Tim Taylor and Taumer Anabtawi
cse5243, Lab 2

What is implemented:

for this lab, we have implemented K-Means and dbscan for our clustering algorithms. For our two distance metrics, we chose to implement Manhattan distance and Euclidean distance.

Tim Taylor created the code for dbscan, and the code for calculating Manhattan distance, entropy, and skew.

Taumer Anabtawi created the code for K-Means, SSE and the code for calculating Euclidean distance. He also adapted his previous lab to output its feature vectors in a manner that could easily be read by this lab.

our projects run from input files that are generated from the previous lab. As such, the run time for our project does not depend at all on the run time of the previous lab. You do not need to run the previous lab to get these input files as we have provided several for you. See the readmes for the two scripts for further explanation

Underlying assumptions:

for K-Means:

1) The centroids are chosen by selecting K random documents' feature vectors to be the initial centroid vectors
2) No post-processing was done to further refine clusters found

for dbscan:

1) If a point is equal to epsilon in distance away, it is counted as a neighbor
2) A point cannot be its own neighbor

for measuring clustering quality:

1) for entropy, I used log base 2 as this class recommended

How to use this project:

There are two python script files for this project, one for K-Means, and one for DBScan. Each has a README file that will explain how to run the file including how to change the parameters for K, epsilon, input data, etc.

Scalability:

for K-Means:

The run time for K-Means depends on many factors including the K value chosen and number of iterations that the algorithm is run for. Our results showed that in general, K-means required more time to complete than dbscan did. For 1000 documents, a K value of 250 and 75 iterations, the algorithm took about 11 minutes to complete. Using the same parameters on 5000 documents, the clustering took around 30 minutes to complete. The relatively long run-times are due to the fact that K-means performs many comparisons for each document on each iteration leading to a runtime

complexity of O(n*K*I*d) were n is the number of input documents, K is the number of clusters, I is the number of iterations and d is the number of attributes.

From running the algorithm on many different parameters, we were able to notice that the K-values and the initial centroid points affected the results the most. In most of the trials, increasing the number of iterations over 50 only marginally altered the results, leading to only slight changes in entropy and skew. We were able to conclude that in order to increase the number of documents, it was imperative to select the right value for K with a sufficient, but not excessive choice for the number of iterations in order to receive usable results in an appropriate amount of time.

Running the K-Means algorithm with the euclidean distance method as our similarity measure takes slightly more time than running the same algorithm with the Manhattan distance. We believe this is because euclidean distances depends on using math libraries and uses “pow” to square numbers, which requires more CPU power to perform.

for dbscan:

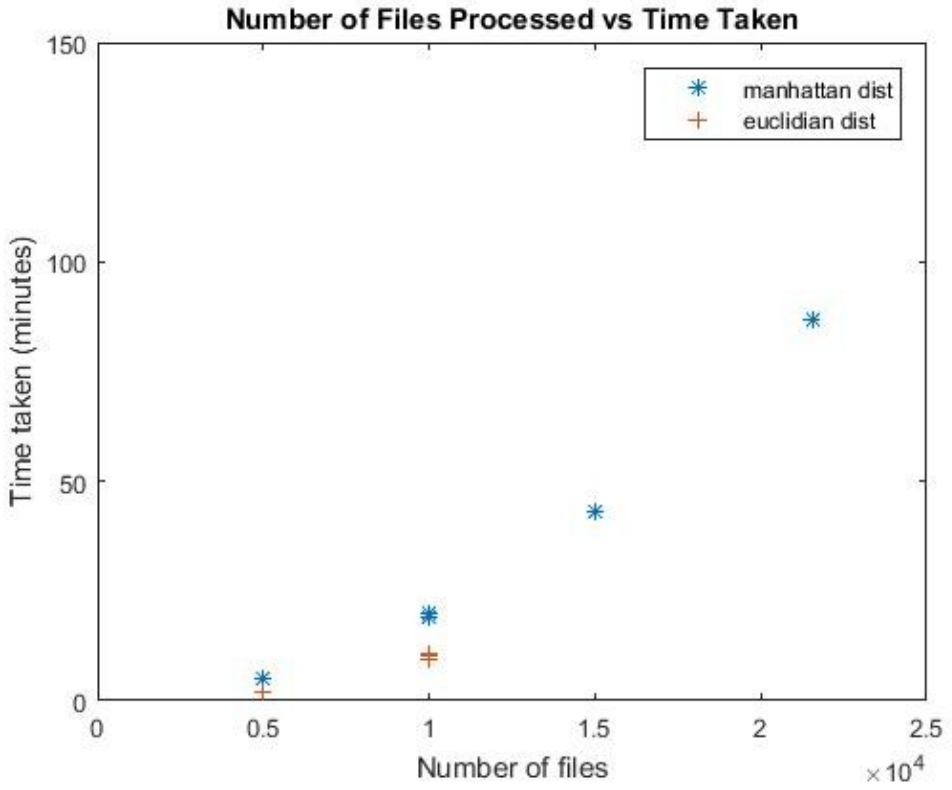
You can see below for a graph that shows how fast dbs can ran for a given amount of files to process (for both Manhattan

