REPORT

Assignment 2b

-Shreshth Saxena (38)
Thongminlien Kuki (46)

Objective:

Create Tf-idf matrix of the collection.
Using Cosine distance, create a similarity matrix.
Cluster the documents using K means clustering, and find the number of clusters (k) that minimizes SSE.
Apply hierarchical clustering. Cut the dendrogram at k and identify clusters of similar documents.

Packages:

NLTK (The Natural Language Toolkit): for text processing like tokenization, stemming, tagging and parsing.
scipy: for clustering
sklearn: for tfidf and cosine similarity matrices' calculation
numpy: for Scientific Computing.
matplotlib and seaborn: for plotting

Tf-idf Matrix:

<22x1610 sparse matrix of type '<class 'numpy.float64'>' with 6062 stored elements in Compressed Sparse Row format>

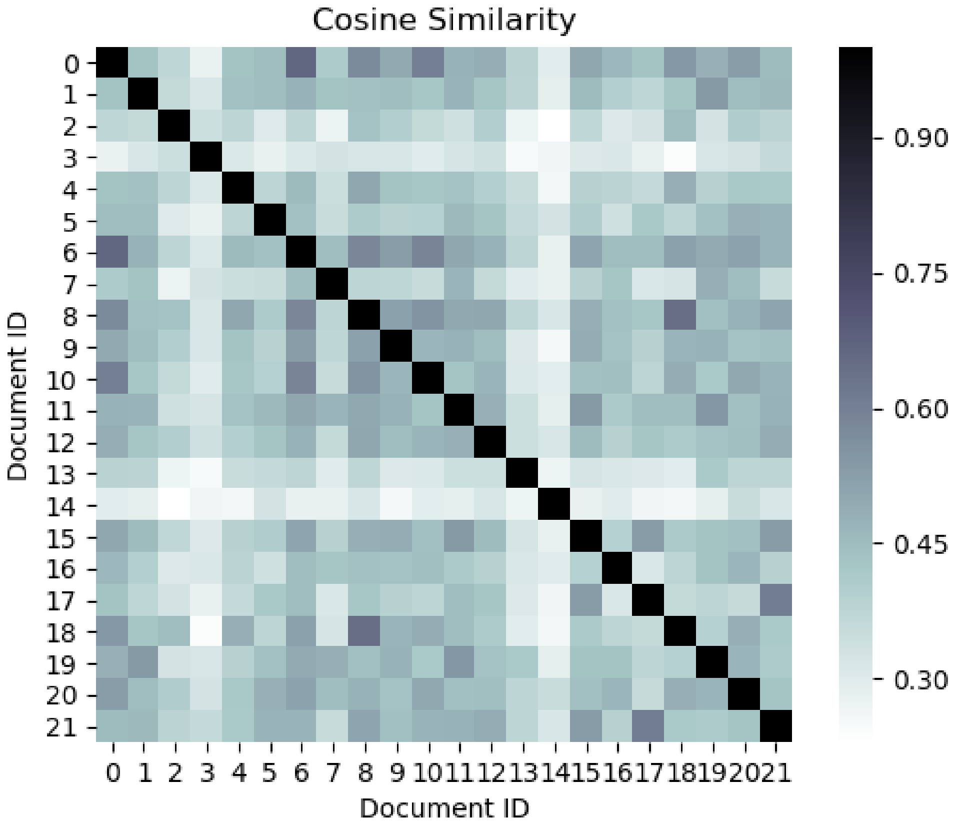
```
>>> print(tfs.todense())

[[0.02962249 0.03192686 0.03192686 ... 0.         0.         0.        ]

 [0.         0.         0.         ... 0.         0.07801384 0.        ]

 [0.         0.06074473 0.         ... 0.         0.         0.08277901]

 ...

 [0.         0.06132381 0.         ... 0.         0.         0.        ]

 [0.         0.         0.03739821 ... 0.         0.         0.        ]

 [0.0264073  0.02846155 0.         ... 0.11635671 0.03097575 0.        ]]
```

COSINE Similarity Matrix:

>>> cosim.shape

(22, 22) >>> cosim

Heatmap:



K-Means Clustering:

The K-Means algorithm was iterated multiple times with different values of k (number of clusters) to compare the SSE (Sum of Squared Errors) in each case and find the optimal value of k. The SSE for K-Means has an obvious decrement with increment in the number of clusters(k).

