Project 3

Chetan Sai Tutika

Somanshu Agarwal(Collaboration)

Problem 1:

Objective: Feature extraction and data cleaning

Solution:

- Remove unnecessary columns such as name and UserId. Name and user Id are unique to the user and give no information regarding the clustering of the dataset
- Extract the year from the column ‘yelping_since’ and replace the dates with the extracted values. The extracted year gives a good approximation of the column ‘yelping_since’ and is feasible to perform operations on
- The ‘column’ is approximated by counting the number years a member has been elite. This is done by counting the number of commas and adding by 1. The none values in the column are replaced by -1
- Normalize the dataset to place all the features in range[0, 1]
    - Data_new[col] = (data[col] – col_min_value)/(col_max_value- col_min_value)

Results:

Input Data Size = (1326100, 21)

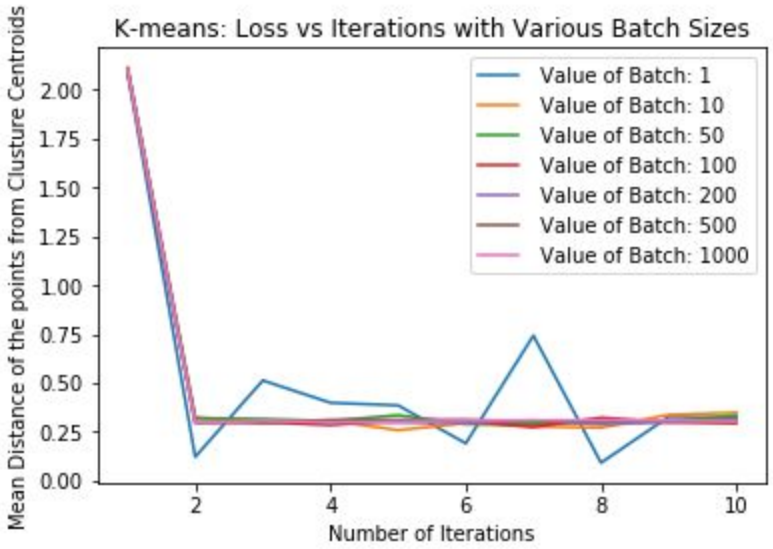
Input Data Size after data cleaning = (1326100, 19)

Problem 2:

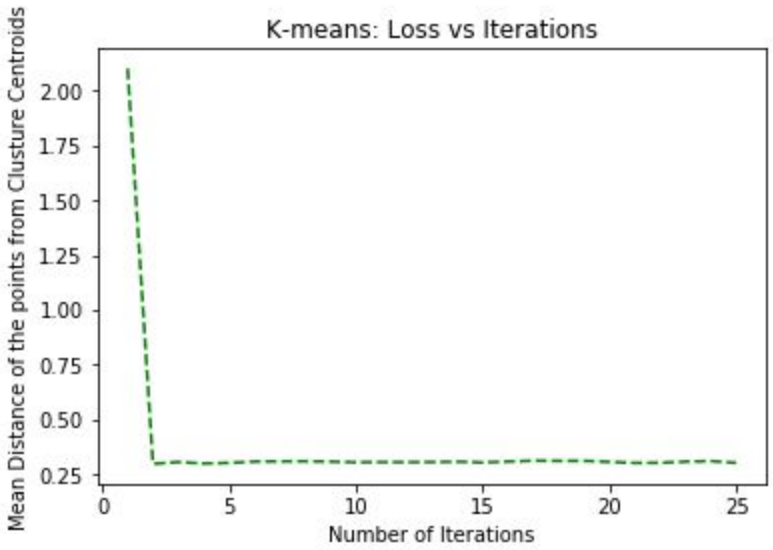
Objective: Implement online version of the k-means clustering algorithm
Solution:

- Fix the number of clusters as k = 10
- Select ‘k’ random points from the dataset
- Assign the centroids to each dataset in the mini-batch
- Update the centroids based on each point assigned to the centroid

Results:



Ideally batch size of the 1000 gives faster convergence with very less deviation in the loss. Since batch size of 500 also gives similar results we can consider 500 to be the ideal batch size. Stochastic process tries to estimate the loss with as less data points as possible. Hence, a batch size of 500 would be better than 1000



Loss at each time step

Out[127]:

array([[2.10117284],[0.29815919],[0.30623734],[0.2997537],[0.30226872],[0.30830575], [0.30908406],[0.31004001],[0.30792183],[0.30525067],[0.30576493],[0.30567567], [0.30600549],[0.30789696],[0.30408619],[0.30868579],[0.3123747],[0.31139272], [0.31172445],[0.30694503],[0.30214064],0.30282541], [0.30790769], [0.31071337], [0.30250016]])

