120A3051

Shreya Idate

Batch: E3

Experiment 7

Aim: Implement K-medoid clustering algorithm using python.

Theory:

K-Medoids (also called Partitioning Around Medoid) algorithm was proposed in 1987 by Kaufman and Rousseeuw. A medoid can be defined as a point in the cluster, whose dissimilarities with all the other points in the cluster are minimum. The dissimilarity of the medoid(Ci) and object(Pi) is calculated by using E = |Pi – Ci|

Algorithm:

1. Choose k number of random points from the data and assign these k points to k number of clusters. These are the initial medoids.
2. For all the remaining data points, calculate the distance from each medoid and assign it to the cluster with the nearest medoid.
3. Calculate the total cost (Sum of all the distances from all the data points to the medoids)
4. Select a random point as the new medoid and swap it with the previous medoid. Repeat 2 and 3 steps.
5. If the total cost of the new medoid is less than that of the previous medoid, make the new medoid permanent and repeat step 4.
6. If the total cost of the new medoid is greater than the cost of the previous medoid, undo the swap and repeat step 4.
7. The Repetitions have to continue until no change is encountered with new medoids to classify data points.
8. The K-medoids algorithm is more robust to outliers than K-means since the medoids are actual data points and not just the mean of the data points in the cluster.

However, the algorithm can be slower than K-means, especially for large datasets, since it involves computing distances between data points.

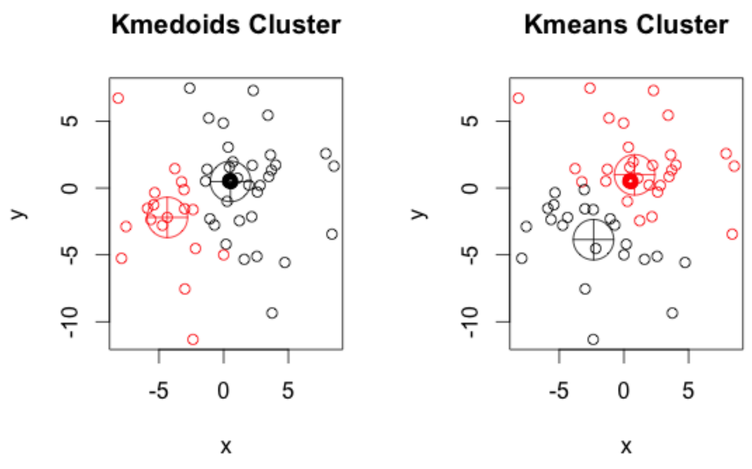
K-medoid has several advantages over K-means, including its ability to handle nonEuclidean distance metrics, its robustness to outliers, and its ability to work with small datasets. However, it can be slower and less scalable than K-means, especially for large datasets.

Advantages:

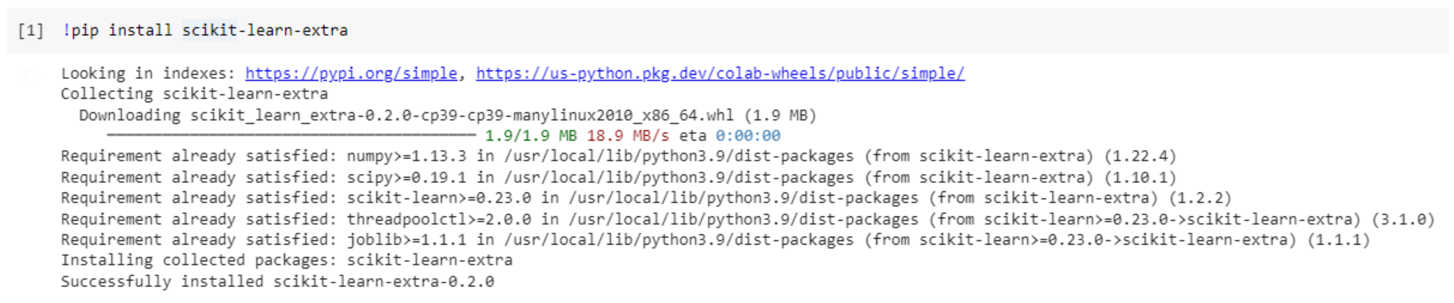
1. It is simple to understand and easy to implement.
2. K-Medoid Algorithm is fast and converges in a fixed number of steps.
3. PAM is less sensitive to outliers than other partitioning algorithms.

