Shashank Agarwal
agarwal.202@osu.edu

Anurag Kalra
kalra.25@osu.edu

Lab report 3

Objective

- **Implement two clustering algorithm:**
  - **k-means clustering -** implemented this clustering algorithm using two proximity measures namely Euclidean distances and Manhattan distance measures.
  - **DBSCAN -** implemented this density based clustering algorithm and report the result for varying the values of epsilon and min number of points.
- **Performance Evaluation -** To evaluate the scalability of the clustering algorithm in terms of time the algorithm takes to cluster the entire data set and also evaluate the quality in terms of topic entropy.

The data matrix which we generated in the first lab assignment is used as the input data for the clustering algorithm.

Procedure:

- We used the feature vector we created in the first assignment and created a distance matrix based on the feature vector.
- We Created two distance matrices for ‘manhattan’ and ‘euclidean’ distances.
- We then calculate coordinates of every document using distance matrix and represent them in the form of a graph
- The graph was used as clustering for different methods in k-means and DBScan

Creating the distance matrix

We used the feature vector from assignment 1 and created a distance matrix. We used Multidimensional scaling (MDS) for this. MDS is defined on Wikipedia as:

> *“Multidimensional scaling (MDS) is a means of visualizing the level of similarity of individual cases of a dataset. It refers to a set of related ordination techniques used in information visualization, in particular to display the information contained in a distance matrix.”*(sic)

This can be done with several manifold embeddings provided by scikit-learn. We use ‘sci-kit’s class ‘sklearn.neighbors.DistanceMetric’ to create this distance matrix. The method DistanceMetric.pairwise_distances gave a distance matrix of with dimension of [14302,14302] as only 14302 documents have topics.

Calculating coordinates of every document

We tried doing clustering using the distance matrix at first but ‘sci-kit’ library would convert these in some kind of graph form and then perform clustering. To speed up the clustering process we calculated the coordinates of every document using the distance matrix.
For this we used sci-kit’s ‘sklearn.manifold.MDS’ class. Using the graph coordinates made the

clustering process very fast with multifold improvements.

But we had issues in fitting everything into memory and our personal laptops only supported '8GB' of RAM. After many tries we finally used AWS's servers and deployed a server with very large RAM (over 60 GB) and converted the distance matrix to graph coordinates.

K-Means Clustering

We used sci-kit's Kmeans library to perform the clustering. For experimentation and to find the best approach we use two libraries:

- k-Means
- MiniBatchKMeans

The MiniBatchKMeans is faster, but gives slightly different results. The MiniBatchKMeans is a variant of the KMeans algorithm which uses mini-batches to reduce the computation time, while still attempting to optimise the same objective function. Mini-batches are subsets of the input data, randomly sampled in each training iteration. These mini-batches drastically reduce the amount of computation required to converge to a local solution. In contrast to other algorithms that reduce the convergence time of k-means, mini-batch k-means produces results that are generally only slightly worse than the standard algorithm.

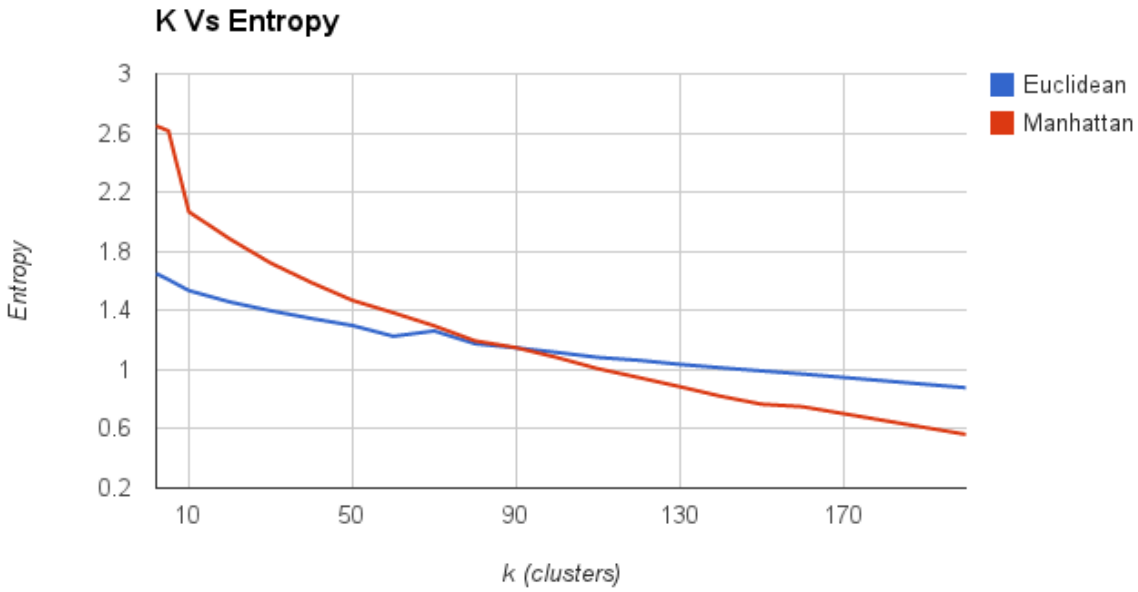
The Entropy for different values for k are given below for both the methods:

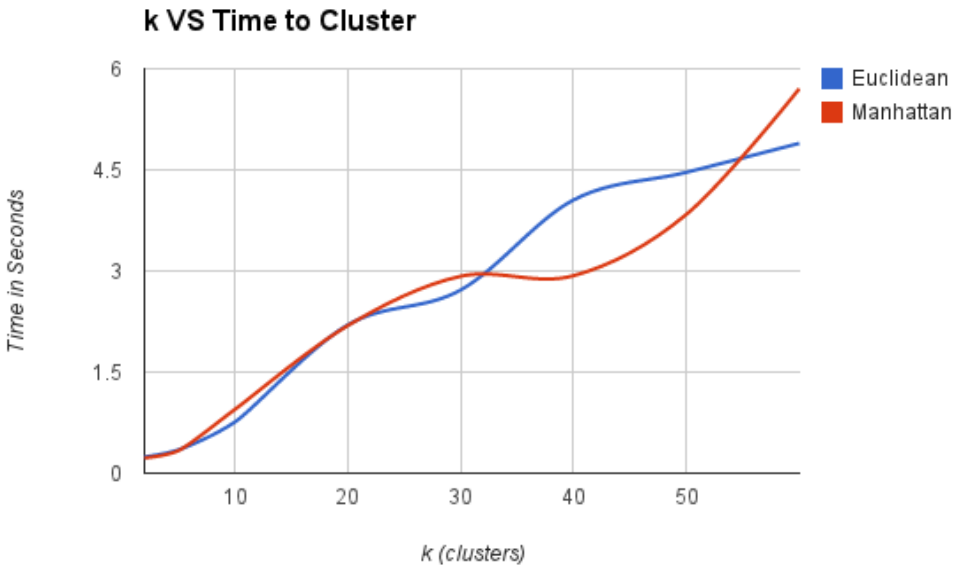
k	Euclidean	Euclidean-Time	Manhattan	Manhattan-Time
2	1.650681557	0.2408881187	2.648805243	0.2270829678
5	1.609127436	0.3456389904	2.614581434	0.3339781761
10	1.534981988	0.7575371265	2.067495442	0.9440288544
20	1.458091459	2.198408842	1.884474621	2.188814878
30	1.398747564	2.717149973	1.721301627	2.921609879
40	1.346354749	4.053508997	1.5887903	2.929509163
50	1.298916257	4.463973999	1.468958647	3.839911938
60	1.226328677	4.894320011	1.385128083	5.705673933
70	1.262177014	4.998895884	1.296788484	6.853452921
80	1.175609267	6.850250006	1.195097433	6.44620204
90	1.148389276	7.032061103	1.149359392	7.503811836
100	1.117126873	7.338573933	1.082926535	8.806655884
110	1.083472044	7.678066015	1.00728452	8.618245125
120	1.063687631	8.811320066	0.9473668142	8.761183023
130	1.036536383	9.358357191	0.8858554194	9.268024921

Shashank Agarwal
agarwal.202@osu.edu

Anurag Kalra
kalra.25@osu.edu

140	1.013624921	9.621115923	0.8212523496	9.557209015
150	0.9923563807	11.26085591	0.7671857394	9.651331902
160	0.9714635865	11.81132007	0.7504742748	11.34897304
200	0.8787003877	13.96971893	0.5628594833	10.86206698