distance and euclidean distance). For Manhattan distance, when the amount of files to process doubled (from 5,000 to 10,000) the

run time increased by a factor of 4 (1.5 minutes to ~6 minutes). Similarly, when the amount of files to process tripled (from

5,000 to 15,000) the runtime took about 9 times longer (1.5 minutes to ~13 minutes). This suggests that we have a polynomial scale

for runtime of N^2. This scale is decent. It isn't great, but it isn't exponential which makes it manageable.



It is worth noting that the euclidean distance using DBScan runs typically took a bit longer than the Manhattan distance runs.

Our group believes this is because of the euclidean distance relying on the power function to square distances. This power

function is much slower than the absolute value function used in Manhattan distance so it likely affected our run times.

Regardless of that, the scale of growth for euclidean distance using DBScan is roughly N^2 still

Quality of clustering:

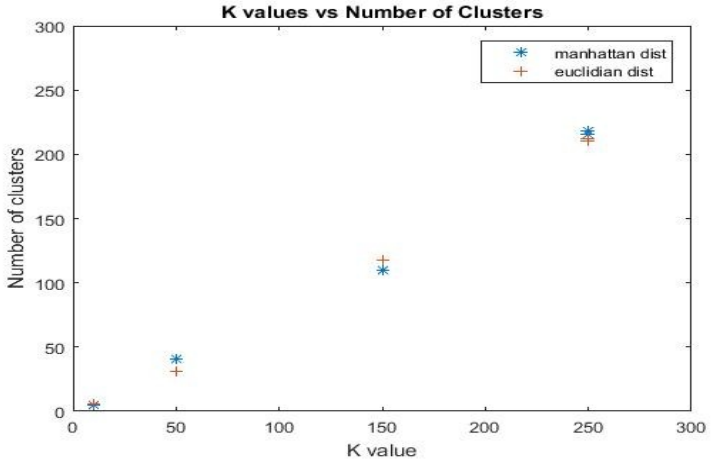
For measuring quality, our group chose to use entropy and skew as the primary measures. Lower entropy means better clusters and

lower skew means that the clusters are all similarly sized.

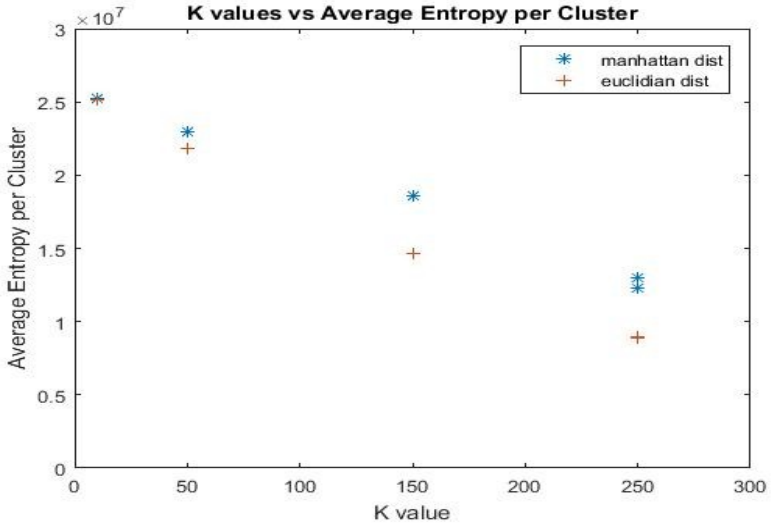
for K-Means:

