Part 1: Parse the Corpus

1.

There are four main steps in this part: remove stopwords, stemming words, calculate TFIDF for each word in each document, calculate cosine similarities.

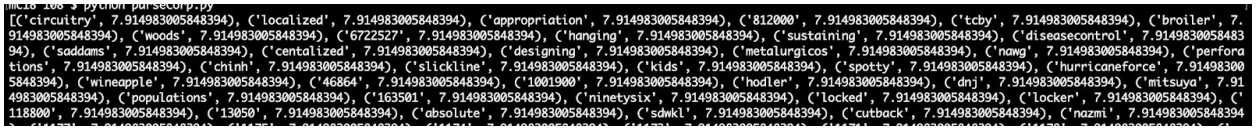
Removing stopwords:

Given a list of documents, find all the unique words and store them in a *bag_of_words*

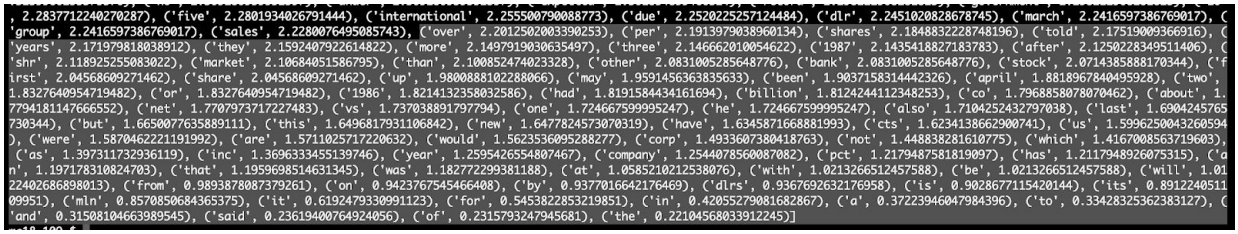
For each word in the *bag_of_words,* count the number of documents that contain such words, denoted as *count*

Calculate the idf using log(N / *count*), where N is the total number of documents in the corpus.

The output shows a large number of words that only occur in one or two documents, which results in a very large idf score (log(N/1) = 7.9). Hence, I chose all the words that have idf larger than 95% of the highest idf score as my stopwords.



Words with low idf (log(N/300) = 2.197) will also be removed because it indicates that this word occurs in most of the documents. Therefore, I chose all words that have idf smaller than 1.05% of the lowest idf score as my stopwords.



In this case, I removed words that only appeared in one document (high idf) and words that appeared in approximately 300 documents (low idf) as my stopwords.

Stemming words:

I used PorterStemmer from nltk package to stem words in each document.

TFIDF:

First of all, I calculated IDF for each word in the same way as above. Get the number of count of each word in each document, find the idf score for such a word. Get TFIDF by multiplying tf with idf, and store the TFIDF score for each word.

Cosine Similarity:
For each document, calculate the cosine similarity with every other document in the corpus (including itself).

2.
Packages used:
BeautifulSoup4 for parsing the sgm file. Using findAll to find the NEWID and BODY
Nltk for tokenize document into a list of words, PorterStermmer for stemming words.

Part 2: Run Complete and Single-Linkage Clustering

I used the dendrogram library from *scipy.cluster.hiearchy* package to plot the graph. We can utilize this to plot both single and complete clusters. Each function returns a matrix z. Z contains four columns, which specifies [cluster, cluster, distance, number of nodes] in each row. I used this matrix z to find all clusters for each document.
Since the first and second column in the matrix z represents the clusters, and each document is also a cluster itself. I put all documents under the same cluster into a list and the key is the corresponding clusterID. In this case, I union two clusters, which already in the dictionary, by grouping all the documents under these two child clusters.
One thing that I need to change is that the highest cluster number in the matrix represents the largest cluster, whereas the smallest cluster number is required to be the largest cluster in this project. I changed the key (clusterID) to meet the requirement.
I did the same thing for complete clustering.
One interesting thing is that documents have much more common top levels clustering in single.txt than in complete.txt. This would result in a higher and thinner single clustering. However, there are less common top level clustering in complete.txt, which means that the dendrogram will be lower and wider. This happens because single clustering uses the nearest point algorithm, whereas the complete clustering uses the farthest point algorithm.

Part 3: Evaluation

1. Number of clusters
If we assume that each cluster only has two children in the dendrogram, then the total number of clusters should be 2N-1, where N is the number of documents in the corpus (each document is a cluster itself).

2. Evaluation Metric Overview
The way I derive the evaluation metric is to utilize the <TOPIC> tag in the corpus. Each document can have one or more topics, and documents with the same topic should be clustered together.

4. Why this is a good measure
I think it is a good measure because topic is a way of clustering documents. If two documents have more common topics, these two documents should have more common clusters.

3. Calculate # of common topics (denoted as common_topic_score)
First, I extracted all the unique topics in the corpurs. There should be one or more documents containing such topic. If two documents are having the same topic, I updated the # of common topics by one each time. I calculated all # of common topics for every two documents by looping through all the unique topic, and this yields a 2D array.

