How K-means clustering actually works?

Long Nguyen Hoang

23520882@gm.uit.edu.vn, Faculty of Information Science and Engineering

February 20, 2025

Abstract

This document introduces the K-means clustering algorithm, a popular method for partitioning data into groups based on their similarities. The process involves minimizing the variance within clusters by iteratively adjusting centroids and assigning data points to the closest centroid. While K-means performs effectively with spherical, evenly distributed clusters, it has limitations when applied to non-convex or irregular datasets. This document explores the core principles of K-means, its mathematical foundation, and the practical implementation of the algorithm in Python. It also highlights the algorithm's strengths and weaknesses, offering insights into situations where alternative clustering methods might be more appropriate. The analysis demonstrates that K-means is best suited for simpler, well-separated clusters, but may struggle with more complex data patterns.

1 Introduction

In the term of Machine Learning, *K-means Clustering* is a powerful algorithm, which allows machines to understand and then separate the original into K clusters, based on specific criteria.

In the following instance, it can clearly be seen that all observations have been grouped into three distinct clusters (Figure 1). In this paper, I do expect to introduce the K-means clustering algorithm, its properties, how it actually works, and how to deploy it in Python.

2 Developing idea

In practice, your job might sometimes need to separate a customers dataset into distinct clusters. However, as a large enterprise in your nation, it is impossible to do it manually, so that we need the assistance from machines like your own computer. Now, your work is not too back-breaking. Now, there is a key problem that need to be resolved:

Which techniques can allow your PC understand the task and perform it effectively?

Original idea

Firstly, simplify the problem with an example with just several observations. Imagine a dataset where we have a small number of customers, each represented by age and income.

