Data Science with Python Programming

- Presentation By Uplatz
- Contact us: https://training.uplatz.com
- Email: info@uplatz.com
- Phone: +44 7836 212635

Uplatz

Clustering

Learning outcomes:

- What is Clustering?
- Clustering Algorithms
- K-Means Clustering
- How does K-Means Clustering work?
- Implementing K-Means Clustering
- Hierarchical Clustering
- Agglomerative Hierarchical clustering
- How does Agglomerative Hierarchical clustering Work?
- Divisive Hierarchical Clustering
- Implementation of Agglomerative Hierarchical Clustering

What is Clustering?

As you know that the **Unsupervised learning** is the training of machine using information that is neither classified nor labelled and allowing the algorithm to act on that information without guidance. Here the task of machine is to group unsorted information according to similarities, patterns and differences without any prior training of data.

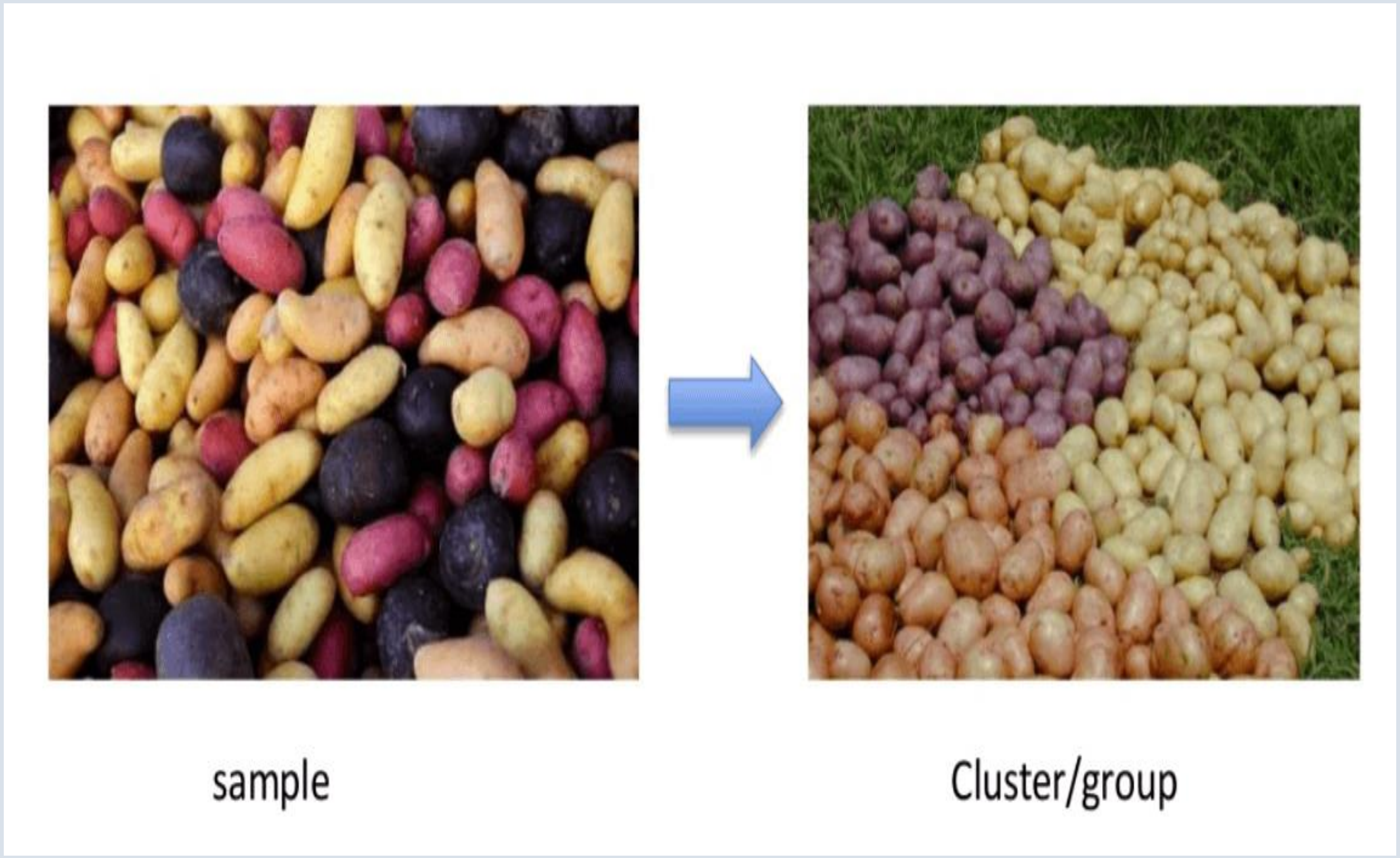
Unsupervised learning classified into two categories of algorithms:

1) **Clustering**
2) **Association**

What is Clustering?

Clustering is an important concept when it comes to unsupervised learning. It mainly deals with finding a structure or pattern in a collection of uncategorized data. Clustering algorithms will process your data and find natural clusters(groups) if they exist in the data. You can also modify how many clusters your algorithms should identify. It allows you to adjust the granularity of these groups. Clustering is one of the most frequently utilized forms of unsupervised learning.

What is Clustering?



Uplatz

What is Clustering?

