PROJECT 3

Karishma Jain
Collaborated with: Disha Jindal

Q1:

Data Transformation:

We have dropped the feature name and user_id from the yelp dataset as every name and user_id is unique. This feature information are not meaningful because in clustering we group the data into clusters with similar features and the above features are unique. Additionally, name and id, do not infer behavioral information.

For the yelping_since feature, we are just keeping the year since the user is on yelp, as the number of months and days wouldn't account for significant information and normalizing the data points would be possible.

If a user has been chosen as an elite user, this tells us that the user has been very active and helpful to the yelp community. In our dataset, we are given the years in which the user was chosen as elite. But, compute the number of times the user was a part of the elite squad as that would be better feature metric to compare it with the other users.

The scale for few of the features are different and so we normalize the dataset for the features. This will also help us in selecting random centroids for online version of k-means clustering algorithm.

Problem Formulation:

We have yelp dataset and we'll cluster the data using 3 techniques: K - Means, K - Means++ and a variation of MCMC. The evaluation criterion that we'll use to compare these models is:

Mean Distance: Average of average distance(for each cluster) across all clusters which has any data points in them.

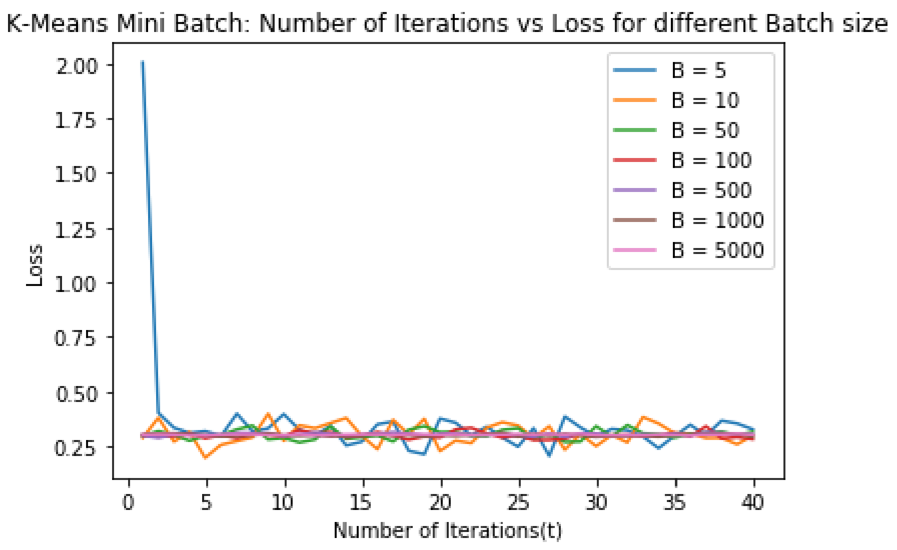
Minimum Distance: Average of minimum distance(for each cluster) across all clusters which has any data points in them.

Max Distance: Average of maximum distance(for each cluster) across all clusters which has any data points in them.

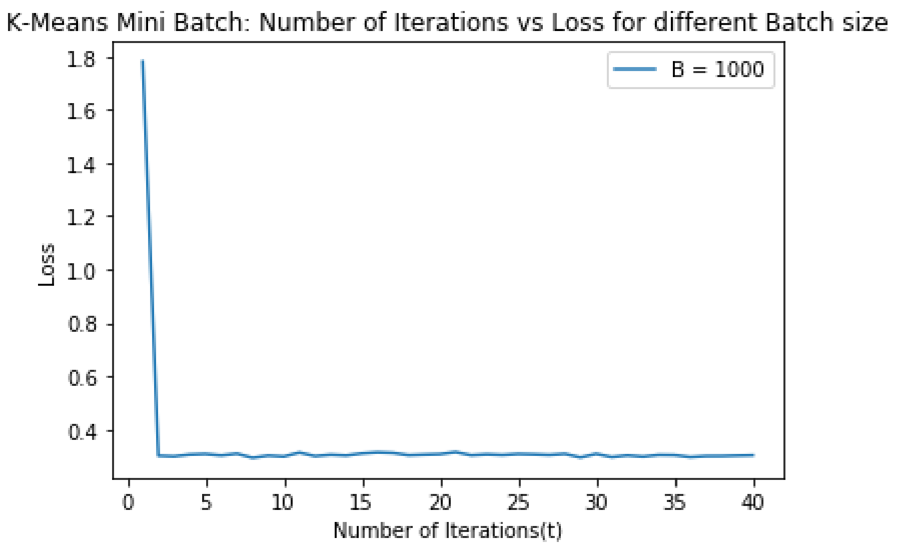
Q2: Online version of k-means clustering algorithm

