ESE 545: Data Mining **Fall 2018**

Project 3 Writeup

Name: *Joseph Lauigan*

1. **Problem 1**

   The clustering problem for this project will be to understand higher level trends and correlations between four different aspects of a Yelp reviewer's online presence: Persona, Prose, Information, and Longevity. I will be defining four new features (based on some of the current features of data) that will serve as high level descriptors of a user's reviews/prescence on yelp:

   (a) "Persona" - Yelp Community's perception of the person's personality. Combination of compliments (hot,profile,cute,cool)

   (b) "Prose" - Yelp Community's perception of the person's writing style. Combination of comments (funny, cool) and compliments (plain,funny photos). Since comments are more passive then compliments, comments will be weighted appropriately to reflect this.

   (c) "Informative" - Yelp Community's perception of how informative a user is. Combination of comments (useful) and compliments (list,write,more,photos)

   (d) Activity/Consistency - user's rate of activity. review counts/(current year - yelping since) (in days)

   These four traits will be used in addition to user id and elite as the basis vectors for clustering. Please see Jupyter notebook for exact implementation of feature construction.

2. **Problem 2**

   For the online K-means clustering algorithm, a min-batch approach was used. The standard min-batch K-means algorithm has a set iteration number, so the free parameters for this implementation are the min-batch size and the number of iterations. A sweep of each free parameter was done (while holding the other to a constant number) and the avg. cluster distance was determined for comparison. For standard K-means++, the **optimum min-batch size is 10** and the **optimum iterations is 20**.

   In addition to a set iteration number, I also implemented a stopping criteria approach which monitors the relative change of the centroids in each round. If the change in the centroid position is less than one percent for all the centroids, the algorithm will stop updating. The min-batch size is still a free parameter in this approach, and thus this was swept through and the avg. cluster distance was used as a metric for comparison. Again, it was found that the **optimum min-batch size is 10**.

3. **Problem 3**

   For the implementation of the K-means++ seeding, a recursive approach was taken, with the definition of the recursive function `k_means_seeding` was defined. This function determines the farthest point from two given centroids, and takes in three input arguments:

   (a) centroid0 distance - array of distances values for all data points for the first centroid passed in

   (b) centroid1 - vector corresponding to the second centroid, which is then used to calculate another distance array

   (c) remaining K - number signifying the remaining amount of centroids to be seeded.

   `k_means_seeding` recursively calls itself until the last centroid starting point is defined, then returns 1. In the main code, the first two centroids are determined by direct computation, then they are passed into the `k_means_seeding` function. The runtime for this function to seed 500 centroids is **6.95 minutes**

4. **Problem 4** For the custom seeding implementation, the main idea is to use numerous random subsets of data to stochastically try to find separated starting points for the centroids. This amounts to taking random subsets of the data, finding the two farthest points within that subset. Then for the next rounds, taking another random subset, finding the point that is farthest from the previously determined centroids, then adding it to the list. The overall process is defined below:

   Parameters:
   n - size of dataset
   K - number of centroids to seed

   Process:

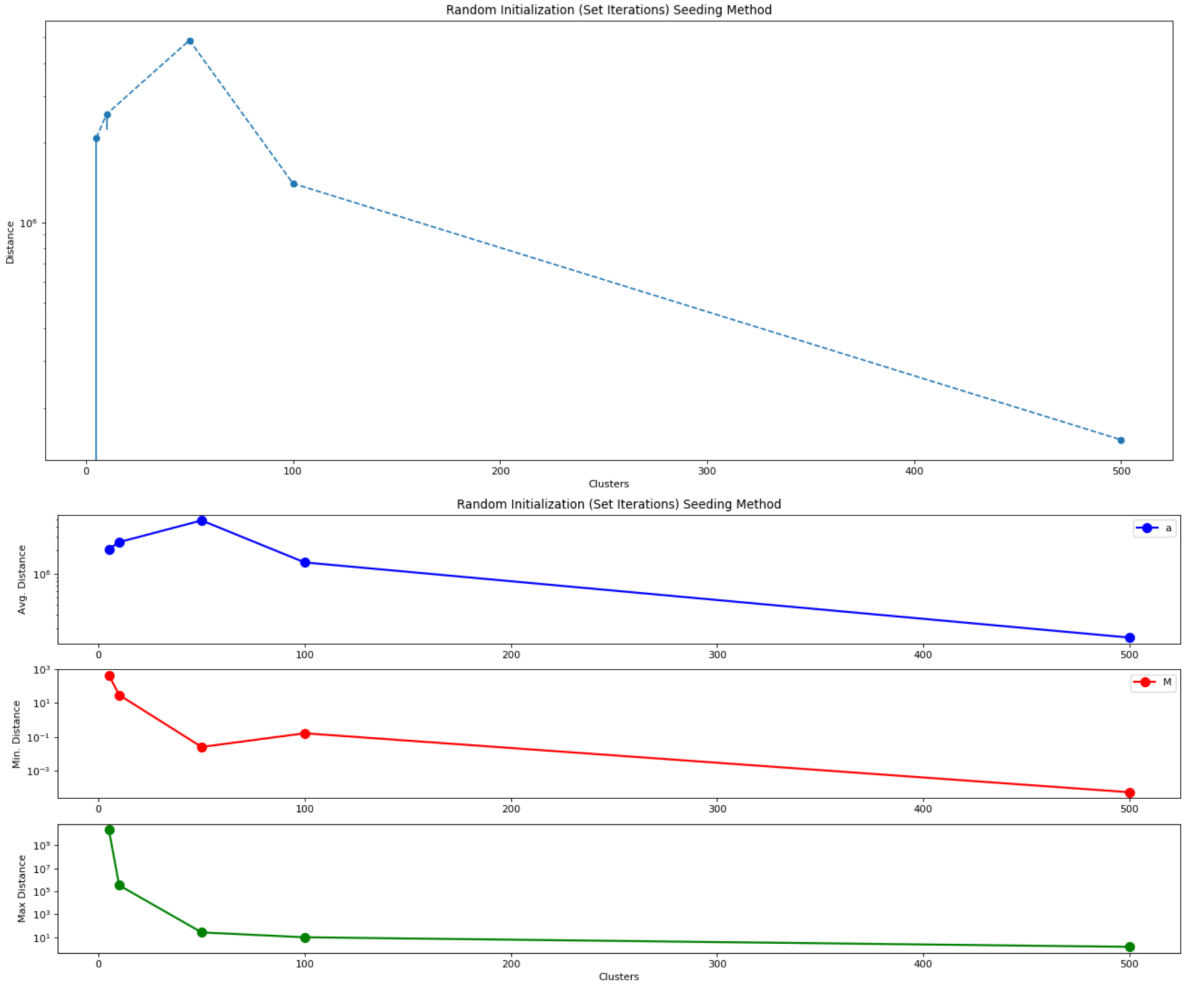
   - Initialize prev-centroids-dist = 0. prev-centroid-idx0 = prev-centroid-idx1 = 0
   - Initialize centroid-list= K*[0]
   - Initial centroid is random point
   - For K centroids:
     - If K = 0: (first/second centroid)
       - For gamma rounds:
         - Create two subsets of (n/K) datapoints.
         - Brute force search of points with farthest distances within the two subsets (point0, point1)
         - If current-dist > prev-centroids-dist --> Set prev-centroids-dist = current-dist. prev-centroid-idx0 = point0 and prev-centroid-idx1 = point1
         - set centroid-list[0] = point0, centroid-list[1] = point1
     - Else: (for rest of K-2 centroids)
       - for Gamma rounds, create a random subsection of (n/K) datapoints and calculate distance from the K-1 centroids already determined
       - save the one with farthest average distance from those points.

