Report: Tweet Sentiment Analysis

Youssef Mostafa, Ahmed Tamer

May 31, 2023

1 Introduction

In this report we are going to discuss our NLP Project deliverable for milestone 1. Our project is a sentiment classification model that targets tweets, as twitter has attracted many users in recent years and has proved to be one of the most engaging platforms for discussions; it makes a good target for an NLP project and with all different kinds of people on the internet, a diverse set of data will be found. We used sentiment140 as our data set, which contains around 1.6M tweets.Then, we prepossessed it and analyzed it thoroughly and this will be discussed in the upcoming sections.

2 Data Analysis

Data analysis is crucial for making sense of the vast amounts of data generated on social media platforms like Twitter. With billions of tweets being posted every day, data analysis can help extract valuable insights from this massive volume of data. By analyzing a Twitter data set, it's possible to gain a deep understanding of the trends, patterns and sentiments of the users, which can be used for various purposes such as: understanding customer preferences, tracking brand reputation and monitoring public opinion on a particular topic. In the case of this project, we first analyzed the skewness of data, by checking the amount of tweets per sentiment and our data set was nearly symmetrical as can be seen in the following figure:

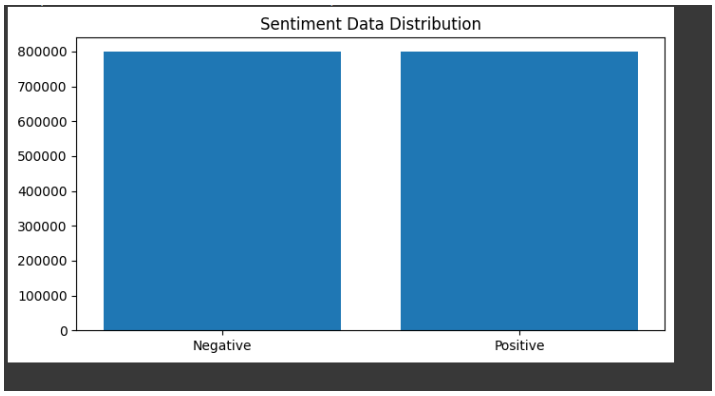

Figure 1: Per sentiment sentence frequency

In addition to this, words used within tweets on a per-sentiment basis were analysed and represented in the form of a word-cloud, shown in the figure below:

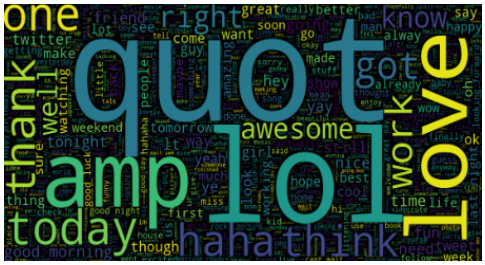

Figure 2: Word cloud for negative keywords

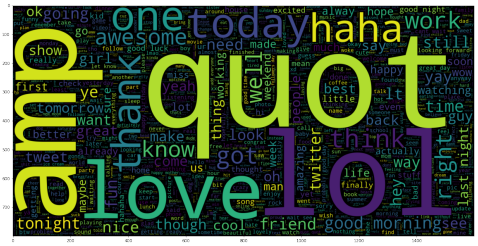

Figure 3: Word cloud for positive keywords

The wordcloud representation is helpful to identify how the different sentiments twitter users portray conform with different words and sentences, as the results from this analysis will be matched and checked with the training results of the model later on while testing in order to measure the accuracy and performance of the model.

