K-Means Algorithm and Automatic k Selection

Mustafa Ocak

1 K-Means Algorithm with Automatic k Selection

In this section, I will describe the idea of the automatic k selection using the elbow method for K-Means algorithm.

1.1 Main Idea

The selection k value for K-Means Algorithm is chosen manually by plotting the WCSS(sum of the squared distance between each point and the centroid in a cluster for each cluster) and k values. Then we choose the "Elbow Point" where the decrease in WCSS values become not worth enough to increase the number of clusters. This process must be handled by hand, however, we want to choose this "Elbow Point" automatically. At this point I tried a basic heurstic to detect that point. The WCSS have non-decreasing values from cluster 1 to $k_{\max}$. I examine consecutive pairs of difference ratios and select the one with the highest value. This approach tends to pick the best-k most of the time due to its focus on the maximum value. Initially, it may appear that the maximum value is associated with the first element in the difference array, but this is not the case. After the true elbow point, the decrease in consecutive differences becomes so small that the corresponding ratio value surpasses the first ratio. My inspiration for this algorithm comes from the research conducted by Satopää, Albrecht, Irwin, and Raghavan [1].

1.2 Code

The code snippet below demonstrates how to determine the optimal number of clusters (k) for a given data set:

Listing 1: Code for Automatic k Selection

```
def elbow_method(data, max_k=10):
    wcss_values = []
    for k in range(1, max_k):
        y, elbow_centroid = k_means(data, k)
        wcss = calc_wcss(data, k, elbow_centroid, y)
        wcss_values.append(wcss)
    return wcss_values

wcss_values = elbow_method(data_set[0])

diff = np.diff(wcss_values)
ratios = []
for i in range(len(diff) - 1):
    ratios.append(diff[i] / diff[i + 1])
max_index = ratios.index(max(ratios))
k = max_index + 2
```

The code performs the following steps:

1. The `elbow_method` function calculates the Within-Cluster Sum of Squares (WCSS) for a range of k values from 1 to `max_k`. It stores the WCSS values in the `wcss_values` list.

2. After obtaining the WCSS values, the code computes the differences between consecutive WCSS values, forming the `diff` array.

3. Next, it calculates the ratios between successive differences and stores them in the `ratios` array.

4. The code identifies the k value where the ratio of differences is the highest, typically indicating an "elbow" point in the WCSS plot. This value corresponds to the optimal number of clusters (k), and it is stored in the variable k.

References

[1] Ville Satopaa, Jeannie Albrecht, David Irwin, and Barath Raghavan. Finding a" kneedle" in a haystack: Detecting knee points in system behavior. In *2011 31st international conference on distributed computing systems workshops*, pages 166–171. IEEE, 2011.

2 Data Generation and Preprocessing

2.1 Data Generation

I generated a synthetic dataset using scikit-learn's 'make_blobs' and 'make_moons' functions.

2.2 Preprocessing

I apply standardization before running K-means for equal feature scaling, better clusters and faster convergence. Standardization: St

Given a 2-dimensional dataset with n samples, each represented as $(x_{j,1}, x_{j,2})$, and two features, the standardization process for each feature i ($i = 1, 2$) involves the following mathematical operations:

Standardize feature i by subtracting the mean and dividing by the standard deviation:

$$x_{j,i} = \frac{x_{j,i} - \mu_i}{\sigma_i} \quad \text{for } i = 1, 2 \text{ and } j = 1, 2, \ldots, n$$

3 K-Means Clustering

3.1 K-Means Algorithm with Improved Initialization

I tried to improve the initial cluster assignments with K-means++. Here is basic idea, first cluster is generated randomly from data points. Then the data points are assigned probabilities (weights) according to their euclidean distance to the other cluster(s), and chosen according to that weight. The point with more distance to the centroids is tend to be selected with higher probability.