It is often used as a data analysis technique for discovering interesting patterns in data, such as groups of customers based on their behaviour. There are many clustering algorithms to choose from and no single best clustering algorithm for all cases. Instead, it is a good idea to explore a range of clustering algorithms and different configurations for each algorithm. It involves automatically discovering natural grouping in data. Unlike supervised learning , clustering algorithms only interpret the input data and find natural groups or clusters in feature space.

What is Clustering?

Clustering techniques apply when there is no class to be predicted but rather when the instances are to be divided into natural groups.

Clustering is the task of dividing the population or data points into a number of groups such that data points in the same groups are more similar to other data points in the same group and dissimilar to the data points in other groups.

It is basically a collection of objects on the basis of similarity and dissimilarity between them.

Clustering Algorithms

There are many types of clustering algorithms. Many algorithms use similarity or distance measures between examples in the feature space in an effort to discover dense regions of observations. As such, it is often good practice to scale data prior to using clustering algorithms. Some clustering algorithms require you to specify or guess at the number of clusters to discover in the data, whereas others require the specification of some minimum distance between observations in which examples may be considered "*close*" or "*connected*."

Clustering Algorithms

Each algorithm offers a different approach to the challenge of discovering natural groups in data. There is no best clustering algorithm, and no easy way to find the best algorithm for your data without using controlled experiments. There are various types of clustering algorithms such as K-means clustering, Hierarchical clustering, Mean-Shift Clustering, etc.
Here we will cover some of the important clustering algorithms.

K-Means Clustering

K-means is considered by many to be the gold standard when it comes to clustering due to its simplicity and performance, so it's the first one we'll try out. K-means clustering is one of the simplest and popular unsupervised machine learning algorithms.

When you have no idea at all what algorithm to use, K-means is usually the first choice.

K-means algorithm partition **n** observations into **k** clusters where each observation belongs to the cluster with the nearest mean serving as a prototype of the cluster .

K-Means Clustering

You’ll define a target number ***k***, which refers to the number of centroids you need in the dataset. A centroid is the imaginary or real location representing the centre of the cluster.
In other words, the K-means algorithm identifies ***k*** number of centroids, and then allocates every data point to the nearest cluster, while keeping the centroids as small as possible.
The *‘**means**’* in the K-means refers to averaging of the data; that is, finding the centroid.

K-Means Clustering

K-means is a centroid-based algorithm, or a distance-based algorithm, where we calculate the distances to assign a point to a cluster. In K-Means, each cluster is associated with a centroid.

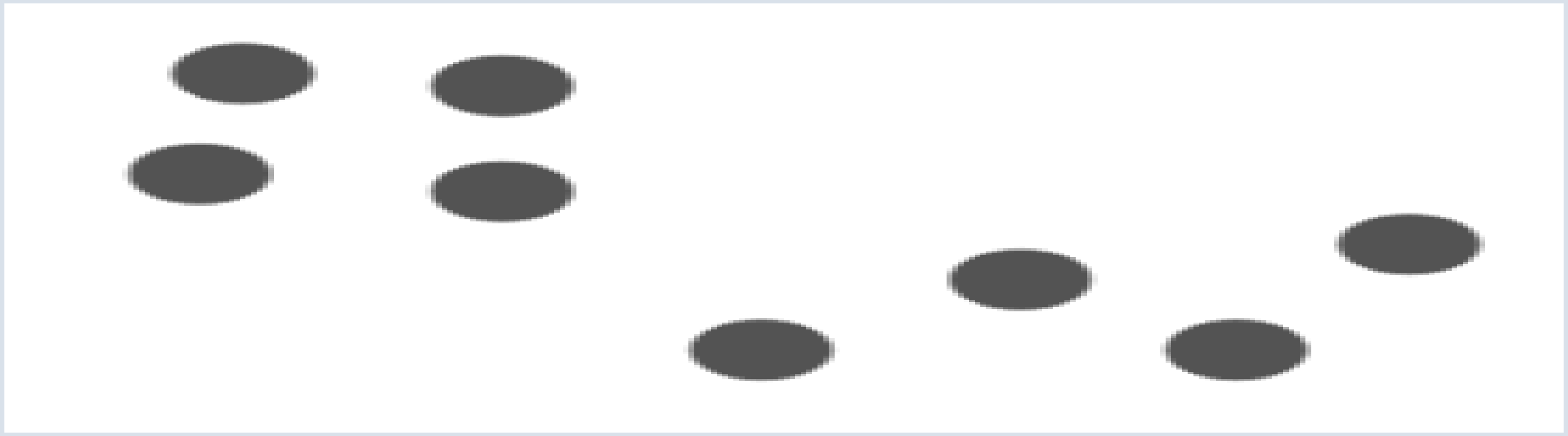
The main objective of the K-Means algorithm is to minimize the sum of distances between the points and their respective cluster centroid.

How does K-Means Clustering work?

K-means is a centroid-based algorithm, or a distance-based algorithm, where we calculate the distances to assign a point to a cluster. In K-Means, each cluster is associated with a centroid.
The main objective of the K-Means algorithm is to minimize the sum of distances between the points and their respective cluster centroid.

How does K-Means Clustering work?

Let’s now take an example to understand how K-Means actually works:
We have these 8 points and we want to apply k-means to create clusters for these points. Here’s how we can do it.



