Natural Language Processing – Twitter Sentiment Analysis

Student Name: Priya Prashar
Student ID: u5590815

I. Introduction

Information from a textual source can be categorized through the process of sentiment analysis to analyse a user's response and feeling towards a certain subject. The main objective of this report is to classify tweets as neutral, negative, or positive, by creating a set of 3 classifiers that sort tweets obtained through various datasets into one of the three sentiments provided.

Twitter sentiment analysis

A. *What is Twitter sentiment analysis*

Twitter sentiment analysis is the process of classifying textual data according to their polarity such as positive, negative, and neutral [1]. Using machine learning techniques, we can understand more regarding whether an individual's post is talking positively or negatively. When we consider the following example:

'The band enjoyed a day of sightseeing in Berlin today. We hope to see you at ROCK IM PARK tomorrow!'

From a human perspective we can identify immediately that the user who has posted the following tweet, 'enjoyed' themselves in Berlin, the word enjoyed highlights their positive outlook of the day followed by a punctuation and a directive that they will be seeing them tomorrow. Sentiment analysis aims to replicate this process to categorize data into user sentiments.

B. *Why is twitter sentiment important*

With an understanding of how users perceive a certain topic, businesses can understand the trends, topics, and negatives regarding their products and services. When considering the importance of this area in business advertising, sentiment analysis can help to [2]:

- Provide objective insights.
- Build better products and services.
- Analyze at scale.
- Real-time results

This report will focus on the sentiment gained from the Twitter dataset, which could potentially be developed further for understanding how users express ideas on social media posts.

II. Developing a model

A. ***Understanding the data:***

For our analysis, we will be using 5 main datasets for development (twitter-dev-data.txt), training, and testing, the amount of tweets/data in these files is described as below:

Text file	Number of tweets
twitter-training-data.txt	45101
twitter-dev-data.txt	2000
twitter-test1.txt	3531
twitter-test2.txt	1853
twitter-test3.txt	2379

The training dataset is used for training and test1,test2, and test3 are used for the testing of the classification models.

Data in the file is stored in the following format:

'110956421929008820 positive Cinema tonight to watch @BeastsMovieUK 😄 😄 in the meantime reading a book about free software licensing… https://t.co/Tm9DSdWWgh'

Each line in the text file contains a sequence of digits representing the tweet id. Followed by a sentiment that could be:

- Positive
- Negative
- Neutral

These are the labels used as part of the classification of the tweets, then a string representing the tweets. The tweets contain various values of data from links, emojis, symbols, and alphanumerical values.

B. ***Preprocessing***

As the data we are working with is stored as a list with each new line representing a separate tweet in the file, we will need to split the data into the tweets, tweet_id, and the sentiment for predictions. With the occurrence of tags, stopwords, emojis, and special characters which affect the overall processing accuracy we need to ensure that our data is correctly formatted.

Separation of data

The text files provided currently contain all information for the sentiments, tweets, and tweet IDs in each line of the file to allow easier processing we begin by separating the information into, tweets, sentiments, and tweet IDs using regex:

```
^[^\d]*(\d+)
```

To obtain the tweet id we use the regex above to find the first set of multiple digits grouped.

```
(neutral)|(positive)|(negative)
```

The following regex then finds the first occurrence of neutral, positive, or negative.

```
(?:[neutral|positive|negative])\b\s*(.+)
```

From here we select all the text after the sentiment to represent our tweet.

Lowercasing all text

To ensure simplicity in our analysis we begin by ensuring all tweets are in lowercase, this reduces the need to differentiate between lower and capitals.

Removing tags and usernames

```
Tag Removal Regex: #[\w]+
Username Removal Regex: @[\w]+
```

Hashtags can often provide us with contextual information regarding a tweet, but most do not indicate the sentiment, for this reason, we remove the hashtags to reduce errors. In the long run, a user associated with a username can be identified as having a certain sentiment, about certain topics however for this analysis we will be focusing on classifying the tweets using data from individual tweets. If we are to consider the association we still would need to know who the user posting is and if the @ is targeting another individual making the overall sentiment from an associated user difficult to obtain.

Removing URLs.

```
URL Removal Regex: https?:[^\s]+
```

URLs do not give any indication of the overall sentiment of the tweet hence we remove URL values.