K-Means clustering was performed in two ways:

a) Using tf-idf matrix

```
dense_tfs = tfs.toarray()
KM = KMeans(n_clusters=i, n_init=50, max_iter=100)
KM.fit_transform(dense_tfs)
```

b) Calculating EigenVectors and EigenValues from the Cosine Similarity matrix and using n EigenVectors corresponding to the topmost n EigenValues to cluster the n-dimensional document vectors.

```
eigen_values, eigen_vectors = np.linalg.eigh(cosim)
km = KMeans(n_clusters=i, init='k-means++')
km.fit_predict(eigen_vectors[:, -4:])
```

Observations:

Method 'b' gave significantly low SSEs for any comparable value of k (Possibly because of the reduced dimensions)

for k=10

a) >>> KM.inertia_

5.763567883967067

>>> KM.labels_

array([7, 5, 8, 0, 3, 2, 7, 5, 3, 3, 7, 5, 3, 9, 6, 1, 4, 1, 3, 5, 2, 1])

b) >>> km.inertia_

0.0077368636560810255

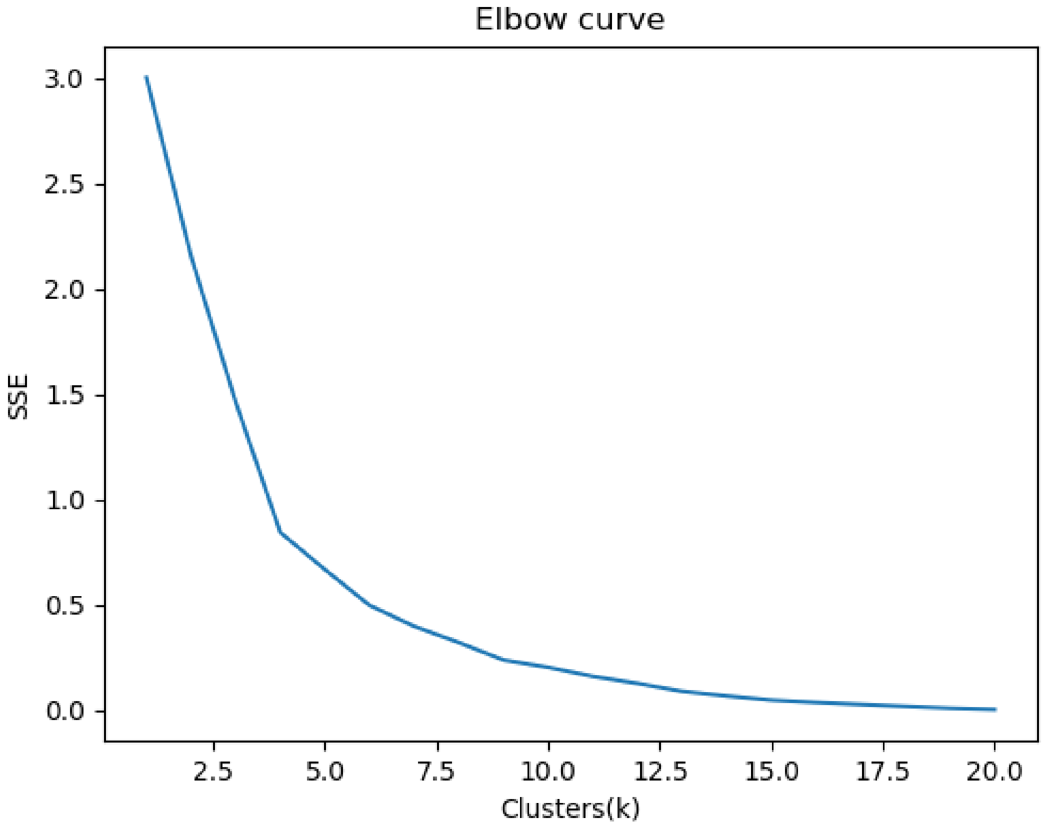
>>> km.labels_

array([7, 5, 1, 2, 8, 9, 7, 2, 3, 8, 3, 0, 4, 9, 2, 4, 4, 0, 6, 9, 4, 0])

