K-Means-Clustering

January 14, 2025

[ ]:
```
'''
    K-Means Clustering -->

    K-Means Clustering is an unsupervised machine learning algorithm used for␣
↪partitioning
    a dataset into a set of distinct, non-overlapping groups, called clusters.
    It minimizes the variance within clusters while maximizing the variance␣
↪between clusters.
    It's widely used in applications like market segmentation, pattern␣
↪recognition,
    image compression, and anomaly detection.
'''
```

[ ]:
```
'''
    Steps in K-Means Clustering -->

    Initialize Centroids :
    Choose k random points from the dataset as the initial cluster centroids.

    Assign Data Points to Clusters :
    For each data point, compute the distance to each centroid (commonly using␣
↪Euclidean distance).
    Assign the data point to the nearest centroid.

    Update Centroids :
    Compute the mean of all points assigned to a cluster and update the␣
↪centroid to this mean.

    Repeat :
    Iterate the assignment and update steps until centroids stabilize or a␣
↪maximum number
    of iterations is reached.

    Output :
    Return the k clusters and their centroids.
'''
```

[ ]:
```
'''
    Advantages -->

    Simple and easy to implement.
    Efficient for large datasets (complexity O(n k t), where n is the number of␣
↪data points,
    k is the number of clusters, and t is the number of iterations).
    Works well with spherical, equally-sized clusters.

    Disadvantages -->

    Sensitive to the initial placement of centroids.
    Requires pre-specifying k, the number of clusters.
    Not suitable for clusters with varying densities or non-spherical shapes.
    Can get stuck in local optima.
'''
```

[ ]:
```
'''
    Choosing the Optimal k -->

    The Elbow Method :
    Calculate the sum of squared distances (inertia) for different values of k.
    Plot k against inertia.
    The "elbow point" (where inertia decreases sharply and then levels off)␣
↪indicates the optimal k.
'''
```

[ ]:
```
'''
    K-Means ++

    K-Means ++ is an enhancement to the K-Means clustering algorithm.
    It improves the initial selection of cluster centroids, which helps K-Means␣
↪converge
    faster and often leads to better clustering results.

    Problem with K-Means Initialization -->

    In standard K-Means, the initial cluster centroids are chosen randomly.
    This can lead to :

    Poor convergence due to bad initialization.
    Suboptimal clustering results when centroids are poorly placed.

    Solution -->

    K-Means++ Initialization
```

```
    K-Means++ ensures better initialization by spreading out the initial␣
↪centroids,
    reducing the chances of convergence to a poor local minimum.
'''
```

[ ]:
```
'''
    Steps in K-Means ++ -->

    First Centroid : Choose the first centroid randomly from the dataset.
    Subsequent Centroids : For each subsequent centroid :
    Calculate the distance D(x) from each point x in the dataset to the nearest␣
↪already chosen centroid.
    Select the next centroid probabilistically, where a point x is chosen with␣
↪a probability proportional
    to D(x)^2.
    Repeat: Repeat until k centroids are chosen.
    Run K-Means: Use these centroids to initialize the standard K-Means␣
↪algorithm.
'''
```

[ ]:
```
'''
    Advantages of K-Means ++ -->

    Improved Convergence: Reduces the number of iterations needed for K-Means␣
↪to converge.
    Better Clustering Quality: Produces better clustering results by starting␣
↪with well-separated centroids.
'''
```

[1]:
```
#   Importing Libraries -->

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
```

[2]:
```
#   Importing Dataset -->

data = pd.read_csv('Data/Mall_Customers.csv')
data.head(10)
```

[2]:

	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40

5	6	Female	22	17	76
6	7	Female	35	18	6
7	8	Female	23	18	94
8	9	Male	64	19	3
9	10	Female	30	19	72

