# Programming Assignment 2 : Vector Space Semantics for Similarity between Eastenders Characters

Sayed Sohail Pasha Peerzade
220541549

**Q1. Improve pre-processing (20 marks)**

Updated the pre-processing function (pre_process_new) to perform the following pre-processing tasks:

1. Removing unnecessary whitespaces (trailing and leading) and extra spaces between words.
2. Lowercasing the text to reduce the dimensionality of data.
3. Tokenization of text using NLTK's word_tokenizer, where each sentence is tokenized into words.
4. Stop words removal using Scikit Learns ENGLISH_STOP_WORDS, to remove common words that have little or no meaning in the context of a sentence. For example: "a", "an", "of" etc. Removing stopwords reduces the noise in the data
5. Punctuation removal,by removing non alphanumeric characters from the tokens.
6. Removed High-frequency words, to remove very frequently occurring words. However I have commented this part of the code as it may be harmful for specific NLP tasks such as Sentiment Analysis. where the word 'not' can change the meaning of the whole sentence.
7. Spelling correction to improve the accuracy of NLP algorithms by ensuring that the text is free from spelling errors. Implemented Spelling correction using SpellChecker Library.
8. Tokens were further lemmatised to their base form by using NLTK WordNetLemmatizer.
9. Implemented Stemming using SnowBallStemmer, but have commented this part of code as Lemmatision was implemented.

After performing these preprocessing tasks the **mean rank** was improved from 4.5 to **1.68** and **mean cosine similarity** was improved from 0.91 to **0.56**. The **accuracy** has been increased to correct classification of **11 out of 16** as compared to 4.

**Q2. Improve linguistic feature extraction**

Implemented 3 linguistic feature extraction techniques, aiming to improve the quality of extracted features.

1. N-gram POS Tagging:

   Each word in the text was marked with corresponding grammatical role, such as noun,verb, adjective,preposition etc using NLTK's pos_tag POS-Tagger. And then these tagged words were used to create Ngrams, which include Bigrams, Trigrams and 4-grams. These were then added to the feature dictionary. This resulted in a mean **rank** of **1.937** and **accuracy** of **56%** and **9 out of 16** correct predictions.

2. Sentiment Analysis:

   Sentiment Analysis was implemented using vader_lexicon of NLTK. Sentiment for each sentence was analyzed and the sentiments (positive, negative and neutral) were stored in the dictionary. This dictionary was then added to the feature dictionary. This resulted in a **mean rank** of **2.687** and **accuracy** of **56.2%** and **9 out of 16 correct predictions.**

3. N-gram POS Tagging and Sentiment Analysis:

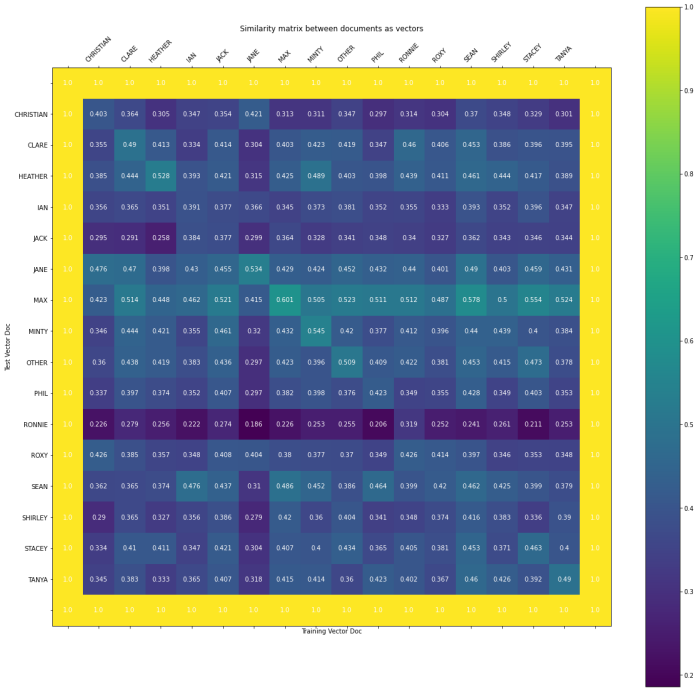
   Both the above feature engineering techniques were then used together. This resulted in a mean **rank of 2.8125**, **accuracy** of **50%** and **8 out of 16 correct predictions.**

**Q3. Analyze the similarity results**

1. By using N-grams with POS Tags we were able to achieve a mean cosine similarity of 0.458. Which means the model was able to distinguish between the vectors to some extent. In the below heat map generated for N-grams with POS Tags we can see that.
   - The Similarity score is highest for each character when the model predicts the character name correctly. In the row for character MAX, the highest similarity score is 0.601 when the model correctly predicts MAX.
   - The Similarity score of MAX against SEAN is 0.578, we can infer that the language constructs used by SEAN

