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

This report is a vector representation of a document containing lines spoken by a character in the Eastenders script data. A character identification system for the script of the TV show " Eastenders". The goal of the system is to identify the character speaking in each line of dialogue. I have approached this problem as an information retrieval task, where I first pre-processed the script to create a collection of documents representing the lines spoken by each character. Then transformed these documents into feature vectors using various techniques and used these feature vectors to compute the similarity between the target (unknown) character's lines and the lines spoken by each of the known characters.

1. Improve pre-processing: To improve the pre-processing, I made several changes to the pre_process function. First, split the input text into sentences using regular expressions. This allowed us to better capture the structure and context of the dialogue. Next, removed punctuation from the sentences. This helped to reduce the noise in the data and make it easier to analyze. I have also normalized the words by lowercasing them, which helped to reduce the dimensionality of the data. Finally, lemmatized the words to reduce them to their base forms, which again helped to reduce the noise in the data.

2. Improve linguistic feature extraction: Defining a pipeline for training a gender_clf to predict the gender of a character based on their dialogue. The pipeline consists of a CountVectorizer, a TfidfTransformer, and the gradient-boosting classifier. The CountVectorizer is used to extract n-grams (unigram, bigram) from the character's dialogue, and the TfidfTransformer is used in the classification pipeline only to transform the resulting feature vectors into a matrix. The gradient boosting classifier is then trained on this matrix and the character's known gender. The to_feature_vector_dictionary function is also defined, which converts a list of pre-processed tokens and extra features into a dictionary representation of the features. Initially, the function extracts n-grams (unigram, bigram) using a CountVectorizer and includes sentiment features (positive, negative, neutral, compound) using a sentiment analyzer.

Results after pre-processing:

	Mean rank	Mean cosine similarity	Accuracy
Validation data score	2.8125	0.321990	8 correct out of 16 / Accuracy: 0.5
Test data score	3.375	0.319927	7 correct out of 16 / Accuracy: 0.4375

3. It is observed that Max has the highest similarity match between the training vector doc and test vector doc. That is because Max has a unique speaking style as compared to others. The similarity of each character’s test vector doc and training vector doc is as follows, Christian=0.765, Clare=0.779, Heather=0.775, Ian=0.796, Jack= 0.798, Jane=0.815, Max=0.842, Minty=0.748, Others=0.787, Phil=0.812, Ronnie=0.806, Roxy=0.795, Sean=0.81, Shirley=0.767, Stacy=0.816, Tanya=0.786. From these values we can analyze that character vectors ranked closest to the training vector may have similar language use, resulting in similar word or n-gram features. On the other hand, the character vectors ranked furthest away may have distinct language use, resulting in dissimilar word or n-gram features. It is possible that there may not be a successful highest match between the target character in the training set and that character's vector in the heldout set due to the distinct language use of the heldout character vector.

4. Add dialogue context and scene features: A function called create_character_document_from_dataframe takes in a Pandas DataFrame containing script data and an integer max_line_count and returns a dictionary with character names as keys and strings of all the lines spoken by each character as values. The function first creates empty dictionaries to store the character documents and line counts, then iterates over the rows of the DataFrame,

adding each line to the appropriate character's document and incrementing the line count. If a character's line count exceeds max_line_count, the function skips the remaining lines for that character. In addition, there are two versions of the function included: one that includes context and scene information and one that does not. The version that includes context and scene information does the following additional tasks: it identifies the unique episodes and scenes in the DataFrame, filters the DataFrame to include only the lines spoken in a given scene for each episode, and for each line, name, gender, and scene_info in the filtered DataFrame, it adds the scene information (if it is 'INT', 'EXT', 'DAY', or 'NIGHT') and the line to the appropriate character's document, separated by an end-of-line marker. The purpose of this function is to create a set of documents, one per character, containing all the lines spoken by that character in the movie script, with the inclusion of context and scene information potentially useful for further analysis.

5. Improve the vectorization method: The function create_document_matrix_from_corpus is being used to transform a list of (class_label, document) pairs (referred to as "corpus") into a matrix. The current implementation of the function uses a dictionary vectorizer (DictVectorizer) to map the feature dictionaries for each character document to a sparse matrix. However, to improve the performance of this function, we can try using a matrix transformation technique like TF-IDF. To do this, we can initialize the corpusVectorizer variable with a TfidfVectorizer instead of a DictVectorizer. This can be done in the same place as the current initialization of corpusVectorizer. We can then fit and transform the data using the TfidfVectorizer, instead of the DictVectorizer. It is important to note that this function is used both in training/fitting (with fitting set to True) and in transformation alone on test/validation data (with fitting set to False). Therefore, it is important to make sure that any transformers we want to try are initialized in the same place as the current initialization of corpusVectorizer, before calling create_document_matrix_from_corpus. TfidfVectorizer is a better choice when the goal is to identify the most important words in a set of documents since it considers the frequency and importance of each word.

If, CorpusVectorizer = TfidfVectorizer the mean rank and accuracy are as follows:

	Mean rank	Mean cosine similarity	Accuracy
Training/Validation data score	1.5	0.79360707	8 correct out of 16 / Accuracy: 0.8125
Test data score	1.1875	0.797582099	8 correct out of 16 / Accuracy: 0.8125

6. Run on final test data: The TfidfVectorizer is used to transform the feature matrix, and the resulting training and test feature matrices are then used to evaluate the character classification performance using the compute_IR_evaluation_scores function. The results on the test data are as follows:

Mean rank= 1.1875

Mean cosine similarity= 0.7975925778561352

13 correct out of 16 / accuracy= 0.8125

In conclusion, the goal was to create a vector representation of lines spoken by characters in the Eastenders script and use that representation to identify the speaking character in each line of dialogue. The vector representation was improved through pre-processing, feature extraction, and transformation techniques. The resulting vectors were then used to compute the similarity between the target character's lines and the lines spoken by each of the known characters. The mean rank is 1.5 and the accuracy on the validation data is 81.25%. The final mean rank is 1.1875 and the accuracy of the system on the test data is 81.25%