Disadvantages:

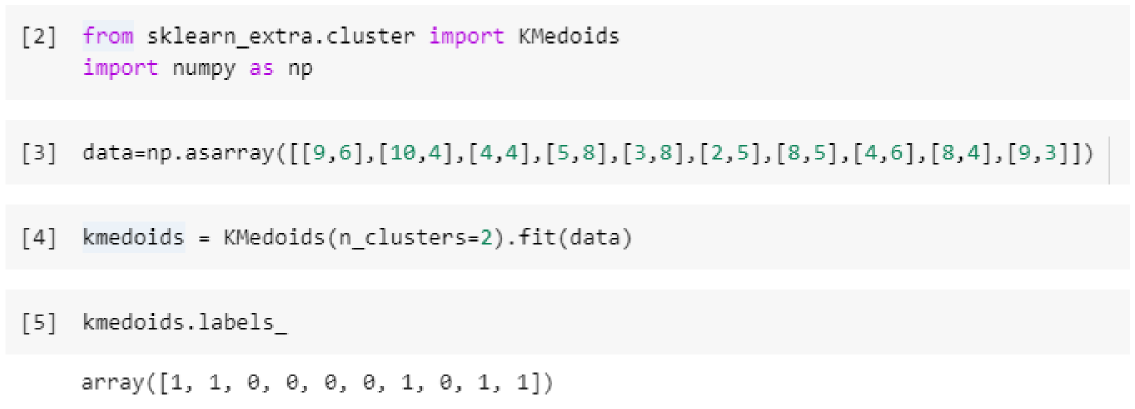
1. The main disadvantage of K-Medoid algorithms is that it is not suitable for clustering non-spherical (arbitrarily shaped) groups of objects. This is because it relies on minimizing the distances between the non-medoid objects and the medoid (the cluster center) – briefly, it uses compactness as clustering criteria instead of connectivity.
2. It may obtain different results for different runs on the same dataset because the first k medoids are chosen randomly.



Output:



```
[1]  !pip install scikit-learn-extra

     Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
     Collecting scikit-learn-extra
       Downloading scikit_learn_extra-0.2.0-cp39-cp39-manylinux2010_x86_64.whl (1.9 MB)
          ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.9/1.9 MB 18.9 MB/s eta 0:00:00
     Requirement already satisfied: numpy>=1.13.3 in /usr/local/lib/python3.9/dist-packages (from scikit-learn-extra) (1.22.4)
     Requirement already satisfied: scipy>=0.19.1 in /usr/local/lib/python3.9/dist-packages (from scikit-learn-extra) (1.10.1)
     Requirement already satisfied: scikit-learn>=0.23.0 in /usr/local/lib/python3.9/dist-packages (from scikit-learn-extra) (1.2.2)
     Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.9/dist-packages (from scikit-learn>=0.23.0->scikit-learn-extra) (3.1.0)
     Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.9/dist-packages (from scikit-learn>=0.23.0->scikit-learn-extra) (1.1.1)
     Installing collected packages: scikit-learn-extra
     Successfully installed scikit-learn-extra-0.2.0
```



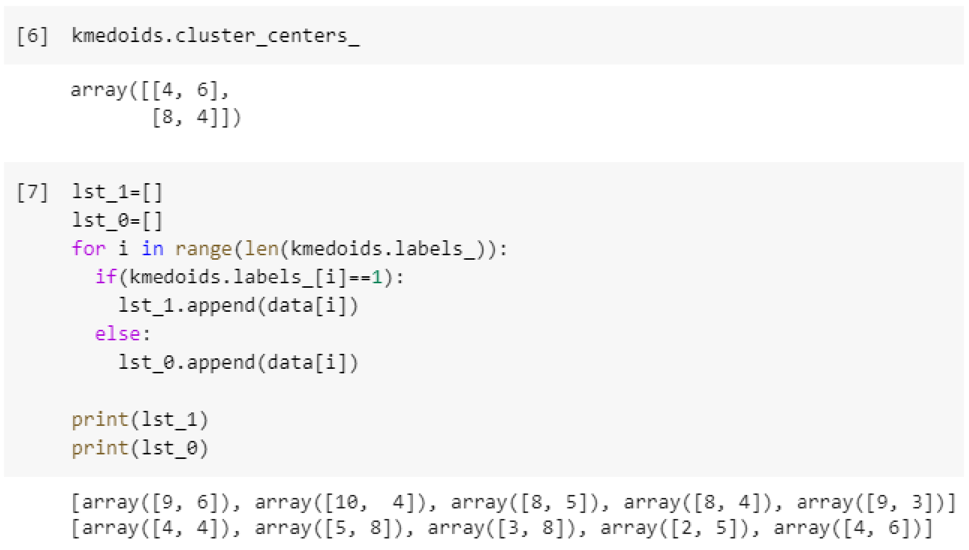
```
[2]  from sklearn_extra.cluster import KMedoids
     import numpy as np
```

```
[3]  data=np.asarray([[9,6],[10,4],[4,4],[5,8],[3,8],[2,5],[8,5],[4,6],[8,4],[9,3]])
```

```
[4]  kmedoids = KMedoids(n_clusters=2).fit(data)
```

```
[5]  kmedoids.labels_
```

```
     array([1, 1, 0, 0, 0, 0, 1, 0, 1, 1])
```