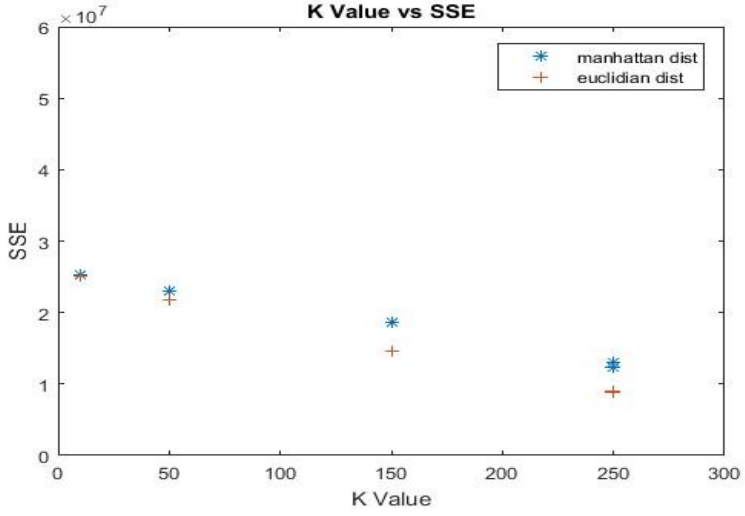
The quality of the clustering for K-Means was mostly dependent on the values chosen for K and the centroid points chosen. Our Lab 1 results provided us with feature vectors that contained a very refined set of words for each document. We were able to conclude that the words filtered out and included in the feature vector for each document were heavily indicative of the overall topic of the document. Keeping that in mind, two feature vectors from our data set could only be considered similar if they were truly similar as each set of words was extremely specific to the document it was filtered from. Our clustering results using K-Means supported our conclusions from Lab 1. The following graph and discussion is based on our data taken for multiple K-values and iterations on a data size of 1000 documents.

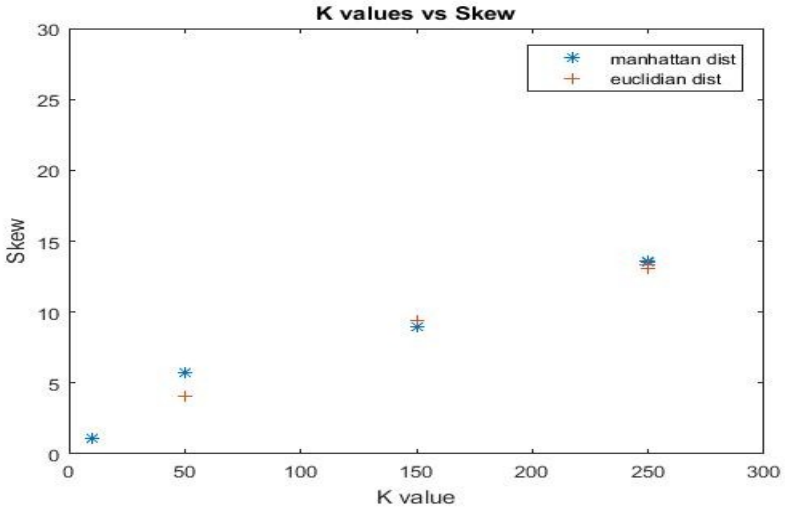
As we increased the K value, we saw that the number of clusters also increased at a similar rate. Although the initial centroid values were chosen at random, the results still showed a linear relationship between number of clusters and K value. This may be due to the fact that each feature vector was highly representative and specific to the document it belongs to, which allowed for more distinct clusters to exist as we allowed for more centroid points. However, in only a few test cases, running the algorithm resulted in one cluster having a disproportionate amount of documents in it. These “outlier” results were attributed to the algorithm selecting poor choices as the initial centroids.



