Statistical Machine Learning Project 2

Arnav Chakravarthy
ASU ID: 1216873944

Required Tasks

The specific algorithmic tasks I needed to perform for this part of the project included:

- Initialize clusters using Strategy 1 and 2. Strategy 1 involves randomly picking the initial centers from the given samples. Strategy 2 involves picking only the first point randomly and picking points with ith center (i>1) such that the average distance of this chosen point to all previous centroids is maximal.
- Test the implementation with the number k of clusters ranging from 2-10.
- Plot the objective function value vs. the number of clusters k. Under each strategy, plot the objective function twice, each start from a different initialization.

1. Initialization

1.1. Strategy 1

I have used the "NumPy" library for the Python programming language. It adds support for large, multi-dimensional arrays and matrices.

And for this whole project I have used vectorization to make calculations more efficient.

Each sample is represented as a 2D Vector.

I have used the random.sample to choose "k" random and unique points to avoid duplicate clusters.

I have then stored this in a Python list called all_centers. The shape of this would be (k x 2) Where k is the number of centroids.

1.2. Strategy 2

For this strategy, I choose the first point randomly using random.randint.

For the rest of the centroids, I built a function called *calc_point_distances()* which calculates the average distance of each point to the previously calculated centroids, stores them in a list in decreasing order and returns them.
I now iterate through this list until I find the first index which is not already present in *all_clusters*. Hence I calculate the point with the maximum average distance which is not already present in *all_clusters*. The reason for doing this and not just returning an argmax is because after considering a certain number of centers, the future centers get repeated. Hence this following method gives us unique centroids.

2. K-Means Algorithm

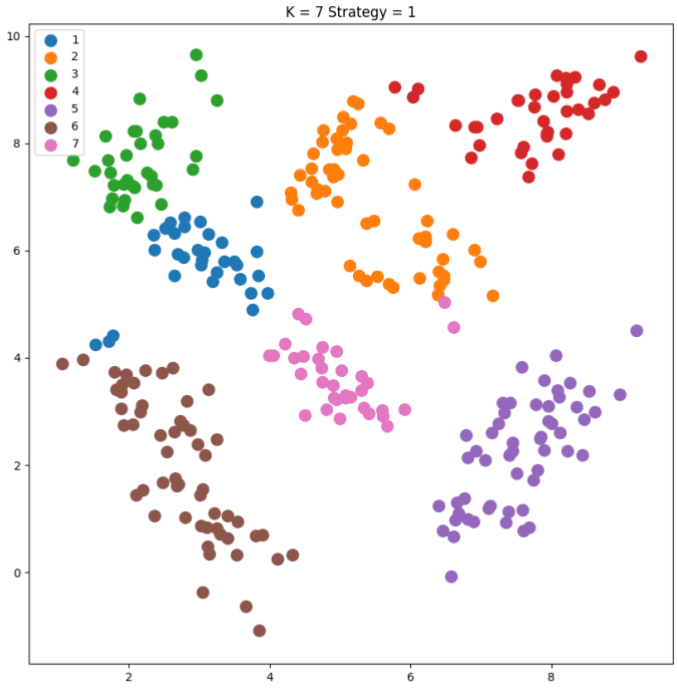
I created a Python dictionary called *center_points_dict* which stores the cluster index as the key, and the points present in the cluster as its value. The key is an integer and the value is a list of lists.

For every iteration of KMeans these points get updated as well as all_clusters get updated.

