Assessment 1: Natural Language Processing

Name: Hamza Khan

SN: 180426342

Q1: Input and Basic preprocessing

I first needed to create a function called parse_data_line that would return a tuple that contained the text and what that text had been classified as. From the given code we had a function called load_data, which contained the parse_data_line function. The load_data function takes my TSV file iterates through the lines using the for loop, if the first element is equal to "Id" it skips because this means that it is the column names. Then for each other line it calls the parse_data_line function that I created to pull the text and the label and create a tuple.

Essentially the whole point of this task is that we have a raw dataset. Part of the dataset there are datapoints that would not be useful in our working, for example, Id number, which can be removed. I am only interested in the text and whether that text is positive or negative.

I then implement my own for loop to make sure that the code that I had created worked. I used load_data("sentiment-dataset.tsv") to see if it would produce the first 5 tuples with the text and label, and it did.

Now that I had loaded the data in and put it into a format with the useful information, I now had to do create the pre_process function. There are many reasons why it is important that I do this. One reason is because there are going to be a lot of text/special characters that are going to be a part of the text that will be of no relevance in the sense that they will not help me to classify text into whether it is positive or negative, so I can remove them.

In this function I kept it very simple focusing on separating punctuation from words (both separating punctuation at the end of words and at the beginning of words). I also performed some normalisation, for example, making everything lowercase. This is extremely useful because this would make sure that the same word in different cases is treated the same. However, there are some negatives to this, for example "Polish" and "polish". The "Polish" is a country but "polish" is not and performing a lowercase will make them have the same meaning, which can be misleading. I finally wanted to mention that the tokenisation was done based on whitespace – this would split the words into a list of words (tokens).

Q2: Basic Feature Extraction

In a nutshell my function to_feature_vector takes a list of tokens (for example from the preprocessing which I just done) and returns a local feature dictionary. As well as creating a local feature dictionary it also has a global feature dictionary that has unique tokkens and their indices.

The first step of my function is to check whether the word from my token is in my global dictionary. If it is, then we assign an index. If not, that means it is not in the dictionary and I need to add it and I need to index it with the total number already in the global dictionary + 1.

The next step focuses on the local feature dictionary, where I have an if statement that says that if index i is already present, the value is increased by 1/len(tokens), if it is not present then we assign a new value to it, which is 1/len(tokens). The reason I did this is because it is possible that a single text can have repeated the same words multiple times, which can skew the numbers.

Q3: Cross-Validation

This step is very important and helps me to see how well my model does with unseen data. The K-Fold allows me to train and test a certain number of times, in my case it is 10. My cross_valdiate function takes as input my dataset and the number of folds I am looking to do. During each iteration the dataset is divided into training and validation sets. This allows me to train the data and get some prediction. Do this is all good, but I need to see how well it is performing which is why I have my predictions and I also have the correct outputs. Using this I can then create a report that allows me to see the performance of each fold. The key metrics that I focused on were precision, recall, f1-score and accuracy. I then stored these in a variable called cv_results, which contains average scores for all folds and be returned.

Q4: Error Analysis:

After I had done all the preprocessing and did my 10-fold cross validation I know had to look at the performance of the classes using a confusion matrix. After producing my confusion matrix, I could see that the False Negative value was 314 and the False Positive value was 814. The confusion matrix is good because it allows me to see a visualisation of the TP,FP,TN and FN. The purpose of the confusion matrix is focused on the FN and FP, I want to see out of all the data we had available and the processes we done how well the modelled work. Ideally, I want a situation where the FN and FP are 0 and the TP and TN are as high as possible. Of course this is very unrealistic scenario.

Once I had created the confusion matrix I focused on FN an FP. I then created a new file where I stored all these FP and FN for further investigation.

Q5: Optimising pre-processing and feature extraction:

Now that I had created a model, tested it and got the results I went back to see where I could have made improvements which could lead to better results. The first area I focused on was the preprocessing.

My first attempt at the preprocessing was a very simple one. In this section I copied across the one I had before because I believe that those steps are key. When I was printing out the text I noticed that there were a lot of words that were being kept that would not help me determine whether a text was positive or negative. So, the first thing I did was find a way to remove them. This was done through removing stopwords. Stopwords are essentially common words – since these common words wont help me with deciding whether a text is positive or negative it is better to remove them.

When printing out some of the text I also saw that some text included links. Again, similar thinking like I did with the stopwords. I felt that the links were not useful in helping me to determine if a text was positive or negative and that it would be better to remove it from my tokenisation. The less noise I had the better it would be for my model.

I also noticed when I first did my preprocessing that words like “I’m” were getting tokenised into “I”, “’” and “m”, which again I did not think was useful. I decided to expand these contractions.

The final addition I done was lemmatisation. This again was done to help me to try and reduce the noise as much as possible and remove anything that was unnecessary. The whole point of lemmatisation is to reduce the words to their root, i.e if we had a word “running” it would become “run”.

I then moved onto feature extraction. Initially I wanted to do a bigram approach since my first approach was done using unigrams. But when I ran the code my results generated a worse result

then when I done unigrams. So I then decided to try the same feature extraction as before and it generated better results, only just. Which shows that my preprocessing that I did now had helped.

The precision of my model from Q1 to Q4 = 83%

The precision of my model from Q5 = 84%

So, I can see that the model did perform a tiny bit better result when I adjusted my preprocessing.

The final point that I want to make in comparison is that the FN and FP for my question 5 are 275 and 779 compared to the ones I got in the previous model which were 314 and 814. This shows that the model performed better as it produced less false results.