K-Means Clustering With Jaccard Distance

Date: March 31, 2024

Siddhesh Maheshwari
(MDS202347)

1 Task

There are three different "Bag of Words" datasets: Enron emails, NIPS blog entries, and KOS blog entries. The task is to cluster the documents in these datasets using K-means clustering for different values of K and determine an optimum value of K. The similarity measure, however, should be the Jaccard Distance instead of conventional Euclidean Distance.

1.1 The Dataset

- The "Enron emails" dataset contains 3710420 entries with 39861 unique documents and 28102 unique words.
- The "NIPS blog entries" dataset contains 746316 entries with 1500 unique documents and 12419 unique words.
- The "KOS blog entries" dataset contains 353160 entries with 3430 unique documents and 6906 unique words.

1.2 Jaccard Distance

The Jaccard index, also known as the Jaccard similarity coefficient, is a statistic used for gauging the similarity and diversity of sample sets. It is generally calculated as the Intersection over Union.

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|}$$

For our task, we will use the Jaccard Disatnce in place of Euclidean Distance, which is given as follows:

$$d_J(A,B) = 1 - J(A,B) = \frac{|A \cup B| - |A \cap B|}{|A \cup B|}$$

1.3 Methodology

- Read the text file into python and converted it into a dictionary of dictionaries, where the keys of the outer dictionary are the unique document IDs and their values are dictionaries, whose keys are unique word IDs and its values are Boolean indicating whether the word is present in the document or not.

{ docID:{ WordID : True,
WordID :True, ... }, ... }

- Self implemented the K-Means and K-means++ algorithm using the Jaccard Distance as the distance metric.

– **Centroid using Jaccard Distance**:

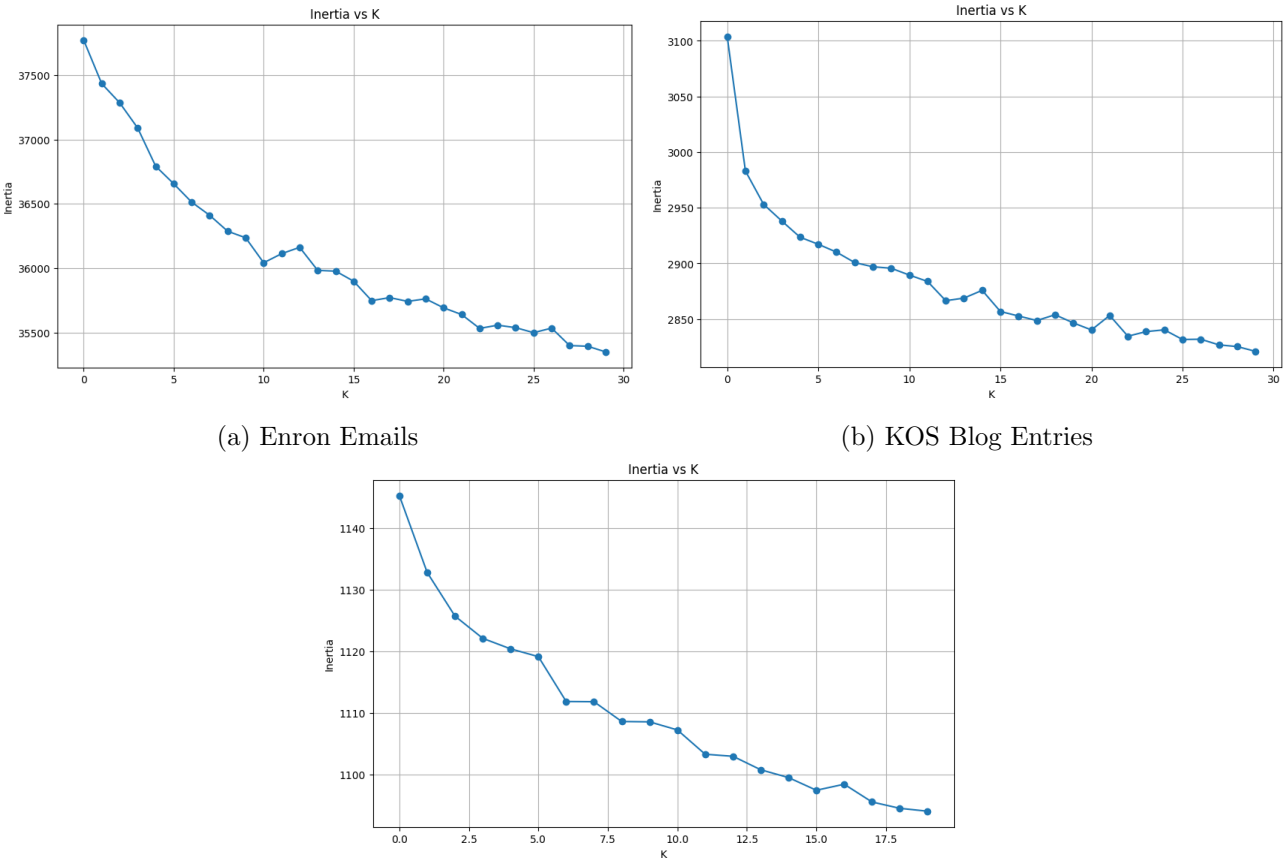
* In K-means, we randomly choose K points to be the initial centroids.
* However, in K-means++, for choosing the initial centroid, we take the first document to be a centroid and find those K documents (centroids), which are at least 0.8 (threshold) distance away from each other and the first centroid.
* Then, we cluster the remaining documents as per their distance to the nearest centroid, using Jaccard distance.
* For calculating new centroids, we calculate the count of each unique word that appears in the documents of a particular cluster. then, we calculate the average length of all the documents in a cluster (Let this be n).
* Then we take the first n words when we arrange the words in decreasing order of their counts.