We were also able to see that as we increased the value for K, we were able to achieve clusters that had smaller levels for entropy and SSE. Our value for skew only increased as the value for K was increased, but occurred at a linear rate. Overall, the values for skew were very similar across the different trials, which supports our notion that the document word sets used were very distinct (similar skew values means similar cluster sizes). In general, we were able to make the conclusion that larger K values, but not too large, provided more useful results than smaller K values. Graphs of K value vs SSE, skew and entropy can be seen below.

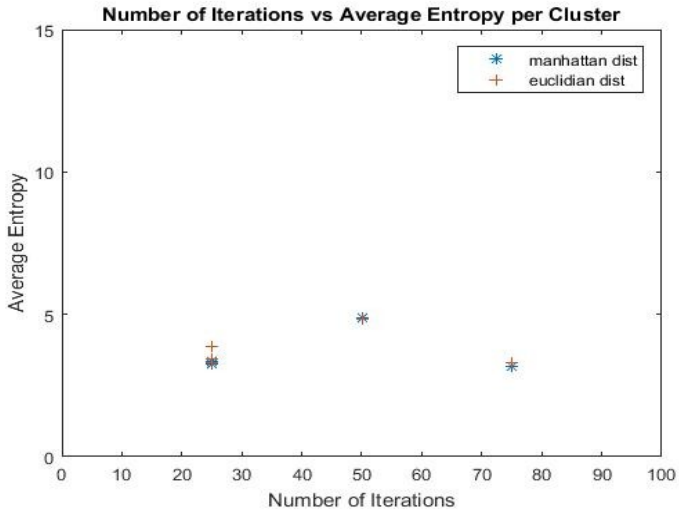


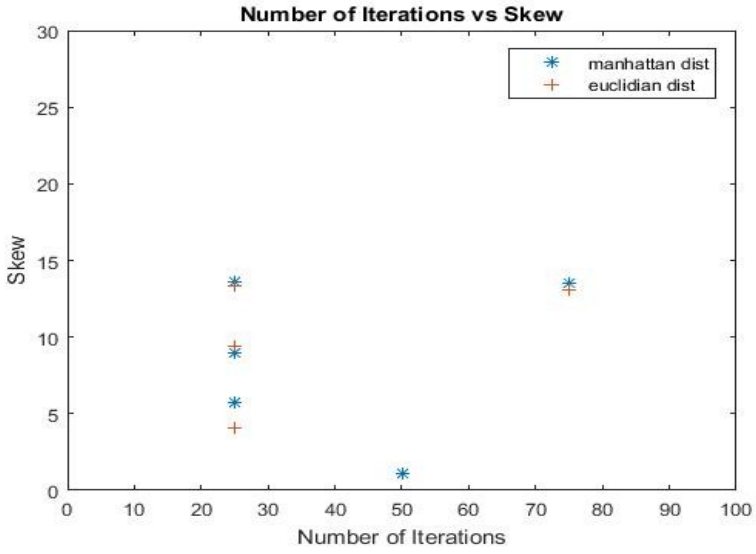


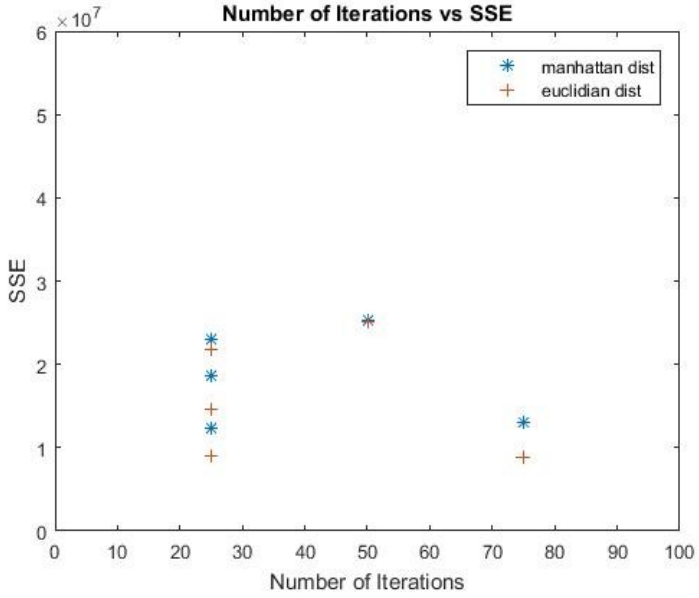


Because of the mostly linear results, we were unable to make a claim on what would be the most optimal K value to choose for clustering. However, based on our results that showed using a larger K value produced lower entropy values, we were able to conclude that larger K values should be favored, especially as the size of the input dataset increases.

