Abstract

K-means clustering is a clustering algorithm that can be used to group data into k clusters. Its applications include segmentation, document clustering, and image compression. I employ two different methods for k-means: the first is to randomly pick the initial centers, and the second is to pick the initial center randomly, and then pick the i-th center to have an average distance from the previous i-1 centers that is maximal.

Implementation

The first strategy for initialization is to randomly pick centers. The second strategy is the following. First, I pick an initial centroid randomly. Then, I iterate from 1 to k to pick k centroids. For every point, I calculate the average distance to the other i-1 centroids. Whichever point has the maximum average distance will be initialized as a centroid. It's also important to note that I must skip a point with continue if the point chosen is already a centroid. Based off the results below, the loss computed for the second strategy is less than that for randomly selected centers. This is expected as centers that are farther separated will tend to produce better results due to better coverage. K-means algorithm has two main steps after initialization. The first step is to classify the points based on the nearest mean. Second, each mean must be recomputed.

K-means stops iterating when there are no changes in the means. Therefore, I used a variable called prev_i_point1 and set it equal to i_point1 at the start of the iteration. If there is no difference between prev_i_point1 and i_point1, k-means ends. I used the following code to see if there is a difference: while (prev_i_point1 != i_point1).any(). After calling assign_centroids and compute_centroids, I printed out the means for every iteration and the loss.

Results

My k-means was run with the following parameters for the last four digits of my student id '61 28'. For strategy 1 I used
k1 = 3, i_point1 = [[1.72614408, 6.81819407],[5.02471033, 8.23879873],[5.57009665, 8.3870942]]
k2 = 5, and i_point2 = [[6.12393256 5.49223251],[2.20011496,1.53863221],[5.33498937,3.07430754],[1.3483716,3.96379638],[2.3537231,6.29810755]].
For strategy 2 I used
k1 = 4, i_point1 = [[5.32508246,7.68399917],[1.20162248,7.68639714],[3.04101702,-0.36138487],[9.26998864,9.62492869]]

k2 = 6, and i_point2 = [[4.50236445,2.9288804],[9.26998864,9.62492869],[8.87578072,8.96092361],[2.95297924,9.65073899],[6.5807212,-0.0766824],[3.85212146,-1.08715226]]

The final centroids for strategy 1 are:
K1 = 3, [[2.62276163,4.39601542],[5.35168695,5.12057986],[7.80248845,5.52193008]] with cost = 2289.994781163356

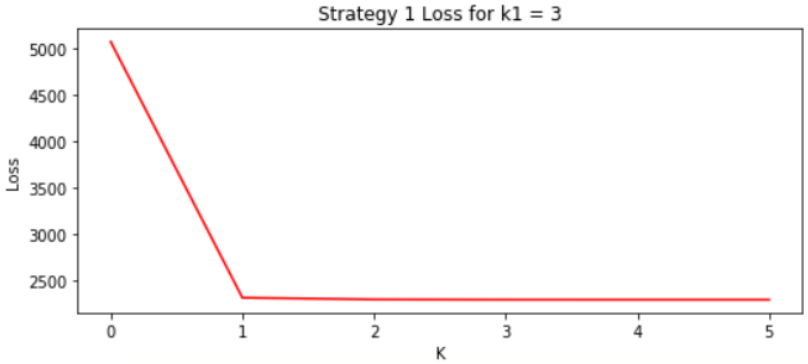
K2 = 5,
[[6.57957643,7.57333595],[3.14506148,0.90770655],[7.41419243,2.32169114],[3.49556658,3.56611232],[2.79053386,6.99912704]] with cost = 649.9115453101182