5. **Problem 5**

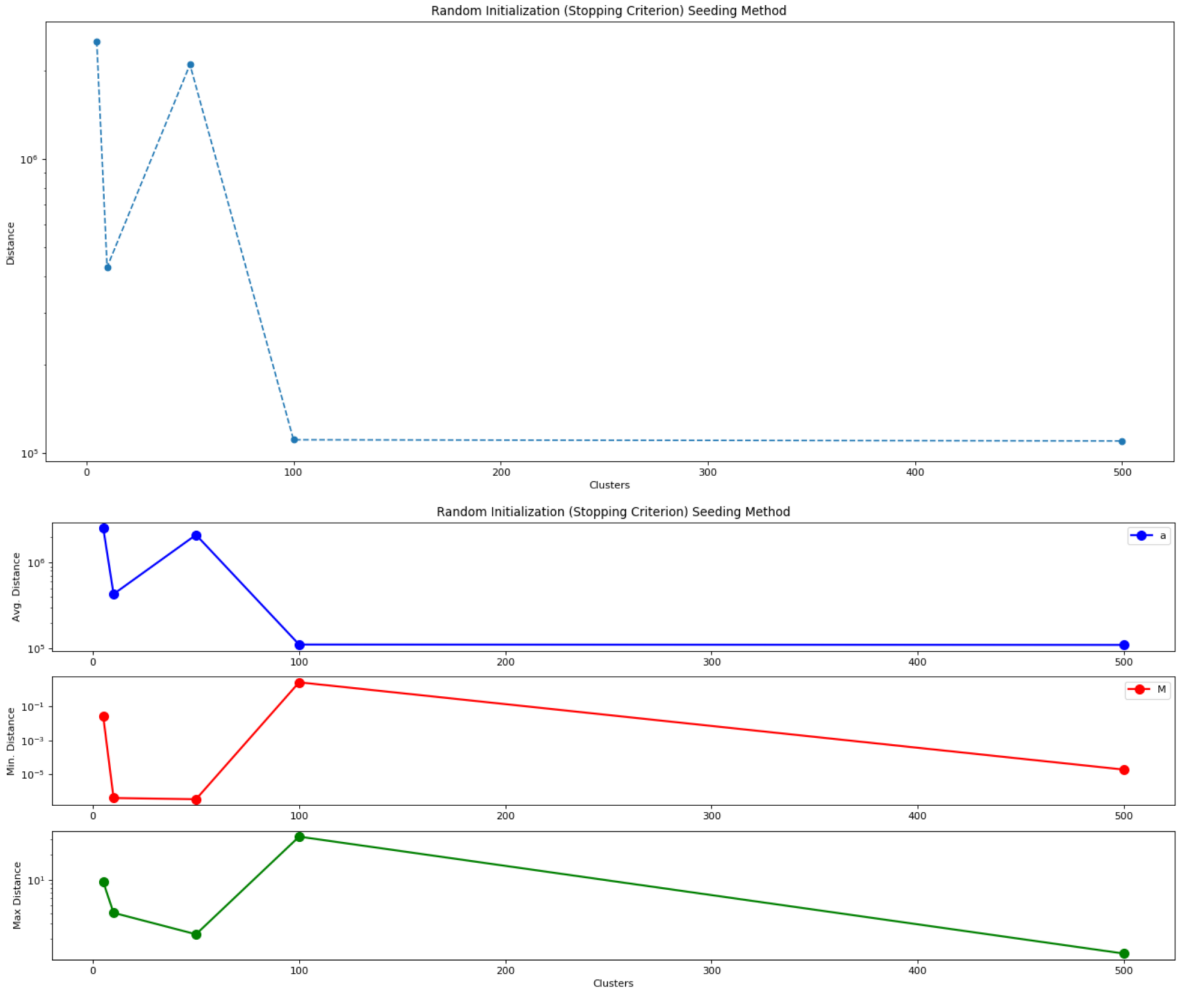
For the comparison between initialization methods, the average distance to each centroid as well as the minimum and maximum distance for each centroid (averaged over all centroids) is determined. Data each initialization method are presented in two plots: one that plots the avg. distance for a given number of clusters with the min/max as error bars, and the second is a multiplot that plots the avg, min, max distance in their own respective graph as a function of the number of clusters. For the purposes of avoiding extremely long runtimes, five numbers of clusters (5,10,50,100,500) were used to sweep.

In analyzing the results and comparing the average distance to each cluster, surprisingly the random initialization with a stopping criteria with K = 500 gave the smallest average distance. Both approaches to random seeding had smallest average distances on the order of 10^5 for K - 500. K-means++ and the custom seeding algorithm gave the smallest average distance on the order of 10^7 for K=500. This result is surprising because K-means++ and the custom algorithm incorporate intelligence to the seeding process and should in theory create better starting points for the K-means algorithm. But all four algorithms are consistent in pointing to K=500 to be the optimal number of clusters, so for this clustering problem, there isn't much correlation between the features constructed.