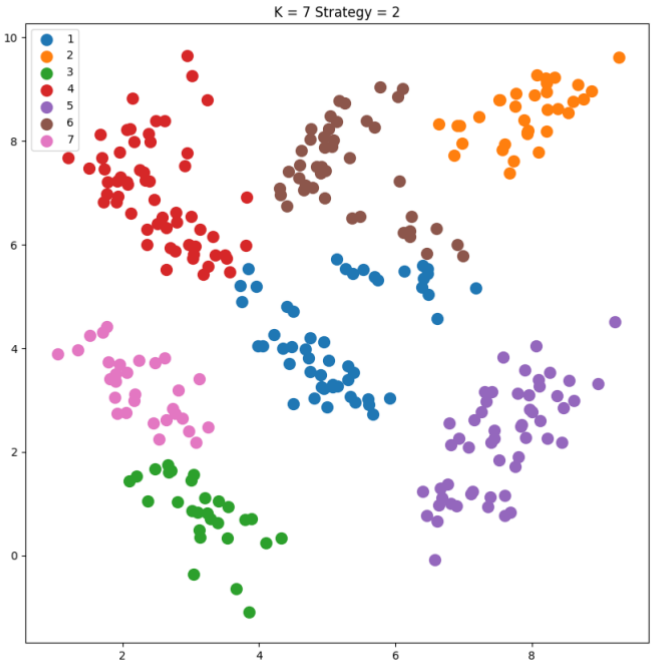
I have used Euclidean distance to assign a point to a cluster such that the Euclidean distance between the point and the centroid of the assigned cluster is the least among that of all the other centroids.

After each iteration I update all_clusters by updating each centroid as the average of all the points present in that specific cluster. These points can be retrieved from *center_points_dict*.

I also created two plotting functions to visualize the centroids as well as all the points in their respective clusters after performing K-Means.



K=7, Strategy 1



K=7, Strategy=2

3. Euclidean Distance

$$d(\mathbf{p}, \mathbf{q}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2}.$$

P and q are the 2 points in a 2D space. q1 and q2 are the x and y coordinates respectively of point q and p1 and p2 are the x and y coordinates of point P respectively.

This is the distance I used to calculate which points belong to which cluster.
For every point I checked the Euclidean distance between the point and all the cluster centroids. The point was assigned to cluster with whose centroid it had the least Euclidean distance.

4. Objective Function

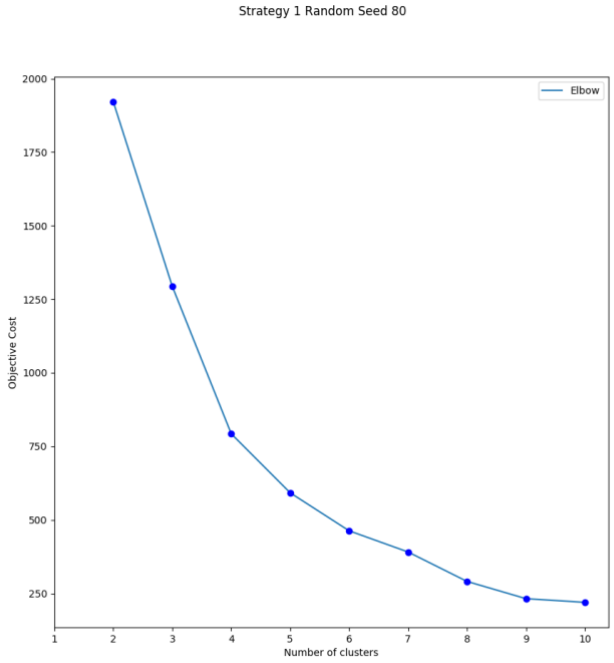
$$\sum_{i=1}^{k} \sum_{x \epsilon D_i} ||x - \mu_i||^2$$

k – number of clusters
D – Total Sample set
D_i- Samples in cluster i
μ_i – Centroid of cluster i.

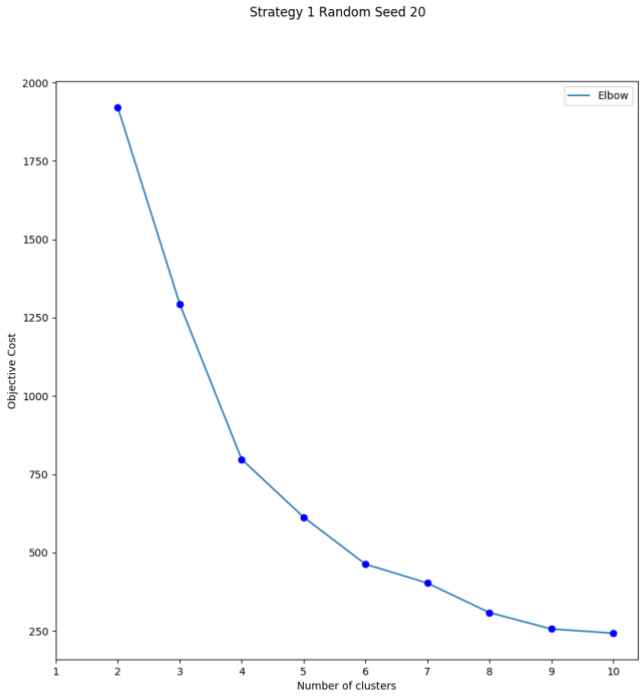
5. Plotting the Objective Function vs Number of Graphs

