https://www.tutorialspoint.com/artificial\_intelligence\_with\_python/index.htm

Unsupervised machine learning algorithms do not have any supervisor to provide any sort of guidance. That is why they are closely aligned with what some call true artificial intelligence.

In unsupervised learning, there would be no correct answer and no teacher for the guidance. Algorithms need to discover the interesting pattern in data for learning.

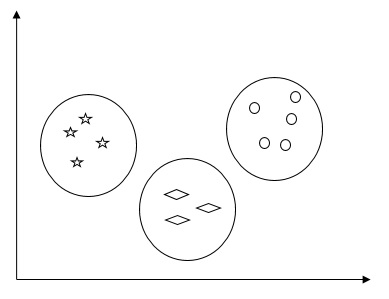
What is Clustering?

It is a type of unsupervised learning method and a common technique for statistical data analysis used in many fields.

Clustering mainly is a task of dividing the set of observations into subsets, called clusters, in such a way that observations in the same cluster are similar in one sense and they are dissimilar to the observations in other clusters.

**In simple words, we can say that the main goal of clustering is to group the data on the basis of similarity and dissimilarity.**

For example, the following diagram shows similar kind of data in different clusters −



Algorithms for Clustering the Data

Following are a few common algorithms for clustering the data

1. **K-Means algorithm**

K-means clustering algorithm is one of the well-known algorithms for clustering the data.

**Step 1** − We need to specify the desired number of K subgroups. (possible groups)

**Step 2** − Fix the number of clusters and randomly assign each data point to a cluster. Or in other words we need to classify our data based on the number of clusters.

In this step, cluster centroids should be computed.

As this is an iterative algorithm, we need to update the locations of K centroids with every iteration until we find the global optima or in other words the centroids reach at their optimal locations.

import matplotlib.pyplot as plt

import seaborn as sns; sns.set()

import numpy as np

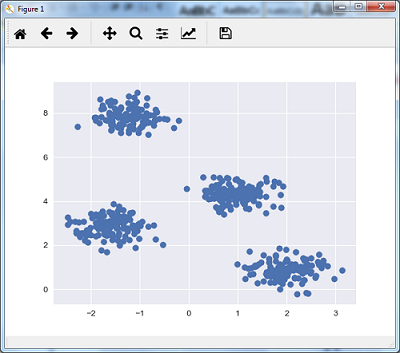
from sklearn.cluster import KMeans

from sklearn.datasets.samples\_generator import make\_blobs

X, y\_true = make\_blobs(n\_samples = 500, centers = 4, cluster\_std = 0.40, random\_state = 0)

plt.scatter(X[:, 0], X[:, 1], s = 50);

plt.show()



Con este grafico podemos asumir que posiblemente hay 4 grupos a clasificar K=4

#Initialize model assuming there 4 groups (according to the graph above!!

kmeans = KMeans(n\_clusters = 4)

#Entrene modelo

kmeans.fit(X)

y\_kmeans = kmeans.predict(X)

#Grafique lo que encontro maquina

#Lo que encontro maquina esta en modelo.cluster\_centers

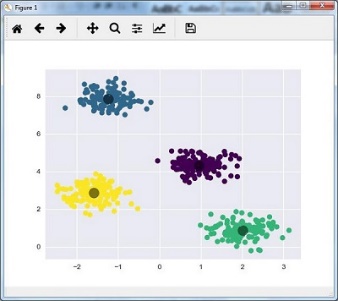
#kmeans.cluster\_centers

plt.scatter(X[:, 0], X[:, 1], c = y\_kmeans, s = 50, cmap = 'viridis')

centers = kmeans.cluster\_centers\_

plt.scatter(centers[:, 0], centers[:, 1], c = 'black', s = 200, alpha = 0.5);

plt.show()



**2) Mean Shift Algorithm**

It is another popular and powerful clustering algorithm used in unsupervised learning. It does not make any assumptions hence it is a non-parametric algorithm. It is also called hierarchical clustering or mean shift cluster analysis. Followings would be the basic steps of this algorithm −

* First of all, we need to start with the data points assigned to a cluster of their own.
* Now, it computes the centroids and update the location of new centroids.
* By repeating this process, we move closer the peak of cluster i.e. towards the region of higher density.
* This algorithm stops at the stage where centroids do not move anymore.

With the help of following code we are implementing Mean Shift clustering algorithm in Python. We are going to use Scikit-learn module.