3.2 Example Iterations of Algorithm for $k = 4$

We will demonstrate the algorithm for $k = 4$ without using auto selection of k. The results are in the next page(s). (Figure 2-9)

3.3 Example Iterations of Algorithm for $k = 6$

We will demonstrate the algorithm for $k = 6$ without using auto selection of k. The results are in the next page(s).(Figure 12-19)

3.4 Objective Function vs. Iteration Count

Figure 8 and Figure 18 show how objective function changes vs iterations count. For k = 4, initial centroids are not assigned well so model needs to iterate more compared to k = 6. For k = 4, it takes more iterations for the objective function to reach stability, largely due to less optimal initial cluster assignments. In contrast, for k = 6, the objective function stabilizes sooner, primarily because the initial clusters are better assigned. (Figure 8 and Figure 18)

3.5 Comparison with scikit-learn

There is no huge difference between scikit-learn's results and mine, it was not coincidence because I tried use same logic while I was writing the algorithm. (Figure 10 and Figure 20)

3.6 K-means on Non-convex Dataset

K-means performs poor for non-convex data set because,for instance, Moon data sets have a cluster based on kind of graph-connectivity rather than distances between points. K-means specifically uses a heuristic based on distances between points. For instance, in the Moon dataset, there are instances where points within the same cluster exhibit greater Euclidean distances from each other compared to points in different clusters. Thus we need to change our heurstic for such dataset. K-means performance on non-convex dataset and results are in Figure 22-30. (The logo belongs to Derrick Rose, the youngest MVP player in NBA history.)

