**Name:** Aditya Somani **Roll No:** BE1851061 **PRN:** 71901204L

**Assignment No 4**

**To study about Unsupervised Learning.**

|  |
| --- |
| **Aim** |
| **To implement K-Means Clustering and Hierarchical clustering on proper data set & to compare their convergence.** |

|  |  |
| --- | --- |
| **Objective(s)** | |
| 1 | To implement K-means clustering on given data set. |
| 2 | To implement Hierarchical clustering on given data set. |
| 3 | To compare their convergence. |

|  |
| --- |
| **Theory** |
| **Introduction:**  **Unsupervised learning:**  Unsupervised learning is the machine learning task of inferring a function to describe hidden structure from unlabeled data. Since the examples given to the learner are unlabeled, there is no error or reward signal to evaluate a potential solution. This distinguishes unsupervised learning from supervised learning and reinforcement learning**.**  Unsupervised learning studies how systems can learn to represent particular input patterns in a way that reflects the statistical structure of the overall collection of input patterns. By contrast with SUPERVISED LEARNING or REINFORCEMENT LEARNING, there are no explicit target outputs or environmental evaluations associated with each input; rather the unsupervised learner brings to bear prior biases as to what aspects of the structure of the input should be captured in the output.  Unsupervised learning is a type of machine learning algorithm used to draw inferences from datasets consisting of input data without labeled responses. The most common unsupervised learning method is cluster analysis, which is used for exploratory data analysis to find hidden patterns or grouping in data.  **K-Means Clustering**  K-means ([MacQueen, 1967](http://home.deib.polimi.it/matteucc/Clustering/tutorial_html/kmeans.html#macqueen)) is one of the simplest unsupervised learning algorithms that solve the well known clustering problem. The procedure follows a simple and easy way to classify a given data set through a certain number of clusters (assume k clusters) fixed a priori. The main idea is to define k centroids, one for each cluster. These centroids shoud be placed in a cunning way because of different location causes different result. So, the better choice is to place them as much as possible far away from each other. The next step is to take each point belonging to a given data set and associate it to the nearest centroid. When no point is pending, the first step is completed and an early groupage is done. At this point we need to re-calculate k new centroids as barycenters of the clusters resulting from the previous step. After we have these k new centroids, a new binding has to be done between the same data set points and the nearest new centroid. A loop has been generated. As a result of this loop we may notice that the k centroids change their location step by step until no more changes are done. In other words centroids do not move any more. Finally, this algorithm aims at minimizing an *objective function*, in this case a squared error function. The objective function:  image009.gif  where  image011.gif is a chosen distance measure between a data point image013.gif and the cluster centre image015.gif, is an indicator of the distance of the *n* data points from their respective cluster centres.  The algorithm is composed of the following steps:   1. Place K points into the space represented by the objects that are being clustered. These points represent initial group centroids. 2. Assign each object to the group that has the closest centroid. 3. When all objects have been assigned, recalculate the positions of the K centroids. 4. Repeat Steps 2 and 3 until the centroids no longer move. This produces a separation of the objects into groups from which the metric to be minimized can be calculated*.*   **Advantages**  1) Fast, robust and easier to understand.  2) Relatively efficient: O(tknd), where n is # objects, k is # clusters, d is # dimension of each object, and t  is # iterations. Normally, k, t, d << n.  3) Gives best result when data set are distinct or well separated from each other.  **Disadvantages**  1) The learning algorithm requires apriori specification of the number of  cluster centers.  2) The use of  Exclusive Assignment - If  there are two highly overlapping data then k-means will not be able to resolve       that there are two clusters.  3) The learning algorithm is not invariant to non-linear transformationsi.e.with different representation of data we get      different results (data represented in form of cartesian co-ordinates and polar co-ordinates will give different results).  4) Euclidean distance measures can unequally weight underlying factors.  5) The learning algorithm provides the local optima of the squared error function.  6) Randomly choosing of the cluster center cannot lead us to the fruitful result.  7) Applicable only when mean is defined i.e. fails for categorical data.  8) Unable to handle noisy data and outliers*.*    **Hierarchical Clustering**  Cluster Analysis (data segmentation) has a variety of goals that relate to grouping or segmenting a collection of objects (i.e., observations, individuals, cases, or data rows) into subsets or clusters, such that those within each cluster are more closely related to one another than objects assigned to different clusters. Central to all of the goals of cluster analysis is the notion of degree of similarity (or dissimilarity) between the individual objects being clustered. There are two major methods of clustering: hierarchical clustering and k-means clustering. For information on k-means clustering, refer to the k-Means Clustering section.  In hierarchical clustering, the data is not partitioned into a particular cluster in a single step. Instead, a series of partitions takes place, which may run from a single cluster containing all objects to n clusters that each contain a single object. Hierarchical Clustering is subdivided into agglomerative methods, which proceed by a series of fusions of the n objects into groups, and divisive methods, which separate n objects successively into finer groupings. Agglomerative techniques are more commonly used, and this is the method implemented in XLMiner. Hierarchical clustering may be represented by a two-dimensional diagram known as a dendrogram, which illustrates the fusions or divisions made at each successive stage of analysis. Following is an example of a dendrogram.  Clustering1.gif  **Agglomerative methods**  An agglomerative hierarchical clustering procedure produces a series of partitions of the data, Pn, Pn-1, ....... , P1. The first Pn consists of n single object clusters, the last P1, consists of single group containing all n cases.  At each particular stage, the method joins together the two clusters that are closest together (most similar).  (At the first stage, this amounts to joining together the two objects that are closest together, since at the initial stage each cluster has only one object.)    **Divisive method**  In this method we assign all of the observations to a single cluster and then partition the cluster to two least similar clusters. Finally, we proceed recursively on each cluster until there is one cluster for each observation.  Clustering_h2.png  **Single Linkage**  In single linkage hierarchical clustering, the distance between two clusters is defined as the *shortest* distance between two points in each cluster. For example, the distance between clusters “r” and “s” to the left is equal to the length of the arrow between their two closest points.  Clustering_single.png  **Complete Linkage**  In complete linkage hierarchical clustering, the distance between two clusters is defined as the *longest* distance between two points in each cluster. For example, the distance between clusters “r” and “s” to the left is equal to the length of the arrow between their two furthest points.  Clustering_complete.png  **Average Linkage**   Average linkage hierarchical clustering, the distance between two clusters is defined as the average distance between each point in one cluster to every point in the other cluster. For example, the distance between clusters “r” and “s” to the left is equal to the average length each arrow between connecting the points of one cluster to the other.  Clustering_average.png  If data is small then Hierarchical Clustering can be done and if it is large then go for K-Means Clustering. |
|  |