The Elbow Method is a simple, but effective, way to tune the k parameter. If the line chart looks like an arm, then the "elbow" on the arm is the value of k that is the best.

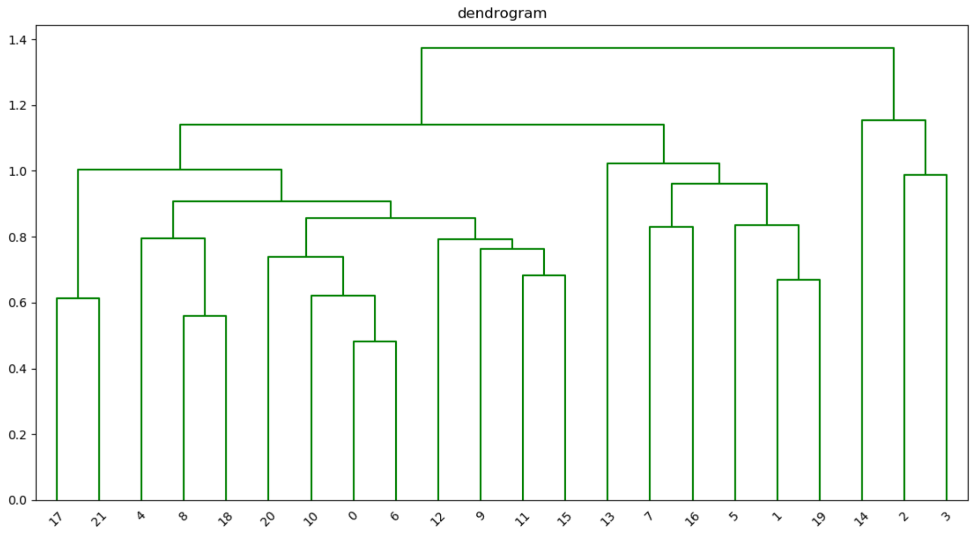
Method 'a' gave a rather smooth SSE curve and the SSE curve for method 'b' is shown below with an elbow near k=5.

Elbow Curve



HEIRARICHAL CLUSTERING

Dendrogram



Cluster Labels for files:

```
>>> cluster.labels_

array([0, 0, 1, 1, 0, 4, 0, 0, 0, 0, 0, 0, 4, 3, 2, 4, 0, 4, 0, 0, 0, 4],
      dtype=int64)
```

File	Cluster
ass1-1019.txt	0
ass1-1037.txt	0
ass1-1046.txt	1
ass1-1138.txt	1
ass1-1147.txt	0
ass1-202.txt	4
ass1-211.txt	0
ass1-321.txt	0
ass1-440.txt	0
ass1-505.txt	0
ass1-532.txt	0
ass1-541.txt	0
ass1-606.txt	4
ass1-743.txt	3
ass1-817.txt	2
ass1-826.txt	4
ass1-909.txt	0
ass1_1349.txt	4
ass1_422.txt	0
ass1_734.txt	0
ass1_808.txt	0
ass1_936.txt	4

Observations:

Takes more time than K-Means
More informative than K-Means
Number of clusters need not be specified