Number of Centroids
k_list = [5, 10, 50, 100, 200, 300]

Batch Size vs Loss:
(a)Plot:

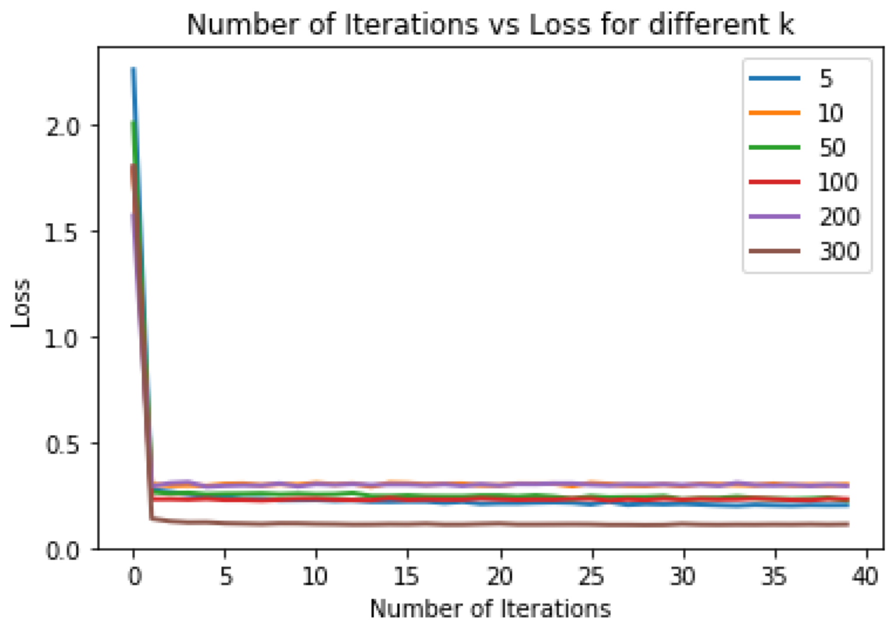


As we can see from the above graph, for initial 3-5 iterations, the loss for batch size 5 is very high and oscillates after 5 iterations and converges to minimum for greater number of iterations. The loss for batch size 10, 15 is also somewhere oscillating for different number of iterations. For batch size greater than 100, the variation in loss is very less and as we can see the loss converges to minimum for very low number of iterations. We can choose any batch size greater than 500 as the loss converges to minimum quickly. We have taken the batch size as 1000.



The above plot of number of iterations vs loss is for batch size 1000 and we have run it for 40 iterations and as we can observe, loss converges almost by 3 to 4 iterations.

Iterations vs Loss :
(a)Plot:



(b)Table:

K	Iterations vs loss

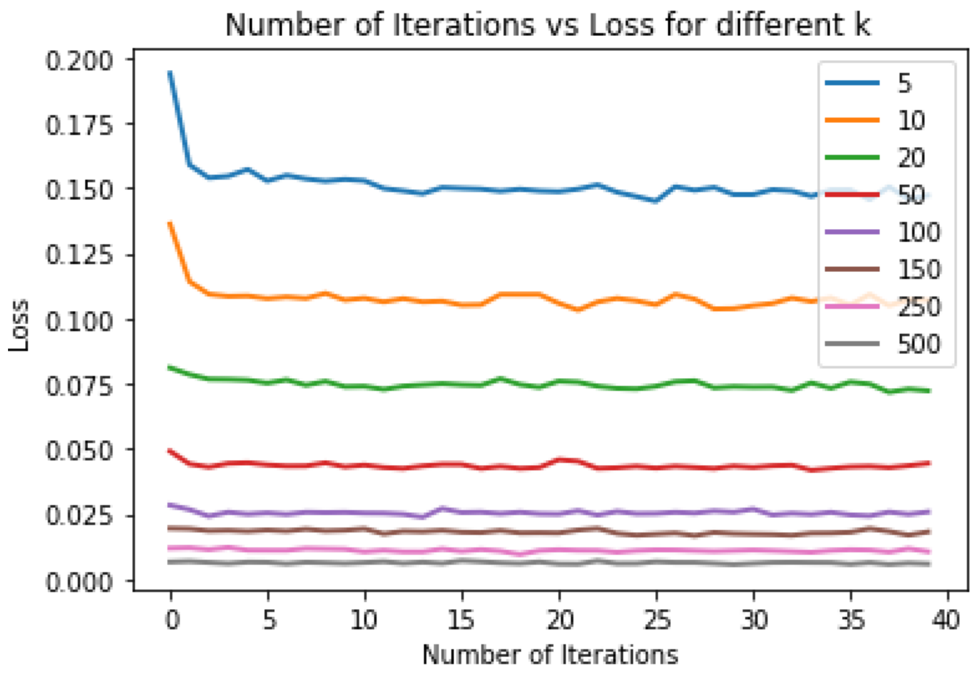
	1	2	3	4	5	6	7	8	9
5	2.253	0.2776	0.2681	0.2570	0.2479	0.2445	0.2358	0.2356	0.2286
10	1.777	0.3038	0.2973	0.2992	0.2964	0.3057	0.3073	0.2988	0.3049
50	2.001	0.2684	0.2603	0.2639	0.2565	0.2585	0.2590	0.2608	0.2565
100	1.7968	0.2337	0.2340	0.2320	0.2356	0.2301	0.2300	0.2273	0.2312
200	1.5672	0.2939	0.3093	0.3137	0.2923	0.2971	0.2993	0.2979	0.3078
300	1.7988	0.1427	0.1303	0.1250	0.1255	0.1205	0.1189	0.1169	0.1200

(c) Analysis:
As we can see from the table and graph above, the loss converges to minimum after 4 to 5 iterations for different values of K, we take the number of iterations to be 40.

Q3: K Means ++

Number of Centroids
k_list = [5, 10, 20, 50, 100, 150, 200, 250]

Iterations vs Loss:
(a)Plot:



(b)Table:

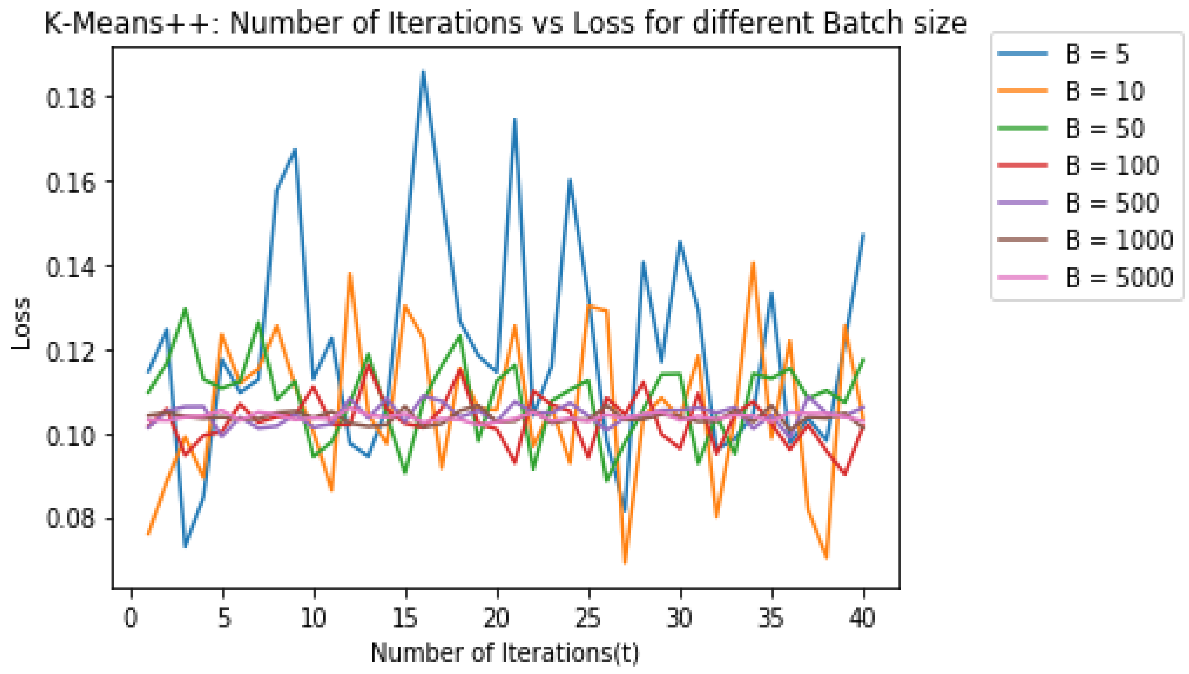
K	Iterations vs loss

	1	2	3	4	5	6	7	8	9
5	0.1937	0.1588	0.1538	0.1544	0.1570	0.1526	0.1547	0.1534	0.1525
10	0.1361	0.1141	0.1092	0.1084	0.1086	0.1075	0.1082	0.1076	0.1095
20	0.0810	0.0784	0.0767	0.0766	0.0763	0.0752	0.0764	0.0744	0.0758
50	0.0491	0.0442	0.0430	0.0444	0.0447	0.0439	0.0434	0.0434	0.0446
100	0.0284	0.0267	0.0242	0.0257	0.0249	0.0254	0.0249	0.0257	0.0255
150	0.0195	0.0194	0.0186	0.0187	0.0184	0.0189	0.0185	0.0192	0.0185
250	0.0119	0.0121	0.0114	0.0123	0.0111	0.0111	0.0111	0.0118	0.0117
500	0.0066	0.0069	0.0064	0.0060	0.0066	0.0065	0.0059	0.0065	0.0063

(c)Analysis:
As we can see from the table and graph above, the loss converges to minimum(is almost the same) after 5 iterations for different values of K, we take the number of iterations to be 40.

B vs Loss:

(a)Plot:



As we can observe from the above graph, if we take the size of the batch to be very small(B = 5,10, 50, 100); the loss is high and it will take large number of iterations to converge. For the values of B above 100, the loss is almost the same and it is converging for almost the same number of iterations. Hence, we select our batch size to be 1000 as the loss for batch size 1000 and 2000 is almost the same.

- As we can clearly see from the iteration vs loss table for online version of k-means and k-means++, for K = 50, the loss after convergence is 0.2565 for online k-means mini batch and 0.0446 for k-means++. There is a significant improvement of 0.2119 in the loss as expected because k-means++ tries to initialize centroids that are far apart from each other, whereas in the online version of k-means we pick our initial centroids randomly. As we know, the performance of k-means depends on the initialization step and the initialization of centroids for k-means++ is better than online k-means and hence we get better clustering, thereby low loss for k-means++.

Q4: Our Algorithm
As, we have seen in k - means ++, the running time of the algorithm is a lot because of the time consumed in initializing the centroids which is clearly not scalable for large datasets and specially the dataset has more number of inherent clusters which requires clustering with a large k.
Initialization is time consuming because it requires entire pass over the data for each centroid calculation by calculating the probability distribution using distances. So, to fasten the initialization step, we need to fasten the probability calculation step by using approximation. Our approach is inspired by Markov Chain Monte Carlo method. As in MCMC, we are choosing samples from a probability distribution and approximating the probability distribution to some stationary distribution so that it doesn’t require the entire pass over data and runs faster.

Algorithm
1. Choose one centroid randomly from the dataset
2. Make one pass over the data and define the initial probability distribution as follows:

Initial probability distribution:
Initial probability distribution has two parts to it.
q(x) = ½(Part1) + ½(Part2)

a. Part 1: It caters to the the distance from the first centroid which will choose far away points as per our requirement. But in case the data has a lot of outliers, it choose outliers with a high probability.

d(x, c)^2 / sum(d(x', c)^2)

b. Part 2: This part deals with giving more importance to the points which are dense and hence, near the mean because that area defines the space where most of the data is clustered. We need this as the other part is giving more importance to the far away points. So, to imitate this notion, we are using gaussian distribution. We are creating gaussian distributions in all p dimensions where p represents the number of features since each feature is almost independent of each other. Then, we are taking the sum of these gaussian distributions to find the probability of each point.

sum(e^(-1*(np.square(x - mu)/ 2 * var)) /sqrt(2 * math.pi * var))

3. To select next centroid, we sample a random point and run the markov chain n number of times. After choosing one point, we sample the next proposal based on the probability distribution qx defined in step 2 and accept the proposal based on the acceptance criterion of Metropolis–Hastings described in the class.
4. We iterate the step 3 k number of times to find k centroids and hence initialize our required centroids.

