CSE 5243: Introduction to Data Mining

Assignment 4

Vaibhav Devekar (devekar.1)
Akshay Nikam (nikam.5)

Goal of the Assignment

This assignment aims at demonstrating various clustering techniques on the Reuters document dataset based on the feature vectors generated in Assignment 1.

Distance Metrics:

We attempted running our algorithms with two different distance metrics:

1. **Cosine distance**
   Cosine similarity between two equal dimensional vectors is the cosine of the angle between the two vectors. Smaller is the angle, larger is the cosine similarity. Cosine distance is defined as:
   Cosine distance = 1 – cosine similarity
   Where Cosine similarity between vector A and B is calculated as:

$$\text{similarity} = \cos(\theta) = \frac{A \cdot B}{\|A\|\|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n} (A_i)^2} \times \sqrt{\sum_{i=1}^{n} (B_i)^2}}$$

2. **Euclidean distance**
   Euclidean distance is the actual point-to-point Cartesian distance between two points in a vector space. For an n-dimensional space, Euclidean distance (d) between points p and q is calculated as:

$$\mathrm{d}(\mathbf{p},\mathbf{q}) = \mathrm{d}(\mathbf{q},\mathbf{p}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \cdots + (q_n - p_n)^2} = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}.$$

These metrics were used to compute distance between a pair of articles based on the feature vector generated.

K-means clustering

This is a convergence based algorithm that works in two steps per iteration. The algorithm expects the number of clusters K as an input. We start with K random mean points, one mean for each cluster. The means are represented as a vector of the same dimensionality as the feature vector of the articles. Each article acts as a point for this algorithm.

In iteration, first step is to find the closest cluster mean for each point and assign the point to that cluster. Each cluster gets a set of points nearby its mean. In the second step, all cluster means are recomputed. This is straightforward by taking a mean of each dimension and forming a mean vector.

The clustering algorithm converges when in two successive iterations no points or only a small number of points are reassigned to different clusters. This can also be represented in terms of change in the cluster means. For the computed data matrix from assignment 1, our implementation of K-means converges in less than 40 iterations.

Results and Performance:

For results and performance computation, we considered topics and place labels of each article in each cluster and used following metrics to evaluate performance and the quality of clustering. For evaluation with entropy, we only considered the labeled data even though all points were used for clustering.

Entropy:

For finding out how the algorithm performed, we used entropy as the performance metric. It is a measure of uncertainty and depicts the number of bits required to represent a point in the given data.

Computing entropy:

Entropy is first individually computed for each cluster and then a weighted average is taken based on the number of points in the clusters. Since each article has multiple topics and places, we gave each label of the article an equal weight, such that the total weight of all topics and place labels for an article is always 1. For example, if an article has 2 topics and 1 place, each are given 1/3 weight. Now, each topic's/place's weight is summed over all the records in that cluster. Once we get a vector of total weights of all class labels, we normalize it to find probabilities of each topic/place. Then entropy for the cluster is computed as:

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log_b p(x_i)$$

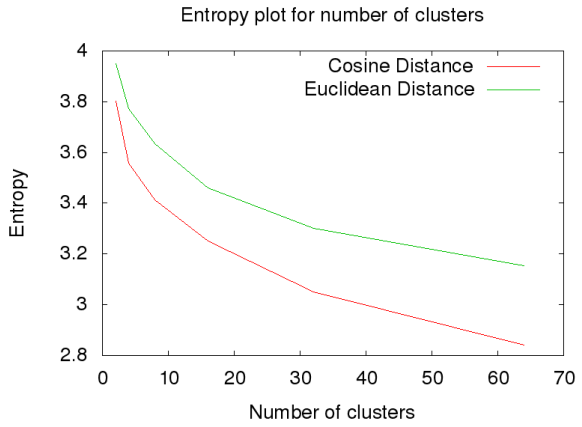
for n class values and b=2.

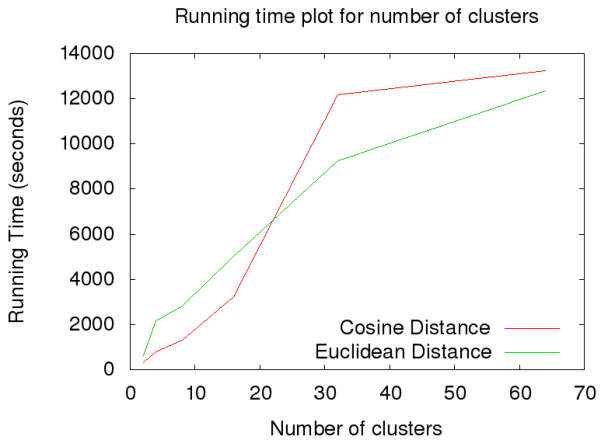
We found that as the number of clusters increase, entropy gradually decreases, meaning a better quality of separation of articles.