As expected the loss for each iteration decreases. This is computed with a batch size of 500. After few iterations the loss value saturates which indicates the centroids are converged to the minimum in the loss function

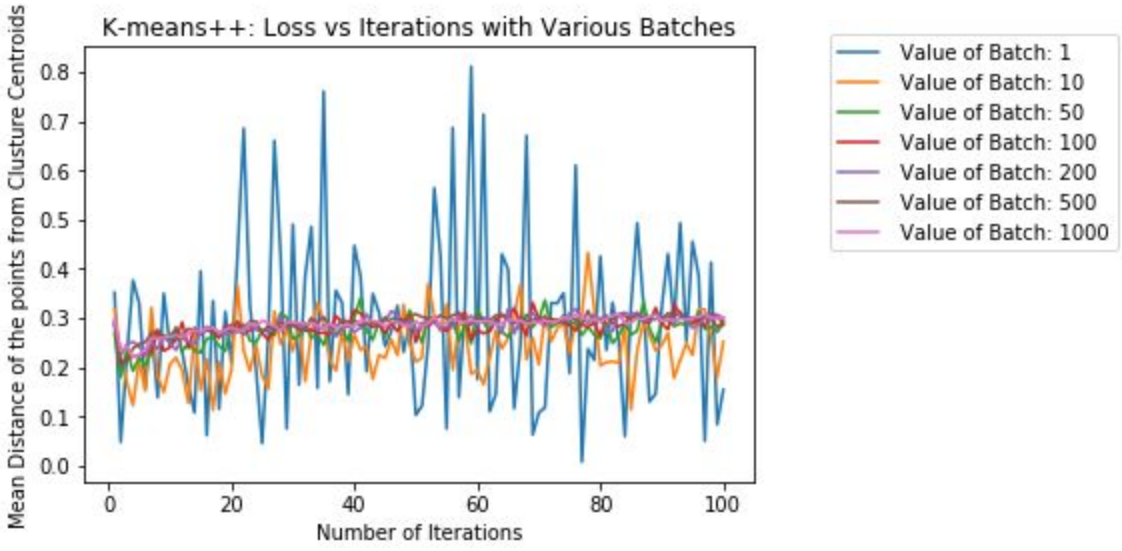
Problem 3:

Objective: Implement the k-means++ initial centroid initialization

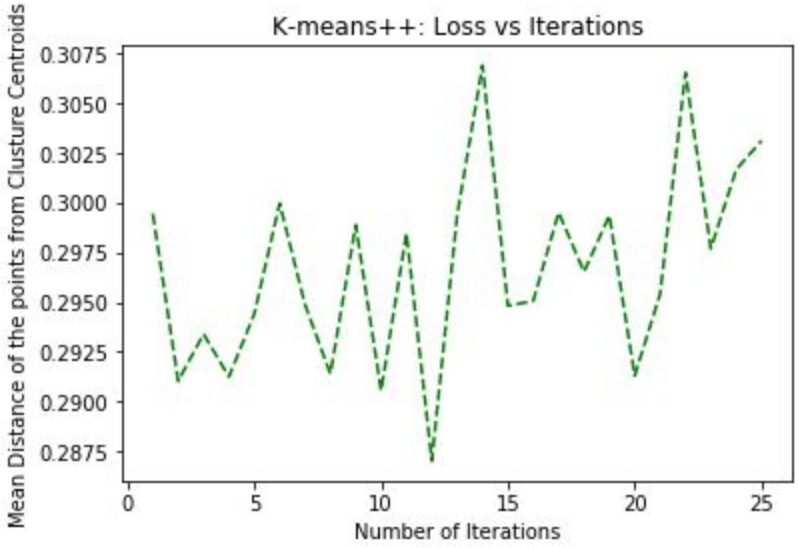
Solution:

- Choose a data point and initialize it as the initial centroid
- Calculate the probability function of each point in the dataset with the centroid
- Sample the new centroid from the calculate probability distribution
- Repeat the steps for N centroids

Results:



At k= 500, 1000 we get very less deviation in loss at different time steps. Since 500 gives similar results as compared to a batch size of 1000, it is better to choose a batch size of 500 for faster computations