For our second database we have a smaller Dataset that has around 55K samples with the sentiment distribution being as follows:

Figure 4: Per sentiment sentence frequency

and the most common words are represented in the following Word Clouds:

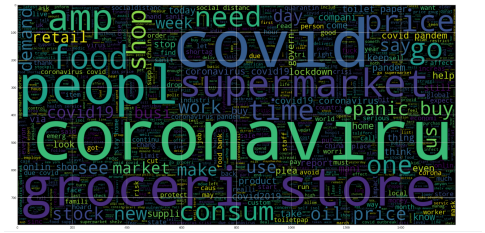

Figure 5: Word cloud for negative keywords

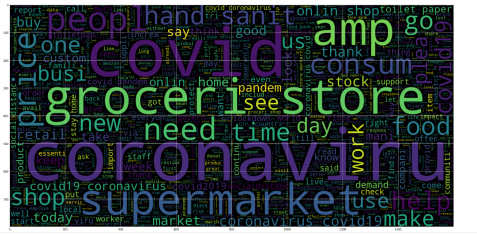

Figure 6: Word cloud for positive keywords

for this specific data set we did oversampling to remove the skewness of the data and have classes with equal size.

Moreover, for this data set extra information were presented and were analyzed these include the Date of sending the tweet and the country of the tweet's originator, where we plotted the date/location vs the the average sentiment and the count of tweets sent:

Following that, the most common words in each sentiment were analyzed to give a better idea of what words carry the most weight in each, while also checking for any common occuring that are present outside of their expected sentiment text.

In addition to that, as Twitter is filled with gibberish and often lots of misspelled words, so we had to use Pyspellchecker to get the amount of misspelled words and their frequencies in order to statistically correct them with the library itself.

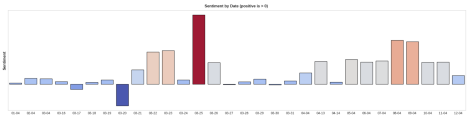

Figure 7: average sentiment of tweets per date

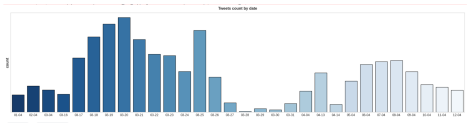

Figure 8: amount of tweets per date

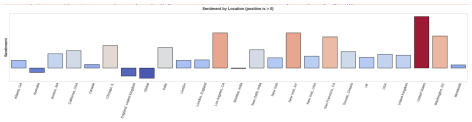

Figure 9: average sentiment of tweets per location for locations with 100+ samples

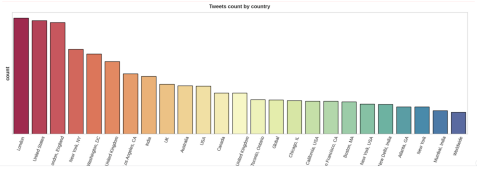

Figure 10: amount of tweets per location for locations with 100+ samples

Misspelled Data can create a large amount of noise which is terrible when training a machine learning model, it will also falsely inflate the size of the model's vocabulary creating multiple different representations for each word.

If not handled well, the points mentioned above can drastically reduce the ability of the model to understand the training data.

3 Pre-Processing Phase

Various techniques were utilized in the pre-processing phase, which will be discussed below. The pre-processing phase is very important in any NLP model, as it allows for smooth operation on the given input by the NLP algorithm, due to the model's ability to more easily understand the input when the unnecessary filler details within the text are filtered out. Techniques used consist of:

- Stemming: the action of reducing words to their base or root form, which is called the stem. The goal of stemming is to group together words that have the same root, so that they can be treated as the same word when performing text analysis.
- Lemmatization: the goal of lemmatization is to group together words that have the same root, so that they can be treated as the same word when performing text analysis. Unlike stemming, which simply removes word endings to create a stem, lemmatization considers the context and part of speech of a word to determine its lemma.
- Spelling Correction: the action of correcting commonly found spelling mistakes within sentences and restoring them to their original form, so that the model can handle them with accuracy.
- Train-Test Split: a common technique used in machine learning to evaluate the performance of a model. It involves splitting a dataset into two subsets: a training set and a testing set. The training set is used to train the machine learning model, while the testing set is used to evaluate the performance of the model on unseen data. By evaluating the model on a separate testing set, we can get a better estimate of how well the model will perform on new data that it has not seen before. The train-test split is typically done randomly, with a certain percentage of the data assigned to each subset. For example, an 80-20 train-test split means that 80% of the data is used for training the model, while 20% is used for testing. The exact split ratio depends on the size of the dataset, the complexity of the model, and the specific task at hand.
- Tokenization: tokenization is the process of segmenting text into meaningful units that can be analyzed by a machine learning algorithm. The tokens created through tokenization are usually the basic units of text that a machine learning algorithm processes, such as words, punctuation marks, numbers, and even emojis. Tokenization is typically the first step in many NLP tasks, such as sentiment analysis, text classification, and machine translation.

- Extras: extra techniques were used in addition to the ones listed, including: slang replacement, converting emoji's to short words representing sentiment as well as hyperlink removal.

when using our BERT fine tuned model we included an extra pre processing step that included removing the samples resulting in less that 4 tokens, the amount of samples with less than 10 tokens can be seen in the following charts:

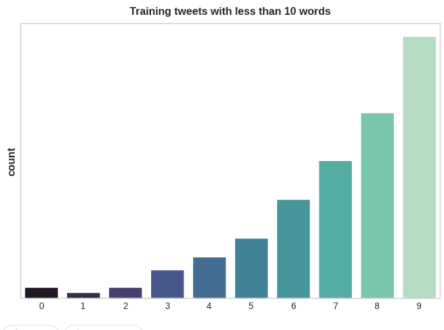

Figure 11: corona 2019 twitter Data set

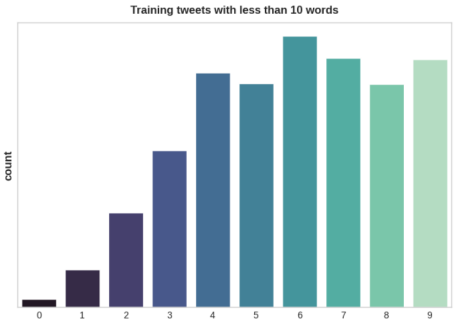

Figure 12: 1.6M sentiment Data set

4 System overview

The current system is made of stacked bidirectional LSTMs following a 1D Convolution layer for features gathering, this allows our model to capture the features from the text and encode them in an interpret-able form to be able to learn from it correctly, the bidirectional LSTMs allow the model to take a look at the input from different directions and better analyze higher order features that aren't easily interpreted. In addition to that, the text input can't be fed directly into the model without being represented as feature vectors and therefore we resorted to word embedding and use both Glove and word2vec in different iterations of our model

the middle layers in the model did include :