Figure 1: Results for k = 4

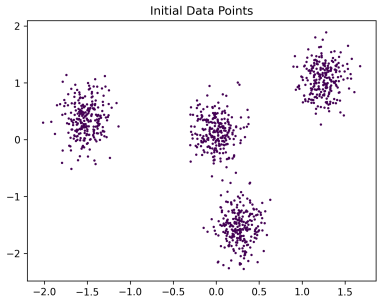

Figure 2: Initial Data Points

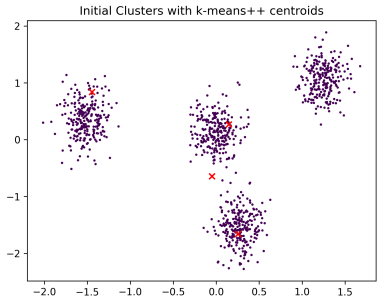

Figure 3: Initial Clusters

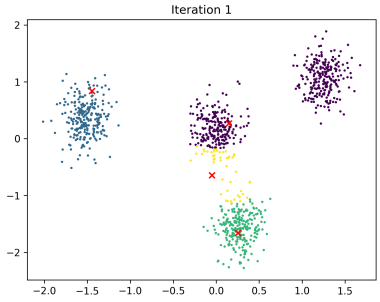

Figure 4: Iteration 1

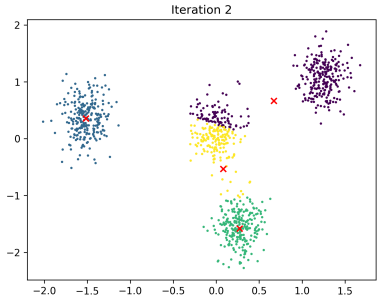

Figure 5: Iteration 2

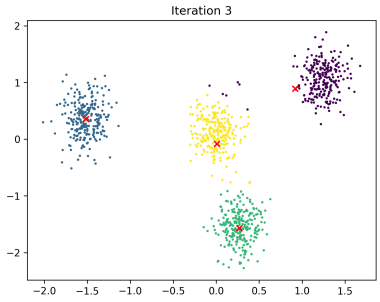

Figure 6: Iteration 3

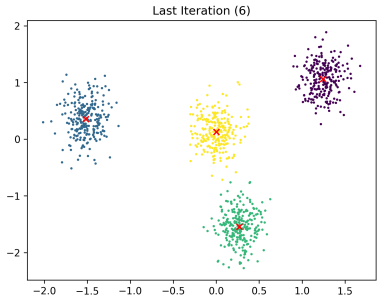

Figure 7: Last Iteration

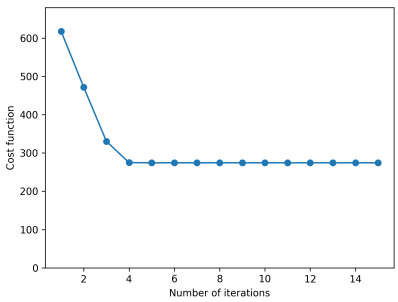

Figure 8: Objective Funtion

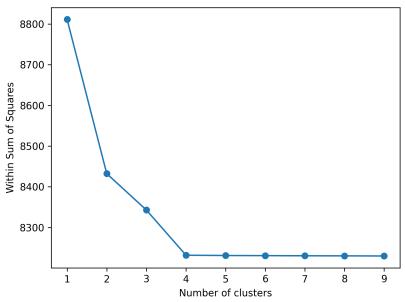

Figure 9: WCSS Graph

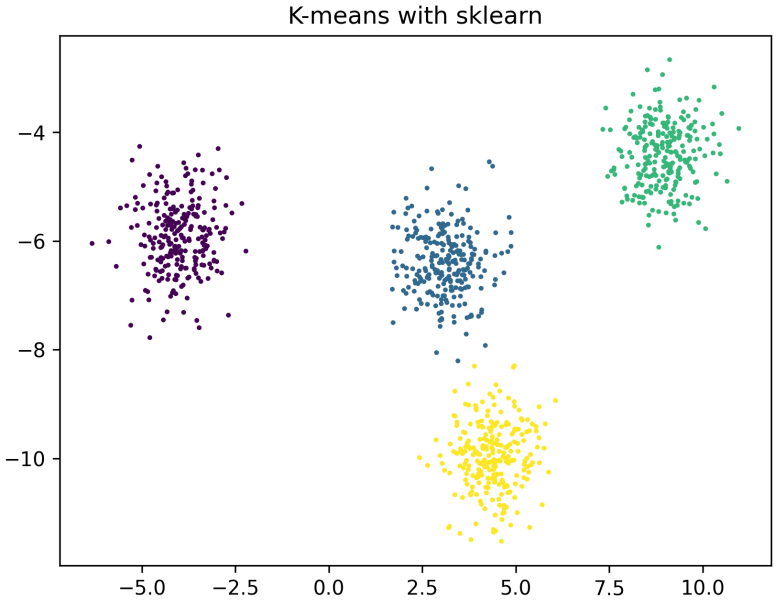

Figure 10: K-means with sklearn for k = 4