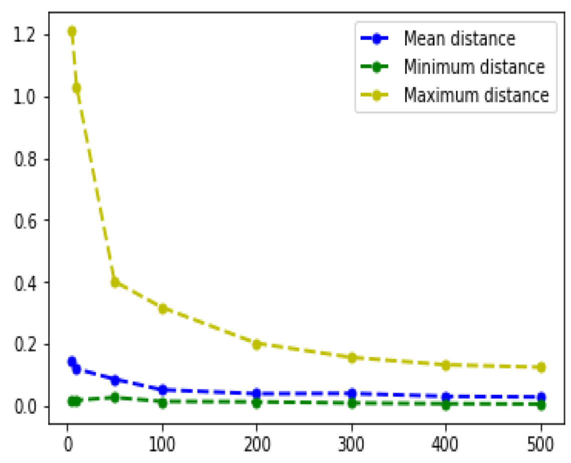
Analysis and comparison with other algorithms
After running this algorithm, two major things we analyzed were the initialization time and the distance metrics.
K means ++
1. The running time of the initialization step of this algorithm(19.27329111099243 For k=500) is way better than the time of K means(128.19120955467224 for k = 500)
2.The distance metrics is almost same as that of one we got in K means.
Hence, this algorithm is definitely speeding up the k means ++ algorithm with providing similar losses.

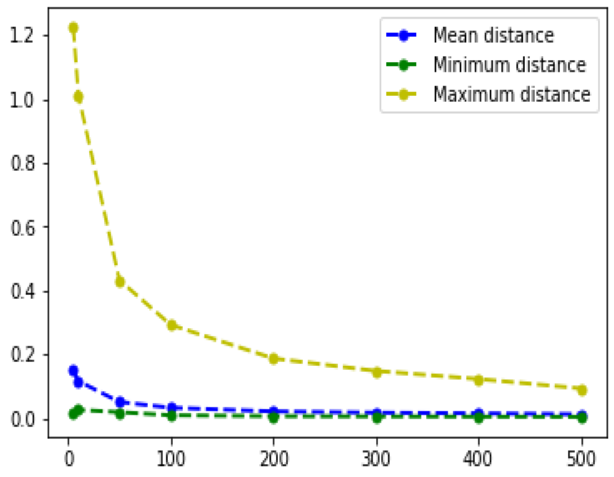
MCMC
We compared the results of our algorithm with the MCMC algorithm explained in the class as well to analyze the performance of our algorithm.

1. The initialization step in our algorithm took little more time as compared to MCMC as part2 of our algorithm is more involved and it uses normal distribution whereas MCMC uses uniform distribution which makes it slightly faster.
2. Following table shows the distance metrics of the algorithms for different values of k. We can see that all of the metrics(min, max and mean) are giving better results with our algorithm on this dataset as compared to MCMC.

	MCMC(Distance Metrics)			**Our Algorithm(Distance Metrics)**		
K	**Max**	**Min**	**Mean**	**Max**	**Min**	**Average**
5	1.21167543	0.01756993	0.14455853	1.22392617	0.01831858	0.14882932

10	1.03085846	0.01739647	0.11859177	1.01122531	0.02573167	0.11581032
50	0.40170732	0.02606197	0.08535385	0.43168177	0.01828475	0.05034636
100	0.31750158	0.01341755	0.05102192	0.29300151	0.00875939	0.0322411
200	0.2014963	0.01193602	0.03817196	0.18426793	0.00577413	0.02079083
300	0.15587269	0.00807122	0.03919516	0.1476231	0.00506833	0.01664273
400	0.1318357	0.00583622	0.02970174	0.122231	0.00429208	0.01477054
500	0.12390876	0.0047718	0.02767405	0.09272358	0.00380835	0.01183338