- Dense Layers
- Concatenation layers
- Flatten layer.
- Dropout layer

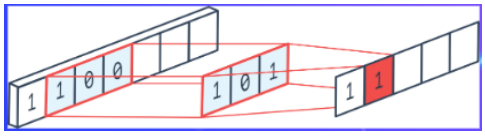

Figure 13: simple convolution

```
Train on 1280000 samples, validate on 320000 samples
Epoch 1/10
1280000/1280000 [==============================] - 106s 83us/sample - loss: 0.5195 - accuracy: 0.
7394 - val_loss: 0.4845 - val_accuracy: 0.7635
Epoch 2/10
1280000/1280000 [==============================] - 101s 79us/sample - loss: 0.4873 - accuracy: 0.
7625 - val_loss: 0.4711 - val_accuracy: 0.7738
Epoch 3/10
1280000/1280000 [==============================] - 101s 79us/sample - loss: 0.4769 - accuracy: 0.
7694 - val_loss: 0.4655 - val_accuracy: 0.7764
Epoch 4/10
1280000/1280000 [==============================] - 101s 79us/sample - loss: 0.4705 - accuracy: 0.
7739 - val_loss: 0.4644 - val_accuracy: 0.7778
Epoch 5/10
1280000/1280000 [==============================] - 101s 79us/sample - loss: 0.4660 - accuracy: 0.
7763 - val_loss: 0.4620 - val_accuracy: 0.7782
Epoch 6/10
1280000/1280000 [==============================] - 101s 79us/sample - loss: 0.4627 - accuracy: 0.
7780 - val_loss: 0.4608 - val_accuracy: 0.7796
Epoch 7/10
1280000/1280000 [==============================] - 100s 78us/sample - loss: 0.4600 - accuracy: 0.
7797 - val_loss: 0.4618 - val_accuracy: 0.7788
Epoch 8/10
1280000/1280000 [==============================] - 100s 78us/sample - loss: 0.4579 - accuracy: 0.
7813 - val_loss: 0.4599 - val_accuracy: 0.7799
Epoch 9/10
1280000/1280000 [==============================] - 100s 78us/sample - loss: 0.4555 - accuracy: 0.
7824 - val_loss: 0.4587 - val_accuracy: 0.7815
Epoch 10/10
1280000/1280000 [==============================] - 100s 78us/sample - loss: 0.4541 - accuracy: 0.
7830 - val_loss: 0.4584 - val_accuracy: 0.7807
```

Figure 14: this is an overview of an LSTM model

these layers serve as connections for the data flow in the model. The dense layer adds trainable parameters to our model as well as adjust the size of outputs to match the inputs for the other layers, the Concatenation layer concatenates outputs of different sizes from different layers and outputs them as one single tensor, similarly is the Flatten Layer as it does reduce the dimensions of our tensors to first rank, finally the dropout layer helps to reduce the over fitting when training.

The final layer in our model is a sigmoid layer to output a probability which is divided into intervals to deduce which class does this input belong to.

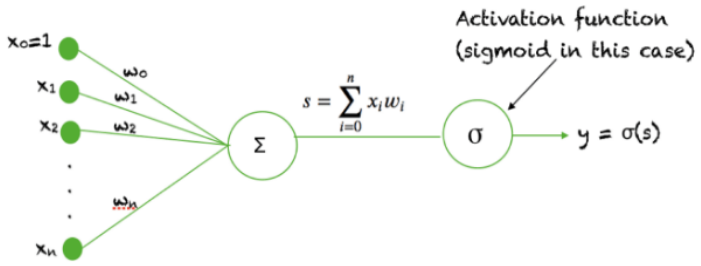

Figure 15: sigmoid diagram

Finally, different iterations of the models are going to be ensembled together to improve the accuracy of the system as a whole through probability average ensemble.

5 Attention & Transformers

Attention is a mechanism used in natural language processing (NLP) that enables models to highlight specific parts of the input sequence when making predictions or generating output. It helps the model to selectively focus on relevant information and assign varying degrees of importance to different parts of the input. The attention mechanism typically involves three components: a query (q), a set of key-value pairs (k,v), and a scoring function. The query is used to compute the compatibility or similarity between the query and each key. The resulting scores are then used to compute attention weights or attention distribution using a softmax function, which ensures that the weights sum up to 1. Finally, the attention weights are applied to the corresponding values to obtain a weighted sum, which is used by the model to make predictions or generate output. NLP models usually rely on an intermediate state of fixed size to process and remember previous data, which can cause potential performance bottlenecks. In cases with long input, the typical usage of intermediate states without an attention mechanism has proved to not be very effective, therefore, attention is utilized in these cases to remove bottlenecks and improve performance. In 2017, the concept of transformers was introduced in the widely renowned paper "Attention is all you need". Transformers rely on the self-attention mechanism to model the relationships between words or tokens in an input sequence, disregarding the use of traditional RNNs (Recurrent Neural Networks) and CNNs (Convolutional Neural Networks). The following figure visually describes the attention mechanism. And, beyond that, a brief explanation regarding the inner workings of a transformer.

