

CMSC 676
HOMEWORK 5

SUBMITTED BY:-
Prajna Bhandary
QC21457

To run the program:

Input:

\$ python3 homework5 filename

Output: 1) Top documents with corresponding scores are displayed
2) The most similar document set containing their similarity score and the two documents id
3) The most dissimilar documents containing their similarity score and the two documents id

Approach:

I have written a code for the clustering algorithm based on the simple hierarchical Agglomerative clustering(HAC) using the single link method where two objects are merged if their centroids are closer to each other than are the centroids of any other pair of objects.

Clustering Algorithm

The first step is to create a similarity matrix and the following steps were followed:

- 1) Consider all the files and compare with one another. If the files are the same then we don't consider them.
- 2) We consider all possible pairs of the documents
- 3) The cosine similarity calculates the tfidf matrix for the two documents.
- 4) It first vectorizes the documents to fit the two documents and then calculates the cosine similarity score by multiplying the the tfidf matrix with its transform.
- 5) The similarity scores are stored in the format set (Score,doc1,doc2)

The second step is to run the clustering algorithm.

- 1) We treat each document as a singleton cluster, that is, each cluster has only one document. The matrix at this point is a measure of closeness from every cluster (only a single document right now) to every other cluster.
- 2) We find out the most similar documents, i.e, the pair with the highest similarity score. Collect the document names and the similarity score of those documents.
- 3) If both these documents thus received do have a similarity score then a new cluster is formed and the similarity score is thus updated.
- 4) The merging of the clusters are done using single linkage strategy
- 5) Remove the previous cluster from the dictionary(similar to hashmap in python) and add the new cluster to it.
- 6) We run the algorithm from steps 2-5 as long as the no pair has a similarity scores is less than 0.4.
- 7) Display the clusters.

Note: The clusters are named as the string of the length of the cluster keys in the dictionary and are incremented accordingly for every cluster formed. For example. Since there are around 501 valid documents the cluster names start from 502 and so on as shown in the output section of this report.

Here is a code snippet of the code function:

```
/*
    cname = str(leng_clusters) //naming of the clusters where total clusters is the length of
the total keys in the cluster
    cosine_score, c1, c2 = near_cluster(centroid, doc_vec) //passes the clusters and the
set
    if cosine_score != -1:
        new_cluster = [c1, c2]
        doc_vec.update(new_cluster) //calls the update function to update the new cluster
        clusters[cname] = new_cluster
*/
```

Data Structures used:

Similarity Matrix:

Hashmaps(dictionaries in python which store a key value pair). The key is the name of the cluster(which is string format) and the value is its priority queue ordered in decreasing order of its similarity score.

The cluster is named after the names of the files here.

Here is a code snippet of my code:

```

/*
cos_sim_matrix = {}
for file1 in documents:
    cos_sim_matrix[file1] = {}
    for file2 in documents:
        if file1 == file2:
            break
        cos_sim_matrix[file1][file2] = cosine_similarity(documents[file1], documents[file2])
*/

```

Finding most similar and Dissimilar documents:

The steps followed were as followed:

- 1) Keep a track of the global minimum score and global maximum scores initially and update whenever we come across a minimum score.
- 2) Iterate over the cosine similarity matrix.
- 3) Find the similarity matrix with respect to the current key.
- 4) Compare the score of the documents similarity with the global minimum and maximum scores and update the scores respectively.
- 5) Repeat the steps 2 to 4 until the most similar and dissimilar matrices are found.

Questions:

- 1) Which pair of HTML documents is the most similar?

The most similar pairs the program calculated was
(1.00000000000000018, '420.html', '417.html')

Where 420.html and 417.html are the names of the files that are most similar and 1.00 is the similarity score between the two.

The approach used was:

The similarity score matrix was passed to the function to check for the most similar documents. Similarity scores were compared with respect to all the files in the cosine similarity matrix. The files with the highest score were thus achieved.

- 2) Which pair of documents is the most dissimilar?

The most dissimilar pair that my program calculated was:
(0.0, '439.html', '205.html')

Where 439.html and 205.html are the files that are most dissimilar with a similarity score of 0. We can also note that there are many files with the dissimilarity score as 0.