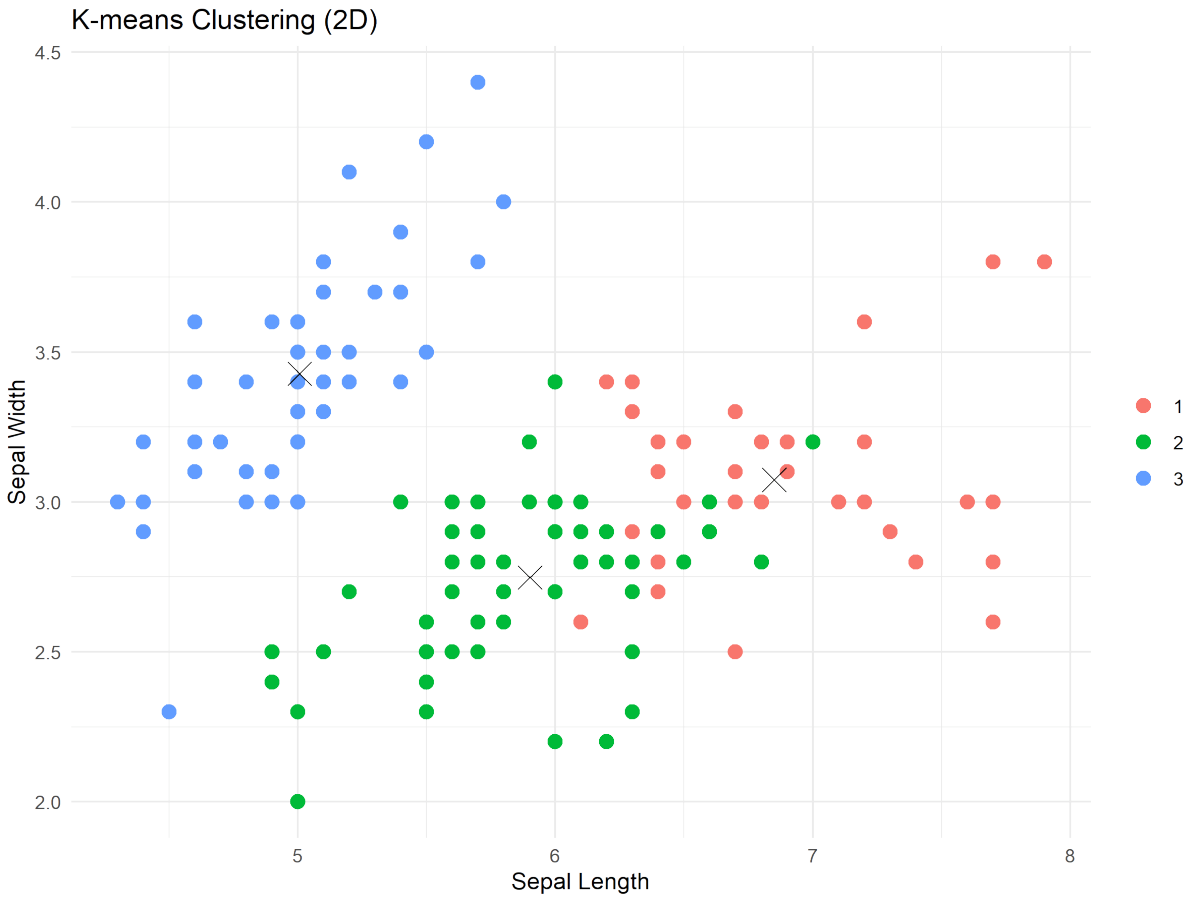

Figure 1: K-means Clustering Result

Each customer has unique characteristics, and our goal is to identify patterns or groupings that could help us understand the customers better.

Here is an example dataset.

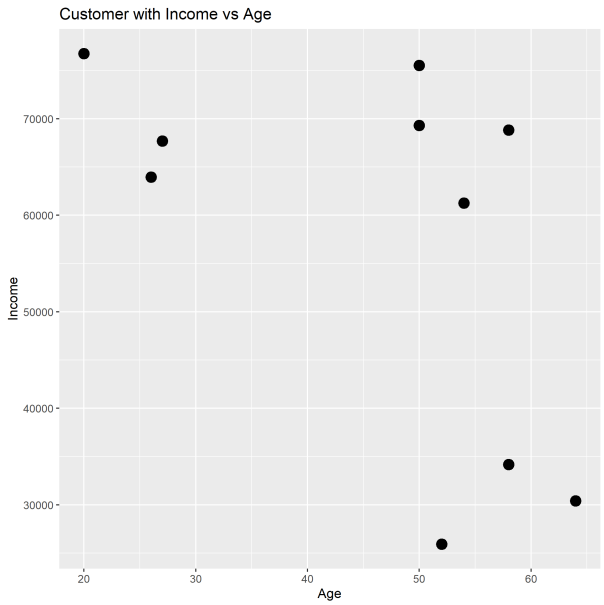

Figure 2: Customer with age and income

To accomplish this, we can employ a technique known as ***clustering***. Clustering refers to

the process of grouping data points (in this case, customers) based on similarities in their features. Instead of manually assigning each customer to a group, clustering algorithms allow the computer to autonomously find natural groupings or patterns in the data. This way, we can categorize customers who share similar traits into the same group.

Now, consider the idea of grouping these customers. At a high level, the computer needs to assess the distances between each customer's characteristics. A key part of the clustering process is determining how "close" or "similar" two customers are based on their features.

Fundamental knowledge

We have known that in an n-dimensional space, the distance between two points $x(x_1, x_2, ..., x_n)$ and $y(y_1, y_2, ..., y_n)$ can be calculated using the following function

$$d(x, y) = ||x - y||_2 = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2} \tag{1}$$

Where $d(x, y)$ represents Euclidean distance between the points x and y, and x_i and y_i are the coordinates of the points in the i^{th} dimension. To avoid square-root results, square Euclid distance ($||x - y||_2^2$) is commonly used [1].

Moreover, when this is deployed as:

$$||x - y||_2^2 = |x - y|^T |x - y|$$

Now, imagine each cluster having a "heart point", which represents the central point of each group. Now your task is more clearly: ***Minimize the distance between each point and the corresponding centroid.*** *(See Figure 1)*

It is crucial to set rules to ensure the accuracy and productivity of the algorithm, is that:

1. Hyperparameter n is defined by the user.
2. Each point only belongs to a specific cluster of the model. In other words, no observation belongs to two more clusters.
3. Each cluster can be represented by a central point (it can be calculated automatically or manually, which will be discussed later).

