Part 1. Clustering: the baseline

1. Write a script that applies the k-means clustering method to the 100K subset. Measure a typical processing time. Experimentally detect the maximum number of clusters k that can be handled with the implementation of the algorithm you are using.

Here's a table of the experiment I have run with k-means on different cluster size.

Cluster Size	Seconds
3	0.496160984
4	0.678535938
5	1.250231981
6	1.936803102
7	1.630746126
8	1.524646997
9	2.209270954
10	2.326575994
11	2.559873104
12	2.101411104
13	2.379818916
14	2.835674047
15	2.988120079
16	3.200045109
17	3.084188938
18	3.673568964
19	4.902256966
20	3.680253983
21	3.680253983
22	4.343153954
23	4.785017967
24	5.042570829

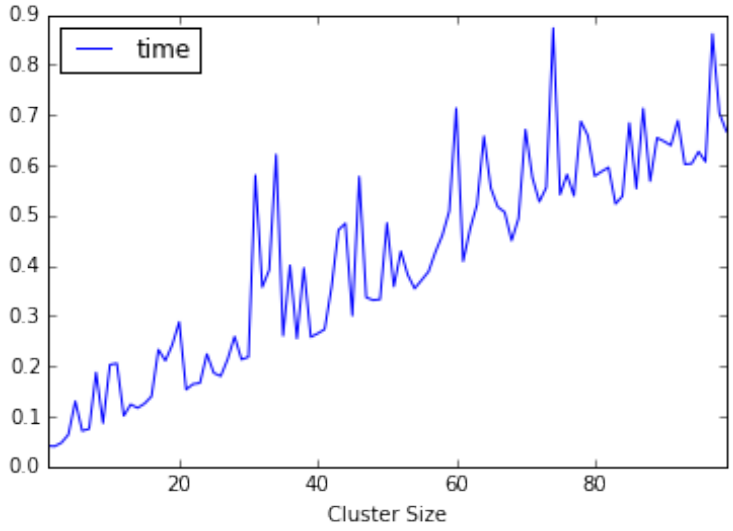
2. Write a script that applies the MiniBatch k-means method to the 100K subset. Select an appropriate value of a batch size. Measure and note the gain in computational time. Evaluate the maximum number of clusters k that can be handled with the implementation of the algorithm you are using.

Before testing the number of cluster size, I tested the optimal batch size by looping through batch size 30 – batch size 4000 (I'm showing a part of the data here). After couple attempts, optimal batch size falls between the range of 900 – 1200.

Batch Size	Seconds
570	0.072448969
600	0.1606071
630	0.053327084
660	0.053051949
690	0.058384895
720	0.052681923
750	0.085420132
780	0.097144127
810	0.065526009
840	0.054008007
870	0.051879168
900	0.051104069
930	0.052051783
960	0.051815033
990	0.060842037
1020	0.058336973
1050	0.070904016
1080	0.092190981
1110	0.076183081
1140	0.097358942
1170	0.100914001
1200	0.092152119
1230	0.093122959
1260	0.090772867
1290	0.085632086
1320	0.150669098

Then I start testing batch size 900 with different cluster sizes.

This graph shows the time it took for different sizes of cluster matched with batch size 900.



3. Write a script that applies the DBScan method to the 100K subset. Fix the min number of samples in a cluster as 100. Experimentally explore the influence of the connectivity threshold

After experimenting with eps and min_samples. The trend seems to be the lower eps is, meaning the more dense the clusters are, the lower the processing time is require. On the other hand, the higher the mini_samples are, meaning the more dense the clusters are, the lower processing time required. This trend is consistent in the manner that if we want a more dense cluster, the running time will be faster.

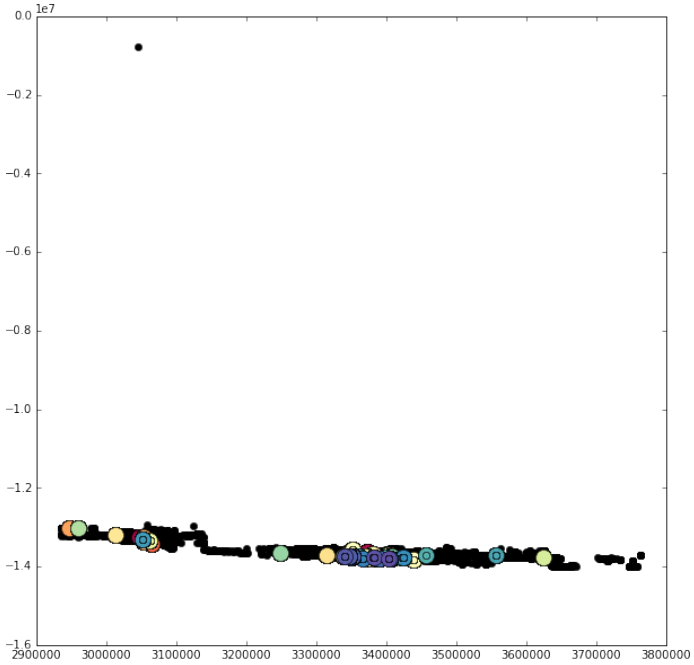
Reference time of clustering 100K samples into k=100 clusters with k-means and mini-batch k-means
With k-means, if we set the k to 100 clusters, we get processing time between a range of 17.254328966140747 seconds to 24.04636287689209 seconds.

