Gil Landau
11/24/18
ESE 545

# Project 3 Report

## Part 1:

My clustering problem tries to cluster together users by desirable characteristics and create a classification for ideal users (engaged users, popular users, etc.).

**Features:**

I use some of the received compliments and the number of fans.

I have scaled the comment compliments received (hot, cool, etc.) to be their ratios over the total reviews, so it is the average score per review. In a real world scenario, it would be useful to understand which users are **positively** engaged in the website (e.g. the number of cool compliments they have), and then promote the reviews of those users (or demote them if they are troll accounts).

I also scale down the number of fans a user has, in order to avoid clobbering the other features. The "number of fans" is now the "number of fans (in thousands)" and this should give more weight to the number of fans at a more reasonable rate.

In order to reduce the burdensome memory requirements, I run the features through SKLearn's PCA and Standardization Scalar. I use the first four principal components, which account for 96% of the variance.

## Part 2:

You can view the code for this part in the section labelled "Part 2." This is a simple implementation of the online mini-batch K-means algorithm from the lecture. It takes as inputs: random points as the initial centroids, the number of iterations, the batch size, and the data matrix.

## Part 3:

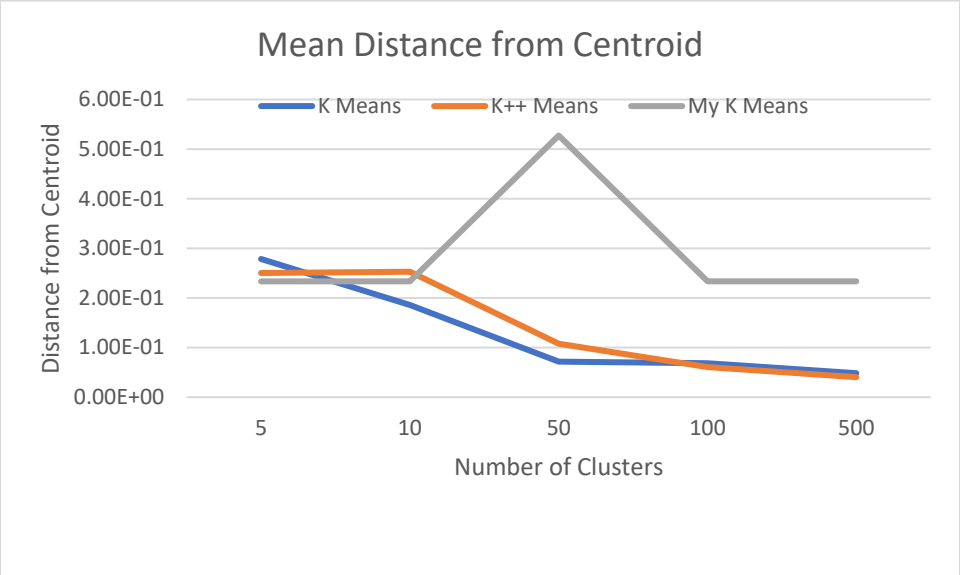
You can view the code for this part in the section labelled "Part 3." This is a simple implementation of the K-means ++ algorithm from the lecture. It is split into two parts: the k-means++ initialization and the standard K-means algorithm (see above).

## Part 4:

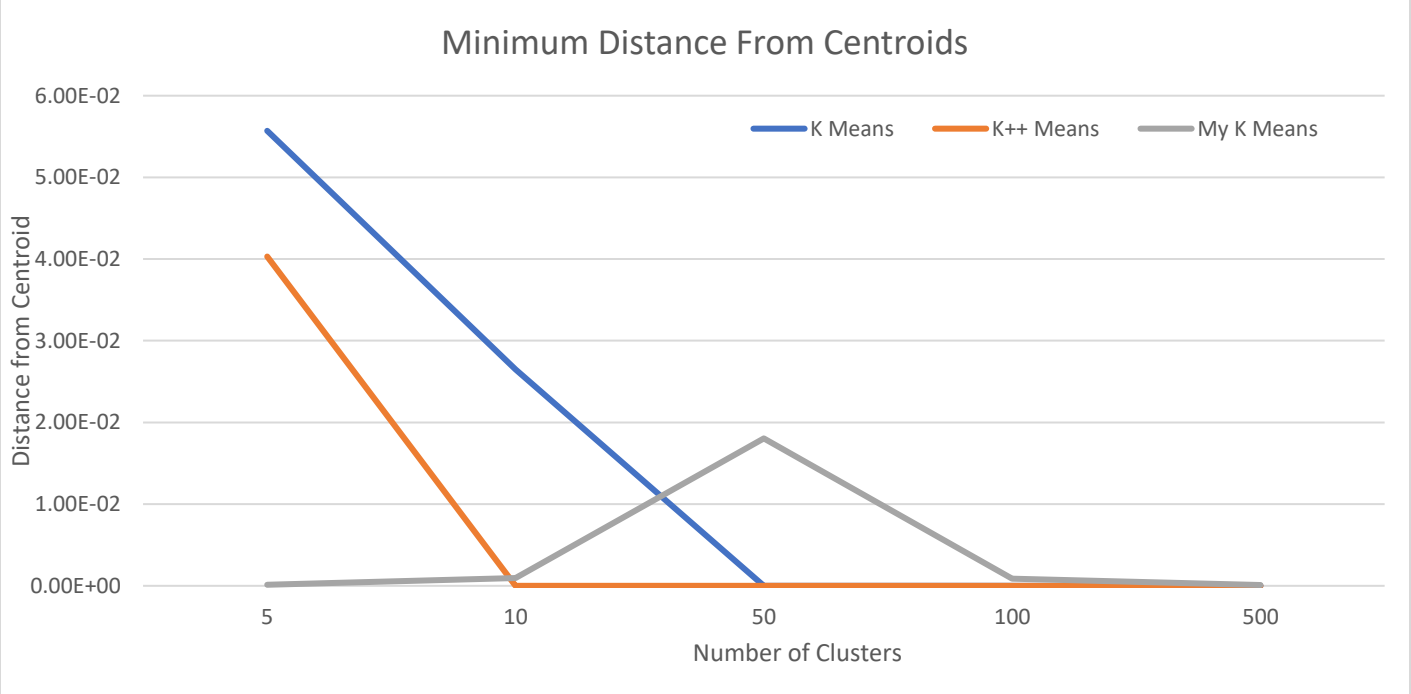
You can view the code for this part in the section labelled "Part 4." My centroid selection algorithm focuses on dividing the user base into extremes of behavior, spiraling to the center. My hope is that I can characterize users by which extreme they are closely aligned to. As the number of centroids increases, it allows me to capture more and more complex behaviors (as users become less aligned to a given feature). The initial four centroids are the four extremes (where one dimension is the maximum value found in the datasets, and all the other dimensions are their respective minimum values). I then initially select a point at random as a fifth centroid. Then, on each iteration of the loop, I compare the current centroid with all previous centroids I selected. I pick the previous centroid with the maximum distance between the two and pick the midpoint. My hope is that I will cover any missing clusters in the vast middle of the extrema.