Remove emojis.

```
Emoji Removal Regex: [\U00010000-
\U0010ffff]
```

Note: This regex matches all Unicode values that fall under an emoji.

Emojis can often provide us with a lot of information regarding the sentiment of the tweets. However more than often they are contextual based and can be used for multiple meanings. While smiley faces can be an indicator of a positive response, we are also dealing with other emojis that aren't as clear. A user's interpretation of an emoji may be different. For the simplicity of this analysis, we are going to remove these values, by using a regex that removes all Unicode values in the range.

Removing special characters.

Punctuation, symbols, and special characters do not provide information regarding the sentiment, and exclamation marks can be an indicator of surprise, excitement, or anger, therefore we can classify this as noise and remove the values.

Removing non-alphanumeric and only numeric values:

```
Removing Non-alphanumeric:  [^a-zA-Z0-9
-]
Removing only Numeric: \b\d+\b
```

Sometimes the tweets contain values that are not alphanumeric, while we have covered most of these cases in our previous preprocessing steps, to ensure our data is in the correct format we remove these values. Moreover, strings of numbers only such as 5000 which do not precede or succeed any other numbers of words are also removed.

Remove words with only 1 character.

Words with 1 character often do not give any indication of the overall sentiment of the tweet due to this we remove all of the words.

Removing stop words.

Stop words are a set of words, or phrases that are commonly used to form sentences in the English language. Examples of this include the, me, you, they, an, or, it, as…

Stop words do not provide any insight into sentiment therefore we remove these words to ensure that the data contains little to no noise. The NLTK library provides us with a set of stop words, we can remove all words that appear in this corpus to ensure that we cover all cases..

Using the tweet from section B as an example after preprocessing our tweets will be in a format similar to below:

'Cinema tonight watch mean time reading book about free software licensing'

These tweets are then tokenised to extract meaningful features from our input data.

C. Features

When extracting features, and tokens from individual tweets we have chosen to look at two main approaches:

Bag of Words (BOW)

As part of feature extraction, we are going to use the bag of words method, this method allows us to gain a representation of the word occurrences in each input. By finding word occurrence we can utilize and extract features from our tweets that are of the most importance, we are also able to visualize how this looks in our data by creating a vocab dictionary that maps each unique word to a unique identifier this is shown below:

```
Training Dev:

{'very': 204, 'late': 586, 'firstenergy': 4099, 'night': 29, 'apologises': 2918, 'teen': 531, 'aid': 2835, 'happen': 573, 'mix': 5096, 'this': 20, 'baggy': 3032, 'example': 3987, 'taking': 529, 'havent': 455, 'disneylands': 3782…}
```

From the above example, we can see that in the training data the most common word occurrences are very, late, and firstenergy. Using a bag of words on the training data we can infer and create a numerical representation of our data. These are used as features in our model training.

Word Embedding - BERT

Another approach we have chosen to use is BERT. Bert unlike bag of words allows us to get a more contextual word embedding. Using a bidirectional approach this method allows us to know the context of a tweet due to being pre-trained on a prior model allowing us to capture better and more useful word embeddings for each tweet.

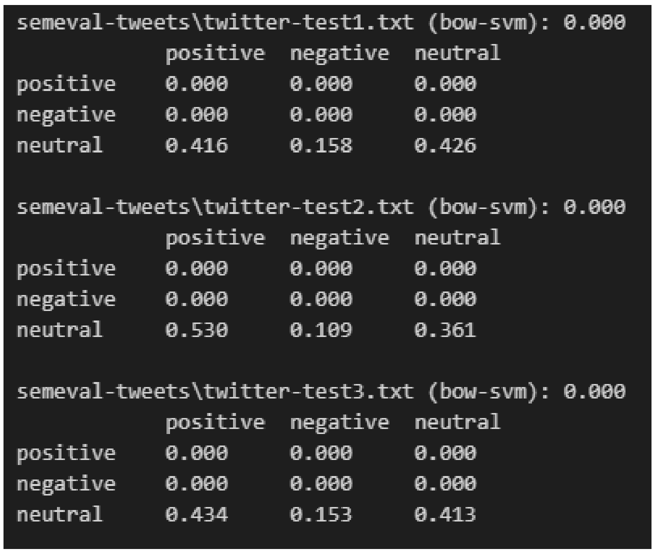
D. Results from Classifiers

After we have obtained our features, we can use them to develop and train a classifier. The architecture of all our classifiers aims to make predictions from the input tweet on whether a tweet is positive, negative, or neutral based on the features we have extracted. For this section, we will be focusing on 3 main classifiers SVM, Max Entropy, and LSTM RNN.