How does K-Means Clustering work?

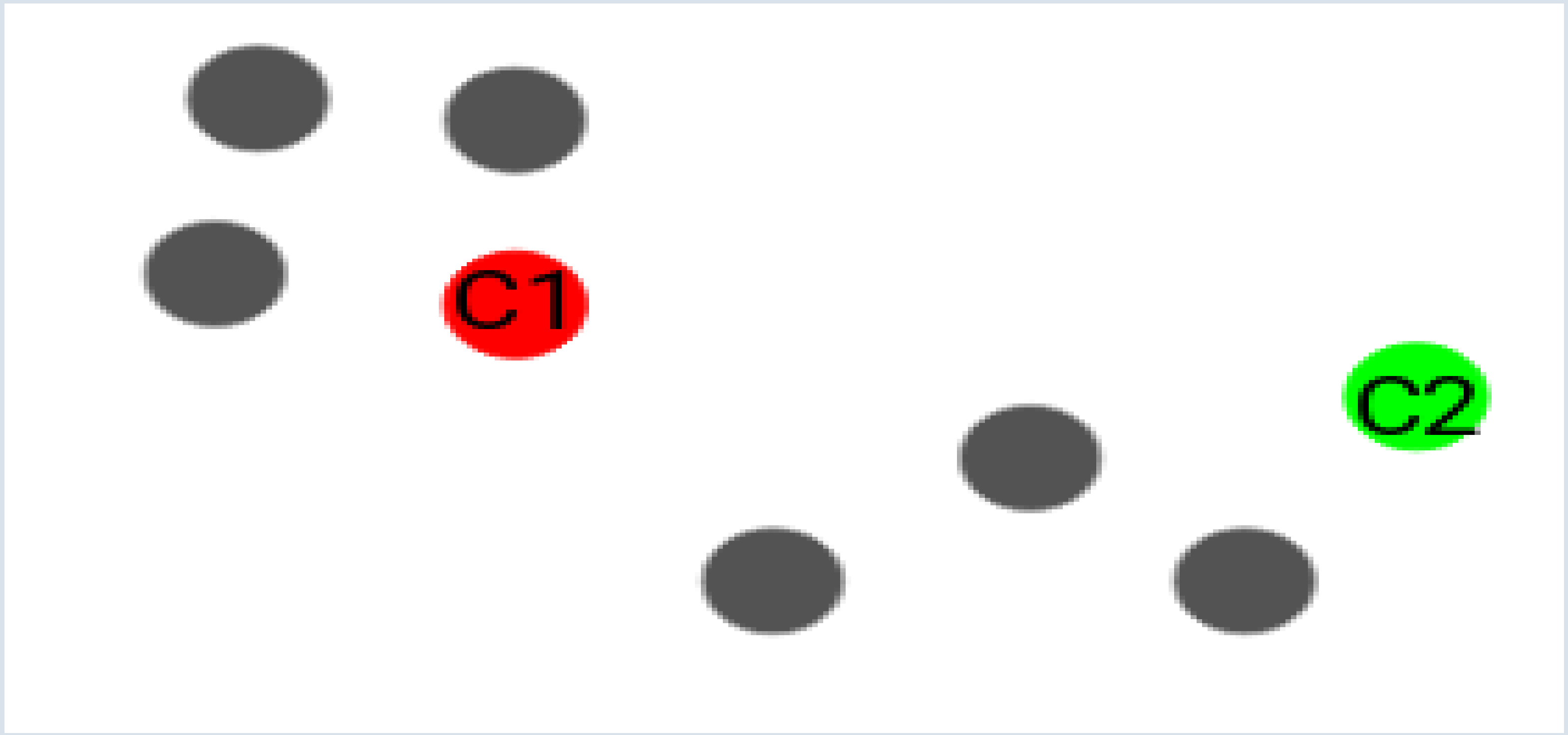
Step 1: Choose the number of clusters *k*

The first step in k-means is to pick the number of clusters, k.

Step 2: Select k random points from the data as centroids

Next, we randomly select the centroid for each cluster. Let’s say we want to have 2 clusters, so k is equal to 2 here. We then randomly select the centroid:

How does K-Means Clustering work?