Let us import the necessary packages −

import numpy as np

from sklearn.cluster import MeanShift

import matplotlib.pyplot as plt

from matplotlib import style

style.use("ggplot")

The following code will help in generating the two-dimensional dataset, containing four blobs, by using **make\_blob** from the **sklearn.dataset** package.

from sklearn.datasets.samples\_generator import make\_blobs

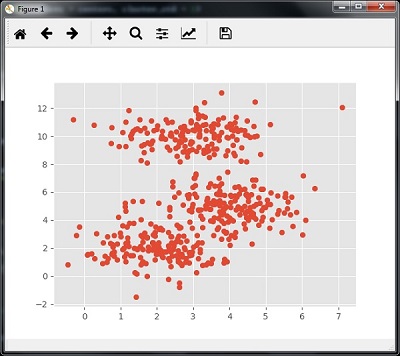
We can visualize the dataset with the following code

centers = [[2,2],[4,5],[3,10]]

X, \_ = make\_blobs(n\_samples = 500, centers = centers, cluster\_std = 1)

plt.scatter(X[:,0],X[:,1])

plt.show()



Now, we need to train the Mean Shift cluster model with the input data.

ms = MeanShift()

ms.fit(X)

labels = ms.labels\_

cluster\_centers = ms.cluster\_centers\_

The following code will print the cluster centers and the expected number of cluster as per the input data −

print(cluster\_centers)

n\_clusters\_ = len(np.unique(labels))

print("Estimated clusters:", n\_clusters\_)

[[ 3.23005036 3.84771893]

[ 3.02057451 9.88928991]]

Estimated clusters: 2

The code given below will help plot and visualize the machine's findings based on our data, and the fitment according to the number of clusters that are to be found.

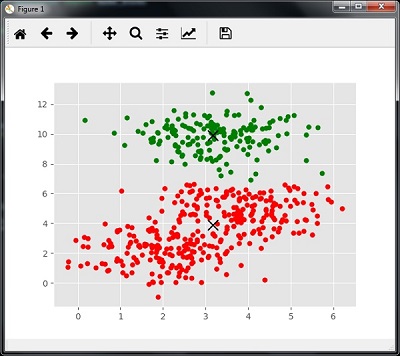
colors = 10\*['r.','g.','b.','c.','k.','y.','m.']

for i in range(len(X)):

plt.plot(X[i][0], X[i][1], colors[labels[i]], markersize = 10)

plt.scatter(cluster\_centers[:,0],cluster\_centers[:,1], marker = "x",color = 'k', s = 150, linewidths = 5, zorder = 10)

plt.show()



Measuring the Clustering Performance

The real world data is not naturally organized into number of distinctive clusters. Due to this reason, it is not easy to visualize and draw inferences.

That is why we need to measure the clustering performance as well as its quality. It can be done with the help of silhouette analysis.

Silhouette Analysis

This method can be used to check the quality of clustering by measuring the distance between the clusters. Basically, it provides a way to assess the parameters like number of clusters by giving a silhouette score. This score is a metric that measures how close each point in one cluster is to the points in the neighboring clusters.

Analysis of silhouette score

The score has a range of [-1, 1]. Following is the analysis of this score −

* **Score of +1** − Score near +1 indicates that the sample is far away from the neighboring cluster.
* **Score of 0** − Score 0 indicates that the sample is on or very close to the decision boundary between two neighboring clusters.
* **Score of -1** − Negative score indicates that the samples have been assigned to the wrong clusters.

Calculating Silhouette Score

In this section, we will learn how to calculate the silhouette score.

Silhouette score can be calculated by using the following formula −

$$silhouette score = \frac{\left ( p-q \right )}{max\left ( p,q \right )}$$

Here, 𝑝 is the mean distance to the points in the nearest cluster that the data point is not a part of. And, 𝑞 is the mean intra-cluster distance to all the points in its own cluster.

For finding the optimal number of clusters, we need to run the clustering algorithm again by importing the **metrics** module from the **sklearn** package. In the following example, we will run the K-means clustering algorithm to find the optimal number of clusters −

Import the necessary packages as shown −

import matplotlib.pyplot as plt

import seaborn as sns; sns.set()

import numpy as np

from sklearn.cluster import KMeans

With the help of the following code, we will generate the two-dimensional dataset, containing four blobs, by using **make\_blob** from the **sklearn.dataset** package.

from sklearn.datasets.samples\_generator import make\_blobs

X, y\_true = make\_blobs(n\_samples = 500, centers = 4, cluster\_std = 0.40, random\_state = 0)

Initialize the variables as shown −

scores = []

values = np.arange(2, 10)

We need to iterate the K-means model through all the values and also need to train it with the input data.

for num\_clusters in values:

kmeans = KMeans(init= 'k-means++', n\_clusters= num\_clusters, n\_init= 10)

kmeans.fit(X)

Now, estimate the silhouette score for the current clustering model using the Euclidean distance metric −

score = metrics.silhouette\_score(X, kmeans.labels\_,

metric = 'euclidean', sample\_size = len(X))

The following line of code will help in displaying the number of clusters as well as Silhouette score.

print("\nNumber of clusters =", num\_clusters)

print("Silhouette score =", score)

scores.append(score)

You will receive the following output −

Number of clusters = 9

Silhouette score = 0.340391138371

num\_clusters = np.argmax(scores) + values[0]

print('\nOptimal number of clusters =', num\_clusters)

Now, the output for optimal number of clusters would be as follows −

Optimal number of clusters = 2