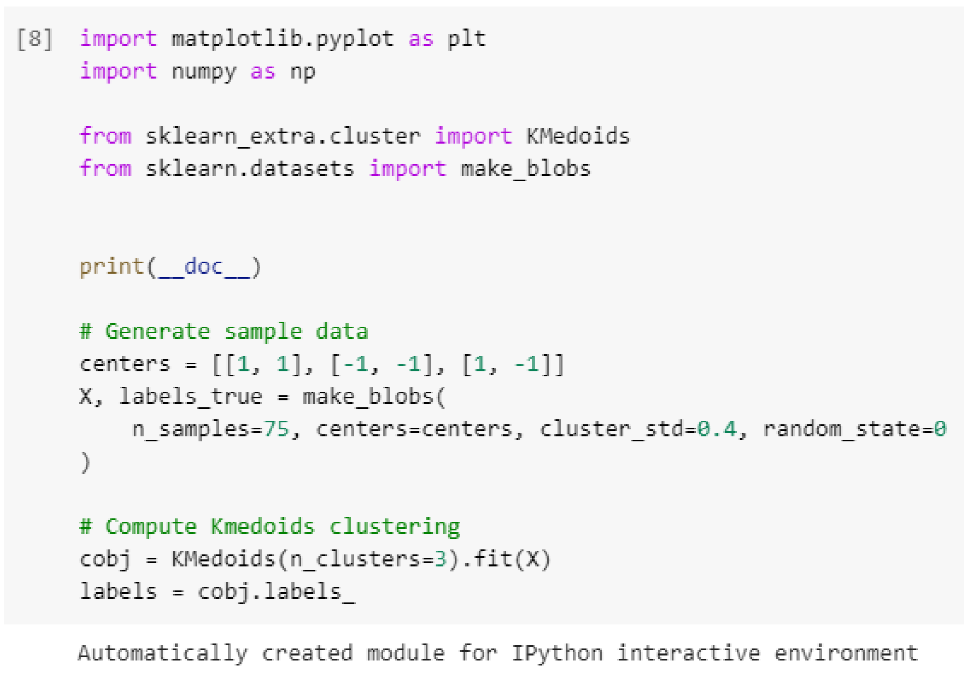


```
[6]  kmedoids.cluster_centers_
```

```
     array([[4, 6],
            [8, 4]])
```

```
[7]  lst_1=[]
     lst_0=[]
     for i in range(len(kmedoids.labels_)):
       if(kmedoids.labels_[i]==1):
         lst_1.append(data[i])
       else:
         lst_0.append(data[i])

     print(lst_1)
     print(lst_0)
```

```
     [array([9, 6]), array([10,  4]), array([8, 5]), array([8, 4]), array([9, 3])]
     [array([4, 4]), array([5, 8]), array([3, 8]), array([2, 5]), array([4, 6])]
```

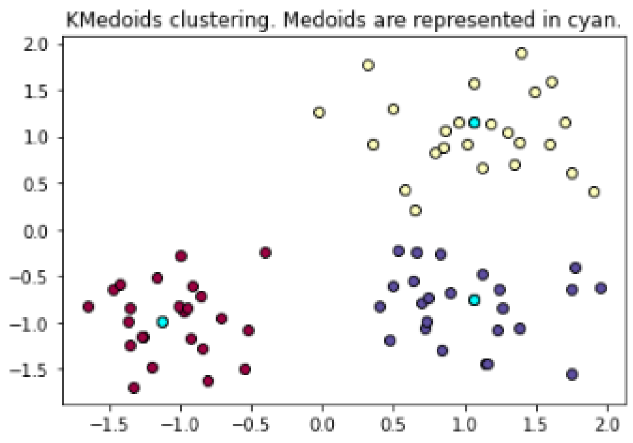


```
[8]  import matplotlib.pyplot as plt
     import numpy as np

     from sklearn_extra.cluster import KMedoids
     from sklearn.datasets import make_blobs


     print(__doc__)

     # Generate sample data
     centers = [[1, 1], [-1, -1], [1, -1]]
     X, labels_true = make_blobs(
         n_samples=75, centers=centers, cluster_std=0.4, random_state=0
     )

     # Compute Kmedoids clustering
     cobj = KMedoids(n_clusters=3).fit(X)
     labels = cobj.labels_
```

```
     Automatically created module for IPython interactive environment
```

```
[9] unique_labels = set(labels)
    colors = [
        plt.cm.Spectral(each) for each in np.linspace(0, 1, len(unique_labels))
    ]
    for k, col in zip(unique_labels, colors):

        class_member_mask = labels == k

        xy = X[class_member_mask]
        plt.plot(
            xy[:, 0],
            xy[:, 1],
            "o",
            markerfacecolor=tuple(col),
            markeredgecolor="k",
            markersize=6,
        )

    plt.plot(
        cobj.cluster_centers_[:, 0],
        cobj.cluster_centers_[:, 1],
        "o",
        markerfacecolor="cyan",
        markeredgecolor="k",
        markersize=6,
    )

    plt.title("KMedoids clustering. Medoids are represented in cyan.")
```

```
Text(0.5, 1.0, 'KMedoids clustering. Medoids are represented in cyan.')
```



Conclusion:

Successfully implemented K-medoid clustering algorithm using Python.