1. Question 1

In this question, the main function to be modified is '**pre_process**'. The primary input of the function is the text of characters, and the output is tokens. Originally, the function use the string split method to tokenize to text, which is quite simple and ineffective (tokens are not clean/good enough). To produce a better result in the preprocessing step, we will aim at 5 potential methods:

+ *Use word_tokenize method from nltk library*
+ *Convert tokens into its lowercase form*
+ *Remove punctuations*
+ *Remove too common words (stopwords)*
+ *Lemmatize tokens*

The issue arising from these 5 methods is whether we need to implement all or just a subset of them. To tackle this, the exhaustive approach has been chosen: evaluate all possible combinations of these 5 preprocessing methods (2^n subsets). The result is as below:

Table 1: Top 5 combinations of preprocessing methods returning the best performance

Method combo	Mean rank	Accuracy
Word_tokenize / lowercase / remove punctuation / remove stopwords / lemmatize	2.5	0.6875
Word_tokenize / remove punctuation / remove stopwords / lemmatize	2.6875	0.6875
Word_tokenize / lowercase / remove punctuation / remove stopwords	2.6875	0.5625
Word_tokenize / remove punctuation / remove stopwords	3.0625	0.6265
Word_tokenize / lowercase	3.25	0.4375

Based on the performance, the combination of all 5 methods will be chosen for implementation, including: *word_tokenize, lowercase, remove punctuation, remove stopwords and lemmatization*.

2. Question 2

There are two things needs to considered:

a. Try adding extra features

The model in question 1 simple use the unigrams and their frequency as the features, which could be insufficient. To add more features, there are some options:

+ *Extract ngrams from the character text (bigrams/trigrams/4grams)*
+ *Extract POS-tagging for the extracted ngrams*
+ *Sentiment analysis of character text (positive/neutral/negative)*

There are two functions involved: '**pre_process**' function helps to extract extra features (tokens) from the character text; and '**to_feature_vector_dictionary**' helps to add those tokens into feature vector dictionary together with their frequency.

The similar exhaustive approach is used to select which extra features (or the combination) should be added to improve the performance.

Table 2: Top 5 combinations of extra features returning the best performance

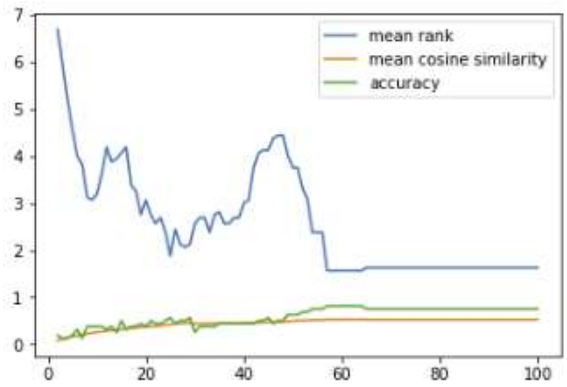
Extra feature combo	Mean rank	Accuracy
Bigrams / 4grams	1.5625	0.8125
Bigrams / Trigrams / 4grams	1.5625	0.8125
Trigrams	1.6250	0.75
Bigrams	1.6875	0.6875
4grams	1.6875	0.75

The extra features which will be added, based on their performance, are: *bigrams and 4grams*

b. Try reducing the number of features

The strategy is to use the maximum document frequency to remove tokens with too high occurrences. Similar exhaustive approach is used to determine the threshold; the value 64 is chosen based on the performance returned.

Graph 1: The chart of performance scores against the maximum document frequency threshold value



3. Question 3;

It can be seen from the heatmap that:

+ There are some cases of high similarities between characters in held-out and training vector but they are not the same person (1)

+ Some cases show low similarity score between the same character in held-out and training set (2)

To investigate, we will print out all the features extracted from character text in each case:
(1) We can take a look at the character 'MAX' in held-out set and 'SEAN' in training one
(2) We can take a look at the character 'RONNIE' in both held-out and training vector

Some insights taken from the observation:

+ In the case of different character in heldout and training vector but returning a high similarity score, there exists some potential trivial words/tokens of high frequency (*'_eol_'/'na'/'oh'/'yeah'//'get'/etc.*)

+ Also, these trivial tokens have been combined into bigrams features as well (like 'oh@eol' or 'eol@yeah'), making their presence even more highlighted, hence hindering the meaningful tokens with lower frequency.

It is sensible to try removing such tokens to see whether it benefit the models. The result of the trial is shown to be super good (mean rank = 1); hence, it could be deemed a good factor to be incorporated in the final model (in question 6)

4. Question 4

The main function which needs to be modified is '**create_character_document_from_dataframe**': the input would be the dataframe all data (character, episode, line and other information). Instead of extracting only character text of each scene, we will also extract other character's line in the same scene, either before or after. The output would be the character text, the text from other characters within the same scene and the scene information.

Accordingly, we also need to modify the '**pre_process**' function to take in the before/after text of other characters as extra feature. Also, we need to consider update the function '**to_feature_vector_dictionary**' to reduce the dimensionality of newly added features – still using the same strategy as question 2: maximum document frequency

5. Question 5

In this question, the '**create_document_matrix_from_corpus**' function needs to be updated to incorporate the TF-IDF transformer. The fitting will still be applied on the training set, neither on held-out nor on test set.

In TF-IDF, there are four hyperparameters: norm, use_idf, smooth_idf and sublinear_tf. There are 16 combinations of the hyperparameters that can be tuned. Still, we will use the exhaustive method to choose the optimal hyperparameter space.

After tuning, there are several optimal sets can be considered:

+ *{norm='l1', use_idf=False, smooth_idf=True, sublinear_tf=True}*

+ *{norm='l1', use_idf=False, smooth_idf=False, sublinear_tf=True}*

+ *{norm='l2', use_idf=False, smooth_idf=True, sublinear_tf=True}*

+ *{norm='l2', use_idf=False, smooth_idf=False, sublinear_tf=True}*

In this part, I will choose *{norm='l2', use_idf=False, smooth_idf=True, sublinear_tf=True}*

6. Question 6

To sum up, there are 4 functions which have been mainly modified/updated to improve the model performance, including: 'create_character_document_from_dataframe', 'pre_process', 'to_feature_vector_dictionary' and 'create_document_matrix_from_corpus'.

The final performance with test set has shown significant improvement, compared to base performance: mean_rank = 1.1875, accuracy = 81.25%.