|  |
| --- |
| **Conclusion** |
| Both clustering algorithms were implemented successfully. Use of  hierarchical Clustering for small dataset, and K-Means Clustering for large dataset is good. |

**CODE :**

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

df = pd.DataFrame({

    'x': [12, 20, 28, 18, 29, 33, 24, 45, 45, 52, 51, 52, 55, 53, 55, 61, 64, 69, 72],

     'y': [39, 36, 30, 52, 54, 46, 55, 59, 63, 70, 66, 63, 58, 23, 14, 8, 19, 7, 24]

})

np.random.seed(200)

k = 3

# centroids[i] = [x, y]

centroids = {

    i+1: [np.random.randint(0, 80), np.random.randint(0, 80)]

    for i in range(k)

}

fig = plt.figure(figsize=(5, 5))

plt.scatter(df['x'], df['y'], color='k')

colmap = {1: 'r', 2: 'g', 3: 'b'}

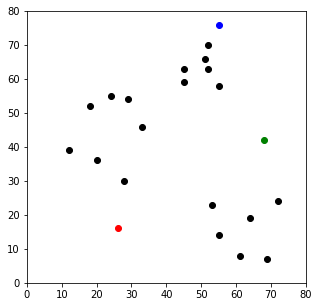
for i in centroids.keys():

    plt.scatter(\*centroids[i], color=colmap[i])

plt.xlim(0, 80)

plt.ylim(0, 80)

plt.show()



def assignment(df, centroids):

    for i in centroids.keys():

        # sqrt((x1 - x2)^2 - (y1 - y2)^2)

        df['distance\_from\_{}'.format(i)] = (

            np.sqrt(

                (df['x'] - centroids[i][0]) \*\* 2

                + (df['y'] - centroids[i][1]) \*\* 2

            )