5.1 Encoder-Decoder Stack

Each encoder has two sub-layers.

- A multi-head self attention mechanism on the input vectors.
- A simple, position-wise fully connected feed-forward network.

Each decoder has three sub-layers.

- A masked multi-head self attention mechanism on the output vectors of the previous iteration.
- A multi-head attention mechanism on the output from encoder and masked multi-headed attention in decoder.
- A simple, position-wise fully connected feed-forward network.

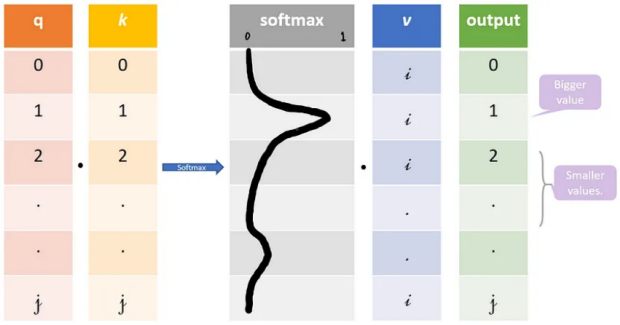

Figure 16: The Attention Mechanism

5.2 Input-Output Preprocessing

The input words are represented using some form of embedding. This is done for both encoder and decoder. Word embedding on its own lacks positional information, unlike RNNs which do achieve due to their positional nature. While in self-attention, this information is lost due to softmax. To preserve the positional information, the transformer injects a vector to individual input embeddings (could be using word embeddings for corresponding to the input words). These vectors follow a specific periodic function (Example: combination of various sines/cosines having different frequency, in short not in sync with each other) that the model learns and is able to determine the position of individual words with respect to each other based on the values. This injected vector is called "positional encoding" and are added to the input embeddings at the bottoms of both encoder and decoder stacks.

5.3 Decoder Stack: Additional Notes

The output of the decoder stack at each step is fed back to the decoder in the next time step — pretty similar to how outputs from previous steps in RNNs were used as next hidden states. And just as we did with the encoder inputs, we embed and add positional encoding to those decoder inputs to preserve the position of each word. This positional encoding + word embedding combo is then fed into a masked multi-headed self attention. This self-attention sub-layer in the decoder stack is modified to prevent positions from attending to subsequent positions — you can't look at future words. This masking ensures that the predictions for position i can depend only on the known outputs at positions less than i. The outputs from the encoder stack are then used as multiple sets of key vectors k and value vectors v, for the "encoder decoder attention" — shown in green in the diagram — layer. It helps the decoder focus on the contextually relevant parts in the input sequence for that step. (The part similar to global attention vectors.) The q vector comes from the "output self attention" layer. Once we get the output from the decoder, we do a softmax again to select the final probabilities of words.

6 BERT

BERT (Bidirectional Encoder Representations from Transformers) is discussed in a recent paper published by researchers at Google AI Language. It has caused commotion in the Machine Learning community by presenting state-of-the-art results in a wide variety of NLP tasks, including Question Answering (SQuAD v1.1), Natural Language Inference (MNLI), and others. BERT's key technical innovation is applying the bidirectional training of Transformer, a popular attention model, to language modelling. This is in contrast to previous efforts which looked at a text sequence either from left to right or combined left-to-right and right-to-left training. The paper's results show that a language model which is bidirectionally trained can have a deeper sense of language context and flow than single-direction language models. In the paper, the researchers detail a novel technique named Masked LM (MLM) which allows bidirectional training in models in which it was previously impossible. BERT makes use of transformers aforementioned. But, before BERT is given any data, 5% of the words in

each sequence are replaced with a [MASK] token. The model then attempts to predict the original value of the masked words, based on the context provided by the other, non-masked, words in the sequence. In technical terms, the prediction of the output words requires:

- Adding a classification layer on top of the encoder output.
- Multiplying the output vectors by the embedding matrix, transforming them into the vocabulary dimension.
- Calculating the probability of each word in the vocabulary with softmax.