Figure 11: Results for k = 6

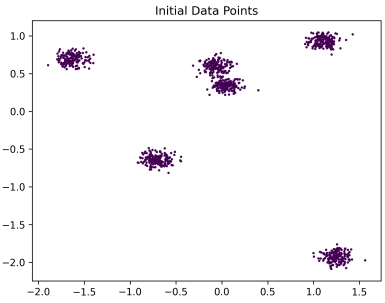

Figure 12: Initial Data Points

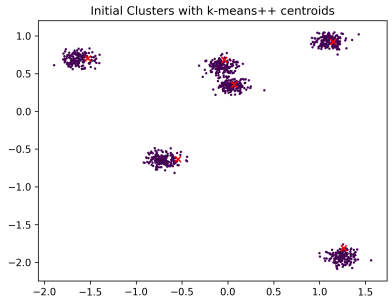

Figure 13: Initial Clusters

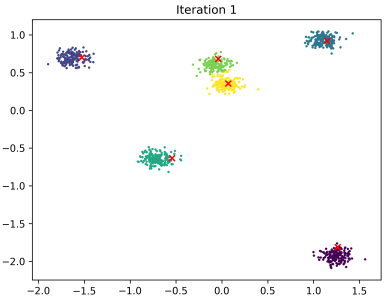

Figure 14: Iteration 1

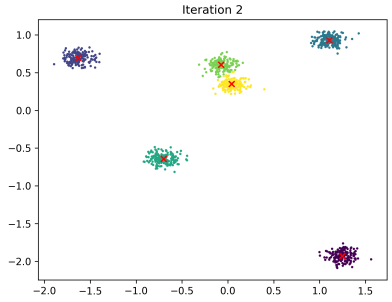

Figure 15: Iteration 2

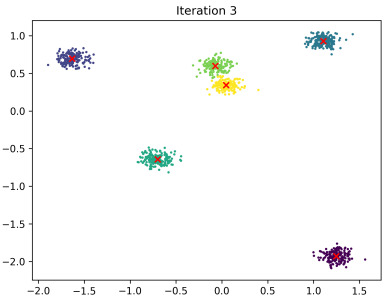

Figure 16: Iteration 3

Figure 17: Last Iteration

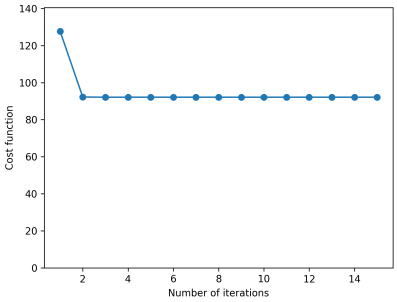

Figure 18: Objective Funtion

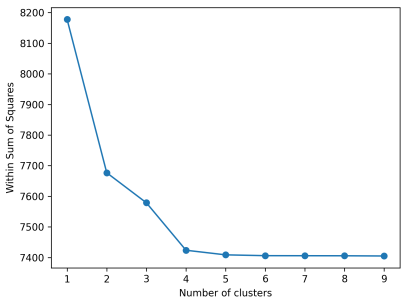

Figure 19: WCSS Graph

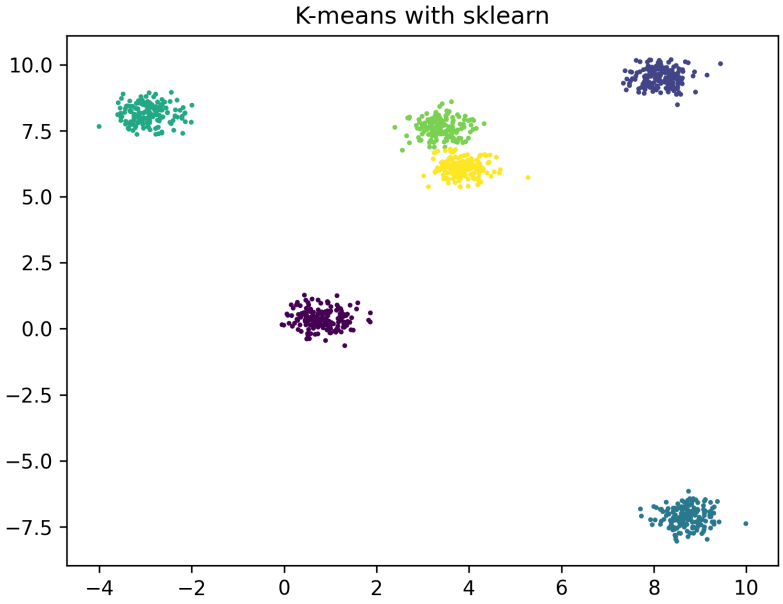

Figure 20: K-means with sklearn for k = 6

Figure 21: Results for Non-convex Dataset