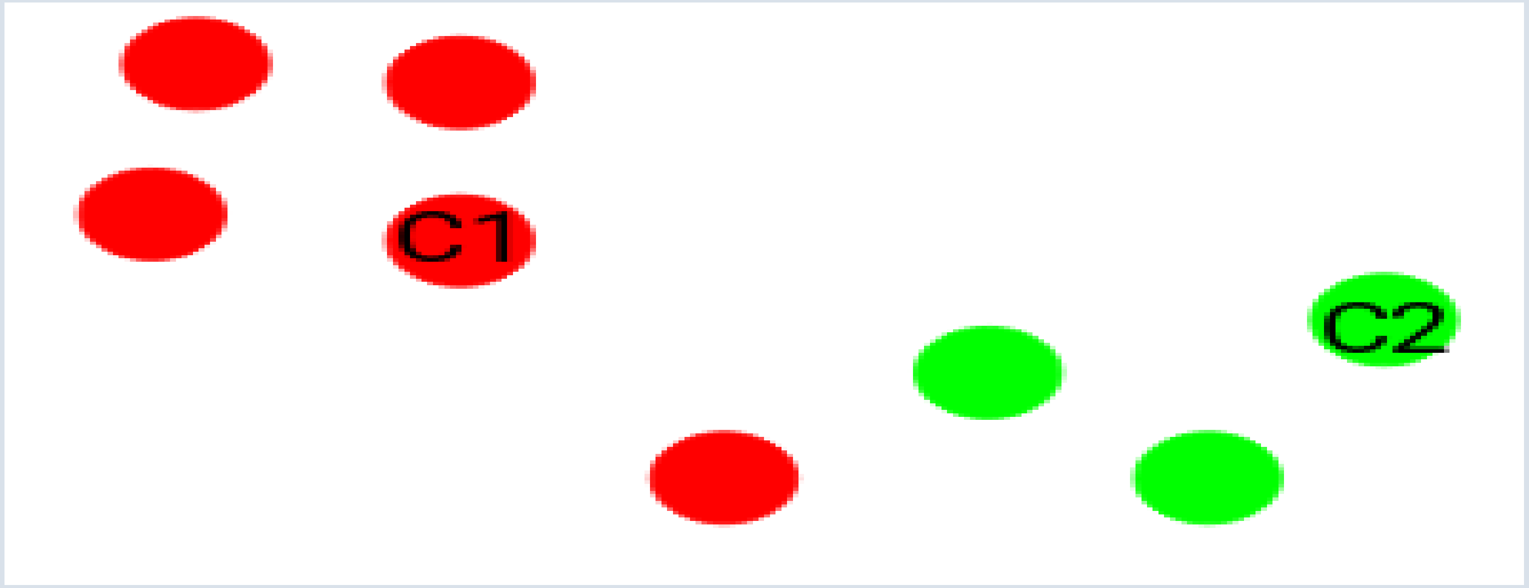


Here, the red and green circles represent the centroid for these clusters.

How does K-Means Clustering work?

Step 3: Assign all the points to the closest cluster centroid

Once we have initialized the centroids, we assign each point to the closest cluster centroid:



Uplatz

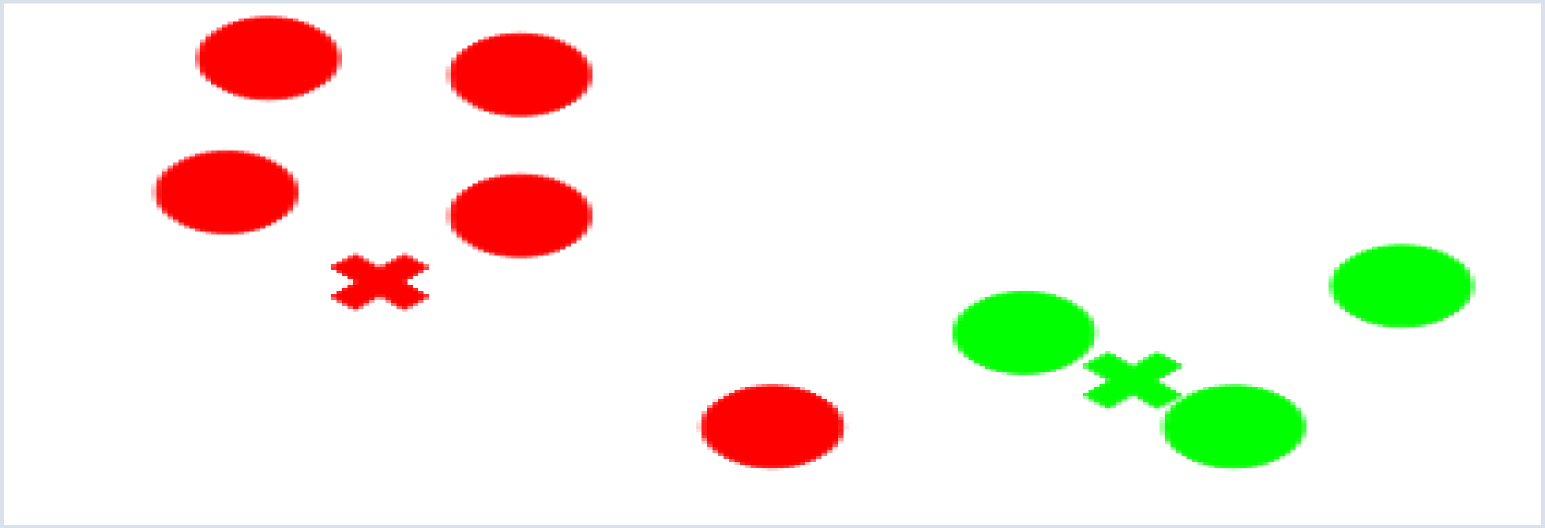
How does K-Means Clustering work?

Here you can see that the points which are closer to the red point are assigned to the red cluster whereas the points which are closer to the green point are assigned to the green cluster.

How does K-Means Clustering work?