[3]:
```
x_data = data.iloc[:, [3,4]].values
x_data
```

[3]:
```
array([[ 15,  39],
       [ 15,  81],
       [ 16,   6],
       [ 16,  77],
       [ 17,  40],
       [ 17,  76],
       [ 18,   6],
       [ 18,  94],
       [ 19,   3],
       [ 19,  72],
       [ 19,  14],
       [ 19,  99],
       [ 20,  15],
       [ 20,  77],
       [ 20,  13],
       [ 20,  79],
       [ 21,  35],
       [ 21,  66],
       [ 23,  29],
       [ 23,  98],
       [ 24,  35],
       [ 24,  73],
       [ 25,   5],
       [ 25,  73],
       [ 28,  14],
       [ 28,  82],
       [ 28,  32],
       [ 28,  61],
       [ 29,  31],
       [ 29,  87],
       [ 30,   4],
       [ 30,  73],
       [ 33,   4],
       [ 33,  92],
       [ 33,  14],
       [ 33,  81],
       [ 34,  17],
       [ 34,  73],
```

```
[ 37,  26],
[ 37,  75],
[ 38,  35],
[ 38,  92],
[ 39,  36],
[ 39,  61],
[ 39,  28],
[ 39,  65],
[ 40,  55],
[ 40,  47],
[ 40,  42],
[ 40,  42],
[ 42,  52],
[ 42,  60],
[ 43,  54],
[ 43,  60],
[ 43,  45],
[ 43,  41],
[ 44,  50],
[ 44,  46],
[ 46,  51],
[ 46,  46],
[ 46,  56],
[ 46,  55],
[ 47,  52],
[ 47,  59],
[ 48,  51],
[ 48,  59],
[ 48,  50],
[ 48,  48],
[ 48,  59],
[ 48,  47],
[ 49,  55],
[ 49,  42],
[ 50,  49],
[ 50,  56],
[ 54,  47],
[ 54,  54],
[ 54,  53],
[ 54,  48],
[ 54,  52],
[ 54,  42],
[ 54,  51],
[ 54,  55],
[ 54,  41],
[ 54,  44],
[ 54,  57],
```

```
[ 54,  46],
[ 57,  58],
[ 57,  55],
[ 58,  60],
[ 58,  46],
[ 59,  55],
[ 59,  41],
[ 60,  49],
[ 60,  40],
[ 60,  42],
[ 60,  52],
[ 60,  47],
[ 60,  50],
[ 61,  42],
[ 61,  49],
[ 62,  41],
[ 62,  48],
[ 62,  59],
[ 62,  55],
[ 62,  56],
[ 62,  42],
[ 63,  50],
[ 63,  46],
[ 63,  43],
[ 63,  48],
[ 63,  52],
[ 63,  54],
[ 64,  42],
[ 64,  46],
[ 65,  48],
[ 65,  50],
[ 65,  43],
[ 65,  59],
[ 67,  43],
[ 67,  57],
[ 67,  56],
[ 67,  40],
[ 69,  58],
[ 69,  91],
[ 70,  29],
[ 70,  77],
[ 71,  35],
[ 71,  95],
[ 71,  11],
[ 71,  75],
[ 71,   9],
[ 71,  75],
```

```
[ 72,  34],
[ 72,  71],
[ 73,   5],
[ 73,  88],
[ 73,   7],
[ 73,  73],
[ 74,  10],
[ 74,  72],
[ 75,   5],
[ 75,  93],
[ 76,  40],
[ 76,  87],
[ 77,  12],
[ 77,  97],
[ 77,  36],
[ 77,  74],
[ 78,  22],
[ 78,  90],
[ 78,  17],
[ 78,  88],
[ 78,  20],
[ 78,  76],
[ 78,  16],
[ 78,  89],
[ 78,   1],
[ 78,  78],
[ 78,   1],
[ 78,  73],
[ 79,  35],
[ 79,  83],
[ 81,   5],
[ 81,  93],
[ 85,  26],
[ 85,  75],
[ 86,  20],
[ 86,  95],
[ 87,  27],
[ 87,  63],
[ 87,  13],
[ 87,  75],
[ 87,  10],
[ 87,  92],
[ 88,  13],
[ 88,  86],
[ 88,  15],
[ 88,  69],
[ 93,  14],
```

```
       [ 93,  90],
       [ 97,  32],
       [ 97,  86],
       [ 98,  15],
       [ 98,  88],
       [ 99,  39],
       [ 99,  97],
       [101,  24],
       [101,  68],
       [103,  17],
       [103,  85],
       [103,  23],
       [103,  69],
       [113,   8],
       [113,  91],
       [120,  16],
       [120,  79],
       [126,  28],
       [126,  74],
       [137,  18],
       [137,  83]], dtype=int64)
```