With mini-batch k-means, if we set the optimal batch size to be 900 (chosen by selecting the minimum processing after testing batch size 30 - 5970), we will get processing time ranging from 0.7106809616088867 seconds - 1.1090731620788574 seconds.

Maximum number of clusters k_max that your implementation of k-means and mini batch k-means can handle. Explain the reasons behind this performance bottleneck.
There's no performance bottleneck now with the new updates.

The value of e (call it e_100) in DBScan resulting in approximately 100 clusters of a minimum of samples (MinPts = 100) and the corresponding processing time.
Eps value 200 would produce approximately 100 clusters given a minimum sample of 100. Processing time ranges from 1.1952118873596191 seconds to 2.001431941986084 seconds.

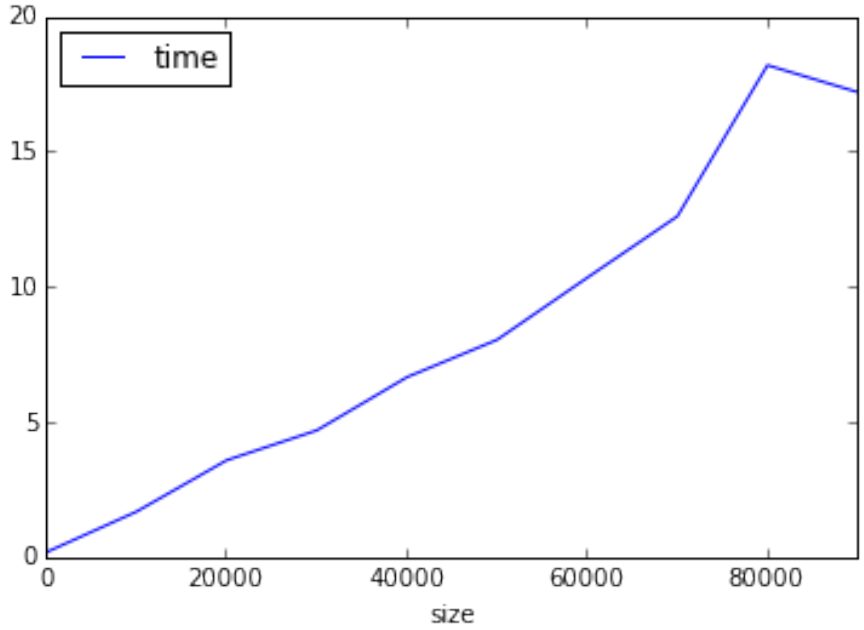
Here's an image of how the clustering looks like.


1e7
0.0
−0.2
−0.4
−0.6
−0.8
−1.0
−1.2
−1.4
−1.6
2900000
3000000
3100000
3200000
3300000
3400000
3500000
3600000
3700000
3800000

Part 2. Clustering: scalability (40 points).