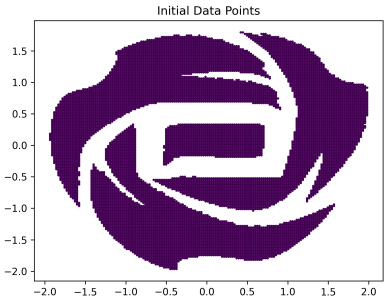

Figure 22: Initial Data Points

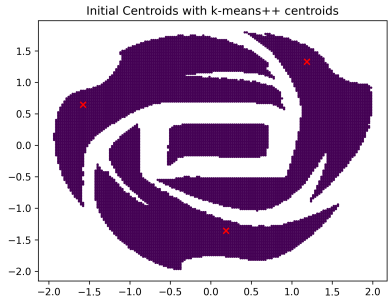

Figure 23: Initial Clusters

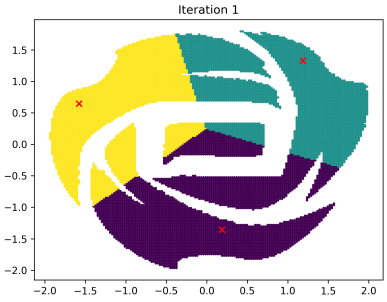

Figure 24: Iteration 1

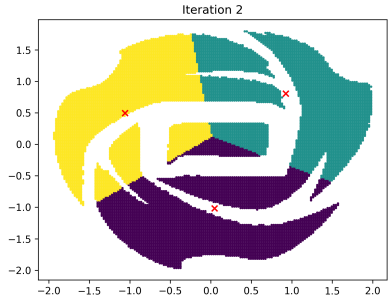

Figure 25: Iteration 2

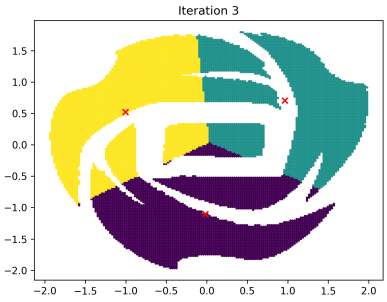

Figure 26: Iteration 3

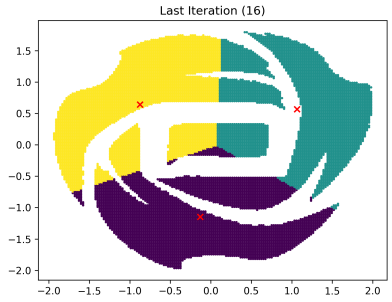

Figure 27: Last Iteration

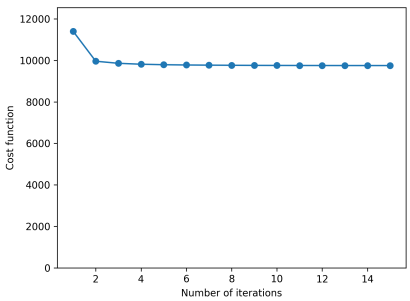

Figure 28: Objective Funtion

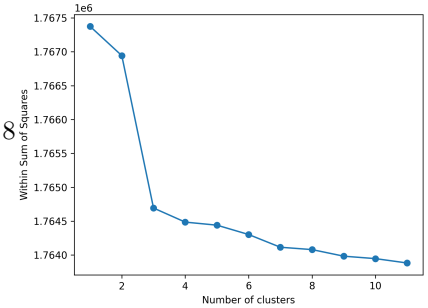

Figure 29: WCSS Graph

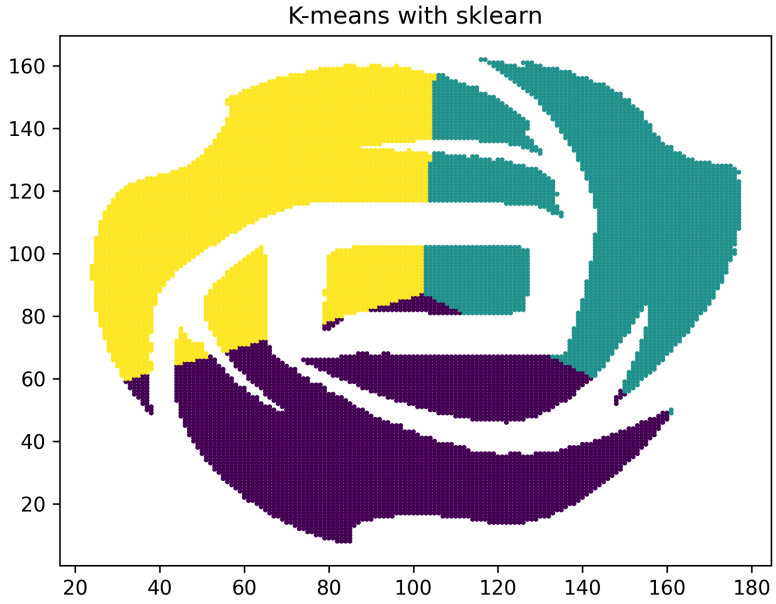

Figure 30: K-means with sklearn for non-convex dataset