Support Vector Machines (SVM)

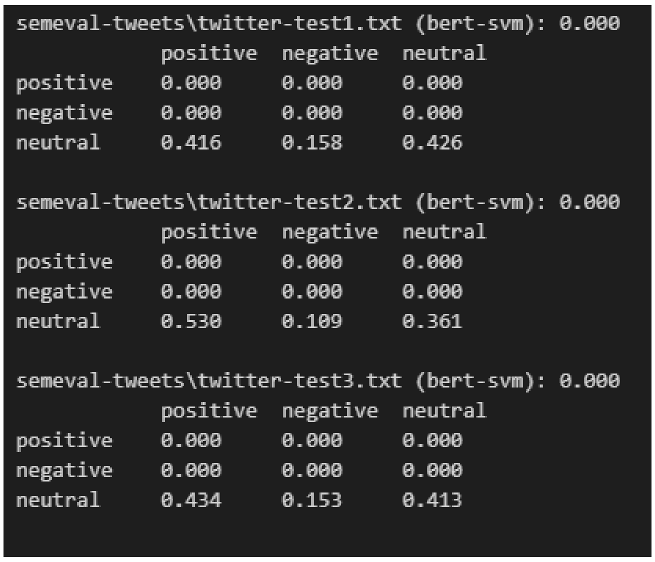
SVM is a supervised machine learning algorithm, for this model, we have chosen to use a linear kernel base for SVM as a parameter. Using a linear kernel reduces the overall computation compared to RBF, and polynomial while providing a more stable result. The results of this classifier are shown below:

Figure 1: Confusion matrices for SVM using BOW:



```
semeval-tweets\twitter-test1.txt (bow-svm): 0.000
            positive  negative  neutral
positive    0.000     0.000     0.000
negative    0.000     0.000     0.000
neutral     0.416     0.158     0.426

semeval-tweets\twitter-test2.txt (bow-svm): 0.000
            positive  negative  neutral
positive    0.000     0.000     0.000
negative    0.000     0.000     0.000
neutral     0.530     0.109     0.361

semeval-tweets\twitter-test3.txt (bow-svm): 0.000
            positive  negative  neutral
positive    0.000     0.000     0.000
negative    0.000     0.000     0.000
neutral     0.434     0.153     0.413
```

Figure 2: Confusion matrices for SVM using BERT:



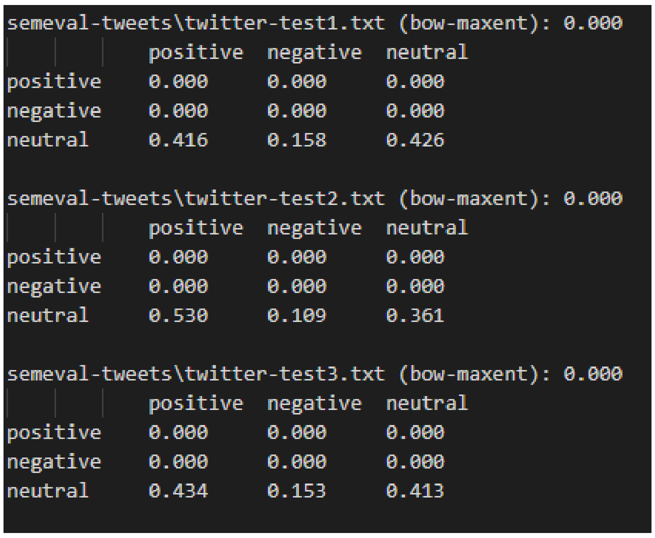
```
semeval-tweets\twitter-test1.txt (bert-svm): 0.000
            positive  negative  neutral
positive    0.000     0.000     0.000
negative    0.000     0.000     0.000
neutral     0.416     0.158     0.426

semeval-tweets\twitter-test2.txt (bert-svm): 0.000
            positive  negative  neutral
positive    0.000     0.000     0.000
negative    0.000     0.000     0.000
neutral     0.530     0.109     0.361

semeval-tweets\twitter-test3.txt (bert-svm): 0.000
            positive  negative  neutral
positive    0.000     0.000     0.000
negative    0.000     0.000     0.000
neutral     0.434     0.153     0.413
```

From our confusion matrices, for both Bert and BOW we can see that our model is heavily biased towards the neutral label. Predictions are mostly neutral. While this could suggest that our model is simply underfitting this is also seen in the later examples. This could suggest that there is some fault within the preprocessing stage. Due to time and schedule difficulties, I have been unable to figure out where this problem occurs. As we get an extremely low f-score this model is not performing well and will most likely need further development to find a solution for.