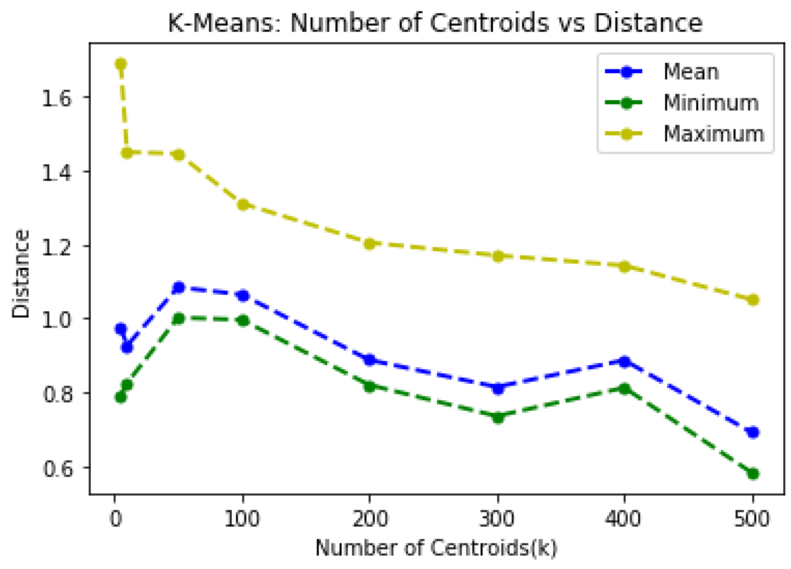




MCMC on the left, Our algorithm on the right

Q5:
We have evaluated all three of the algorithms with the parameters chosen in training phase and a list of different K's.
Batch Size = 1000
List of different number of centroids = [5, 10, 50, 100, 200, 300, 400, 500]
k_list = [5, 10, 20, 50, 100, 150, 250, 500]
No. of Iterations = 40

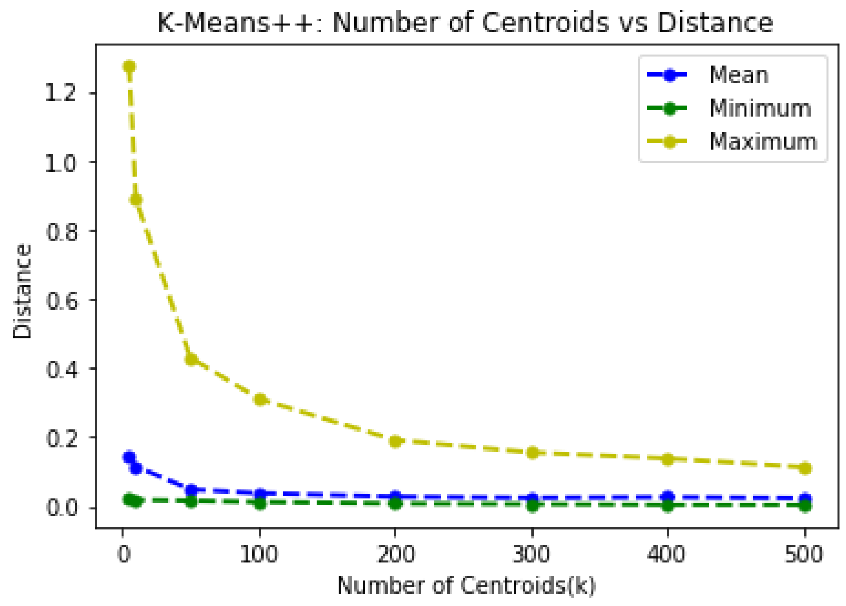
1. **K - means**
   plot(1)



