Q1 – Improve Pre-processing

After receiving the east ender’s dataset our first step is to pre-process the data. Initially, the data achieves a mean rank of 5.12 which through a variety of different pre-processing techniques I managed to bring it down to 2.5. We did this through lemmatizing the text, removing punctuation, stemming the text, and removing all stop words.

Lemmatization is the process of grouping together the different inflected forms of a word so they can be analysed as a single item. Lemmatization is like stemming but it brings context to the words. So, it links words with similar meanings to one word. An example would be converting the word “better” to “good”, which in turn will increase our rank as a standard becomes set for words. This also converts the text to lowercase.

Of course, removing the stop words helped our score as removing these words removes all our low-level information which will give more focus to the important information. At this stage I was happy with my mean rank as I made use of a variety of pre-processing techniques and not too low in that it would be hard to improve even greater in the further stages.

Q2 – Improve linguistic feature extraction

After we have implemented our pre-processing techniques to the corpus, we apply feature extraction methods to improve our mean rank further. As you can see from the ipynb file there were many different options applied, but the best proving to be using sentiment analysis, of 1.875. I managed to obtain better results using gender classification and sentiment analysis of 1.4375 which I have included in the report, however, because of the next question not allowing us to make use of the gender column I didn’t want to do something illegal. The classifier would be trained using a Naïve bayes classifier on our words by predicting who said what word. The documentation for this is on nltk website at nltk.org/book/ch06.html. So, I decided on the sentiment analysis approach, if I were to say why this is so effective would be because of Eastenders being quite a dramatic show, there will be lots of correct tagging. Another option that I tried was going into using bigrams and trigrams of particular words, however it was producing very poor results; my thoughts on this is because of the sheer amount of pre-processing that we have done, particularly removing stop words to the point where it just wasn’t working effectively as the text was too broken up.

To quickly run through our final to\_feature\_vector\_dictionary approach, we do some small processing techniques removing the startofscene and endofscene words which I implement in question 3 as markers. We then remove any empty elements. Finally, every word is tagged with a positive, negative or neutral tag that is based on compound score. The compound score uses an algorithm that uses a sentiment lexicon approach as well as grammatical rules and syntactical conventions for expressing sentiment polarity and intensity.

Q3 – Add dialogue context data and features

This section was all about providing more context with the scene. Specifically, we had to find a way to incorporate the previous and next lines into our character documents. I managed to find a fantastic function online that helped me tremendously in retrieving previous/next lines from a user on stack overflow which can be found here: https://stackoverflow.com/questions/323750/how-to-access-the-previous-next-element-in-a-for-loop/22030004#22030004. We of course implement these new lines inside of the create\_character\_document\_from\_dataframe function. Inside of here we do a couple of conditions to check if we are on the last sentence within a scene or not and if it’s the first sentence being passed through the function – this is of course as we don’t want to get next and previous sentences for other scenes as this will of course only affect our mean rank negatively, as there will be even less context. The main benefit of this is of course providing more data and providing more context around that data.

So, my best result from adding dialogue context data and features was just bringing through the previous and next context lines – achieving a mean rank total of 1.375. I did also bring through more data that relates to the sentences, specifically tagging the words with their corresponding scene number or their corresponding scene name. With the corresponding scene info my results were still quite effective, achieving a mean rank of 1.4375. Since we were not allowed to implement the character\_name or gender then there was not a whole lot more we could do with the dataset except for the above methods.

Q4 – Improve the vectorization method

The objective of this question is to change the way we pass our data through to our ranking algorithms, specifically I went with using a TF-IDF vectorizer. One of the main changes in using this was instead of passing through a Counter of the words in a dictionary format, we pass through strings of the words. The TFIDf stands for Term frequency inverse document frequency that is a technique for text vectorization based on the Bag of words (BoW) model, although it is deemed much more effective as it considers the importance of the word in a document into consideration – which Is strange as at this point in the coursework we have lots of context so I would have thought this would have increased my scores significantly.

The tf-idf vectorizer has parameters that can be tuned that may increase or decrease performance. However, in my case they didn’t seem to prove too effective. I first started with manipulating the min and max df parameters which to be honest only made things worse for me. I think was because of my amended corpus, the words that get passed through are all vital as I stripped out quite a lot in the pre-processing stages.

I also tried to make use of the max\_features parameter, which creates a new vocabulary that only consider the top max\_features ordered by term frequency across the corpus. I changed around with number ranging from lower to higher and all it seemed to do was decrease my performance significantly. My optimal performance was with no parameters, which also gave me a mean rank of 1.375 and 12/16 predictions but with a greater mean cosine similarity. The mean cosine rank just validates how optimal our model is, although it’s not a brilliant indicator having a higher score is better than having a lower one.

I also implemented K-best at this stage, with max\_features of 5000 and using chi squared as the parameter which yielded me results of 8.5, proving ineffective. The SelectKBest package from scikit-learn selects features according to the k highest scores. Seeing as we already have quite a low score with effective pre-processing it appears to not be of use to us.

Q5 – Select and test the best vector representation method

As you can see from the notebook my best vector representation method achieved a mean rank of 1.0 for the training set and for the testing set 1.375 with an accuracy of 0.75 and a mean cosine similarity of 0.99, making use of a tfidf vectorizer. I have included all my tests/adjustments throughout the process of this project inside of the ipynb file. Overall, I am happy with my scores, and I felt this project was fantastic from start to finish and has given me much more of an insight into the realm of natural language processing.