From our confusion matrix, we can also see that the number of actual neutral values compared to the values that were predicted is low therefore for both models this classifier does not perform well.

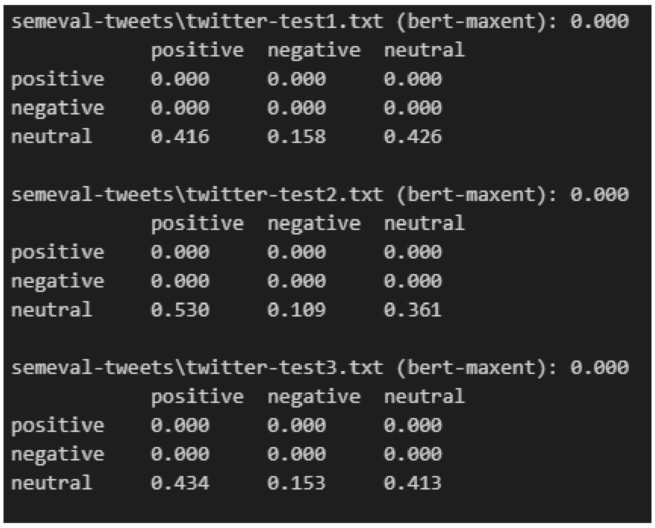
Max Entropy:

Figure 3: Confusion matrices for Max Entropy using BOW:



```
semeval-tweets\twitter-test1.txt (bow-maxent): 0.000
                positive  negative  neutral
positive        0.000     0.000     0.000
negative        0.000     0.000     0.000
neutral         0.416     0.158     0.426

semeval-tweets\twitter-test2.txt (bow-maxent): 0.000
                positive  negative  neutral
positive        0.000     0.000     0.000
negative        0.000     0.000     0.000
neutral         0.530     0.109     0.361

semeval-tweets\twitter-test3.txt (bow-maxent): 0.000
                positive  negative  neutral
positive        0.000     0.000     0.000
negative        0.000     0.000     0.000
neutral         0.434     0.153     0.413
```

Figure 4: Confusion matrices for Max Entropy using BERT:



```
semeval-tweets\twitter-test1.txt (bert-maxent): 0.000
                positive  negative  neutral
positive        0.000     0.000     0.000
negative        0.000     0.000     0.000
neutral         0.416     0.158     0.426

semeval-tweets\twitter-test2.txt (bert-maxent): 0.000
                positive  negative  neutral
positive        0.000     0.000     0.000
negative        0.000     0.000     0.000
neutral         0.530     0.109     0.361

semeval-tweets\twitter-test3.txt (bert-maxent): 0.000
                positive  negative  neutral
positive        0.000     0.000     0.000
negative        0.000     0.000     0.000
neutral         0.434     0.153     0.413
```

Similar to our previous model these results show a similar trend and will need further investigation to improve model performance.

Neural Network Architecture:

The neural network architecture we have used is shown below:

```
self.embedding = nn.Embedding(max_words,
embedding_dim)
self.embedding.weight.requires_grad = False
        self.lstm = nn.LSTM(embedding_dim,
        hidden_dim,
         num_layers = n_layers,
         bidirectional = bidirectional,
        dropout = dropout,
        batch_first = True
)
self.fc = nn.Linear(hidden_dim *
2,output_dim)
self.dropout = nn.Dropout(0.2)
```

The model has 4 main layers and 2 dropout layers. The first layer's input size of our data is determined by the GloVe embedding matrix size where the matrix size is based on the max words and the embedding dimensions. Embedding layers allow for data to be represented as continuous vector mapping using the pre-trained word embeddings from GloVe. The LSTM (Long short-term memory) captures dependencies found in sequential data. The final fully connected layer provides us with an output for 3 of the three labels, this is defined as positive, negative, or neutral. This network aims to classify the tweets given in our training and test data. The function creates_LSTM_model acts training this model on our X_train making predictions on all test sets stored in X_tests. The return value of this model is the associated tweet_id and predicted labels for the tweet.