The following diagram details the process in a clearer light for the masked LM:

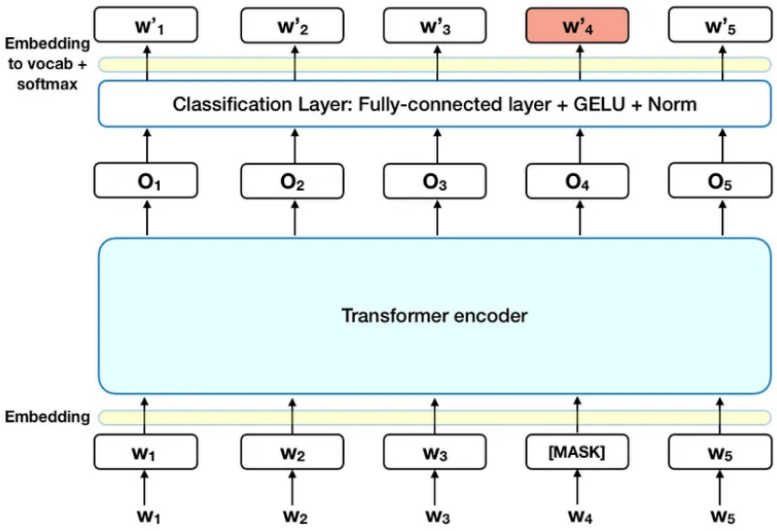

Figure 17: Masked LM

During the BERT training process, the model is trained using pairs of sentences as input or given a Label as in our case. It learns the relation between words and another from the input using Self Attention further Aiding and enhancing it's prediction capabilities. In training, masks are generated to mask the input based on calculations from the inputs own text, while the other part consists of the unmasked tokens left from the previous masking.

To facilitate the model in Predicting an output during training, the input undergoes a specific processing procedure before being fed into the model:

- Sentences are prepended with a special [CLS] token, and each sentence is appended with a [SEP] token to mark their boundaries.
- Each token is assigned a sentence embedding indicating whether it belongs to Sentence A or Sentence B. Similar to token embeddings, sentence embeddings have a vocabulary of only two.
- Additionally, a positional embedding is assigned to each token to indicate its position in the sequence. The concept and implementation of positional embeddings are described in the Transformer paper.

In order to determine the connectivity between the Tokens and the Sentiment, the following procedures are executed:

- The entire input sequence is processed through the Transformer model.
- The output generated by the [CLS] token is transformed into a 2×1 shaped vector using a straightforward classification layer, which consists of learned weight and bias matrices.

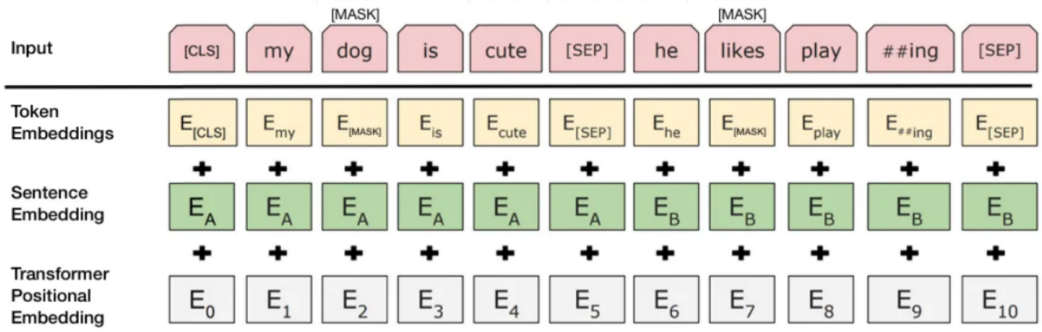

Figure 18: Input Preprocessing

- The softmax function is applied to calculate the probability of the second sentence being the subsequent sequence.
- In other similar NLP tasks, During the training of the BERT model, both Masked LM (Language Modeling) and Next Sentence Prediction strategies are trained simultaneously. The objective is to minimize the combined loss function, which takes into account both strategies, aiming for optimal performance in predicting masked words and determining sentence connectivity.