For K-means clustering algorithm, we found following entropies:

# Clusters	Entropy for Cosine Distance	Entropy for Euclidean Distance
2	3.80375332771	3.94993482586
4	3.55697535925	3.77195219452
8	3.41406922379	3.63348425172
16	3.25008232272	3.46016354507
32	3.05113063415	3.29978197132
64	2.84177028704	3.23467349883

According to the following plot, it can be observed that Cosine distance produces lower entropy and hence is a better distance metric for clustering given data.



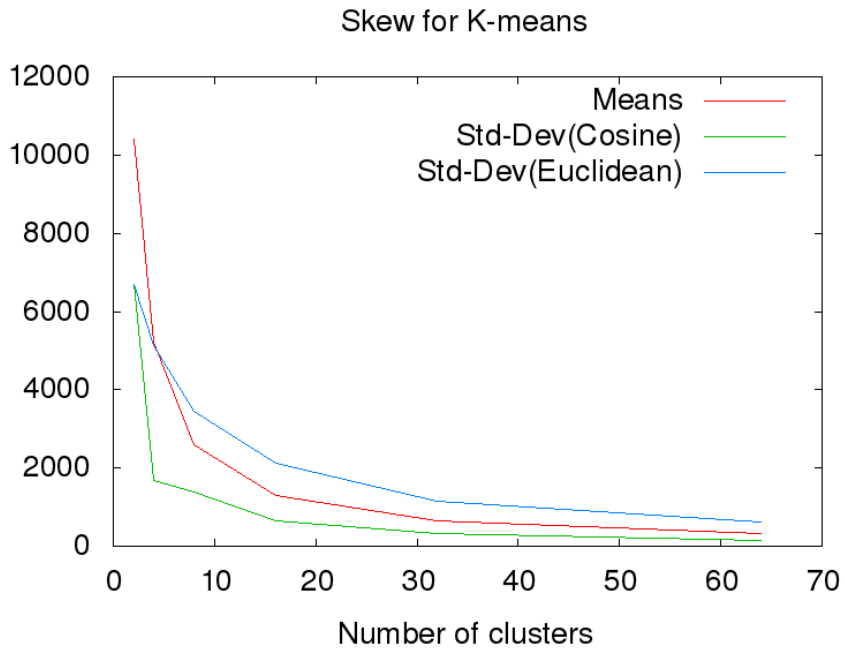


Scalability:

The running time is roughly directly proportional to the number of clusters K provided as input. We took results for K=2, 4, 8, 16, 32, 64. It is found that running time is directly proportional to the number of clusters and the number of iterations required to converge. The above plot shows the results for the same.

Skew:

We also determined the quality of clustering based on the skew in the resulting clusters. Ideally, all clusters are of the same size. But skew is more if some clusters are very small while some are very large. Standard deviation of cluster sizes can be a good measure to determine skew. Lower the standard deviation, lower is the skew in the clustering. We found that as we increase the number of clusters, skew decreases. Following plots show the mean and standard deviation to represent the degree of skewness for the algorithms we implemented.



For K-means, again we can note that cosine metric works better as it has less standard deviation, meaning less skew in the clustering.

Hierarchical Clustering

The algorithm was first implemented in python for which the running time proved expensive. Consequently, the clustering was implemented in 4 stages in python and C++:

1. Preprocessing data matrix(Python)
   The data matrix from Assignment 1 contains term counts for each document. The matrix is converted to space-separated format and can be stored in two ways:
   a. Using the actual values i.e. term count
   b. Converting the counts to 1/0

   Also instead of including all rows, only labeled data are included.

2. Distance Matrix(C++)
   The data matrix output by stage 1 is read and used to compute distance matrix using Cosine or Euclidean distance metric. Only the lower triangular part is calculated and also excludes the diagonal elements. The matrix is written in binary format to a file.

3. Hierarchical Clustering(C++)
   The distance matrix from stage 2 is read and the clusters are computed. The results are written to files in the form of article numbers for various cluster sizes.

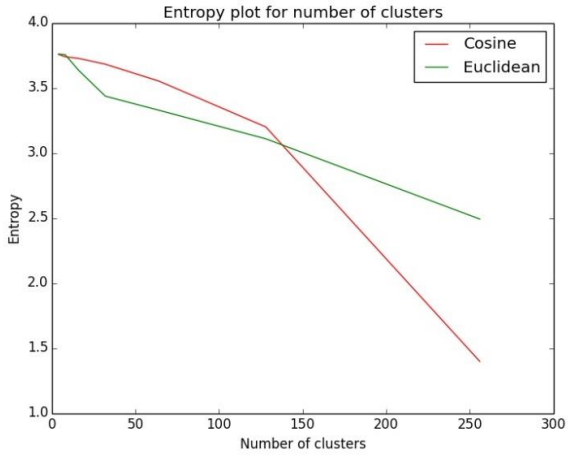
4. Compute Entropy and Display Results(Python)
   The files produced in stage 3 are read and their entropy is calculated. The graph showing variation of entropy against cluster sizes is produced accordingly.