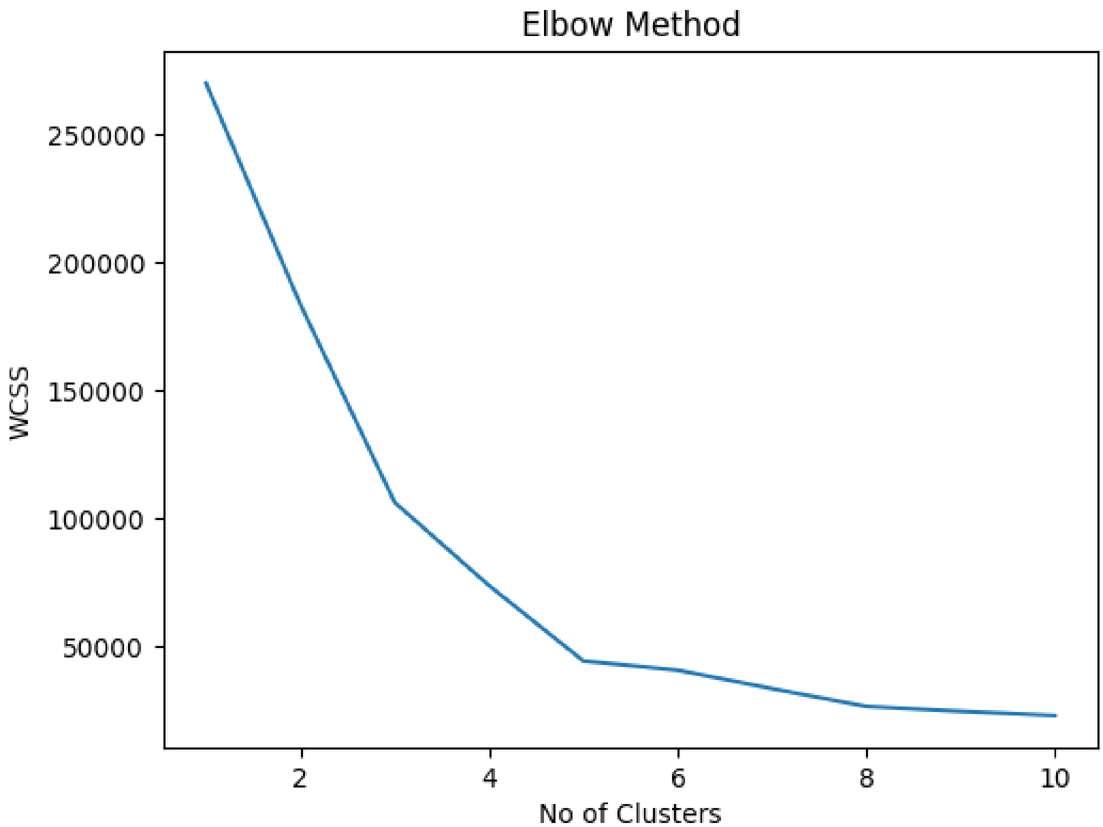
[4]:

```
#   Elbow Method To Find Optimal no of Clusters -->

wcss = []

for i in range(1,11) :
    model  = KMeans(n_clusters=i, init='k-means++', random_state=42)
    model.fit(x_data)
    wcss.append(model.inertia_)

plt.plot(range(1,11), wcss)
plt.title("Elbow Method")
plt.xlabel("No of Clusters")
plt.ylabel("WCSS")
plt.show()
```



[5]:
```
#   Building The Model -->

model = KMeans(n_clusters=5, init='k-means++', random_state=42)
model.fit(x_data)
```

[5]:
```
KMeans(n_clusters=5, random_state=42)
```

[6]:
```
#   Predicting Results -->

y_pred = model.predict(x_data)
y_pred
```

[6]:
```
array([4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2,
       4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 2, 4, 0,
       4, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 3, 1, 0, 1, 3, 1, 3, 1,
       0, 1, 3, 1, 3, 1, 3, 1, 3, 1, 0, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1,
       3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1,
```

```
3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1, 3, 1,
3, 1])
```

[8]:
```
#   Visualizing Results -->

plt.scatter(x_data[y_pred==0,0], x_data[y_pred==0,1], s=50, c='red',␣
 ↪label="Cluster 1")
plt.scatter(x_data[y_pred==1,0], x_data[y_pred==1,1], s=50, c='blue',␣
 ↪label="Cluster 2")
plt.scatter(x_data[y_pred==2,0], x_data[y_pred==2,1], s=50, c='green',␣
 ↪label="Cluster 3")
plt.scatter(x_data[y_pred==3,0], x_data[y_pred==3,1], s=50, c='cyan',␣
 ↪label="Cluster 4")
plt.scatter(x_data[y_pred==4,0], x_data[y_pred==4,1], s=50, c='magenta',␣
 ↪label="Cluster 5")
plt.scatter(model.cluster_centers_[:,0], model.cluster_centers_[:,1], s=70,␣
 ↪c="black", label="Centroids")
plt.title("Cluster Of Customers")
plt.xlabel("Annual Income (K$)")
plt.ylabel("Spending Score (1-100)")
plt.legend()
plt.show()
```