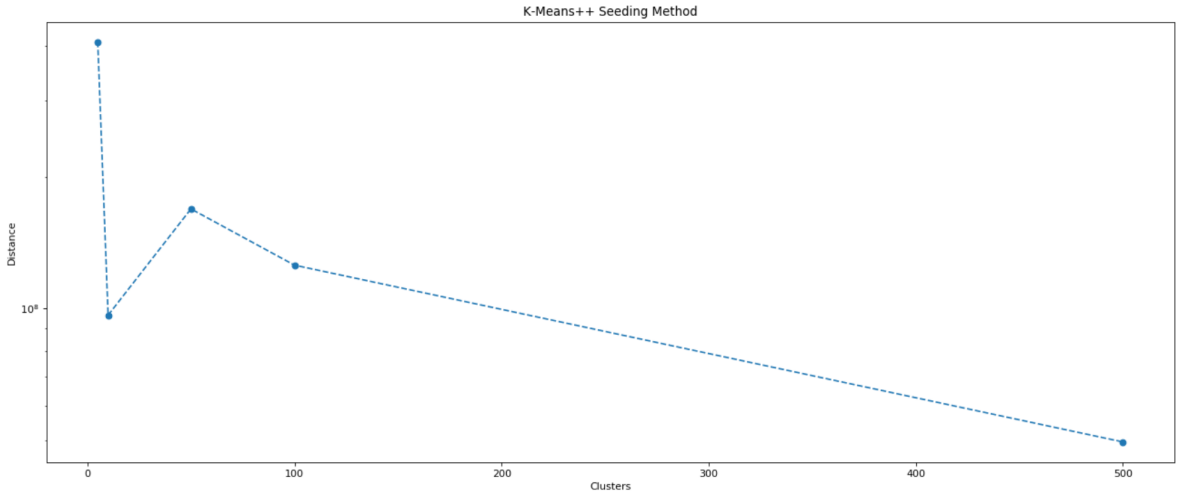
(a) **Random Seeding - Set Iterations Method**

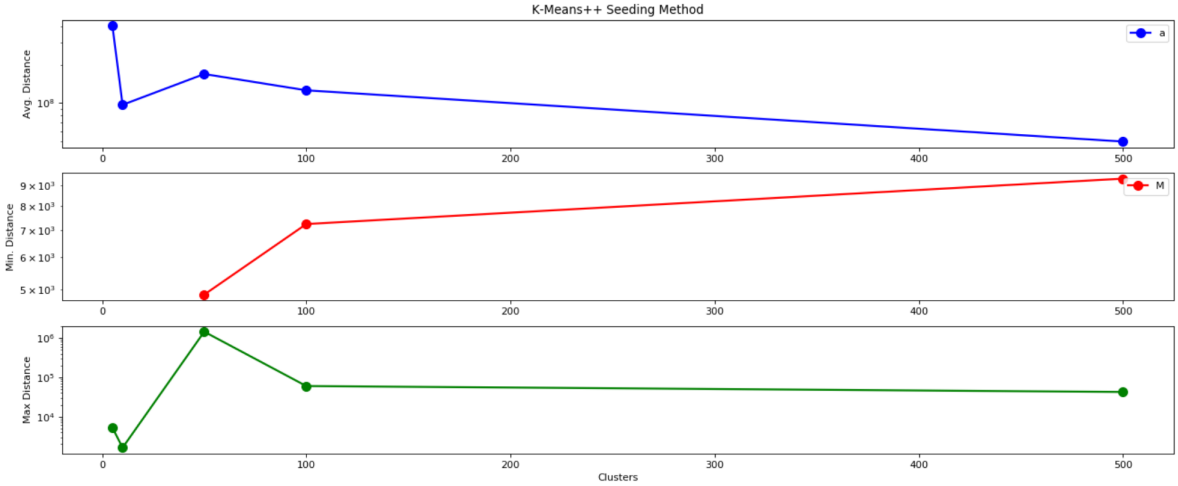


(b) **Random Seeding - Stopping Criterion Method**



(c) **K-Means++ Method**





(d) **Custom Method**