IR Assignment 5: Clustering

Ambrose Tuscano
atuscan1@umbc.edu
RE90613

Introduction

In Programming Assignment 5, I have analysed the 504-document HTML corpus by using K Means document clustering. I have used my code from earlier phases of the project for tokenizing, and used sklearn library for calculating normalized term weights and building a term document matrix. I have used Python to code the entire program. To execute the program from a terminal (after setting right permissions for the file), type
$python cluster.py ../input_files ../output_files
The command line parameter is the path to the input and output files directory .

You need to install the NLTK and Sklearn to run the program.

Output

The output is written to the file cluster.txt . For every merge between clusters or documents, a line was written into the file as:

cluster1 with cluster2 clusters to new_cluster with similarity ----|| cosine_score

I have included the first 100 lines of the output in the zip file.
Also, I have attached the centroid file generated.

Implementation

I extended the program from previous phases for implementing document clustering. I followed the approach below.

1) I created the tokens and maintained a documents dictionary for the files as keys with values as the tokens.
2) I used the same loop and created a dictionary for clusters with the name centroid. I referred centroid of each cluster to be the name of the file, my plan was to use the documents dictionary key as the key of the cluster, so It shows which files were in the cluster to form it.
3) Later I used the TfidfVectorizer function from the library sklearn.feature_extraction.text for the Tfidf matrix calculation, this was referred from a blog on temple Universities website(https://sites.temple.edu/tudsc/2017/03/30/measuring-similarity-between-texts-in-python/) and some Stackoverflow answers, which helped regarding building a nested dictionary

structure, where I had used the dot product tfidf matrix and its transpose to calculate the similarity between two files.
Here we got something like:
Similarity Matrix = { “001.html”: {“002.html”:0.00223,“004.html”:0.4214....},.....}
Which can be traversed as:
Similarity Matrix[“001.html”][“002.html”] = 0.00223

4) This similarity matrix is used for the clustering.
5) I choose to do agglomerative clustering , for this i started with creating a new matrix which saved the clusters as they are made so, they could be used later on for the clustering and so that the same documents don't get clustered again with other documents directly rather the centroid gets clustered. This helped reduce redundancy.
6) For clustering documents, I used the maximum similarity score from the similarity matrix implemented using the dictionary structure.
7) Here I checked for the condition that the files being clustered were not already clustered and that they were not the same.
8) Now I saved the clustered files in the clustered set created earlier and I added them in the centroid dictionary.
9) Also, I handled the addition of the new centroid in the similarity matrix by calculating similarity between the new cluster and individual files. The similarity here was found by the Group Average Link method, where the intuition was that the similarity between the clustered files centroid and the centroids would be the average ratio of the similarity between the previous similarity scores to the number of the files being clustered.
10) I checked for the condition as mentioned in the requirements that files with similarity less than 0.4 didn’t have any significant difference while clustering and hence such files clusters were ignored.

Data Structures:
I used Dictionary and nested Dictionary implementations for Similarity matrix creation. Additionally, since a similarity matrix is upper triangular, only one entry was made for a pair of documents.

Similarity Matrix
The Similarity matrix was constructed using the cosine similarity score between pair documents. For Finding the Cosine Similarity matrix, I used the TfIdf Matrix approach, which was referenced from Temple Universities Blog{1}. To calculate the Tfidf Matrix I imported the TfidfVectorizer function from sklearn.feature_extraction.text library.

Cosine Similarity matrix(F1,F2) = ((tfidf_matrix * tfidf_matrix.T).A)

The above implementation gives a 2*2 Matrix, and since the similarity matrix is upper triangle, we calculate the cosine score as

Cosine Score = ((tfidf_matrix * tfidf_matrix.T).A)[0,1]

This Score is assigned as value against a file, which acts as a value to another file as key, in the nested Dictionary form of the Similarity Matrix.

Group Average Link

After forming a new cluster, the Similarity matrix was updated with the scores between the new cluster and existing ones. Group Average Link method was used for computing the similarity score between two clusters or one cluster and another document

D(Cluster1,Cluster2) = ($\sum_{n \text{ in } C1} \sum_{n \text{ in } C2}$ Cosine Similarity between Cluster1 and Cluster2) / Number of files in Cluster1 and Cluster2

Evaluation

1. Which pair of HTML documents is the most similar?
Documents 130.html and 102.html are most similar as they were first to be merged into a cluster.

2. Which pair of documents is the most dissimilar?
Documents 144.html and 111.html are most dissimilar with a similarity score of 0.0 (the first such occurrence was chosen). Also, the condition checking for 0.4 plus similarity was removed to check this part.
With the thresholding checking, The most dissimilar are , 211.html and 188.html

3. Which document is the closest to the corpus centroid?
After document clustering (without the similarity score threshold of 0.4, the centroid of documents belonging to the cluster was found out (the median of magnitude of term weights of documents belonging to cluster). The document with least distance to this centroid is 031.html.
And, with the Threshold of 0.4 is 077.html