We also recorded results that compared the number of iterations to the number of clusters found, SSE, entropy and skew. The results of these tests showed that K value had a much larger effect on the number of clusters than the number of iterations. In graphs that showed more of a horizontal trend, we were able to conclude that number of iterations did not have much of an effect on the results. This can be seen in the Number of Iterations vs Average Entropy per Cluster graph. In general, it is hard to make conclusions on the rest of the graphs as the number of clusters and K value are not considered which would heavily contribute to any correlation in the results.







In general, we did not see any significant differences in any of the values tested when using Manhattan distance versus euclidean distance as our similarity measures.

for dbscan:

The quality of the clustering was highly dependent on both epsilon and minPoints. To better get a conclusion on how quality

this clustering can be, I spent some time varying these parameters on a smaller data set in order to find the values that would

give optimal clustering in the full set. Below are several charts showing the resulting average entropy per cluster and skew vs

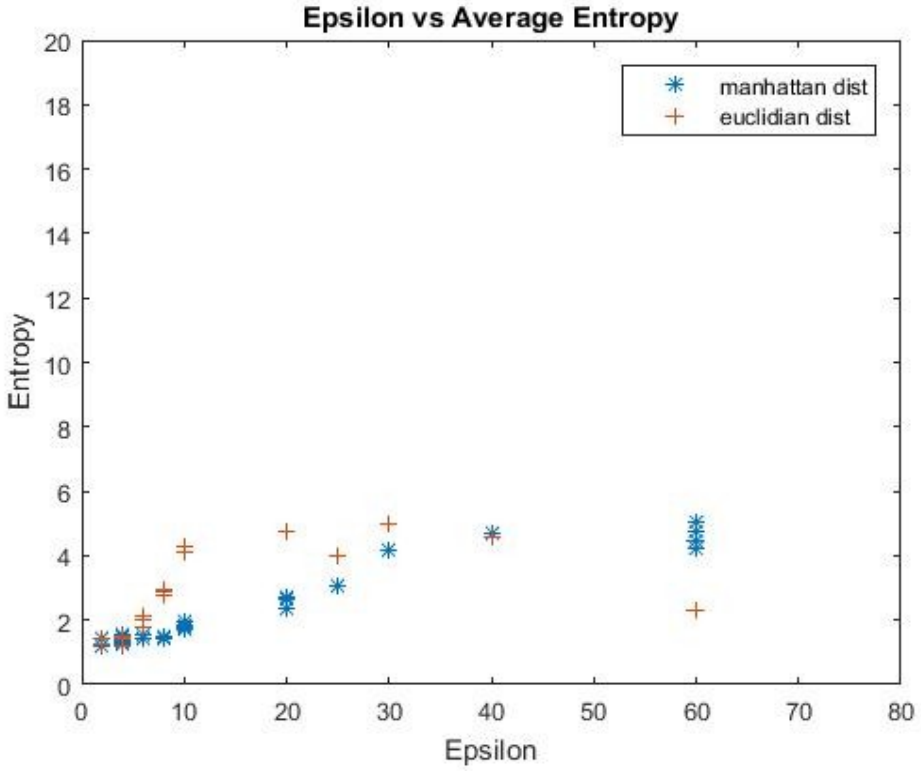
minPoints and epsilon (for both Manhattan distance and euclidean distance)