<u>When evaluating the model, we have chosen to use the following loss function and optimizer:</u>

Adam: We have chosen Adam as an optimizer due to its ability to adapt and reduce overall bias. Adam further allows us to use memory efficiently.

Cross entropy: Cross-entropy is a common loss function for classification, we can evaluate how well the model performs against the true values.

Test Set 1

Epoch	Train_loss	Val_loss	Val_Acc
1	1.050	1.028	0.426
2	1.041	1.026	0.361

Test Set 2

Epoch	Train_loss	Val_loss	Val_Acc
1	1.046	1.012	0.362
2	1.042	1.011	0.360

Test Set 3

Epoch	Train_loss	Val_loss	Val_Acc
1	1.051	1.031	0.422
2	1.042	1.30	0.361

From evaluating our model's accuracy for each epoch, we can see consistent results on validation and accuracy. The results show that some of the values do match their actual values, where the test set 1 provides us with the highest accuracy. However, from our previous results, we need to consider that this also may be due to randomness in our results leading to this match. To improve this model, we can increase the number of epochs and adjust the learning rate.

F-score across all Test1, Test2, and Test3: Due to recurrent Kernel failures and memory overload, I was unable to get a conclusive score for the following classifier.

Overall, the CNN model we have created produces a low accuracy on the test data the next section covers some of the reasons why and how we can potentially fix this.

E. Errors, Optimisation, and Model Improvements

When working with this model one of the most important performance factors has been the use of hardware and the hardware capabilities. As the model we are working with is complex, when developing the model, there are certain changes, I have made due to the complexity.

For example, using a smaller file twitter-dev-data.txt for training and reducing the number of epochs when training. The model has been tested on a desktop PC with the following specifications with the majority of the model utilizing the CPU:

Operating System	Windows 64-bit operating system
Processor/CPU	12th Gen Intel(R) Core™ i7-12700K 3.60 GHz
Graphics/GPU	Nvidia GeForce RTX 3070
RAM	64.0 GB

<u>As this task is highly hardware the following changes were made to improve the efficiency of the model:</u>

Smaller File for data processing:
The original data file for development contains a significantly larger number of data samples compared to the Twitter-dev-data.txt file that we have chosen to use. Using a larger sample can affect the model's accuracy as having a larger and more diverse set of data allows for different trends to be identified.

Reducing the batch size:
One common error was in memory, when training and making predictions with the model, I often had the error of memory usage, due to this I reduced the batch size of my data to a size of 30 samples. Having a higher batch size may be useful in the generalization of the model.

Reducing the number of epochs:
By reducing the number of epochs, I am reducing the learning time for the model, increasing the number of epochs can also help to make better predictions from the data, we can see from our small test that the accuracy does improve with additional epochs.

To improve this model, there are most definitely multiple approaches we can take. Firstly, by investigating the format in which the data is preprocessed. From our debugging we were able to see that the format of our data is clean however after tokenization it seems we are losing some of the features causing the results to be mostly neutral. Hardware seemed to be a big factor in the processing of this model, If the code is running on a higher spec, then the computation load may decrease leading to a more efficient model. A potential solution to this is to utilize the GPU (cuda) or multi-threading to improve the computation load and time.

The code is also running multiple steps and ensuring that these are not all processed in one file ensuring that models are not overlapping or affecting one another's results By segmenting data and potentially considering a multi-threaded approach we can look at preventing the issue of there being high memory usage.

III. Conclusion

To conclude, the classifier models I have built in its current form only provides a foundation for development. Further investigation into the methods is most definitely required. Understanding each process and ensuring that it is producing the correct results at each stage. In the previous section, I have outlined some steps that can be taken to solve and improve on this model and while there was a lot of debugging, most of the issues raised come from the model's optimization.

References

[1] "Getting Started with Sentiment Analysis on Twitter." https://huggingface.co/blog/sentiment-analysis-twitter

[2] "What is Sentiment Analysis? - Sentiment Analysis Explained - AWS," Amazon Web Services, Inc. https://aws.amazon.com/what-is/sentiment-analysis/