I used matplotlib as my plotting library of choice.
I first created a function called *compute_all_objective_costs()*, which calculated the final objective function cost after convergence for clusters ranging from 2-10, and also take the strategy as an input.
Since I created KMeans as a class, I just had to pass in number of clusters, strategy type, maximum number of iterations and the dataset path as parameters to the class instance.
I was then able to get the final objective cost for each instance each time with different number of clusters and a specific strategy type.
These costs were stored in a list and returned.
I then used matplotlib to plot these costs on the Y-axis and the number of clusters on the X-axis.

Strategy 1

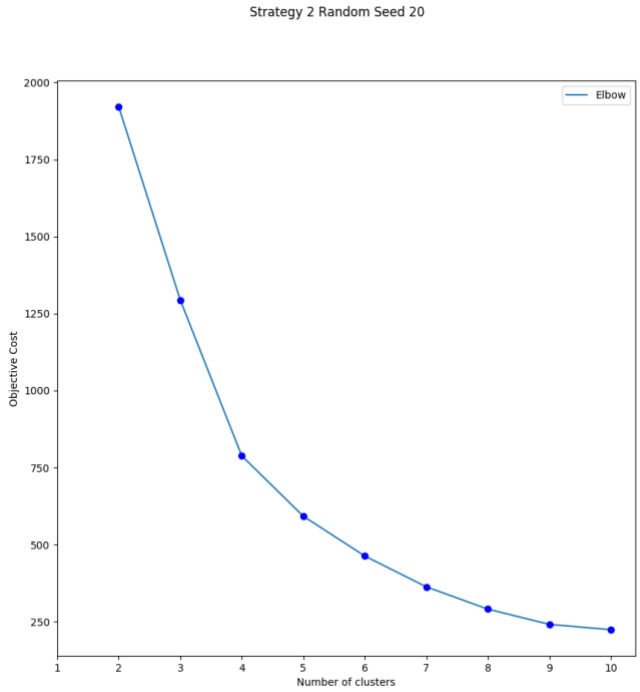


Random Seed 80

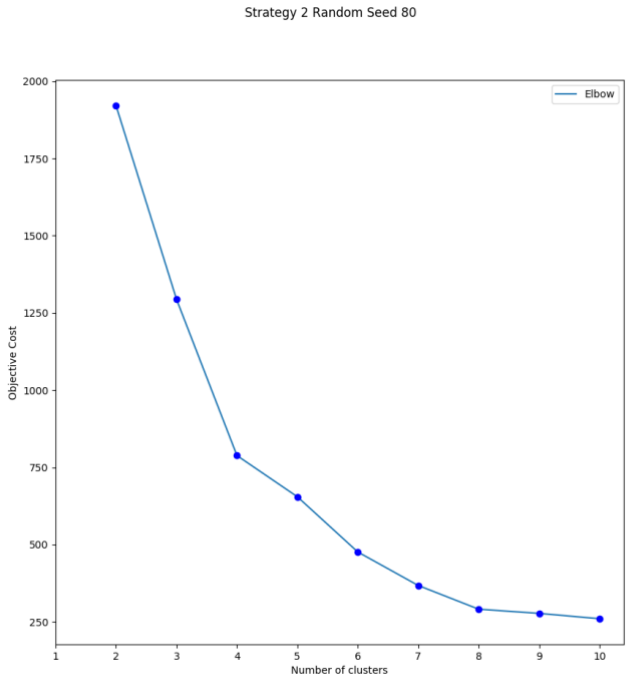


Random Seed 20

Strategy 2



Random Seed 20



Random Seed 80