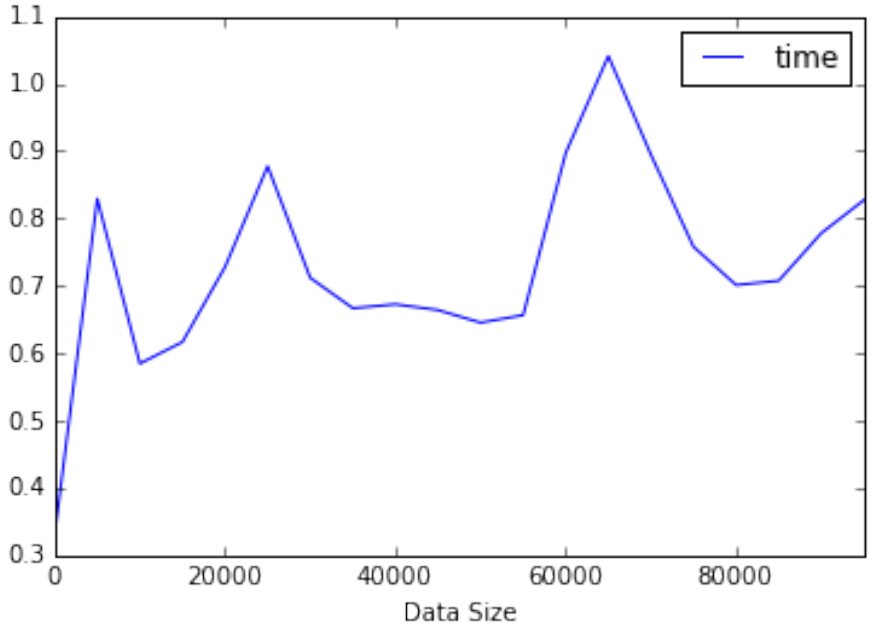
a) Number of data samples (consider the range of 100 to 100'000) for a fixed k =100.

K-means Data Size



By eye-balling the growth rate, it will take approximately 200 - 250 seconds to complete a million tweets since it's following a rather linear trajectory.

Mini-batch k-means data size

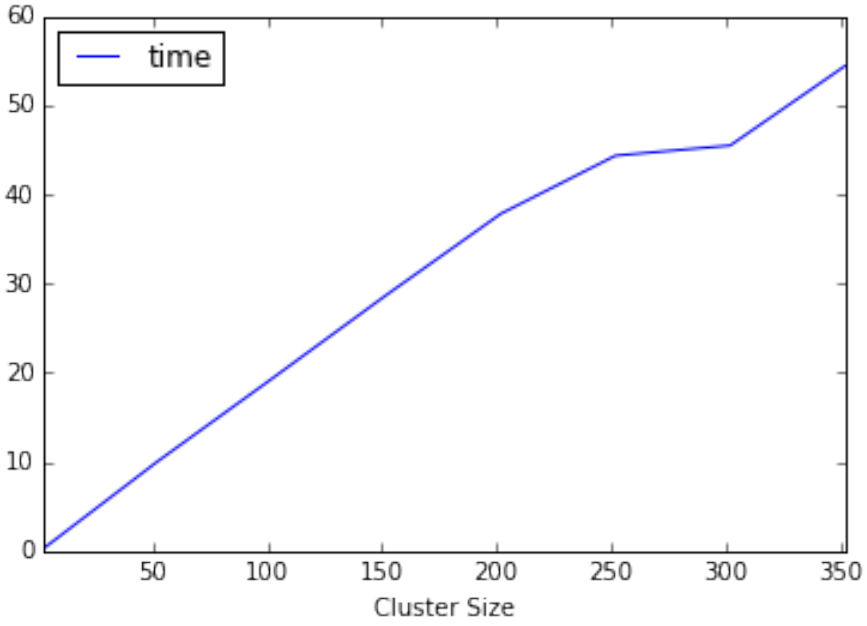


Mini-batch is trickier to eye-ball, so I just ran the a million tweets on mini-batch. It only took around 2 seconds. To be exact, it took 2.4107940196990967 seconds.

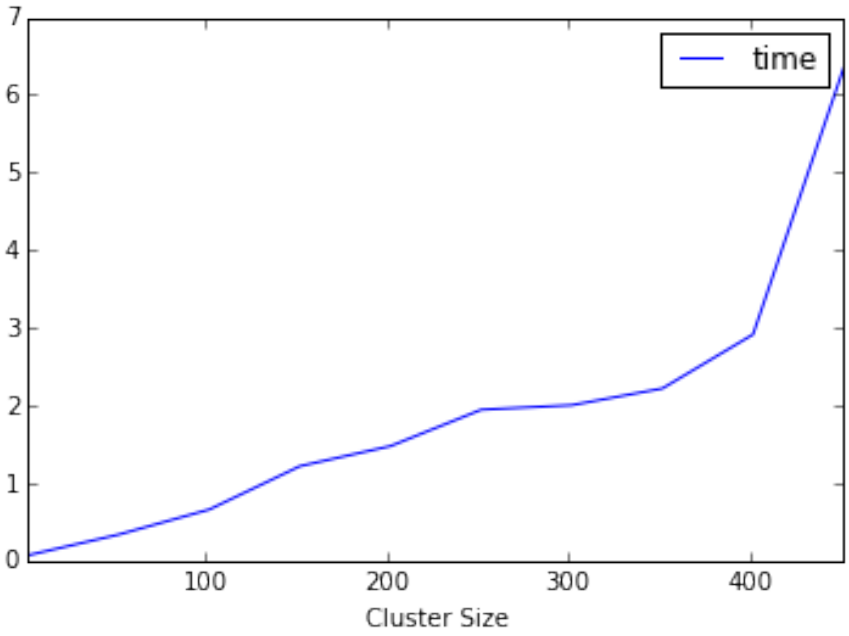
By comparison, mini-batch is much more efficient in handling increasing number of data.