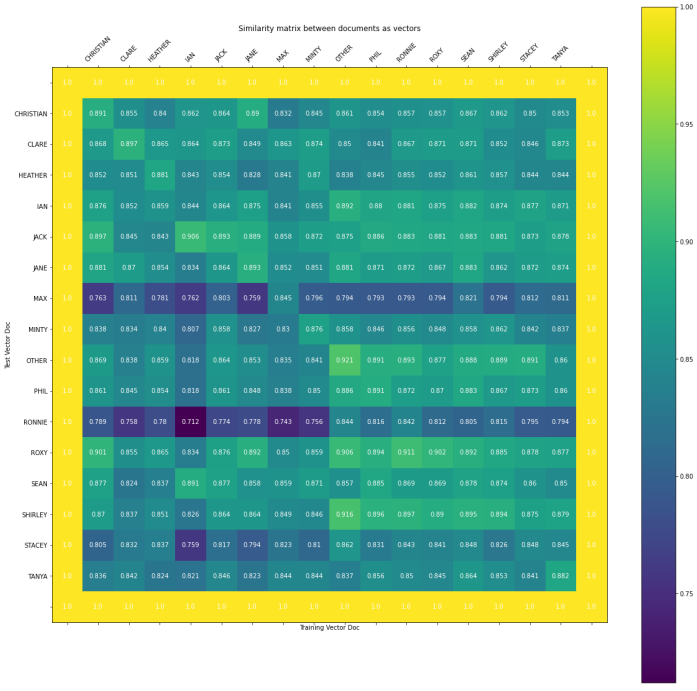
and MAX maybe similar.

- The lowest similarity score in between RONNIE and PHIL which means that Ronnie and PHIL do not use similar language structure.




*Heat map for Ngrams with POS Tags*

2. By using Sentiment Analysis we were able to achieve mean cosine similarity of 0.879, which means that the sentiment of lines spoken by characters is generally similar.




* We can see that the Similarity score between RONNIE and IAN is lowest at 0.712, which means IAN and RONNIE generally do not share similar sentiments.

## Q4. Add dialogue context and scene features

To incorporate the context of the line spoken by characters in the dataset, 'create_character_document_from_dataframe' is updated to fetch contextual dialogues by taking the previous and next lines from different characters to create a context for the current dialogue for each character. The approach used for analyzing the context of each dialogue is to take the previous and next lines spoken by different characters for the current character within the same scene and use them as the PRE-context and POST-context respectively. The current dialogue of the character is used with the CURRENT key. The previous and next lines of dialogue in the same scene will be used as the PRE-context and POST-context, respectively, unless there is no previous or next dialogue or the dialogue is spoken by the same character, in which case they will be set as None.

During pre-processing, the previous and next sentences are converted into a list of words with the prefixes "PREV_" and "POST_" added respectively. Additional techniques such as removing stop words, punctuation are also applied. As seen in Q2, incorporating n-gram features improved the mean rank, so n-grams were added as part of feature engineering. Adding contexts of the line spoken has resulted in a notable improvement in the mean rank. This resulted in a **mean rank of 1.43** and **accuracy** of **87.5%** and **14 out of 16** correct predictions.

## Q5. Improve the vectorization method

To improve the vectorization method, TfidfTransformer of sklearn.feature_extraction was used.In order to generate vectors, Tfidf object was used to transform the matrix obtained from DictVectorizer's transformed data when fitting the training data.
DictVectorizer takes in a list of dictionaries where the keys are feature names and the values are feature values whereas TfIdfTransformer, is used to scale down the impact of tokens that occur very frequently in a given corpus and that are hence empirically less informative than features that occur in a small fraction of the training corpus. It's a transformer that is used to scale a count matrix to a normalized tf-idf representation.

By comparing the vectors generated by TfIdfTransformer and DictVectorizer and measuring the similarity between the training and validation vectors, a substantial improvement in the mean rank was noticed, resulting in a value of **1.31**, and an accuracy of **87%**(**14 out of 16 characters**).

**Q6. Run on final test data**

From the above questions we have noticed that using context of the lines spoken by the characters and by using TfidfTransformer we can achieve better mean rank and accuracy. To create the best system to train on all of the training data and to test on the test file, In the preprocessing step the training and heldout labels were added to train and test data, The context of the line spoken was obtained by create_character_document_from_dataframe_q4 function created in Q4. Several preprocessing steps are applied in pre_process_q4 created for Q4. Ngrams of each of these words are created in the function to_feature_vector_dictionary_new. The feature dictionary created is then used to represent the text words in vectors. TfIdfTransformer is then used in create_document_matrix_from_corpus_q5 function from Q5 to scale the count matrix to normalized tf-idf representation.
IIn the end, the model is trained using the complete training data (all_train_data) and tested using unseen test data, the resulting mean rank achieved is **1.125** and accuracy achieved is **87.5%** and **14 out of 16** correct predictions.

In conclusion, the model is more effective when the data is effectively pre-processed and when the model has meaningful and important features.

**NOTE: Solutions to Questions 4,5 and 6 can be found in the second notebook**