K-Means Clustering Report

RABEMANANTSOA Andriamianja Tiana

May 27, 2025

Contents

1 Introduction

Clustering is an unsupervised learning technique used to group similar data points. This report focuses on the **K-means** algorithm, applied to a real-world customer segmentation dataset.

The link of the notebook is here :

`https://colab.research.google.com/drive/1bR7X1wELiaoKA7wzopuVBg0a5kXjMvex#scrollTo=G_F7UiCCWNA_`

2 Methodology

2.1 Data Preprocessing

The dataset used in this project is available on Kaggle: `https://www.kaggle.com/datasets/vjchoudhary7/customer-segmentation-tutorial-in-python/data`

After inspecting the data, we found that it contains **200 rows** and **5 columns**. All columns are useful for our analysis, except `CustomerID`, which does not provide any relevant information for clustering.

Most of the columns are of type `int64`, except for the `Gender` column, which is of type `object`. Since K-Means requires numerical inputs, we needed to encode the `Gender` column before applying the algorithm.

Finally, we observed that there is no strong correlation between the features, meaning that each variable brings independent information to the model.

2.2 Manual K-means Implementation

The K-Means algorithm follows these main steps:

2.3 K-means Pseudocode

Algorithm 1 K-means

```
Choose k initial centroids randomly.
repeat
    for each point x_i do
        Assign x_i to the nearest centroid.
    end for
    for each cluster do
        Update the centroid: compute the mean of the assigned points.
    end for
until centroids do not change (convergence)
```

Because of these drawbacks, a more robust variant called **K-means++** was developed.

3 Results

3.1 Elbow Method

We can see at the bottom that the best number of clusters, k, to choose is 5. We observe this using the **elbow method**, where the curve starts to bend, showing that increasing k further doesn't improve much.

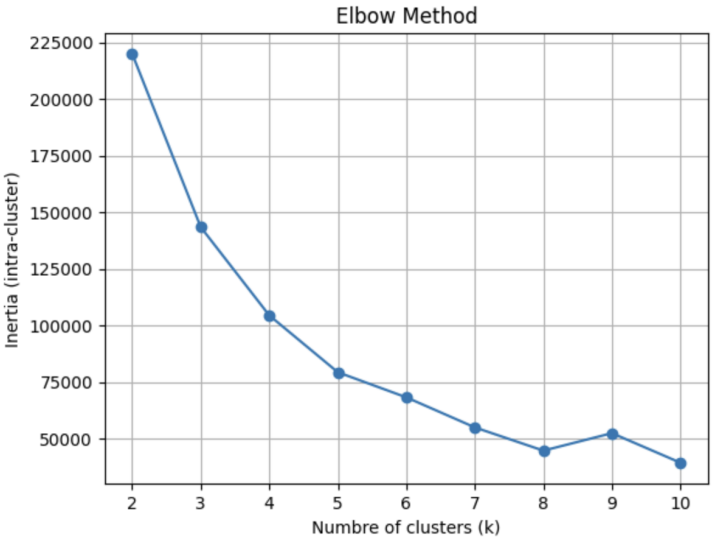

Figure 1: Elbow Method

4 Comparison and Evaluation

- Compare manual implementation vs Scikit-learn.

1. Centroids

My function:

$$\begin{bmatrix} 0.468 & 37.43 & 79.39 & 50.22 \\ 0.404 & 40.62 & 37.08 & 50.18 \end{bmatrix}$$

Scikit-learn:

$$\begin{bmatrix} 0.447 & 28.95 & 62.18 & 73.62 \\ 0.435 & 46.17 & 59.37 & 32.89 \end{bmatrix}$$

Observation: The centroids are different, showing that the two methods group the data differently. For example, the third and fourth values vary significantly, indicating that the data points were not clustered in the same way.

2. Labels

My function: Most data points at the beginning are labeled as `1`, then followed by many labeled `0`. This shows a clear block separation.

Scikit-learn: The labels are more mixed, alternating between `0` and `1`, especially at the start.

Observation: This suggests that my function might have grouped data based on their position in the list rather than based on actual data features. On the other hand, scikit-learn adapts more accurately to the natural structure of the data.

3. Inertia

My function: 219942.06

Scikit-learn: 212889.44

Observation: The inertia is lower with scikit-learn, which is better. It means the points are closer to their centroids, making the clusters more compact and well-formed.

Discussion on K-means and Introduction to K-means++

Even though I admit that the K-means implementation from `scikit-learn` provides better results than my own version, we must also recognize that the K-means algorithm has some limitations:

- It is sensitive to outliers (abnormal values), which can distort the clustering.
- The algorithm is sensitive to the initial positions of the centroids.
- The number of clusters k must be defined in advance, which is not always obvious or easy to determine.

K-means++ Algorithm

K-means++ improves the initialization step of the classic K-means algorithm. Instead of choosing the initial centroids randomly, K-means++ selects them in a smarter way to spread them out:

Algorithm 2 K-means++

1: Choose one data point at random as the first centroid.
2: **for** each remaining point **do**
3: Compute $D(x)$: the distance to the nearest existing centroid.
4: **end for**
5: Select the next centroid with probability proportional to $D(x)^2$.
6: **repeat**until k centroids are chosen.
7: Then apply the standard K-means algorithm.

This initialization technique helps to produce more consistent and often better clustering results by reducing the chance of poor initial centroid placement.

Evaluation using the Silhouette Score

Mathematical Formula

For a point i:

$$s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))}$$

where:

- $a(i)$: the average distance between i and all other points in the **same cluster**.
- $b(i)$: the minimum average distance from i to points in the **nearest cluster**.

Interpretation

- $s(i) \approx 1$: well clustered (close to its own cluster, far from others).
- $s(i) \approx 0$: on the border between two clusters.
- $s(i) < 0$: potentially misclassified (closer to another cluster).

To assess the quality of the clustering produced by our algorithm, we used the **Silhouette Score**. This metric evaluates how well each data point fits within its assigned cluster compared to other clusters.

In our case, we obtained a **Silhouette Score of 0.2546**. This indicates that the separation between clusters is moderate. It suggests that there is some structure in the data, but clusters may partially overlap or some points may be misclassified. This result highlights some limitations in our clustering model, particularly in terms of the quality of separation.

5 Conclusion

- This project helped to understand and implement the K-means algorithm step-by-step and evaluate its performance on a real dataset.
- Despite its limitations, K-means remains a valuable method to segment data and obtain coherent groups.
- Improvements via K-means++ and the use of the silhouette score help optimize and assess cluster quality.
- Future work could explore alternative algorithms and apply preprocessing techniques to refine the results.