Shashank Agarwal
agarwal.202@osu.edu

Anurag Kalra
kalra.25@osu.edu

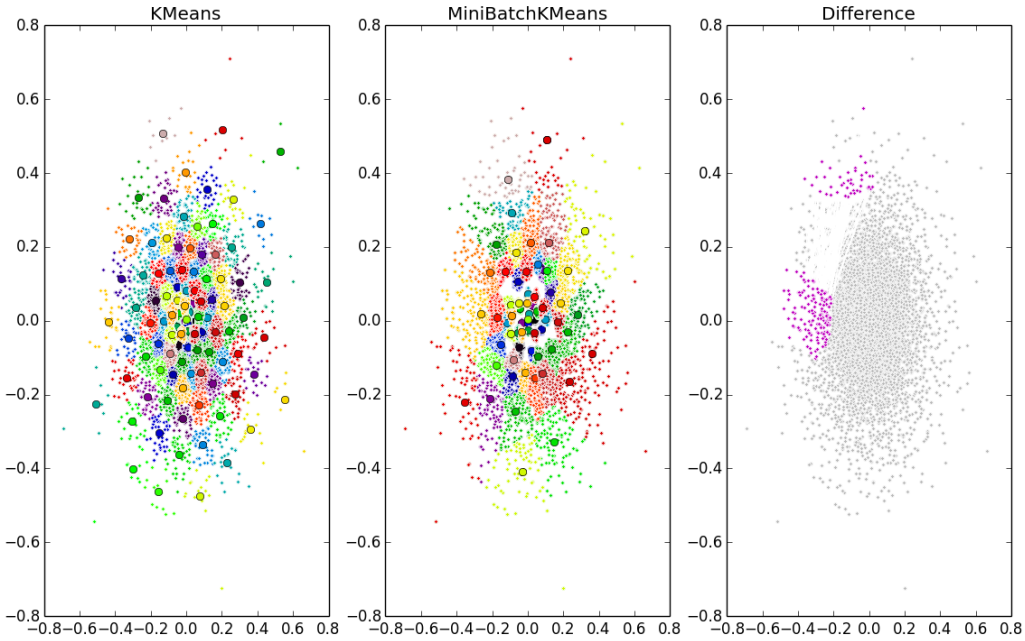


Fig: The above figure shows KMeans and MiniBatchKMeans clustering for k=200 on whole dataset. The Difference is shown in third graph.

DBScan

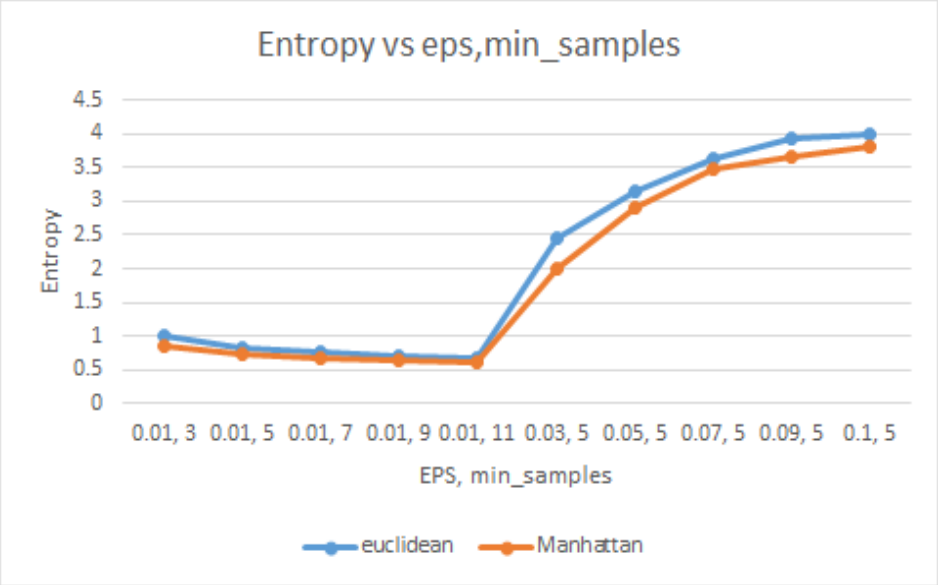
We use sci-kit's DBScan library for clustering. We passed the graph coordinates to the library. And the data for different value of epsilon and minimum value which represents the number of samples in a neighborhood for a point to be considered as a core point. For different set of Eps, min_samples values, we ran the code twice. Each time for different graph -created using Euclidean or Manhattan metric.