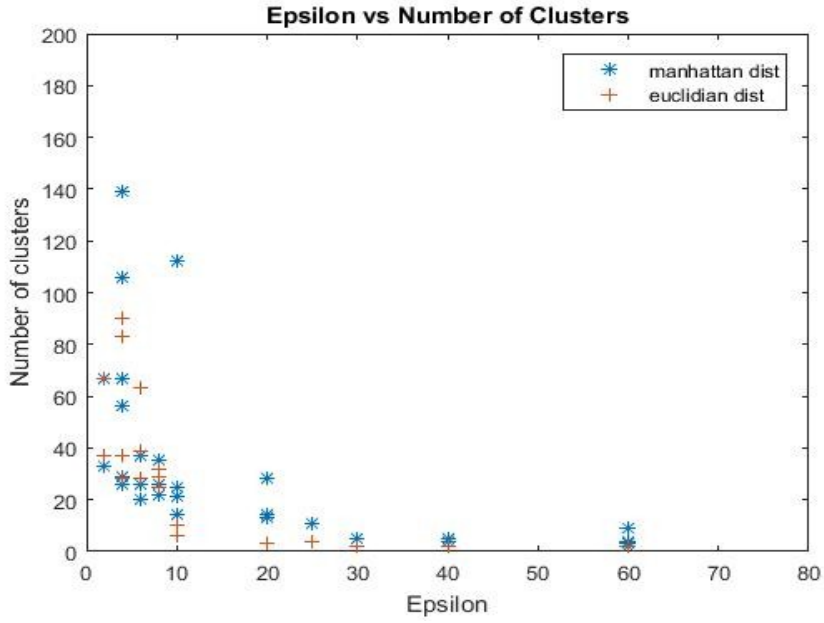
We found that results tended to have to balance average cluster entropy against skew. Higher epsilon and minPoints yielded

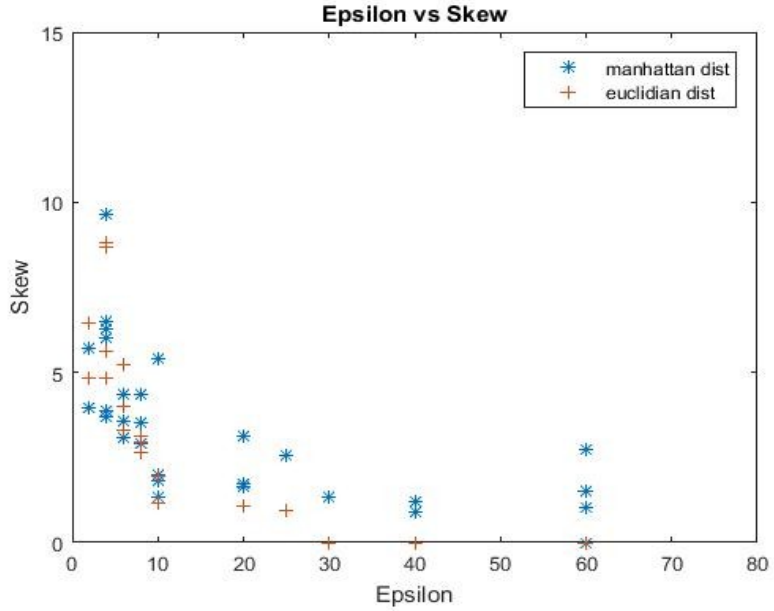
lower skew, but higher cluster entropy. Also, higher epsilon and minPoints yield lower number of clusters, so that is an

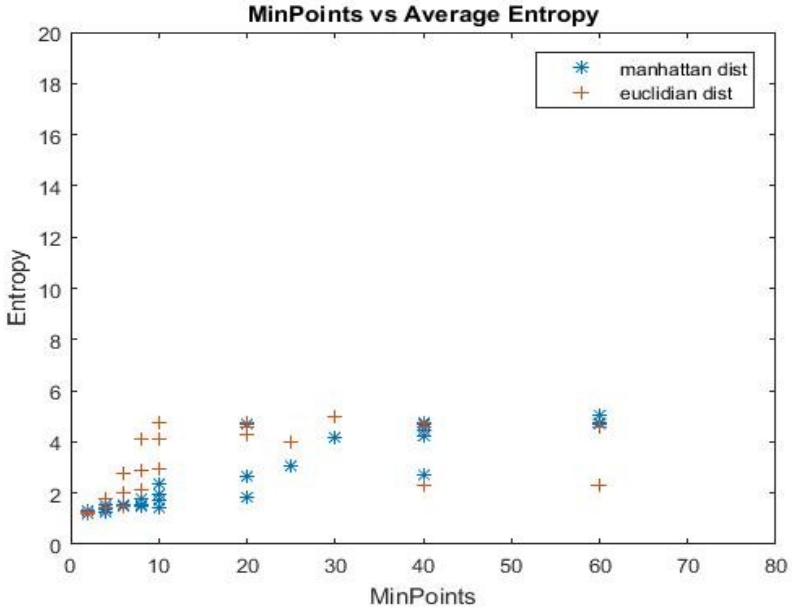
additional consideration for picking the parameters. Because of these two balances to strike, there was no "best" configuration

