ECS763P: Natural Language Processing

Assignment_2_distributional_semantics

Solution 1: The most frequent word for each character consists of EOL and stop words, which provide no unique information. A function *pre_process_q1* has been updated with the following pre-processing techniques – **Tokenization** is done at a sentence level using the NLTK library. Training data for each character consists of all the lines spoken by that character, these are split using _EOL_ and each sentence is then used to tokenize into words. Each token is then compared against the list of **stop words** and **punctuations**. '**SpaCy'** library's stop words list has been used as it contains 326 stop words as compared to NLTK's 179 stop words which include, "n't", "'s", "'ve", etc., which acts as a noise in the data. Tokens are further **lemmatized** to their base form using NLTK *'WordNetLemmatizer'*. Subsequently, the mean rank on the validation data has improved from 4.5 to **1.56** and the accuracy has jumped to the correct classification of 12 documents out of 16 as compared to 4 before the pre-processing. Additionally, Stemming was also applied in the pre-processing which decreased the mean rank and therefore it has been removed as Lemmatization is already used.

Solution 2: The second problem focuses on feature engineering principles in which different feature extraction techniques have been applied and tested. Firstly, we have tried to leverage the **POS tagging** feature of SpaCy and NLTK libraries and compare the results using both. Current training data is in the form of sentences; therefore, POS tagging is done in the pre-processing step where we are already looping over each sentence and tokenizing each sentence into words. These words are then replaced by the combination of word and corresponding POS tag concatenated using a delimiter **'|',** for example, *'Lesley|PROPN'*. The resulting Mean rank decreased slightly. Subsequently, another feature **N-grams** is added by using the NLTK n-grams function. A combination of n-gram which includes Bigrams, Trigrams, and 4-grams have been added ti the feature dictionary and tested using the validation data. This resulted in an improvement in the Mean rank from **1.93 to 1.87**. A similar experiment is done using NLTK POS Tags with the combination of 4-grams in the feature dictionary and the final mean rank achieved is **1.5** with an accuracy of **0.68**. Additionally, another feature; **Sentiment analysis** is used by importing the tool - *vader_lexicon* of NLTK *SentimentIntensityAnalyzer.* Sentiment for each sentence is analyzed using the range of compound value and the count of sentences in each sentiment category *('positive', 'negative', and 'neutral'*) is recorded in a dictionary. An example of feature generated is a dictionary *{positive_sent:166, negative_sent: 47, neutral_sent: 147}*. These sentiment counts are then appended at the end of the feature dictionary for the respective document. Testing with Sentiment Analysis fields yields decrement in the mean rank to **2.68.** It was also tested without the use of n-grams and the performance recorded was similar. Finally, SelectKBest has been used to select the k number of best features for classification. Tested the KBest for different values of k like 1000, 2000, 3000 and achieved the best mean rank of **1.437 and accuracy of 81.25%** (13 out of 16 characters) for value 2000 as it started increasing again at k = 3000.

Solution 3: In the third task, we are doing the contextual analysis for each dialogue. We are taking the previous and the next lines from different characters to create a context for the current dialogue for each character. The first task in order to implement this is to update the *'create_character_document_from_dataframe'* function and fetch the contextual dialogues. Below is the approach used for the same: For a current character, **in the same scene**, if the previous or the next dialogue is from some other character, it will be used as PRE-context or POST-context respectively. The character's current dialogue is used with the CURRENT key. If the previous dialogue is spoken by the same character or if there is no previous dialogue, PRE context will be taken as None otherwise the previous line in that same scene. If the next dialogue is spoken by the same character or if there is no next dialogue, POST context will be taken as None otherwise the next line in the same scene. Context is created by leveraging pandas dataframe operations.

In the pre-processing part, the previous and the post sentences are tokenized into words with added prefix *PREV_* and *POST_* respectively. Pre-processing techniques like stop words removal, punctuations removal, and POS tagging is also done in the pre-processing function as implemented in problem 1 and 2. As observed in the previous problem, using n-grams features helped in improving the mean rank, therefore, n-gram features are added on the current context words only in the function *to_feature_vector_dictionary.* Validation data used with 40 lines is used to evaluate the performance and calculate the final mean rank of the system with context data. There is a significant improvement observed in the mean rank after adding the contexts. The final mean rank achieved in this case is **1.18**. There is also an improvement in the accuracy of the model which from 68% to 81% (13 out of 16 characters)

Solution 4: In the fourth problem, we are trying to test the different vectorizer approaches. Currently, we are using *DictVectorizer* which transforms lists of feature-value mappings to vectors. We are using pre-processing and features explored in the third problem which includes context data and POS tags, Bi-grams as the features. Imported *TfidfTransformer* from *sklearn.feature_extraction* library. Created an object of TfidfTransformer object. In the case of fitting the data using training data, Tfidf object is fit using DictVectorizer's transformed matrix and perform the transformation to generate the vectors. On comparing the vectors generated using TfIdfTransformer, and calculating the similarity between training and validation vectors, a significant improvement has been observed in the mean rank i.e., mean rank has improved to **1.06** and the accuracy achieved is 0.9375 i.e., **93.75% (15 out of 16 characters**)

Solution 5: In the given problem, we are selecting the best approach from the experiments performed in each of the above problems. For Data extraction, we started with the extraction of lines for each character, however, in question 3, we identified that using the previous and next context lines can result in better classification of the character. Therefore, we will be using the context data and implementing the updated function 'create_character_document_from_dataframe' for data extraction. For pre-processing, we have explored different techniques and different libraries namely tokenization using NLTK and SpaCy library, stop words removal using SpaCy as it contains more relevant stop words for our data. We have also explored the removal of punctuations present in the data. These are assessed based on the training data. Lemmatization and stemming were also explored whereas stemming provided poor performance and thus only lemmatization has been used for the final model. POS tags have been used and appended to each word including Pre and Post. In the featured engineering step, we are using bi-grams of the current words in context only. Pre and Post words are used as it is. Counts are calculated for each of these words and bi-grams of current context words as well. The feature dictionary created in this step is then used to vectorize the text words. DictVectorizer has been used to take the feature dictionary and the matrix has been created which is then used to fit and transform into vector format using TfidfTransformer. As observed in problem 4, TfidfTransformer gives much better performance as compared to the DictVectorizer alone, we will be using the same in our final transformation.
Finally, the model is trained on the complete training data (all train data) and tested using the unseen test data (40 lines) from the file '*test.csv*" and the resulting mean rank achieved for the given data is ***1.0*** with an overall accuracy of **100% (16 out 16 characters)**

In conclusion, the pre-processing(removal of noise from the data), the addition of relevant features (pos tags, n-grams, etc) in the model increases the accuracy of the classification and rank the similarity coefficients better among different documents.