Step 4: Re-compute the centroids of newly formed clusters

Now, once we have assigned all of the points to either cluster, the next step is to compute the centroids of newly formed clusters:

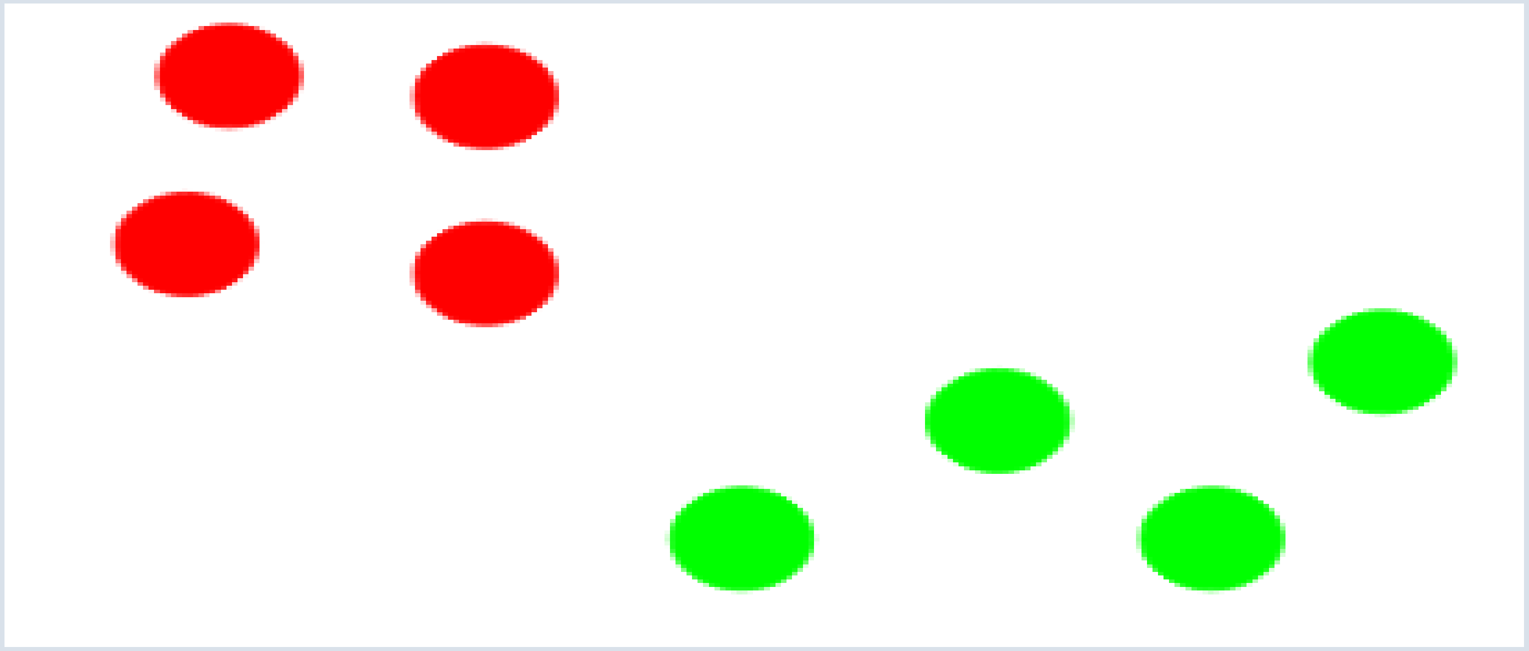


Here, the red and green crosses are the new centroids.

How does K-Means Clustering work?

Step 5: Repeat steps 3 and 4

We then repeat steps 3 and 4:



How does K-Means Clustering work?

The step of computing the centroid and assigning all the points to the cluster based on their distance from the centroid is a single iteration. But wait – when should we stop this process? It can't run till eternity, right?

Stopping Criteria for K-Means Clustering

There are essentially three stopping criteria that can be adopted to stop the K-means algorithm:

Centroids of newly formed clusters do not change

Points remain in the same cluster

Maximum number of iterations are reached

How does K-Means Clustering work?

We can stop the algorithm if the centroids of newly formed clusters are not changing. Even after multiple iterations, if we are getting the same centroids for all the clusters, we can say that the algorithm is not learning any new pattern and it is a sign to stop the training.

Implementing K-Means Clustering

Let’s see the steps on how the K-means machine learning algorithm works using the Python programming language. We’ll use the Scikit-learn library and some random data to illustrate a K-means clustering simple explanation.

Step 1: Import libraries

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import Kmeans
```

Numpy for carrying out efficient computations
Matplotlib for visualization of data

Implementing K-Means Clustering

Step 2: Generate random data

Here is the code for generating some random data in a two-dimensional space:

```
X= -2 * np.random.rand(100,2)
X1 = 1 + 2 * np.random.rand(50,2)
X[50:100, :] = X1
plt.scatter(X[ : , 0], X[ :, 1], s = 50, c = ‘b’)
plt.show()
```

A total of 100 data points has been generated and divided into two groups, of 50 points each.

Here is how the data is displayed on a two-dimensional space:

Implementing K-Means Clustering

Step 3: Training the K-means algorithm, Use Scikit-Learn

We'll use some of the available functions in the Scikit-learn library to process the randomly generated data. As we have got the number of clusters, so we can now train the model on the dataset.

```
from sklearn.cluster import KMeans
Kmean = KMeans(n_clusters=2)
Kmean.fit(X)
```

Implementing K-Means Clustering

Step 4: Finding the centroid

Here is the code for finding the centre of the clusters:

```
Kmean.cluster_centers_
```

```
array([[-1.1028412 , -1.09877634],
[ 1.90423995, 2.06748866]])
```

Let’s display the cluster centroids (using green and red colour).