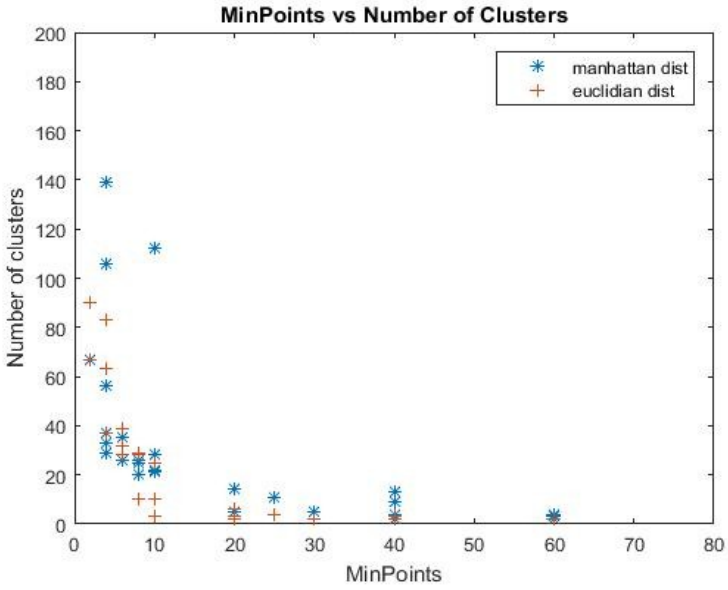
of minPoints and epsilon. It would depend on if you want small clusters or big clusters, low skew or high skew, etc.

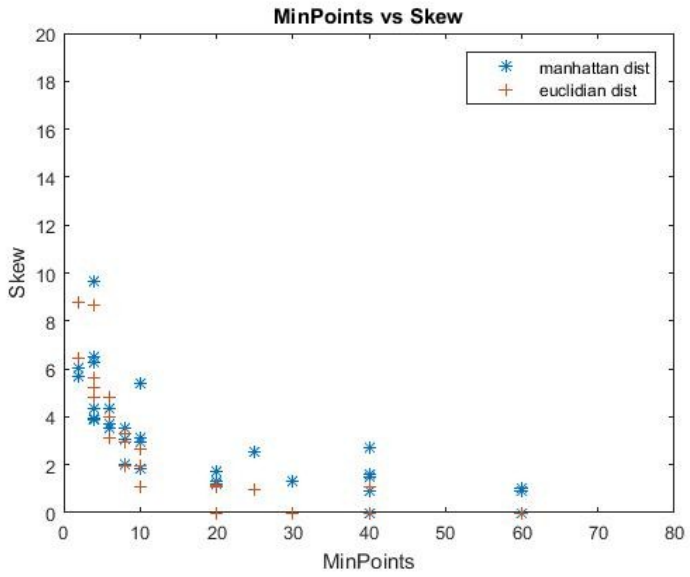





Epsilon vs Skew
manhattan dist
euclidian dist
Skew
Epsilon
0
5
10
15
0
10
20
30
40
50
60
70
80


MinPoints vs Average Entropy
manhattan dist
euclidian dist
Entropy
MinPoints
0
2
4
6
8
10
12
14
16
18
20
0
10
20
30
40
50
60
70
80





In general, Manhattan and euclidean distance did not affect the results very much. They both followed the same trends such as

where higher minPoints would lead to larger clusters (in conjunction with higher epsilon).