b) Number of requested clusters k (consider the range of 2 to the k_max)

K-Means Cluster Size

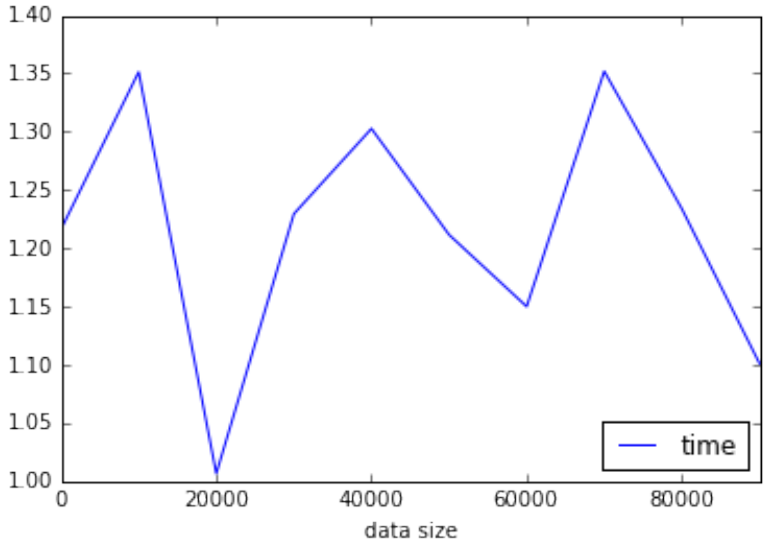


Mini-batch Cluster Size



Mini-batch is much faster than k-means, but they all are growing at an exponential rate when the size of cluster increases. Since there's no bottleneck with this new version, I didn't go further to test larger cluster size since it is not necessary because we can the runtime is just growing on a linear fashion.

a) Number of samples (consider the range of 100 to 100'000) for a fixed ε_100, MinPts = 100.



DBScan is trickier since the run time fluctuates and it does not seem to be taking up too much time when run on 100K tweets. But after many attempts to run DBScan on a million or by looping the DBScan on multiple subsets of 100K tweets, I had no success but crashes on the local server.

Part 3. Clustering: 1 million samples problem (40 points).
Describe your approach to the design of the system in a write-up document and provide the total number of cluster detected. Submit your codes as a standalone script.

After spending time to experiment with the parameters with different clustering algorithm, I have developed a clear idea of the nature of each algorithm. Mini-batch is the fastest among all since the algorithm only runs the batch size instead of all the data points. K-means is the slowest and is affected by the outliers. The outliers also affect mini-batch, despite the speed advantage of mini-batch. DBScan does not get affected by the outliers since it clusters based on the density instead of the variance between centroid and each data points. After knowing this, it's obvious that we will select mini-batch to process the tweets then pass off the data to DBScan since DBScan has the power of identifying dense area which in our case, would be the area that generates most tweets. With mini-batch generating 100 clusters and passing it off to DBScan, DBScan is able to produce a total of 1606 clusters based off that. You can play around with the cluster size of mini-batch to experiment with the results. If you try 200 clusters for mini-batch, you get around 820 cluster size. Because it's harder to identify density when there are more clusters preprocessed. I have also attached the graph below to show one of the cluster and conducted further analysis on such cluster. The cluster I'm showing has three DBScan identified core center and around 2200 tweets are in the area. I have pulled out all the text being tweeting in the area and identified a few things. There might have been earthquake that had happened around the area since there's around 4128 mentions of such words,

not including people reporting earthquake in Japanese language. There is 8xxx mentions of earthquake in Japanese in the area. Also another thing worth noting is the people in the area are of relatively young age since there's mentions of school of around 4000 times and a lot of usages of words such as 'shit', 'fuck', and 'fucking'(all around 3000 mentions each).

Extra Credit (Max 20 oints)
Produce a visualization of a typical cluster of your choice, and describe its origins by inspecting its spatial location and/or content of the tweets formingthe cluster.