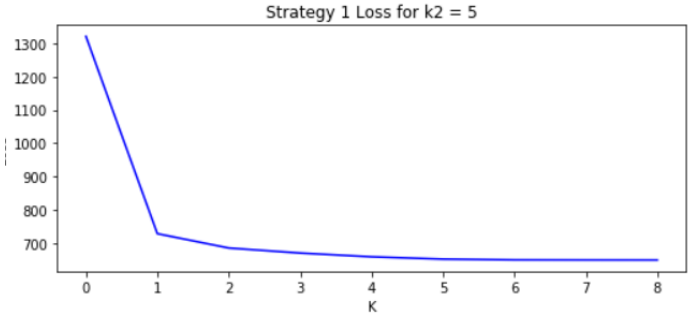
The final centroids for strategy 2 are:

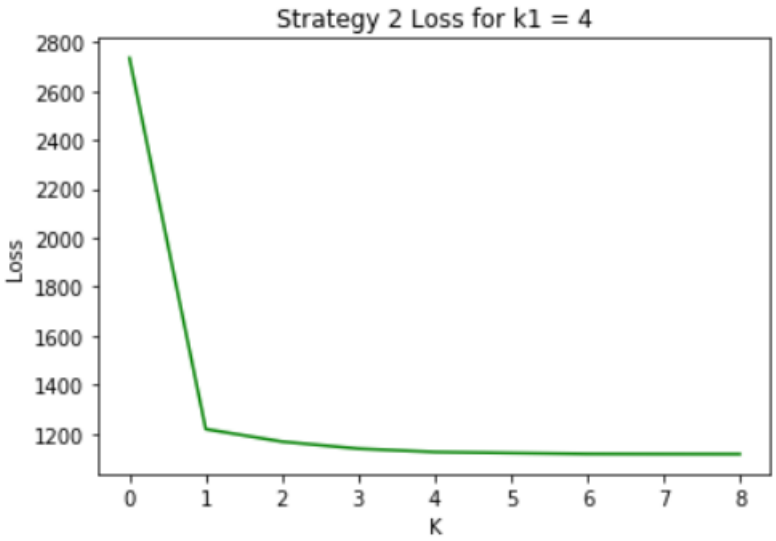
K1 = 4,

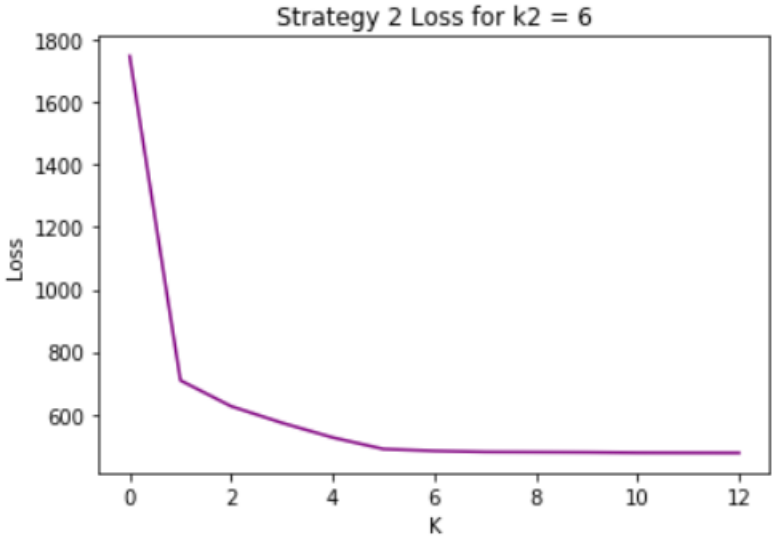
[[5.51381131,6.6871291],[5.46810558,2.23600141],[7.75648325,8.55668928],[2.45162074,6.08990448]] with cost = 1116.2626578188188

K2 = 6,
[[3.502455,3.62870476],[3.14506148,0.90770655],[7.75648325,8.55668928],[5.46427736,6.83771354],[2.52382885,7.02897469],[7.41419243,2.32169114]] with cost = 476.2965705269664









The loss for strategy 2 was less than that of strategy 1 for their respective k's. Also, as the number of clusters increase, there is a decrease in loss. These two facts highlight the fact that when there is better coverage with the centroids, the loss function decreases.