## Part 5:

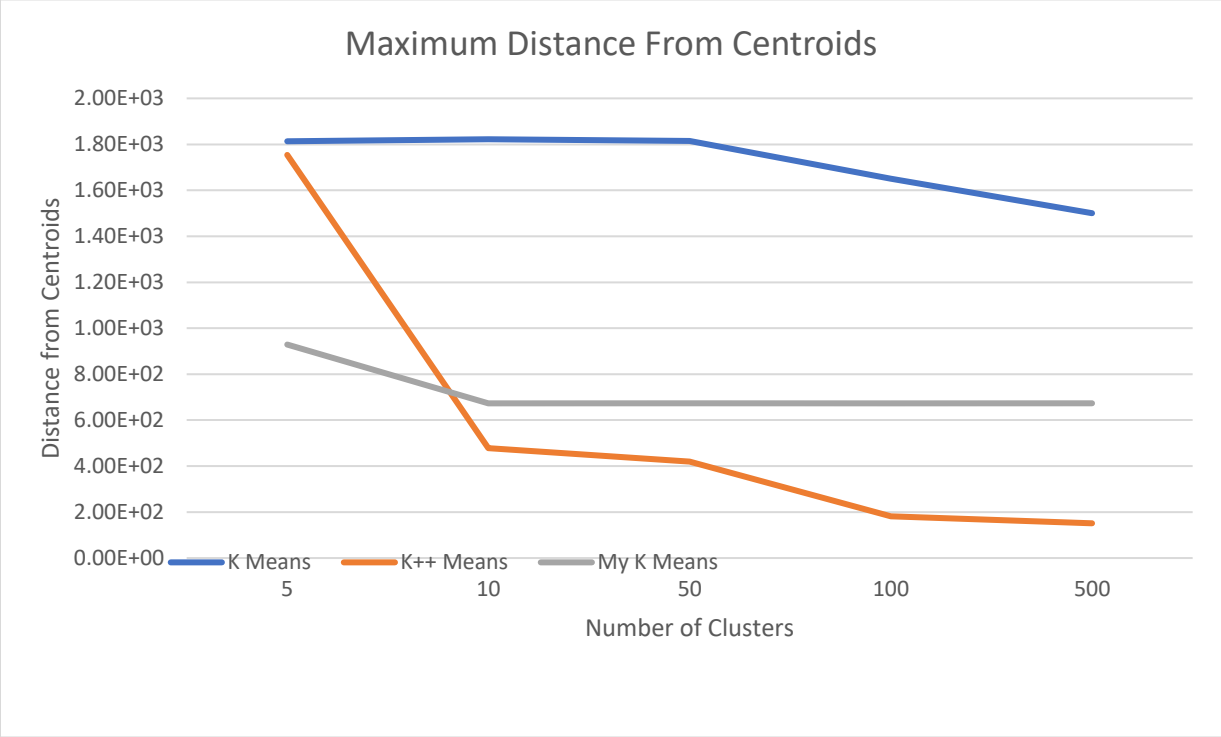
**Mean Graph:**




**Minimum Graph:**



**Maximum Graph:**




The *k* with the lowest error is the K++ means algorithm. I think the main difference between the initializations is how quickly they converged to a distance and how low that distance. Since this problem is highly non-convex, it can become very easy for centroids to become stuck on local minima. In my initialization especially, it became clear that the centroids became stuck at a certain value. K++ means is probably the best initialization. It not only converged the quickest, but also to the lowest errors across the board.