Returning to our problem, with the definition mentioned above and together with the example (Figure 2), we can manually divide them into 3 distinct groups (high age - high salary; high age - low salary; low age - high salary). For each observation, the shorter the distance between an observation and the "central point," the more accurately it is reflected by the cluster.

3 Solving problem

Now, we'd get started to solve the aforementioned problem with using ***Expectation - Minimization***. We present the (unlabeled) patterns in the training set to the algorithm one-by-one. For each pattern, x_i, presented, we find that cluster seeker, C_i ,that is closest to S_i and move it closer to S_i [2]. That means:

$$S_i \longleftarrow (1 - w_i)S_i + w_i S_j \tag{2}$$

Modeling

Given a set of observations $X = (x_1, x_2, \ldots, x_N)$, where each observation is a N-dimensional vector, **k-means clustering** aims to partition the N observations into $K \leq N$ sets $S = \{S_1, S_2, \ldots, S_K\}$. These are points in an n-dimensional space that we want to adjust so that they each move toward the center of one of the clusters of patterns.

As the whole, the total error for the entire dataset is:

$$\mathcal{L}(Y, M) = \sum_{i=1}^{N} (\text{distance between } x_i \text{ and the corresponding centroid point})^2 \tag{3}$$

Commonly, it can be encoded as a vector $y_i = \begin{bmatrix} y_{i1} & y_{i2} & \ldots & y_{iK} \end{bmatrix}$ for data point x_i as [1]

$$y_{ij} = \begin{cases} 1 & \text{if } x_i \text{ belongs to } S_j \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

For example, if x_i belongs to S_2, then the following y_i is:

$$\begin{bmatrix} 0 & 1 & \ldots & 0 & 0 \end{bmatrix}$$

Let $\mu_k \in \mathbb{R}^K$ be the centroid of cluster k. Let the data point x_i be assigned to cluster k. If the data point x_i is replaced by μ_k, we want the data point to be as close as possible to the centroid, i.e., x_i is close to μ_k. This can be achieved by minimizing the squared Euclidean distance $\|x_i - \mu_k\|^2$. Moreover, if x_i is assigned to group k, then $y_{ik} = 1$, and $y_{ij} = 0$, for all $j \neq k$. Thus, the squared Euclidean distance can be written as

$$\|x_i - \mu_k\|_2^2 = \sum_{j=1}^{K} y_{ij} \|x_i - \mu_k\|_2^2 \tag{5}$$

From (3) and (5), thus, the average error for the entire dataset is:

$$\mathcal{L}(Y, M) = \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{K} y_{ij} \|x_i - \mu_k\|_2^2 \tag{6}$$

Solving problem

The approach K-means following to solve the problem is called ***Expectation - Minimization***. Particularly, E-step assign the data points to the closet cluster, whereas the M-step computes the centroid of each cluster. [3]. The objective function is:

$$\arg\min_{S} \sum_{i=1}^{K} \sum_{x \in S_i} \|x - \mu_i\|^2 = \arg\min_{S} \sum_{i=1}^{K} |S_i| \cdot \text{Var}(S_i)$$

where μ_i is the mean (also called centroid) of points in S_i, $|S_i|$ is the number of points that belong to cluster S_i i.e.,

$$\mu_i = \frac{1}{|S_i|} \sum_{x \in S_i} x$$

.

It's a minimization problem of two parts. We first minimize J with respect to y_{ik} and treat μ_k fixed. Then we minimize J with respect to μ_k and treat y_{ik} fixed. Technically speaking, we differentiate J with respect to μ_k first and update cluster assignments (E-step). Then we differentiate J with respect to y_{ik} and recompute the centroids after the cluster assignments from previous step (M-step).

Therefore, the E-step is:

$$y_{ik} = \begin{cases} 1 & \text{if } k = \arg\min_j \|x_i - \mu_j\|^2 \\ 0 & \text{otherwise} \end{cases}$$

In the **M-step**, the centroids are updated. Each centroid μ_k is computed as the mean of all points x_i assigned to cluster k. Mathematically, this is:

$$\begin{aligned} \frac{\partial J}{\partial \mu_k} &= 2 \sum_{i=1}^{m} y_{ik} \left(x^i - \mu_k \right) = 0 \\ \Rightarrow \mu_k &= \frac{1}{|S_k|} \sum_{x_i \in S_k} x_i \end{aligned} \tag{7}$$

where $|S_k|$ represents the number of points assigned to cluster k and S_k is the set of points in cluster k.

