Report: Vector Space Semantics for Similarity between Eastenders Characters

Q1. Improve pre-processing (20 marks)

We have tried multiple techniques in pre-processing. Firstly, we have used regex, this method separates the punctuation at the end and the start of the word. It also converts written numbers into their numerical format. Further, we have filtered out the special characters present in the corpus. Then we tried to convert all the letters into lowercase. After that, we removed the Stop words using nltk library. At last, we have tried to use lemmatization. By performing all these techniques together in the same sequence, we have got the best **Mean Rank of 4.5** with the cosine similarity as 0.99. Different combinations of pre-processing were tried, whose Mean Rank and cosine similarity are printed as a part of the output in jupyter notebook.

Q2. Improve linguistic feature extraction (20 marks)

Different techniques were used to improve the linguistic feature extraction. Below all techniques were tried, and verified with the training and validation dataset:

1. **Sentiment Analysis: VADER Sentiment Analysis**, is used to improve linguistic feature extraction. VADER is a lexicon and rule-based analysis instrument that utilizes a mix of lexical corpus. The VADER not only tells us about the Polarity score but also about the positive or negative sentiment conclusion. Just broadly using the Vader technique on the overall corpus was improving the mean score. But it makes more sense to get the positive and negative sentiment analysis for each sentence in the corpus. To achieve this, the character doc was converted into sentences separated by “EOL”. By using the polarity scores produced by Vader, the sentences were appended with Positive, Negative, or Neutral sentiments.
2. **POS tag:** As POS tagging is used to assign parts of speech to each word based on its definition and context, this is added to improve feature extraction. After performing sentiment analysis, each word from the newly generated character set is again split into words. And for each word a POS tag is appended with ‘/’ between them.
3. **Ngram:** Different ngrams such as Unigram, Bigram, and Trigram were used on the training corpus. We have tried these ngram techniques with POS tagging and Sentiment analysis. Out of them, the Bigram and Trigram provide Mean ranks of **6.12** and **6.18**, which increases their mean rank comparatively to **Unigram which provides a mean rank of 3.375**. **Thus, the unigram technique was finalized with Sentiment Analysis and POS tagging.**

Q3. Analyse the similarity results (10 marks)

From the plotted heatmap of the confusion matrix, we can see many of the characters are getting similar words and that is the reason we are finding big numbers outside diagonal values. For example, the worst values can be seen for JACK. When comparing words spoken by **JACK with PHIL** which has a similarity of **0.924,** which is almost the same JACK vs JACK diagonal values, 0.926. This is due to the number of similar words they are speaking. In code, I have printed these words and the length of common words which are spoken by both is **3338**, which is huge.

On the contrary, the similarity between **RONNIE and IAN is 0.785**, which is quite less. As the number of common words spoken by them is only **2578**, which is quite less than the above case. So, the plotted heatmap row for Ronny is better as it has a higher value of similarity only with himself and the similarity is less with other characters.

The worst and good similarity values between characters as explained above and similar words they spoke are printed in jupyter notebook.

Q4. Add dialogue context and scene features (20 marks)

To add dialogue context and the scene features, here we are first trying to compare the episode scene number with the previous and next characters. If the episode scene number is matching with episode scene number of the next character, then we are adding the scene line as the new column in the data frame as the next line. If the episode scene number is matching with episode scene number of the previous character, then we are adding the scene line as the new column in the data frame as the previous line. So basically, we are comparing the lines spoken by characters in the same scene. After adding this technique, **the overall mean rank improved to 2.5**.

Q5. Improve the vectorization method (20 marks)

To further improve the mean rank score, we are going to use **TFidf-Transformer**, which transforms a count matrix to a normalized tf or tf-idf representation. The value for norm will be the same as by default as 'l2'. i.e., the sum of squares of vector elements is 1. The cosine similarity between two vectors is their dot product. We are keeping 'use_idf' as True as we want to enable inverse-document-frequency reweighting.

Adding the TF-IDF with dialogue context and scene features further improve the Mean Rank to 1.43. But when trying to just use the TF-IDF transformer technique without adding dialogue context and scene features, we can see the **Mean Rank further improves to 1.25**. It means, just adding the TF-IDF, we are getting a better mean rank on the validation dataset.

Q6. Run on final test data (10 marks)

All the selected features, which provide better accuracy on Training data will be finally used to run on Test Data. We will be using **the Pre-processing techniques, Sentiment Analysis, Unigram, POS Tagging, and TF-IDF Transformers**. The heatmap diagonal values look better than non-diagonal for all characters. **The Final Mean Rank on test data with all above-used techniques is 1.0625, which gives the result as 15 correct out of 16.**

Note: All the used and non-used techniques results are printed in code, to understand which techniques were finalized.