        )

    centroid\_distance\_cols = ['distance\_from\_{}'.format(i) for i in centroids.keys()]

    df['closest'] = df.loc[:, centroid\_distance\_cols].idxmin(axis=1)

    df['closest'] = df['closest'].map(lambda x: int(x.lstrip('distance\_from\_')))

    df['color'] = df['closest'].map(lambda x: colmap[x])

    return df

df = assignment(df, centroids)

print(df.head())

x y distance\_from\_1 distance\_from\_2 distance\_from\_3 closest color

0 12 39 26.925824 56.080300 56.727418 1 r

1 20 36 20.880613 48.373546 53.150729 1 r

2 28 30 14.142136 41.761226 53.338541 1 r

3 18 52 36.878178 50.990195 44.102154 1 r

4 29 54 38.118237 40.804412 34.058773 3 b

fig = plt.figure(figsize=(5, 5))

plt.scatter(df['x'], df['y'], color=df['color'], alpha=0.5, edgecolor='k')

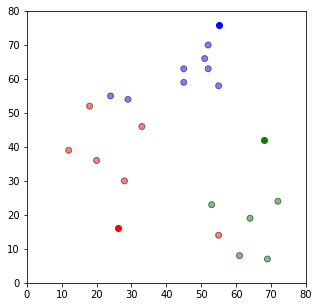
for i in centroids.keys():

    plt.scatter(\*centroids[i], color=colmap[i])

plt.xlim(0, 80)

plt.ylim(0, 80)

plt.show()



import copy

old\_centroids = copy.deepcopy(centroids)

def update(k):

    for i in centroids.keys():

        centroids[i][0] = np.mean(df[df['closest'] == i]['x'])

        centroids[i][1] = np.mean(df[df['closest'] == i]['y'])

    return k

centroids = update(centroids)

fig = plt.figure(figsize=(5, 5))

ax = plt.axes()

plt.scatter(df['x'], df['y'], color=df['color'], alpha=0.5, edgecolor='k')

for i in centroids.keys():

    plt.scatter(\*centroids[i], color=colmap[i])

plt.xlim(0, 80)

plt.ylim(0, 80)

for i in old\_centroids.keys():

    old\_x = old\_centroids[i][0]

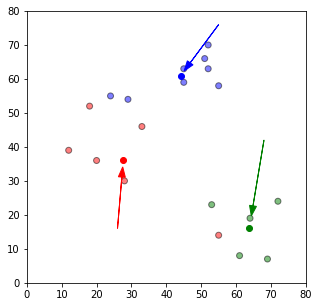
    old\_y = old\_centroids[i][1]

    dx = (centroids[i][0] - old\_centroids[i][0]) \* 0.75

    dy = (centroids[i][1] - old\_centroids[i][1]) \* 0.75

    ax.arrow(old\_x, old\_y, dx, dy, head\_width=2, head\_length=3, fc=colmap[i], ec=colmap[i])

plt.show()



df = assignment(df, centroids)

fig = plt.figure(figsize=(5, 5))

plt.scatter(df['x'], df['y'], color=df['color'], alpha=0.5, edgecolor='k')

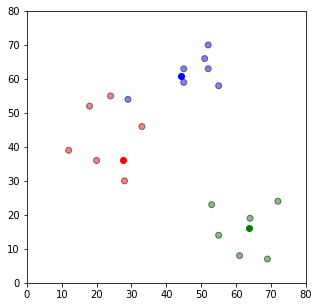
for i in centroids.keys():

    plt.scatter(\*centroids[i], color=colmap[i])

plt.xlim(0, 80)

plt.ylim(0, 80)

plt.show()



while True:

    closest\_centroids = df['closest'].copy(deep=True)

    centroids = update(centroids)

    df = assignment(df, centroids)

    if closest\_centroids.equals(df['closest']):

        break

    fig = plt.figure(figsize=(5, 5))

    plt.scatter(df['x'], df['y'], color=df['color'], alpha=0.5, edgecolor='k')

    for i in centroids.keys():

        plt.scatter(\*centroids[i], color=colmap[i])

    plt.xlim(0, 80)

    plt.ylim(0, 80)

    plt.show()