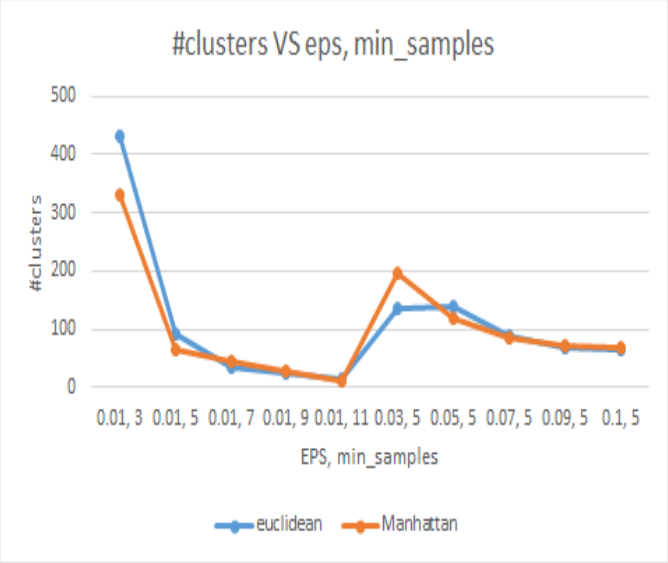
	Entropy		#Clusters		Time	
EPS, Min_samples	Euclidean	Manhattan	Euclidean	Manhattan	euclidean	Manhattan
0.01, 3	0.9992	0.8611	430	332	2.342	2.141
0.01, 5	0.8318	0.7317	92	67	2.345	2.05
0.01, 7	0.745	0.6796	35	44	2.126	1.969
0.01, 9	0.7067	0.6435	24	30	2.133	2.009
0.01, 11	0.6771	0.6139	14	12	2.112	2.006
0.03, 5	2.46	1.998	135	196	3.082	2.516

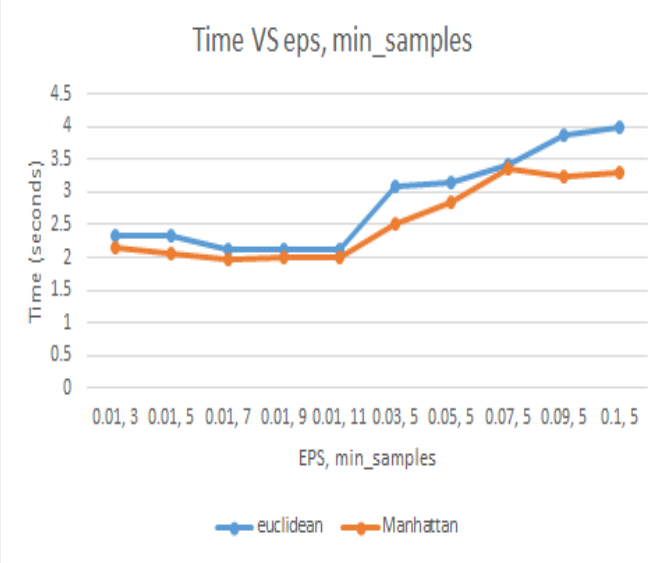
Shashank Agarwal
agarwal.202@osu.edu

Anurag Kalra
kalra.25@osu.edu

0.05, 5	3.1396	2.901	140	120	3.147	2.858
0.07, 5	3.6319	3.472	88	86	3.417	3.372
0.09, 5	3.9193	3.671	70	72	3.881	3.234
0.1, 5	4.0042	3.812	67	68	4.001	3.312







Entropy

The entropy was calculated for every cluster and also the overall weighted entropy using the formula:

$$H(X) = \sum_i P(x_i)\, I(x_i) = -\sum_i P(x_i) \log_b P(x_i)$$

For both K-means and DBScan we show the Entropy variance in above charts.

Shashank Agarwal
agarwal.202@osu.edu

Anurag Kalra
kalra.25@osu.edu

Scalability

With our experimentation we learned that clustering on big data can be very time consuming. So at first when we were clustering directly on feature vector it took really long to cluster.
Further using a distance matrix improved the time but still there were too many calculations.
Finally we created a graph for our documents, although the graph could only be made on a huge AWS Server with over 64GB of RAM and took around 2 hours to be made, it significantly saved our clustering time as the data was already preprocessed and ready for clustering.

Hence to scale the algorithms we recommend preprocessing the data to a simple form and also it is always good to reduce the dimensionality.

Contribution

Shashank Agarwal**:** K-Means clustering algorithm
Anurag Kalra**:** DBScan clustering algorithm.

Credits

I. Scikit-learn. We used multiple modules available in Scikit-learn library for calculating Distance metric and for Clustering.
II. Article on ‘Finding the coordinates of points from distance matrix’ http://math.stackexchange.com/questions/156161/finding-the-coordinates-of-points-from-distance-matrix