These steps (E-step and M-step) are alternated iteratively until convergence. The algorithm converges when the cluster assignments y_{ik} and the centroids μ_k no longer change significantly.

4 Python code

Preparation

First, please download the dataset from this link for the project deployed in the following steps.

After download and visualize it with `matplotlib`, it displays the following figure:

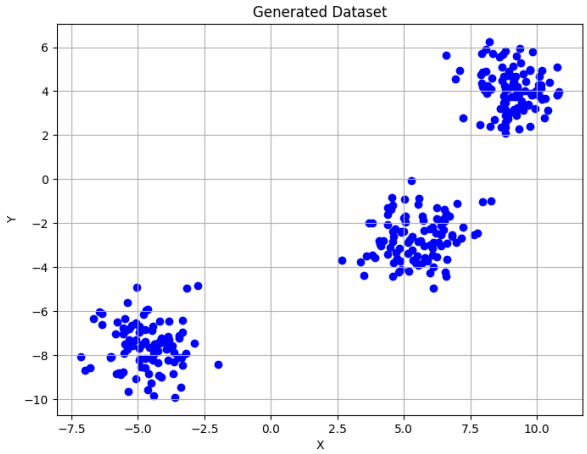

Figure 3: Visualization of the dataset

Now, I'll create an Jupyter Notebook named as `k-means-clustering.ipynb`. First, necessary libraries will be imported like this

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

Then import the dataset.

```
df = dp.read_csv("generated_dataset.csv")
df
```

The notebook's cell returns the following result (Figure 4).

	x	y
0	5.135889	-2.862346
1	6.506228	-1.386944
2	5.625100	-2.808703
3	5.306448	-3.204297
4	5.210431	-2.776417
...	...	...
295	9.759568	3.905837
296	9.399271	5.273777
297	8.938098	4.135821
298	8.334630	5.701751
299	9.899493	3.856729

Figure 4: The figures of the dataset

If we visualize them with `matplotlib`, it should be returns the result as the Figure 3. The following code can be written like this

```
X = df["x"]
Y = df["y"]
plt.scatter(X, Y)
```

As now, the data is ready for performing K-means algorithm.

Deploying algorithm

Now, we will initialize random centroids (S) from the first K points.

```
# Initialize K (number of clusters)
K = 3
N = len(df)  # Number of data points
df["cluster"] = -1  # Initialize cluster column with -1 (unassigned)

# Initialize random centroids (S) from the first K points
S_temp = [[float(df["x"][i]), float(df["y"][i])] for i in range(K)]
S = np.array(S_temp)

# Plot initial centroids
plt.scatter(S[:, 0], S[:, 1], color='red', label='Centroids')
plt.title("Initial Centroids")
plt.xlabel("x")
plt.ylabel("y")
plt.legend()
plt.show()
```

The cell should return the Figure 5.