As compared to k means with random initialization with kmeans ++ initialization the initial loss is very less. With kmeans ++ we are close to the optimal loss at the initialization step itself. Hence, the loss is consistant over ‘t’ iterations. The loss at the end of the ‘t’ time steps is similar to Kmeans with random initialization. We can say that kmeans ++ converges faster due to intial seeding optimization

Loss at each time step

array([[0.3087949], [0.29840997], [0.30745345], [0.30108392], [0.30157947], [0.30055851], [0.29896767], [0.30318801], [0.30048864], [0.29897627], [0.30758161], [0.30099028], [0.30680965], [0.30049549], [0.30411831], [0.30025396], [0.29889503], [0.31029407], [0.30240155], [0.29764345], [0.30332067], [0.29706884], [0.30384958], [0.30604701], [0.31023215]])

Due to the initialization algorithm for selecting centroids, we choose the centroids which are close to the global minima. Hence, there is not much decrease in loss at each time step.
The value of loss at the final time step is close to the convergence value for random initialization of centroids.
We can conclude by saying that Kmeans ++ initial seeding is much more efficient than random seeding

Problem 4:

Objective: Implement our own initialization algorithm

Solution:

For faster initialization of the centroids the algorithm selected must approximate the posterior distribution of the dataset with very few passes over the data. Some of the algorithms which give good approximation of the distribution are the random walk Monte Carlo algorithms(Metropolis–Hastings algorithm, Gibbs Sampling, etc). These algorithms make use of the markov chains algorithm for a quick and efficient approximation.

Hence, implementing an algorithm which makes use of markov chain would give us the best results.

We would like to approximate the probability distribution of our data by constructing a sequence of markov chains and passing through them

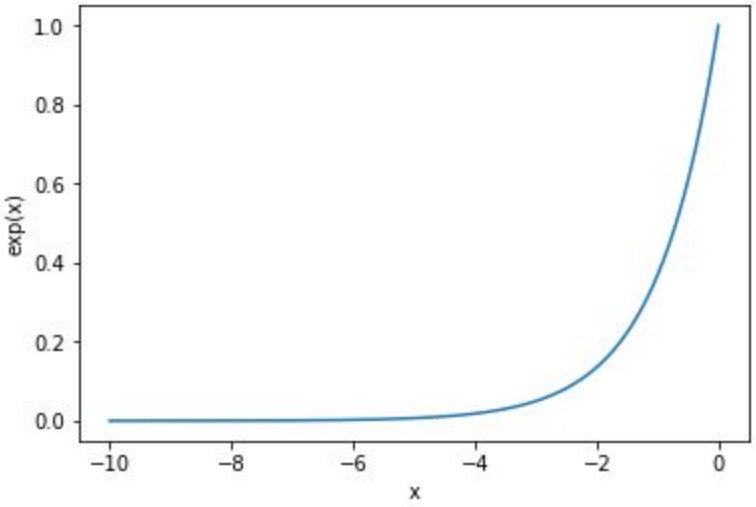
Algorithm:

- Initialize an arbitrary centroid which lies on the data at random
- Compute the probability distribution of all the data points with reference to the assigned centroid
- Pick a point at random from the data set to start the markov chain. This point will act as the initial state in our sequence of points
- Sample the next state form the dataset based on the probability distribution. Compute the probabilities of staying in the same state and moving to the next state
- Depending on the probabilities move to the desired state by introducing randomness in the decision

Our Contribution:

- Probability of State change:
    - The state change or stay probability is computed by taking the sign of the distance between the present and future states to their respective centroids
        - dist $(x_{j+1} - x_j) = D(x_{j+1},C) – D(x_j,C)$
          where D(x, C) is the distance square of the point x from the known centroid set C.
    - This gives a good estimation of movement in markov chains to different chains at time ‘t’. The estimation is similar to the Metropolis Hashing technique and gives similar results.
    - By comparing the difference between the distances and taking the sign we can say which state is farther to the centroid based on the sign value. The new state

