings. There is, however, a second part that is specific to the Transformer architecture. So far, no where have we provided any information on the order of the elements inside the sequence. How can this be done in the absence of the sequential RNN architecture? Well, we have the positions, let's encode them inside vectors, just as we embedded the meaning of the word tokens with word embeddings. The resulting post-processed vectors,

carrying information about both the word's meaning and its position in the sentence, are passed on to the Encoder and Decoder layers.

5.4.3 BERT



The new language representation model called BERT (Bidirectional Encoder Representations from Transformers)

BERT is the Encoder of the Transformer, it is a technique for NLP (Natural Language Processing) pre-training developed by Google. BERT was created and published by Jacob Devlin and his colleagues from Google in 2018. Google uses BERT to improve the understanding of user searches. More precisely, it was pretrained with two objectives:

1. Masked language modelling (MLM)
2. Next sentence prediction (NSP)

The choice of using "cased" or "uncased" Bert model depends on whether we think letter casing will be helpful for the given task or not.

5.4.3.1 BERT BASE CASED

Pretrained model on English language using a masked language modelling (MLM) objective. This model is **case-sensitive**: it makes a difference between english and English.

BERT-Base, Cased: 12-layers, 768-hidden, 12-attention-heads, 110M parameters
BERT-Large, Cased: 24-layers, 1024-hidden, 16-attention-heads, 340M parameters

5.4.3.2 BERT BASE UNCASED

Pretrained model on English language using a masked language modelling (MLM) objective. This model is **uncased**: it does not make a difference between english and English.
BERT-Base, Uncased: 12-layers, 768-hidden, 12-attention-heads, 110M parameters
BERT-Large, Uncased: 24-layers, 1024-hidden, 16-attention-heads, 340M parameters

5.4.3.3 BERT TOKENIZER

The choice of "cased" vs "uncased" depends on whether we think letter casing will be helpful for the task at hand. This tokenization is beneficial when dealing with out of the vocabulary words, and it may help in better representation of complicated words. The sub-words which are constructed during the training time and that depend on the corpus of the model that was trained on. We get the best results if we tokenize with the same tokenizer the BERT model was trained on. The PyTorch-Pretrained-BERT library provides us with tokenizer for each of BERTS models.

5.4.3.4 CROSS ENTROPY LOSS

Syntax:

CLASS torch.nn.CrossEntropyLoss(*weight: Optional[torch.Tensor] = None, size_average=None, ignore_index: int = -100, reduce=None, reduction: str = 'mean'*)

This criterion combines **nn.LogSoftmax()** and **nn.NLLLoss()** in one single class.

The loss can be described as:

$$\text{loss}(x,class) = -\log\left(\frac{\exp(x[class])}{\sum_j \exp(x[j])}\right) = -x[class] + \log\left(\sum_j \exp(x[j])\right)$$

Parameters

- **weight** (*Tensor, optional*)
- **size_average** (*bool, optional*)
- **ignore_index** (*int, optional*)
- **reduce** (*bool, optional*)
- **reduction** (*string, optional*)

5.4.4 HYPERPARAMETER TUNING

The hyperparameters which were involved in training this model are Batch_size, Max_len, number of epochs, choice of optimizer, loss function, learning rate. Batch_size is used while training to decide how many numbers of tweets will get trained in a particular epoch at the same time. While training it was observed that our model was not performing that well if it was provided with a higher batch size i.e., (256, 128), while on lower batch size (32, 16) it performed comparatively well. The Max_len is used to pad every tweet to a particular single value i.e., if the length of tweet is smaller than Max_len it will get padded to Max_len, while on the other hand it will get shrinked. The mean value of the length of all the tweets in the dataset was around 117, so in order to not lose information, and at same time just not doing high padding for every sentence, it was observed that taking Max_len around 140-180 gave a good performance. The number of epochs decide how many times the model will be trained with the same dataset. We started with 10 epochs and observed that it was overfitting on the training set, and it's validation accuracy started to decrease, so, we reduced the number of epochs to 5 and observed that again in the last 2 epochs it was overfitting, thus we went with 3 epochs, and it gave good performance without overfitting. The choice of optimizer was between Adam, SGD (Stochastic Gradient Descent), RMSprop, and we decided this by applying all of these in our training and it was observed that there was not a very high difference between the accuracy or the loss but overall, Adam optimizer had the best performance. This is a multiclass classification problem, so, in the last layer of training, we gave the activation to be softmax and thus the loss function we gave is categorical cross entropy. The learning rate is an optimization algorithm that determines the step size at each iteration in order to reduce the value of loss function. We started with a learning rate of 1e-3, then we increased the l.r. To 1e-2, and the accuracy deteriorated, so we reduced our l.r. After few alterations we observed that it was performing fairly well in the order of 1e-5.

5.4.5 MODEL EVALUATION

We are getting accuracy on test data of around 74.86 %.

5.5 BERT CLASSIFIER USING KTRAIN

KTRAIN is a lightweight wrapper for the deep learning library TensorFlow Keras (and other libraries) to help build, train, and deploy neural networks and other machine learning models. Inspired by ML framework extensions like *fastai* and *ludwig*, it is designed to make deep learning and AI more accessible and easier Model Architechture.

Same bert model is used here and training is done on model using ktrain library.

5.5.1 HYPER PARAMETER TUNING

Model is trained on different learning rates which is decided based on graph loss vs learning rate. On learning rate, (5e-5) we are getting maximum accuracy. Also using earlystopping and reduce_on_plateau is used to optimize model

5.5.2 MODEL EVALUATION

We get a maximum testing accuracy of 75.89% by using this model.

6. RESULT ANALYSIS

Based on the analysis on the different methods used, here are the final results.

Models	Testing Accuracy (in %)
Naive Bayes (TD-IDF)	60.1
Naive Bayes	61.13
MLP	62.78
USE in TensorFlow	66.59
Bert in TensorFlow	74.85
Bert in PyTorch	75.43
Distilbert in ktrain	73.25
Bert in ktrain	75.9

We have achieved an accuracy of 75.891% with training on around 21,465 tweets and testing on 5,398 tweets. We have used the Bert model with ktrain incorporated to get a better accuracy. Our method achieves good accuracy with relatively small data size.

7. FUTURE CONSIDERATIONS

The future of sentiment analysis is going to continue to dig deeper, far past the surface of the number of likes, comments and shares, and aim to reach, and truly understand, the significance of social media interactions and what they tell us about the consumers behind the screens. Sentiment Analysis has been a safe-haven for analysts mainly because of its functions and nuances were too complicated for machines to challenge. The ability to understand sarcasm, hyperbole, positive or negative feelings has been difficult for machines that lack feelings. Although several literatures are available that study the challenges of Sentiment Analysis in the big data frameworks, such as through the volume, velocity and variety issue, the value, veracity and volatility have not been explored as much, though in fact taming the data is key for big data analytics.

8. REFERENCES

1. https://ieeexplore.ieee.org/document/7066632
2. https://www.ijrte.org/wp-content/uploads/papers/v7i4s/E1989017519.pdf
3. https://pytorch.org
4. https://huggingface.co/
5. https://web.stanford.edu/~jurafsky/slp3/4.pdf
6. https://www.analyticsvidhya.com/blog/2014/10/ann-work-simplified/
7. https://realpython.com/python-keras-text-classification/
8. https://www.tensorflow.org/tutorials/text/transformer
9. https://www.tensorflow.org/hub/tutorials/semantic_similarity_with_tf_hub_universal_encoder
10. https://tfhub.dev/google/universal-sentence-encoder/4