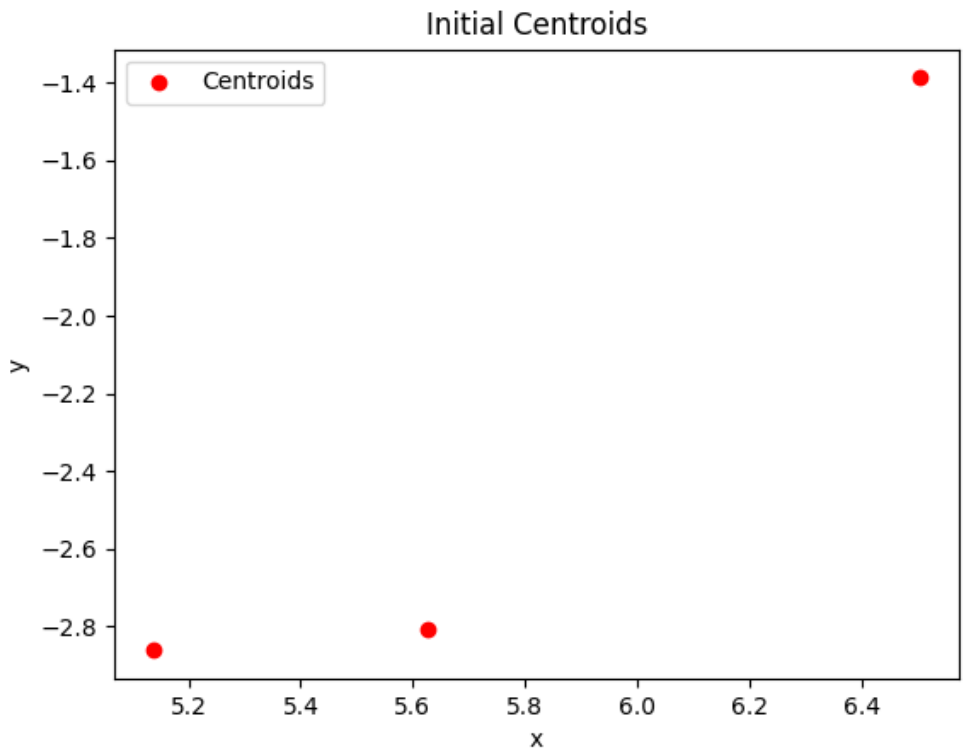

Figure 5: The initial centroids

Then, deploy the algorithm.

```
def EuclideanDistance(x1, y1, x2, y2):
    return np.sqrt((x1 - x2)**2 + (y1 - y2)**2)

# K-means algorithm loop
while True:
    prev_cluster = df["cluster"].copy()
    Total = 0

    # Assign points to the nearest cluster
    for i in range(N):
        min_distance = float('inf')
        min_index = -1
        for j in range(K):
            distance = EuclideanDistance(df["x"][i], df["y"][i],
               S[j][0], S[j][1])
            if distance < min_distance:
                min_distance = distance
                min_index = j

        # Update cluster assignment
        df.loc[i, "cluster"] = min_index
        Total += min_distance

    # Update centroids (mean of points assigned to each cluster)
    for j in range(K):
        cluster_points = df[df["cluster"] == j]
        if len(cluster_points) > 0:
            S[j] = [cluster_points["x"].mean(), cluster_points["y
               "].mean()]

    # Check for convergence (if cluster assignments don't change)
    if (df["cluster"] == prev_cluster).all():
        break

    print("Total distance:", Total)
```

Now, visualize the result by using `matplotlib`.

```

plt.figure(figsize=(8, 6))
for j in range(K):
    cluster_points = df[df["cluster"] == j]
    plt.scatter(cluster_points["x"], cluster_points["y"], label=f
       'Cluster {j}')
    plt.scatter(S[j][0], S[j][1], color='red', marker='X', s=100,
        label=f'Centroid {j}')

plt.title("K-means Clustering")
plt.xlabel("x")
plt.ylabel("y")
plt.legend()
plt.show()
```

The cell returns this result (Figure 6). As you can see, the K-means algorithm has effectively grouped the observations into distinct clusters. This clustering process relies on minimizing the within-cluster variance by iteratively adjusting the cluster centroids and reassigning the data points.

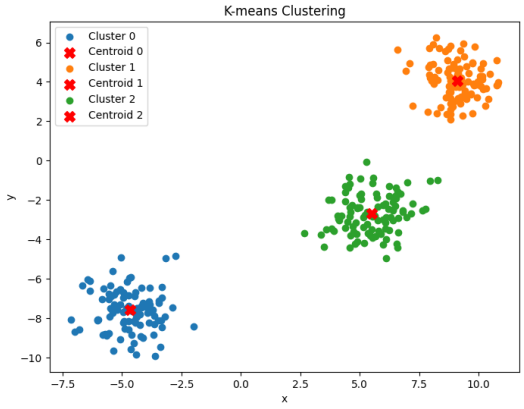

Figure 6: Clustered dataset

5 Evaluation

Returning to the example, it looks like the algorithms works well (a cluster is significantly different from the others). See Figure 7