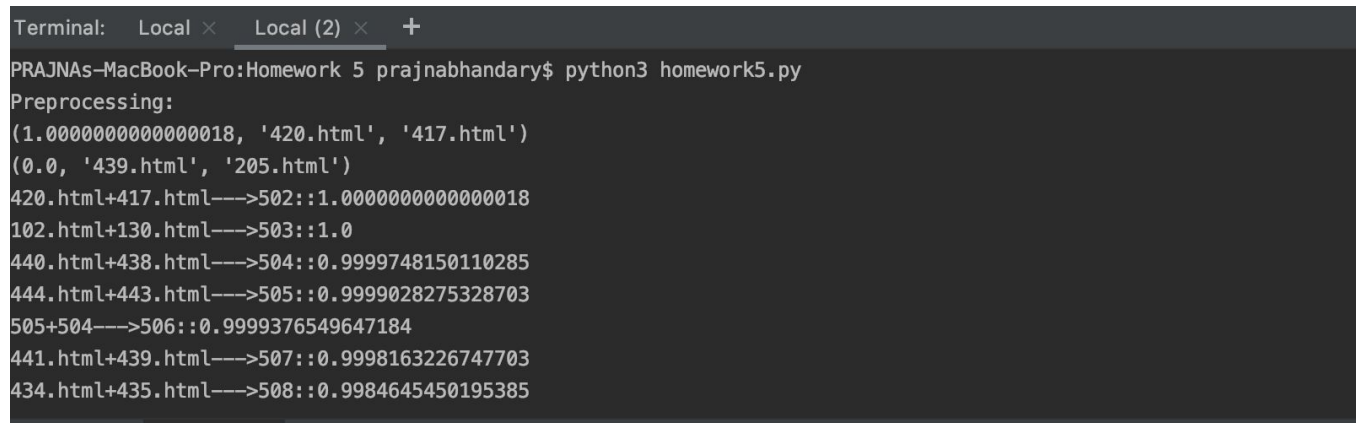
3) Which document is the closest to the corpus centroid?

The document thus found was the file 333.html.

The corpus centroid is a vector that is the average of all document vectors. We consider the average term weight for the different terms across all docs, e.g. for moon, sun, stars, cat. etc.. So for each term, we add up all the term weights (tf*idf) and divide by the total number of documents.

Output format:

A screenshot of the first few lines of the execution of the program is as given below:



```
Terminal: Local × Local (2) × +
PRAJNAs-MacBook-Pro:Homework 5 prajrabhandary$ python3 homework5.py
Preprocessing:
(1.000000000000000018, '420.html', '417.html')
(0.0, '439.html', '205.html')
420.html+417.html--->502::1.000000000000000018
102.html+130.html--->503::1.0
440.html+438.html--->504::0.9999748150110285
444.html+443.html--->505::0.9999028275328703
505+504--->506::0.9999376549647184
441.html+439.html--->507::0.9998163226747703
434.html+435.html--->508::0.9984645450195385
```

The first two lines are the output for the most similar and dissimilar documents respectively as explained above. The next lines show the clusters thus formed in a descending order.

To better explain the output consider the following first few lines of the output:

420.html+417.html--->502::1.000000000000000018 //cluster 1

```
102.html+130.html--->503::1.0
440.html+438.html--->504::0.9999748150110285
444.html+443.html--->505::0.9999028275328703
505+504--->506::0.9999376549647184 // merge of two clusters which have similarity
score greater than 0.4
441.html+439.html--->507::0.9998163226747703
434.html+435.html--->508::0.9984645450195385
436.html+433.html--->509::0.9950385963880606
509+508--->510::0.997704654799924
421.html+423.html--->511::0.9926127767342392
416.html+418.html--->512::0.9870090275591261
405.html+403.html--->513::0.9850086870853417
```

Two documents are compared to find the similarity scores and are printed with their cluster name and the similarity score between the two documents.

As we can observe above the clusters 505 and 504 are merged to form a new cluster with a different similarity score as mentioned in the comment.