ECS763P- NLP Assignment 2: Vector Space Semantics for Similarity between Eastenders Characters

Introduction:

In this assignment, we will be creating a vector representation of a document containing lines spoken by a character in the Eastenders script data (i.e., from the file training.csv), then improving that representation such that each character vector is maximally distinguished from the other character documents.

Q1. Pre-processing-

I. **Remove Punctuation:**
The punctuation removal process helps in identifying each word equally. For e.g., 'wonder' and 'wonder!' are identified as the same after the removal of punctuation (!). In this case, it reduced the mean_rank to 3.875.

II. **Conversion to lowercase:**
In this technique, we convert all the words in the token list 'tokens' to lowercase form. This is used to treat each word equally without being affected by capitalisation. This technique further reduced the mean_rank to 3.0.

III. **Stop word removal:**
Stop word removal is a process of eliminating all prepositions/very frequent functional words. By removing these words, we remove the low-level information from our text to give more focus to the valuable information. Mean_rank = 2.3125.

IV. **Lemmatising:**
Lemmatization considers the context and converts the word to its meaningful base form called Lemma. E.g., started, starting → start. Mean_rank = 2.1875

V. **Stemming:**
Stemming is a Text Normalisation technique that stems or removes the last few characters from a word. Mean_rank = 2.0

Overall, pre-processing had a positive effect on the mean_rank.

Q2. Feature extraction:

I. **N-gram:**
In this technique, we take an N sequence of words and count how often these sequences occur in the texts. This way we can add contextual information to a word which in turn would increase the quality of the information received by the classifier. Although in this case, any sequence of n>1 had a negative effect on the mean_rank.

II. **Ngram POS-tagging:**
POS (Part-Of-Speech) In this technique, we first extract ngram sequences and then assign them respective POS tags using nltk.pos_tag() function. This will create POS tag sequence. POS tagging is a task of labelling each word in a sentence with its appropriate part of speech. POS tagging gives more meaning to a sentence.
This technique had a negative effect on the mean_rank

III. **Gender classifier:**
In this technique, a KNN classifier is trained on train_data (data split into 70-30 train-val) from all_train_data with the feature as Character_name and label 'Gender'. The best parameter as well as cross-validation is obtained using GridSearch() and the classifier has an accuracy of approx. 82% on val data.
The model is then added as an extra_feature which takes the character_name as input and gives their gender as an extra_feature.
This technique had no effect on mean_rank and just a slight effect on the cosine_similarity score

IV. **Average dialogue length:**
In this technique, we calculate the average number of words spoken by a character. This method would be useful if some characters have considerably longer dialogues than other characters.
This did not have any effect on the mean_rank in this case.

V. **Min_max doc frequency**:
In this technique, we eliminated words/features that appear less than or more than the minimum and maximum threshold respectively.
This technique had a negative impact on the mean_rank.

Q3. Analyse the similarity results:

Cosine similarity tells how similar 2 documents are based on the angle

between their feature vectors irrespective of their sizes. In this scenario, let's take an example of target character SEAN who has the highest match with IAN and not with SEAN himself in the training vector, here can see that even though SEAN(val) and SEAN(train) have a higher number of common characters, IAN has feature frequency which is more similar to the target. For e.g., comparing word frequency of SEAN and IAN: 'want' – (3, 17), 'yeah' – (7, 45), 'alright' – (3, 15), 'go' - (4, 37), 'come' – (3, 26), 'turn'- (3, 6) we can see the target document have counts which are much higher than other words this is also the case with IAN's document. While in the case of target and JACK(train) frequencies of features are not similar, 'want' – (3, 30), 'yeah' – (7, 25), 'alright' – (3, 19), 'go' - (4, 25), 'come' – (3, 16), 'turn'- (3, 7) we can observe that 'yeah' which has the highest count in target but this is not the case in train.
Meanwhile, the least similar character HEATHER had a completely different feature frequency (word – (target_count, train_count) 'want' – (3, 11), 'yeah' – (7, 16), 'alright' – (3, 9), 'go' - (4, 22), 'come' – (3, 31) and also lesser number of common words.
This proves that cosine similarity is not solely dependent on the number of common features but the angles between the feature vectors.
Other such characters are SHIRLEY, JACK and ROXY who display similar characteristics.

Q4. Add dialogue context and scene features:

In this part, we have added some context to the lines spoken by the target character. The context here is the lines spoken by other characters in the same scene (before and after the target character's lines). This is achieved by adjusting the create_character_document_from_dataframe where the character_doc dictionary is of the format e.g. - {Character_name: <episode scene number> <Line spoken before in the same scene> character_line 1<line spoken after in the same scene> _EOL_ <episode scene number><Line spoken before in the same scene> character_line 2<line spoken after in the same scene> _EOL_.....and so on}.
This process can add more clarity and help distinguish the target character as more information is added.
This technique improved the mean_rank from 2.0 to 1.5625.

Q5. Improve vectorization method:

1. **TF-IDF**:
Term frequency-inverse document frequency is one way of assigning higher weights to the more discriminative words. TF: Frequency of the word in the document. IDF: Inverse document frequency. The fewer documents in which a term occurs, the higher this weight. Because of the large number of documents in many collections, this measure is usually squashed with a log function.
This technique is achieved by creating a Pipeline which first fits the training_corpus data into DictVectorizer and then to TfidfTransformer().

2. **Binarizer():**
Binarize data (set feature values to 0 or 1) according to a threshold. Values greater than the threshold map to 1, while values less than or equal to the threshold map to 0. With the default threshold of 0, only positive values map to 1.
This technique was achieved in the same manner as TFID and had a negative effect on the mean_rank

Mean_rank before matrix transformation: 1.5

Technique	Mean_rank
TFIDF	1.3125
Binarizer	2.5

The best train-validation system result achieved so far is 1.3125.

Q6. Run on final test data:

We now test the system on actual test data from test.csv. The only modification required for this was adding an 'episode-scene' column to the test data frame which is used by create_character_document_from_dataframe. The final score on test data obtained is a mean_rank of 1.25 and an accuracy of an 87.5%