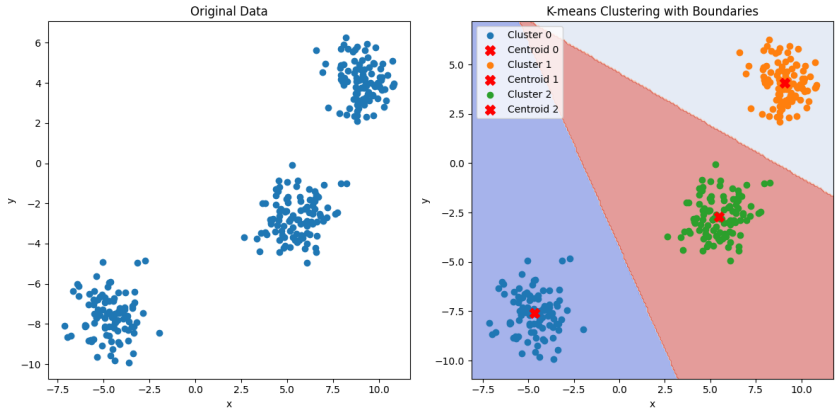

Figure 7: The above example with boundaries

So, there is a question: ***Does the K-means Clustering work effectively in each situation?*** Here are some example:

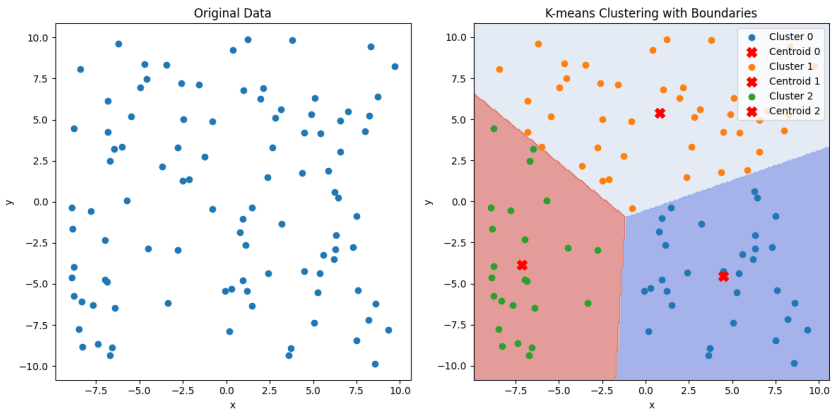

Figure 8: Uniformed-distribution dataset

As you can see in Figure 8, it still works well, although the difference between two patterns at the boundary is not very clear.

However, this method does not yield satisfactory results in this particular case (Figure 9, as the K-means algorithm struggles to properly segment the non-convex data points, leading to poor cluster boundaries and an inaccurate representation of the underlying structure.

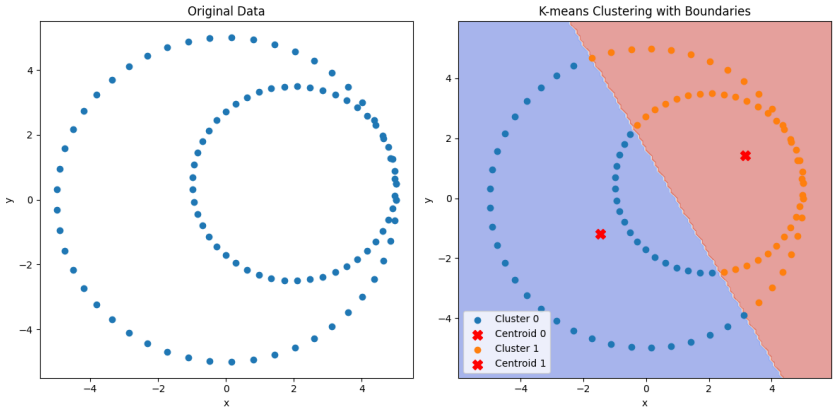

Figure 9: Moon distribution

References

[1] V. H. Tiep, *Machine Learning co ban.* 2022. GitHub repository available at https://github.com/tiepvupsu/ebookMLCB.

[2] R. L. Nils J. Nilsson, "Introduction to machine learning - an early approach of a proposed textbook," November 3, 1998.

[3] I. Dabbura, "K-means clustering: Algorithm, applications, evaluation methods, and drawbacks." `https://medium.com/towards-data-science/k-means-clustering-algorithm-applications-evaluation-methods-and-drawbacks-aa03e` 2018.