– **Termination Condition**:
We calculated the inertia after every iteration. We observed that the inertia values were fluctuating too much. So we ran 50 iterations for every value of K and considered the minimum value of inertia that we obtained.

- Calculated Inertia for different values of K (from 1 to 25).
- Plot the values of inertia against the different values of K to find the "elbow" point for optimum value K.

1.4 Results

Below are the plots of Inertia vs K for different values for K for the three datasets.

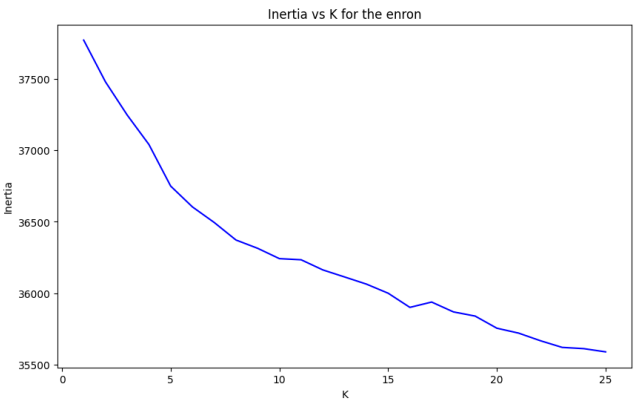


(a) Enron Emails

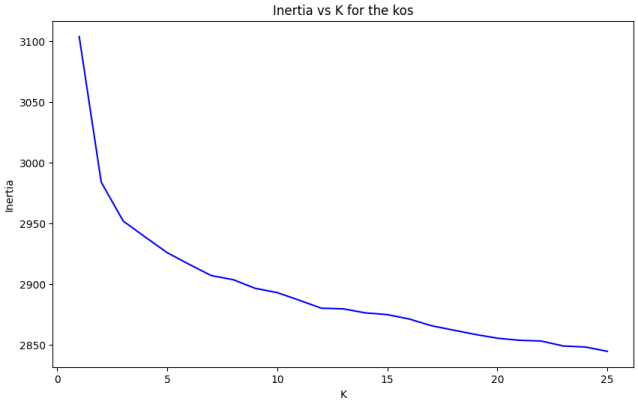
(b) KOS Blog Entries

(c) NIPS Blog Entries

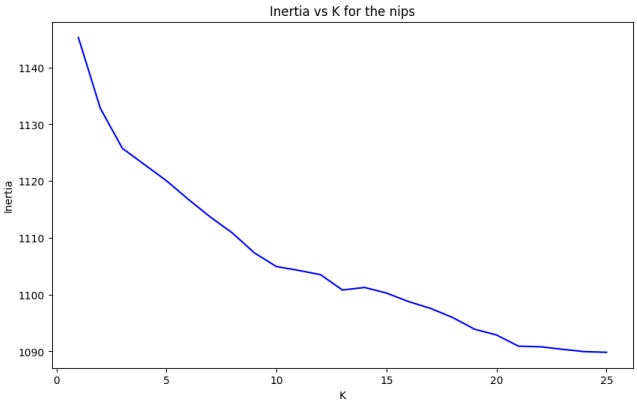
Fig. 1: Inertia vs K plots. for K-Means Clustering



(a) Enron Emails



(b) KOS Blog Entries



(c) NIPS Blog Entries

Fig. 2: Inertia vs K plots. for K-Means++ Clustering

2 Conclusions

- From the graph, we can see that for the K-means algorithms our optimum values of K turns out to be 12, 10, 8 for Enron emails, KOS, and NIPS respectively. And, for the K-means++ algorithms, the optimum values of K are 8, 5, 10 for Enron emails, KOS, and NIPS respectively.
- The time taken to run this program is
- For every value of K, we are loading only K keys and K their corresponding values from the main dictionary into another dictionary. The only extra space that is consumed by the program is the space taken by this new dictionary.

3 External Libraries

- **pandas**: For creating and managing data-frames.
- **numpy**: For all arrays and matrix related operations.
- **matplotlib**: For plotting and visualization.
- **time**: For calculating the running time.