Data mining Report - Clustering

Clustering:

For this exercise we have chosen the following clustering methods.

a. KMeans clustering using Euclidean distance.
b. KMeans clustering using jaccard similarity.
c. Agglomerative clustering using cosine
d. Agglomerative clustering using Euclidean
e. Agglomerative clustering using manhattan

Of these, KMeans using jaccard is implemented from scratch. Other four are done using Scikit learn module. KMeans using euclidean is as well implemented but performance wise scikit was superior and hence used.

Number of clusters to be formed is fixed at 8. But it can easily changed in the code. Entropy of each cluster is calculated in an optimistic way, where in an article with multiple topics is considered a candidate for each of the class it represents. Since to apply entropy we need class labels, articles only with topics are considered.

Entropy is calculated considering all the labels present in that cluster.

Variance is calculated to see the difference in distribution of articles in different clusters. Agglomerative with cosine has given a good result, followed by KMeans using Euclidean.

KMeans using jaccard: The initial mean points are chosen randomly and the result of clustering depends on the initial points. While developing we have seen for some initial points the clustering is very bad. But on an average run, the clustering looks good and even for random picks. After the initial pick, each element is put into different lists based on the proximity to the centre. Then a new point is chosen from this list, which is closest or most representative of the batch as new medians and the process repeats until the medians converge or 10 times.

Results:

The statistics are printed on the console. Following is an example output.

<u>For ten SGM files:</u>

##################################

KMeans clustering, Euclidean

('**Time to cluster** ', 63.702091932296753)

Number of elements in each cluster

[3132, 106, 823, 100, 99, 494, 125, 329]

Entropy for each cluster

[4.8349746144136665, 7.3475148030780835, 0.025287139053555521, 4.1967534917402256, 1.9680946116241791, 4.0757001041238041, 4.0870003835100803, 6.5712478467971085]

Variance
936910.5
/home/4/ruan/local/lib/
  X = self._check_test_data(X)
################################
Agglomerative clustering, cosine
('**Time to cluster** ', 663.68481016159058)
Number of elements in each cluster
[3228, 71, 163, 77, 393, 1073, 199, 4]
Entropy for each cluster
[5.6012035785394714, 0.82573573407781575, 6.8950406955098211, 3.4103538469696932, 0.0, 0.0, 5.6520541685546384, 7.0]
Variance
1051563.75
################################
Agglomerative clustering, manhattan
('**Time to cluster** ', 710.28125596046448)
Number of elements in each cluster
[98, 17, 33, 2, 6, 38, 4978, 36]
Entropy for each cluster
[5.6180455178579765, 2.7616509120768926, 4.3016068001842154, 0.0, 4.9866140897573334, 1.4616669339724064, 4.6708144104317473, 1.5946299249026423]
Variance
2675479.75
################################
Agglomerative clustering, Euclidean
('Time to cluster ', 616.97443199157715)
Number of elements in each cluster
[524, 978, 89, 619, 165, 48, 106, 2679]
Entropy for each cluster
[6.9896328210191401, 0.0, 4.8478043392207493, 3.429259521043845, 5.792926041998574, 3.3548722373413726, 0.0, 4.8999464572531828]
Variance
681192.5
################################
KMeans clustering, Jacard
[[709], [4405], [44], [4210], [4421], [3034], [5120], [3409]]
('Time to cluster ', 3647.7688150405884)
Number of elements in each cluster
[505, 716, 542, 1502, 146, 520, 510, 759]
Entropy for each cluster
[4.0573771890644386, 4.9519080078515367, 4.8783769326096884, 5.0743274648522458, 3.8490841774449729, 4.4865643891088123, 5.30219755966026, 4.0924229123064437]

Variance
133168.25

Performance varies as the number of articles in the corpus. For KMeans if the distance matrix is precomputed the performance is better but requires more memory to save the matrix. The performance of the clustering algorithm coded from scratch(KMeans clustering using jaccard similarity) pales when compared to suites, as the corpus size is increased.

Agglomerative cosine and Kmeans euclidean shows good entropy within each cluster. But as observed, for Kmeans the initial centroids are crucial.

Withnin K means, the Jaccard metric resulted in lower variance than the euclidean metric, but since we implemented from scratch, it took much longer. But entropy of euclidean was better than entropy of jaccard.

Agglomerative clustering had lower variance than K means for the same distance metric. Kmeans had lower entropy values for the same distance metric.

However agglomerative clustering was much slower than K means, so when choosing a clustering mecahnism one must consider the tradefoffs between performance and time taken to cluster.

Within agglomerative clusering, manahattan had good entropy values but the worst variance or skew. Cosine had worse variance(higher) than euclidean but better entropy values than euclidean.

Ouput file with the results of all 20 sgm files:

/home/2/tantri/data_mining_lab4_output