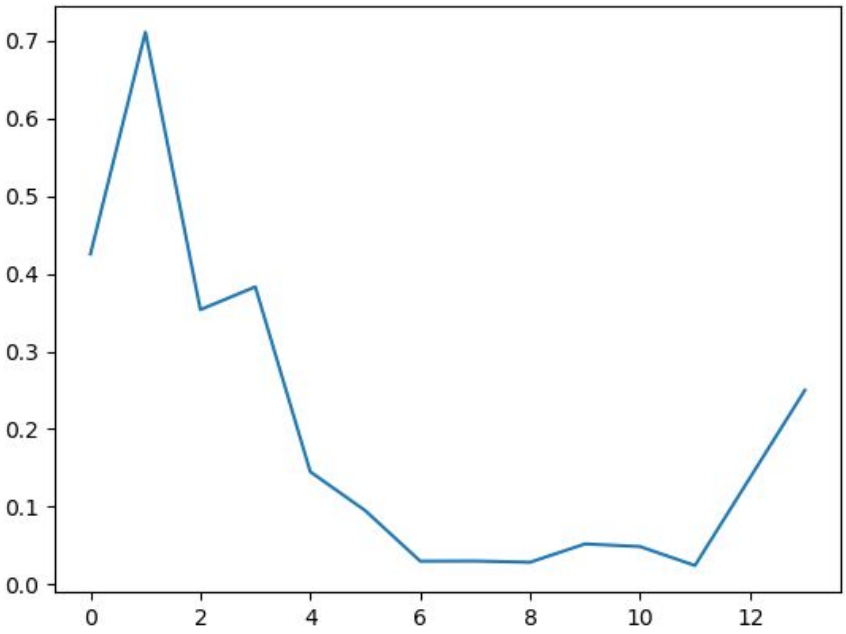
4. Calculate # of common clusters (normalized) (denoted as clustering_score)
Second, for each two documents, union their clusters and get the total number. Additionally, find the number of intersecting clusters of these two documents. Using the number of intersecting divided by the number of union should give us a normalized score.

5. Get the relationship between common_topic_score : clustering_score
In the above steps, we get the number # of common topics and # of common clusters (normalized). However, different # of common topics might have a list of different clustering_score. There might exist different cluster_score for a same common_topics_socre. (e.g: {1: [0.2223, 0.3334, 0.007]}. In this case, I chose to use the median of the list because the median is not affected by outliers.

6.
Single Percentage:
{0: 0.4253968253968254, 1: 0.7110608455189744, 2: 0.35374149659863946, 3: 0.383133637313193, 4: 0.1450653983353151, 5: 0.09512292164007204, 6: 0.030017921146953404, 7: 0.030175337971186107, 8: 0.02857142857142857, 9: 0.0521759697256386, 10: 0.04878048780487805, 11: 0.024390243902439025, 13: 0.25}

Best-Score achieved: 0.7110608455189744

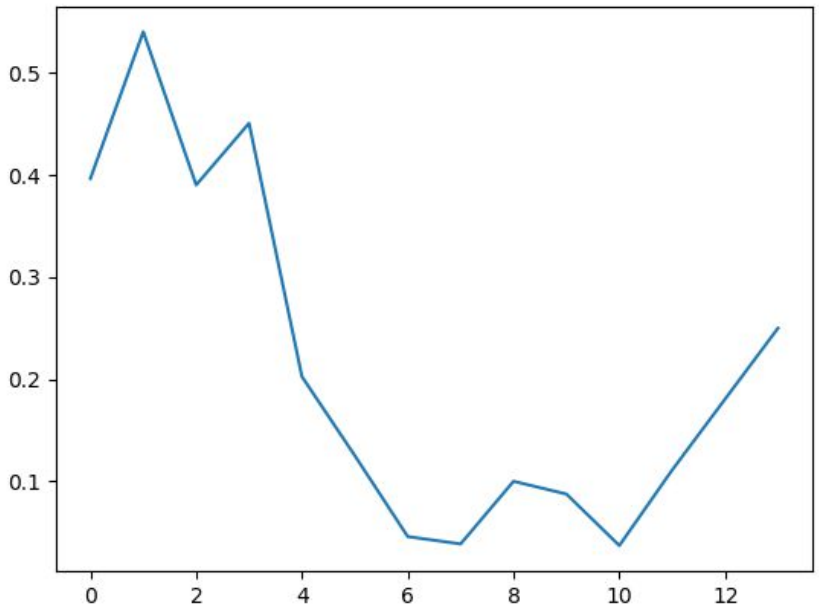
The x-axis is the common_topic_score
The y-axis is clustering_score



Complete Percentage:
{0: 0.3964497041420186, 1: 0.5403225806451613, 2: 0.3902439024390244, 3: 0.450713453698311, 4: 0.20253164556962025, 5: 0.12508218277449046, 6: 0.04583189207886544, 7: 0.0386302294197031, 8: 0.1, 9: 0.0875, 10: 0.037037037037037035, 11: 0.1111111111111111, 13: 0.25}

Best-Score achieved: 0.5403225806451613

The x-axis is the common_topic_score
The y-axis is clustering_score



As we can see above, I think the clustering is not very good. The expected result should be a positive relation between common_topic_score and clustering_score. As the number of commont_topic increases, the number of common clusters should also be increased. However, the graph does not show an explicit positive relationship.

6. Discuss the different in performance between single-linkage and complete-linkage
The single-linkage results in a longer time than complete-linkage does.
I think the reason is that the height of single-linkage is higher than complete-linkage. Just like a binary tree, if the depth is large, it would result in a long time to get to the bottom. On the other hand, complete-linkage has a small depths and it spreads more. Hence, complete-linkage would result a better performance.