```
plt.scatter(X[ : , 0], X[ : , 1], c='b')
plt.scatter(-1.1028412, -1.09877634, s=200, c='g',
marker='s')
plt.scatter(1.90423995, 2. 06748866, s=200, c='r',
marker='s')
plt.show()
```

Implementing K-Means Clustering

Step 5: Testing the algorithm

Here is the code for getting the labels property of the K-means clustering example dataset; that is, how the data points are categorized into the two clusters.

```
Kmean.labels_
```

As you can see the result, 50 data points belong to the **0** cluster while the rest belong to the **1** cluster.

Implementing K-Means Clustering

Now let's use the code below for predicting the cluster of a data point:

```
sample_test=np.array([-3.0,-3.0])
second_test=sample_test.reshape(1, -1)
Kmean.predict(second_test)
```

Output: **array([0])**

It shows that the test data point belongs to the 0 (green centroid) cluster.

Hierarchical Clustering

Hierarchical clustering is another unsupervised machine learning algorithm, which is used to group the unlabeled datasets into a cluster and also known as **hierarchical cluster analysis** or HCA.
In this algorithm, we develop the hierarchy of clusters in the form of a tree, and this tree-shaped structure is known as the **dendrogram**.
Sometimes the results of K-means clustering and hierarchical clustering may look similar, but they both differ depending on how they work. As there is no requirement to predetermine the number of clusters as we did in the K-Means algorithm.

Hierarchical Clustering

Why hierarchical clustering?

As we already have other clustering algorithms such as **K-Means Clustering**, then why we need hierarchical clustering? So, as we have seen in the K-means clustering that there are some challenges with this algorithm, which are a predetermined number of clusters, and it always tries to create the clusters of the same size. To solve these two challenges, we can opt for the hierarchical clustering algorithm because, in this algorithm, we don't need to have knowledge about the predefined number of clusters.

Hierarchical Clustering

The hierarchical clustering technique has two approaches: OR you can say hierarchical clustering algorithms falls into following two categories.

Agglomerative: Agglomerative is a **bottom-up** approach, in which the algorithm starts with taking all data points as single clusters and merging them until one cluster is left.

Divisive: Divisive algorithm is the reverse of the agglomerative algorithm as it is a **top-down approach.**

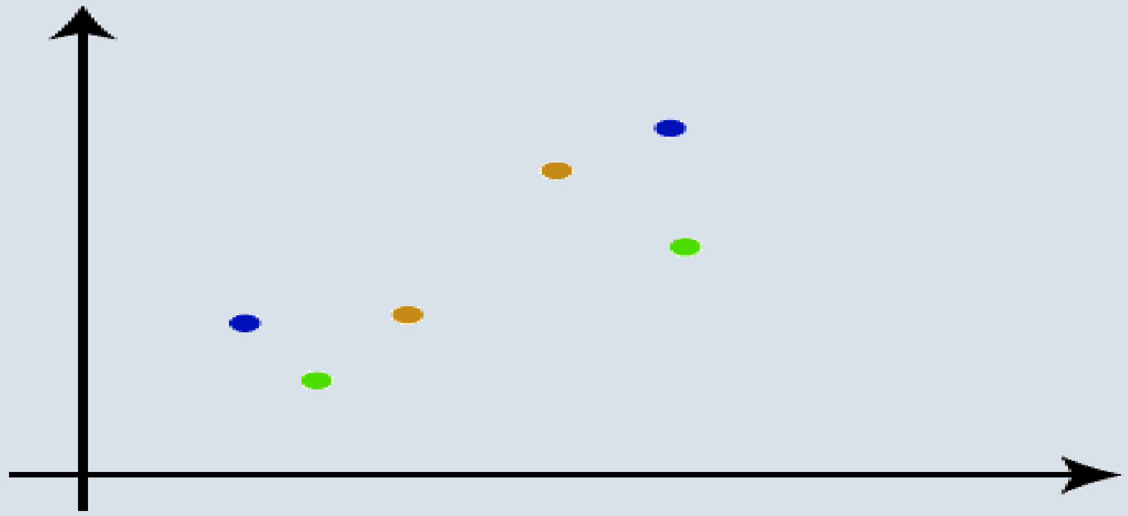
Agglomerative Hierarchical clustering