7 Results

After using the previously mentioned techniques on 2 different Datasets, these are the results we came up with:

7.1 LSTM

for the LSTM the performance on the corona 2019 Dataset was horrible because of the low number of samples, it witnessed a significant increase to it's performance when using the bigger 1.6 Dataset, the results were as follow:

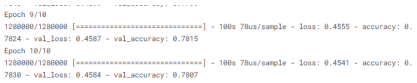

Figure 19: metrics for the LSTM model training on the 1.6M sentiment Data set

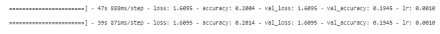

Figure 20: metrics for the LSTM model training on the Corona 2019 Data set

it can be clearly seen that the LSTM under fitted on the smaller Data set however when provided with a bigger sample it gave better results.

7.2 BERT

BERT proves its efficacy when testing on both datasets, achieving an accuracy higher than 90% and a validation accuracy exceeding 80% on both models. More specifically:

- Sentiment 140: accuracy = 93.56%, validation accuracy = 83.21%
- Corona 2019: accuracy = 92.06%, validation accuracy = 87.96%

This shows that transformer based approaches are superior to typical RNN solutions

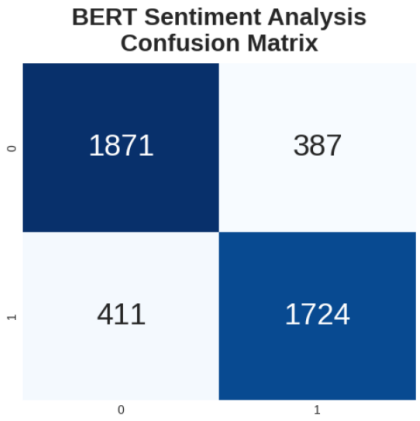

Figure 21: confusion matrix for the BERT model using a sample of the 1.6m Data set

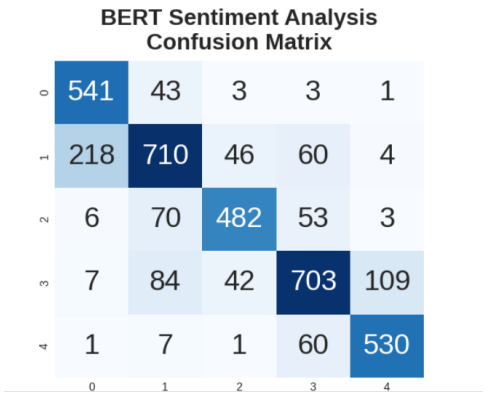

Figure 22: confusion matrix for the BERT model using the corona 2019 tweet dataset

8 Conclusion & Future Work

In conclusion, this report has served as a means of discussion about the importance of sentiment analysis and what it can be used for, as well as portraying how sentiment analysis can be broken done into a series of steps. More specifically: data analysis, pre-processing and detailed operational functionality. Each of those sub-categories were explained thoroughly with listed examples underneath each of them. In the near future, the models that were created were able to classify the sentiment into 5 different classes with high accuracy using the latest state of the art tech that was demonstrated in the BERT model fine tuning.

References

[1] BERT Explained: State of the art language model for NLP, Rani Horev

[2] Transformers, Beyond Data Science, Ria Kulshrestha

[3] Understanding Attention In Deep Learning, Ria Kulshrestha

[4] Attention Is All You Need, Ashish et al.

[5] BERT :Pre-training of Deep Bidirectional Transformers for Language Understanding, Jacob et al.