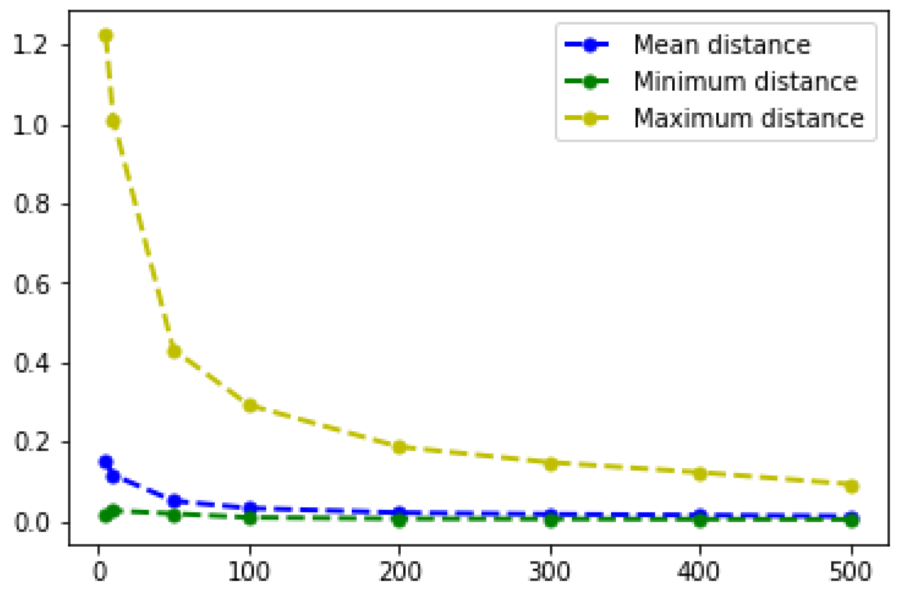
2. K - means++

Initialization: Sequential as per K means ++
Initialization Time(in seconds):
K = 5: 0.969397783279419
K = 10: 2.327777147293091
K = 50: 12.720983028411865
K = 100: 24.848555326461792
K = 200: 50.09703612327576
K = 300: 73.57429575920105
K = 400: 102.8738853931427
K = 500: 128.19120955467224

plot(2)



3. Our Algorithm (Variation of MCMC):
plot(3)



Initialization Time(in seconds):
K = 5: 0.3388187885284424
K = 10: 0.4327847957611084
K = 50: 1.5155539512634277
K = 100: 3.599907159805298
K = 200: 7.35718297958374
K = 300: 10.845945119857788
K = 400: 14.924752950668335
K = 500: 19.27329111099243

Comparison of online mini batch k-means, k-means++, our method of selecting initial cluster centroids (Variation of MCMC) :
The performance of k-means depends on the initialization step. The initial centroids determine which solution is ultimately obtained. But, some local minima are better than others. In online k-means mini-batch, the initial centroids that are chosen are random in the space [0,1]. There are chances that the initial centroid chosen could be very far away from the complete dataset or that the two centroids are very close to each other, and that kind of an initialization is not good. The final clustering will not be good and as we can observe from the above two graphs. The mean for online k-means is higher than the mean for k-means++.

Also, for online k-means (k = 500) has the lowest error, while for kmeans++ (k = 50) has the lowest error. This means that for larger value of k, the error for online k-means will converge to some local minima which could not be a good local minima, whereas for k-means++ only for k = 50, the error converges to good local minima.

This could be so because random selection leads to more number of iterations also selecting different centroid points every time gives different clusters thus leading to almost different output each time. The error converges to local minima for higher value of k(= 500) for online k-means, as compared to k-means++ (k = 50) online k-means, because having higher number of clusters in online k-means will make sure that we get atleast some good clusters. To overcome this problem, in k-means++ we choose our centroid from the data points as well as we make sure that the centroids are far away from each other so that we have better initialization of centroids and convergence will be faster.

Initialization for k-means++ is better than online k-means. But the problem with choosing initial cluster centroids for k-means++ is that since it assigns high probability to points that are far away from the, outliers could also be chosen as a centroid. We want to make sure that we not only select those data points as centroids that are far away from each other. Now, for our method of choosing initial cluster centroids, we define a gaussian distribution which kind of makes sure that the area that has more number of data points also get the opportunity to be chosen as centroids along with the points that are far away. As we can observe from the above plot for variation in MCMC, when compared to kmeans++, on convergence the error for variation in MCMC is less than the error for k-means++ and variation in MCMC yields better results as we are improving our centroid initialization.

As we can observe from the above plots plot(2) and plot(3), for k-means++ (k = 50) has the lowest error, and for variation in MCMC (k = 50) has the lowest error.

Hence, the initialization for our variation in MCMC is the best.