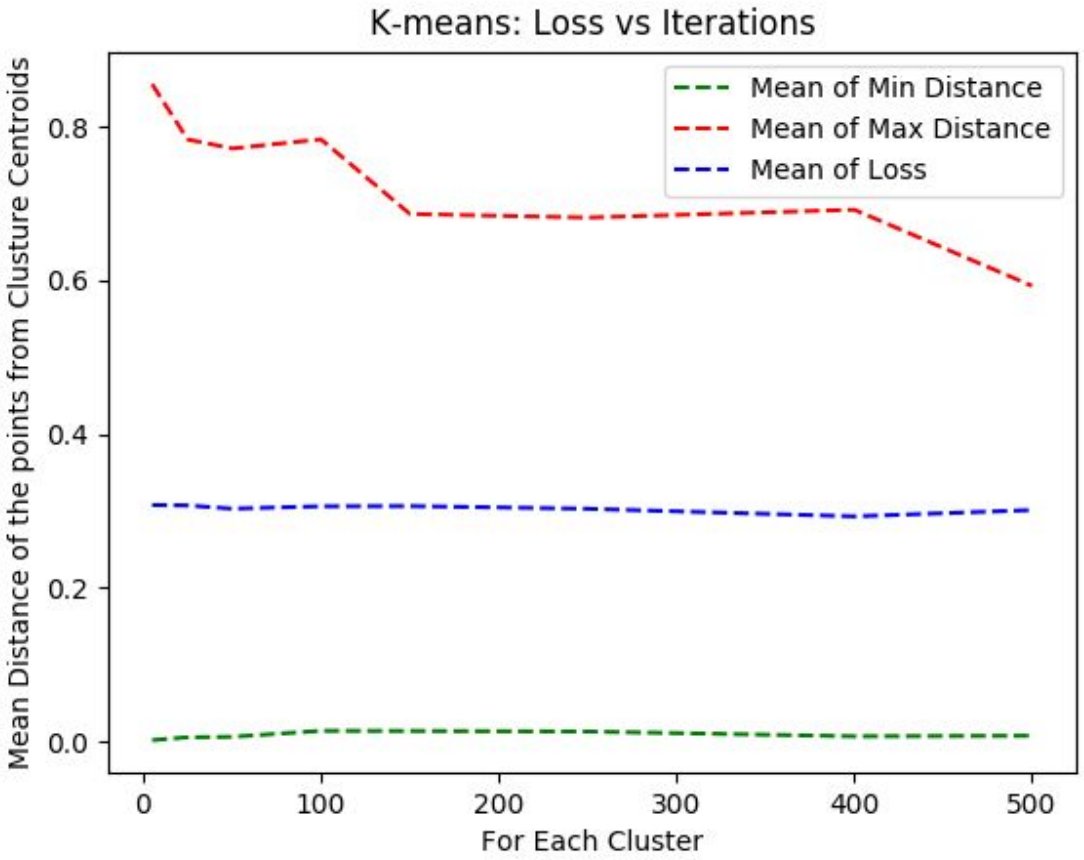
will be assigned based on the sign value which gives a good approximation of the distances to the centroids
    - The points with huge distances and with negative sign are mapped with the function e^x. This maps (-inf, 0] to [0,1]
- Update Rule
    - The difference is computed between the next state and the current state in that order. If the distance is positive, it means the next state is farther to the centroids than the current state. Hence, we move to the next state
    - If the distance is negative the current state is farther than the next state. Hence, we stay in the same state. The distance here is mapped from (-inf, 0] to [0,1]. We then sample from uniform distribution to decide the change in state based on the mapped distance values.
        - $p(x_{j+1}/x_j) = \exp(dist(x_{j+1} - x_j))$.
        - If the sampled uniform probability is less than the mapped distance, we move to the next state else we stay in the same state



The graph shows how the distances are mapped after passing the function e^x

Problem 5:

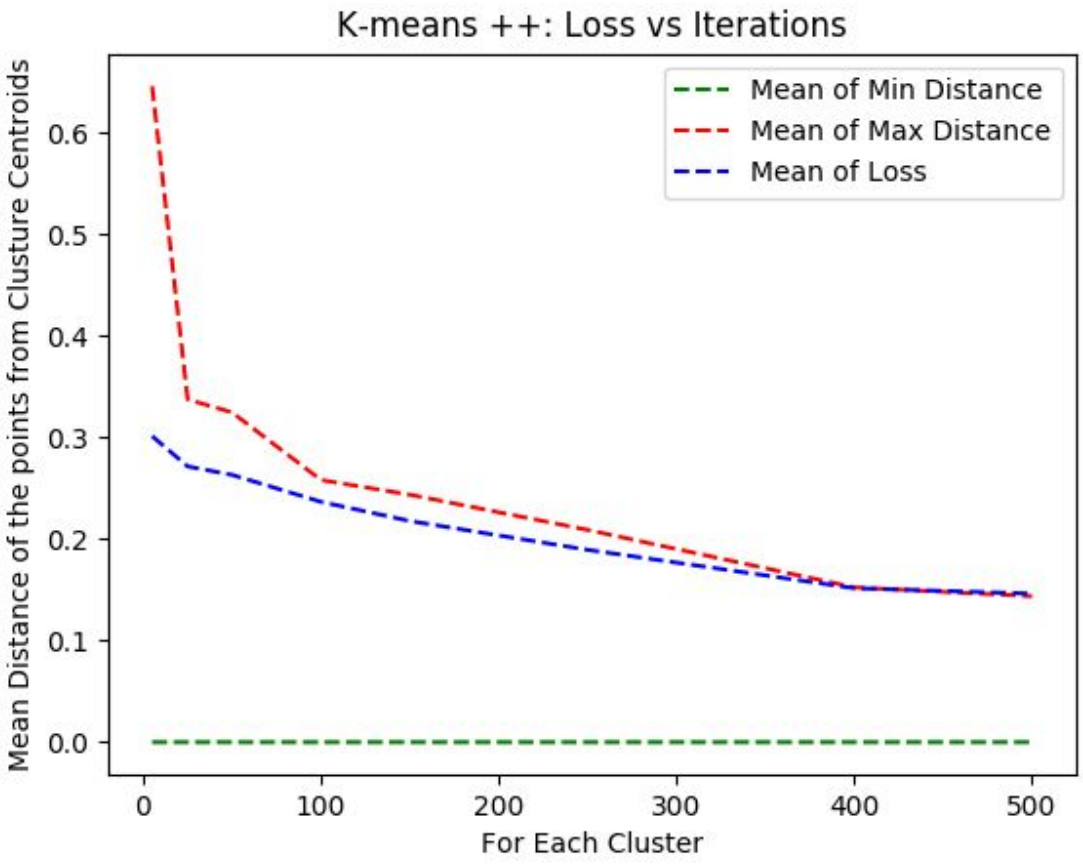
K-means with random initialization of Centroids



Inference:

- From the graph we can observe that the max distance of a data point in each cluster is decreasing for increasing centroid number. As the number of centroids increases the clusters are increased which in turn decreases the size of each cluster. Hence, the max distance changes with each increase in cluster size decreases.
- The minimum value and loss over varying centroid count remains constant as the size of the cluster has very little impact on these

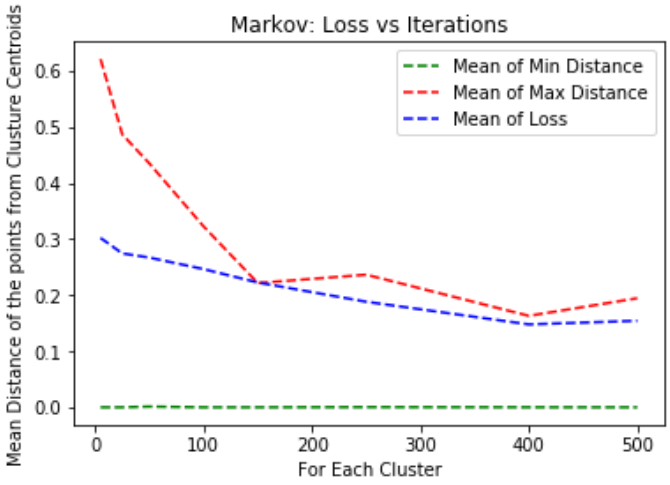
K-means++ with Optimized initialization of Centroids



Inference:

- It can be inferred that the loss decreases steadily for different value of k (number of centroid).
- When compared with kmeans the values of the loss are less. This can be inferred such that kmeans++ initial seeding gives better convergence because it converges to global minima.
- The max distance of points in each cluster decreases with increase in k because the cluster size decreases with the increase in k. With decrease in cluster size the data points very close to the centroid are clustered together. Hence, the max distance decreases

Mini Batch Kmeans Using our own Initialization of Centroids



Inference:

- It can be observed the loss and other characteristics such as min and max over k centroids is similar to the Kmeans ++ algorithm. This serves as a good indicator that the proposed algorithm is compaarable to kmeans++.
- The algorithm proposed is much faster than Kmeans++ since we approximate the probability distribution with just a few passes over the data for each centroid.
- The proposed method which uses markov chain approximation takes 42 seconds with a batch of 500 data points to converge. For the same batch value, Kmeans++ initialization takes 218 seconds. This difference is largely due to the initialization of the centroids.

From the above graphs we can see that the loss is minimum when the initialization is done by using markov chains. Across ‘k’ values(number of clusters) the loss is consistently decreasing. Hence, the proposed algorithm is better in terms of accuracy and time