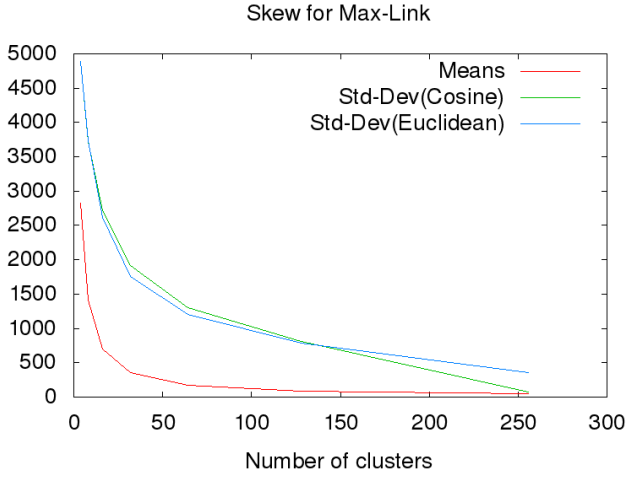
Computing the distances for the entire dataset (20841 articles) takes a large amount of time. Hence we decided to include only topic-labeled articles (11305 articles). Accordingly stage 1 generates only labeled data matrix.

Entropy:

The clusters are skewed with most articles (70%) in a few clusters and the remaining distributed among the other clusters. A sample result and comparison is provided in 'CLUTO' section below.

# Clusters	Entropy for Cosine Distance	Entropy for Euclidean Distance
4	3.75693066406	3.75960564751
8	3.73908154017	3.75483842574
16	3.7263998765	3.6347700202
32	3.6818774862	3.4361708798
64	3.55229254978	3.32822322466
128	3.19986385296	3.10870895911
256	1.39849314748	2.49199695245





We observed that the cosine metric produces lower entropy when the number of clusters is high.

Scalability:

Stage	Time in seconds
1	26.0988128689
2 – Cosine	475
2 – Euclidean	754.48
3	2747.63
4	29.0881068614

Since stage 3 works directly on distances, the choice of metric does not affect its running time. Computation time for cosine distances is less than that of Euclidean since it uses magnitudes which can be pre-computed and reused. Computation time is considerably reduced since only 11305 features are used instead of entire dataset.

Other Approaches

MIN-LINK: We tried two methods of Hierarchical Clustering: Min Link and Max Link. Since Min Link clustering resulted in highly skewed clusters (most points in one cluster and others one in each cluster), we decided to go with Max Link Clustering.

DBSCAN: We also implemented DBSCAN algorithm. However, it again resulted in highly skewed clusters for Cosine distances. For the Euclidean distances which can range from 0 to as large as 244, estimating the optimal EPSILON and MINPTS proved to be difficult. Nevertheless, we found the optimal clustering at EPSILON=0.45 and MINPTS=4. We have provided the code for this approach in the submission.

Comparison with CLUTO:

To verify our implementation of Max Link clustering, we also tried off-the-shelf software CLUTO with clink(max-link) option. Only Cosine metric was tested as Euclidean was not available in CLUTO.

Preprocessing: Stage 1 from our implementation was used.

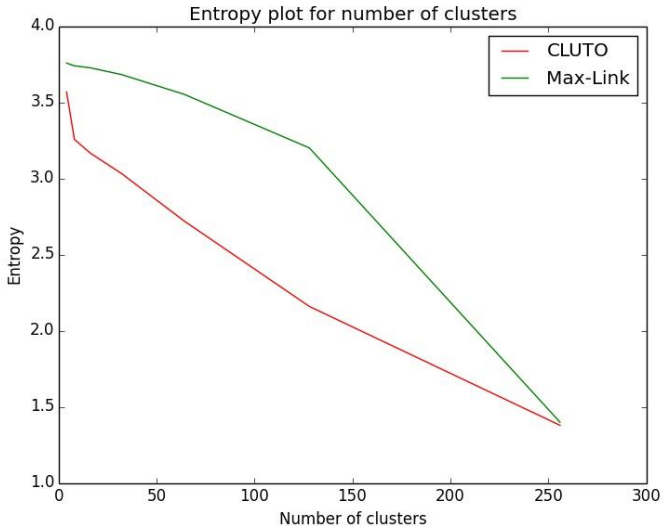
Post-processing: Stage 4 with slight modification was used to display results and entropy.

The individual cluster sizes for 16 clusters are given below:

Max-Link: [10823, 108, 206, 92, 8, 3, 23, 1, 12, 14, 1, 10, 1, 1, 1, 1]

CLUTO: [7567, 1428, 32, 68, 71, 9, 78, 92, 20, 99, 223, 1387, 23, 27, 119, 62]

Though the CLUTO clusters are better balanced, it still suffers from skew in sizes as in our code.



Individual Contributions

Akshay implemented the K-mean clustering algorithm while Vaibhav implemented Max-link hierarchical clustering and also implemented and tried Min-Link and DBSCAN. Akshay and Vaibhav worked together on CLUTO.

REFERENCES: http://glaros.dtc.umn.edu/gkhome/views/cluto