The agglomerative hierarchical clustering algorithm is a popular example of HCA. To group the datasets into clusters, it follows the **bottom-up approach**. It means, this algorithm considers each dataset as a single cluster at the beginning, and then start combining the closest pair of clusters together. It does this until all the clusters are merged into a single cluster that contains all the datasets.
This hierarchy of clusters is represented in the form of the dendrogram.

How does Agglomerative Hierarchical clustering Work?

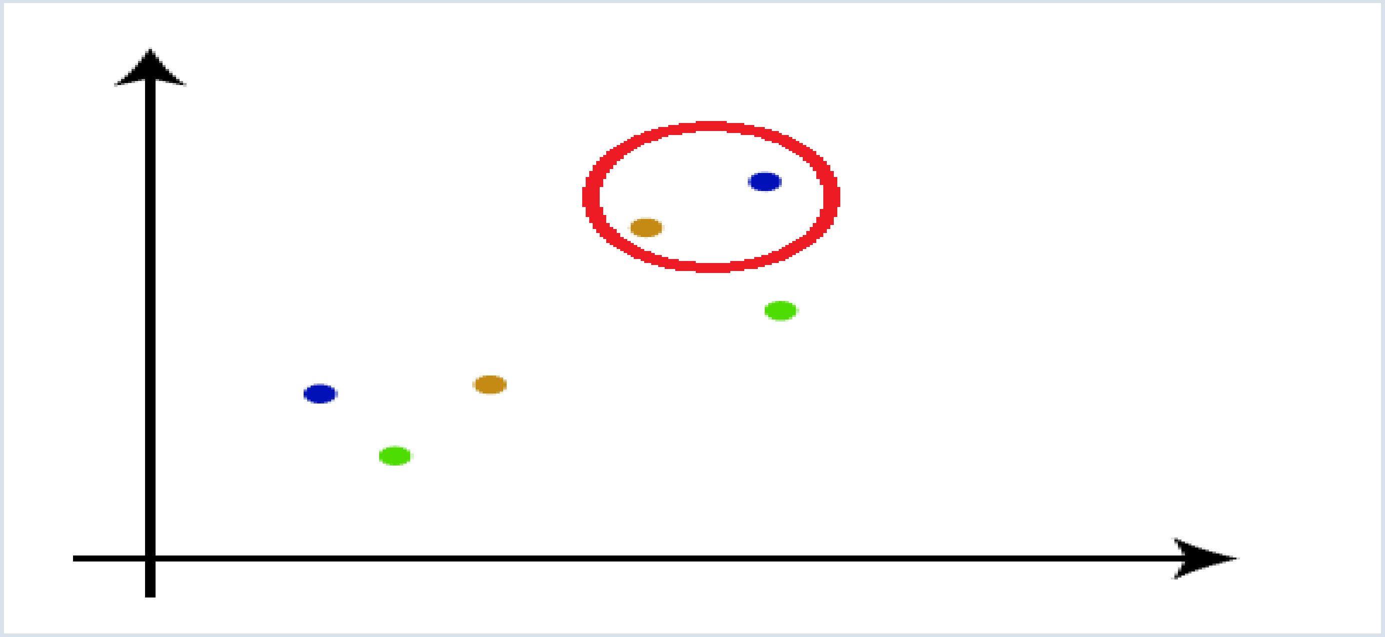
The working of the AHC algorithm can be explained using the below steps:

Step-1: Create each data point as a single cluster. Let's say there are N data points, so the number of clusters will also be N.



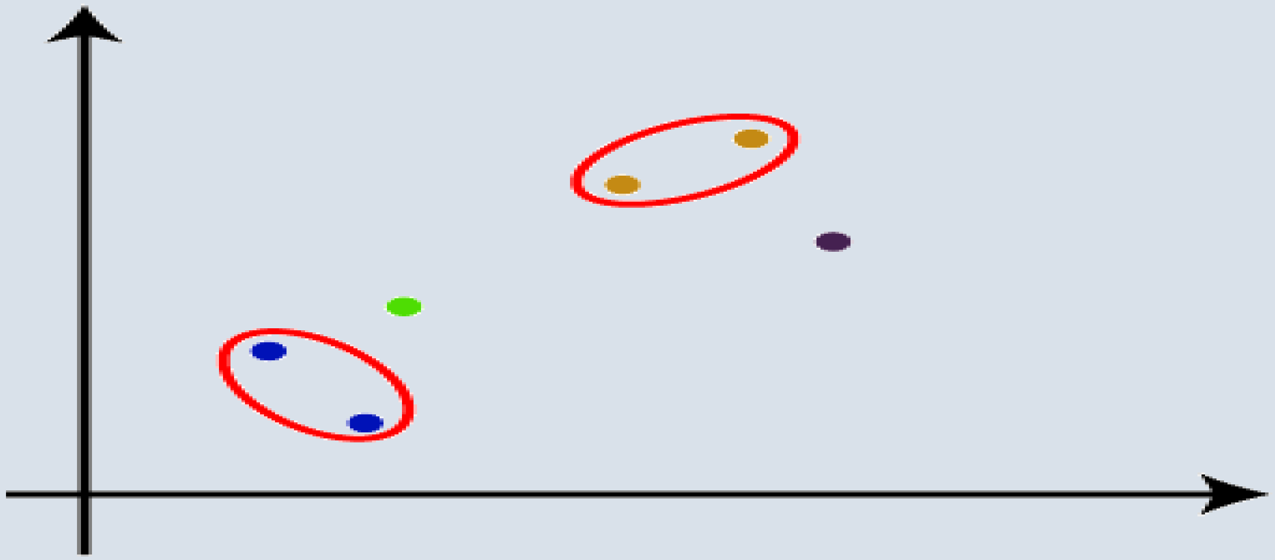
How does Agglomerative Hierarchical clustering Work?

Step-2: Take two closest data points or clusters and merge them to form one cluster. So, there will now be N-1 clusters.



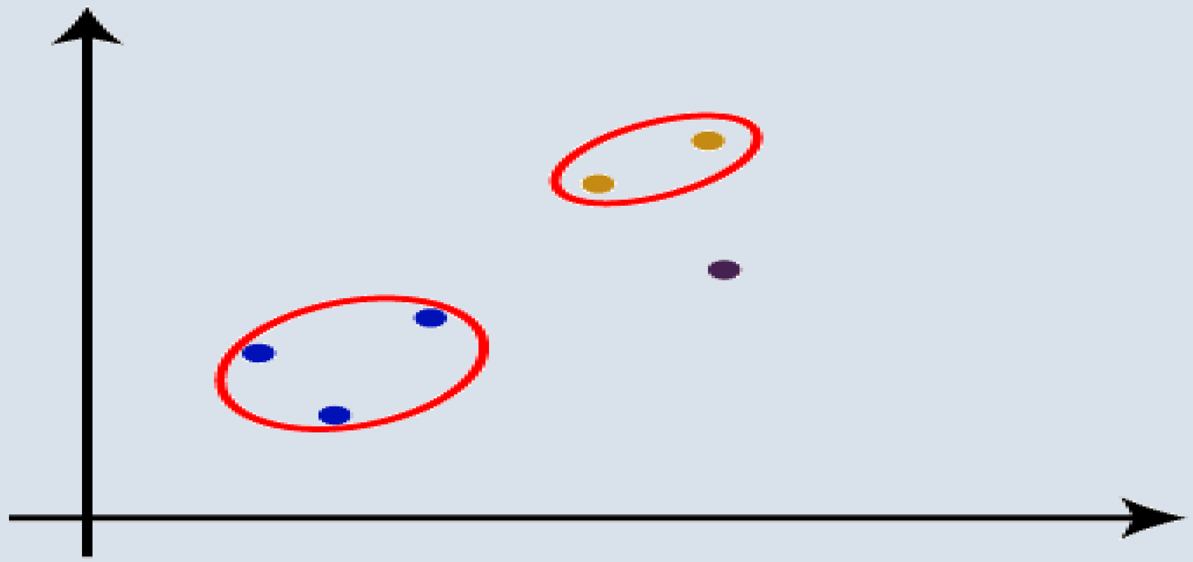
How does Agglomerative Hierarchical clustering Work?

Step-3: Again, take the two closest clusters and merge them together to form one cluster. There will be N-2 clusters.

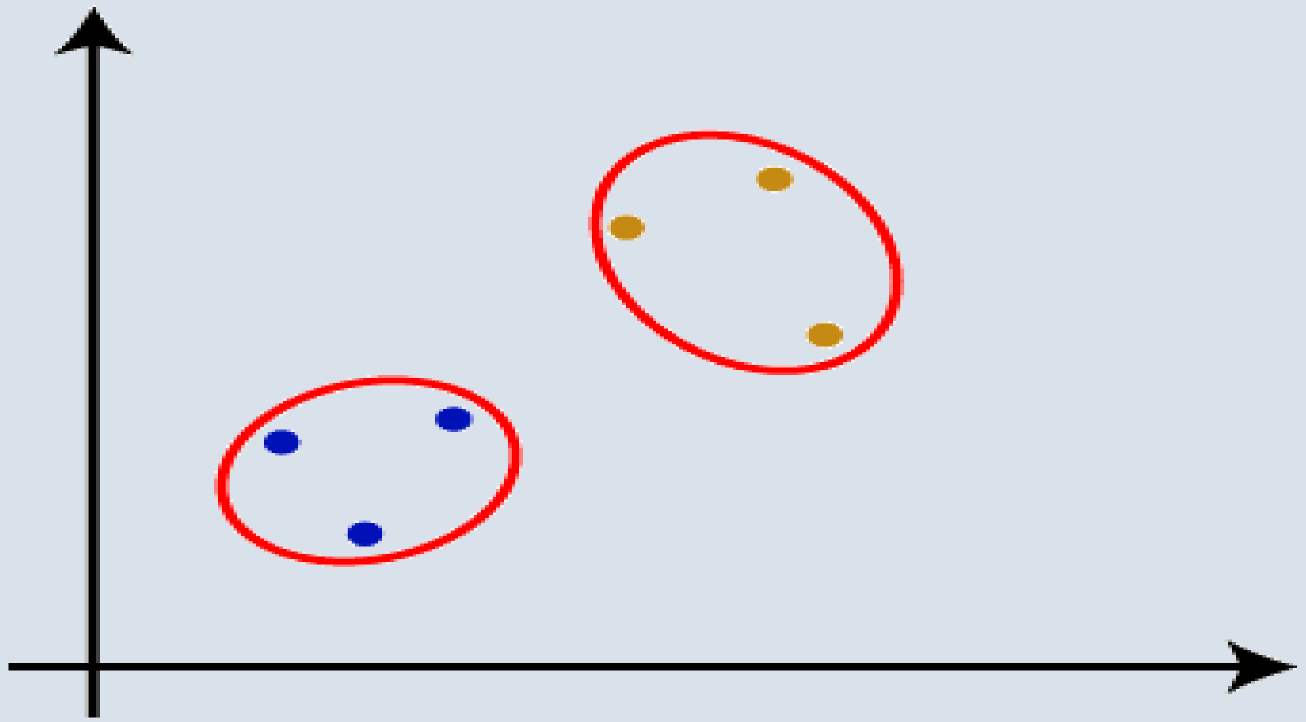


How does Agglomerative Hierarchical clustering Work?

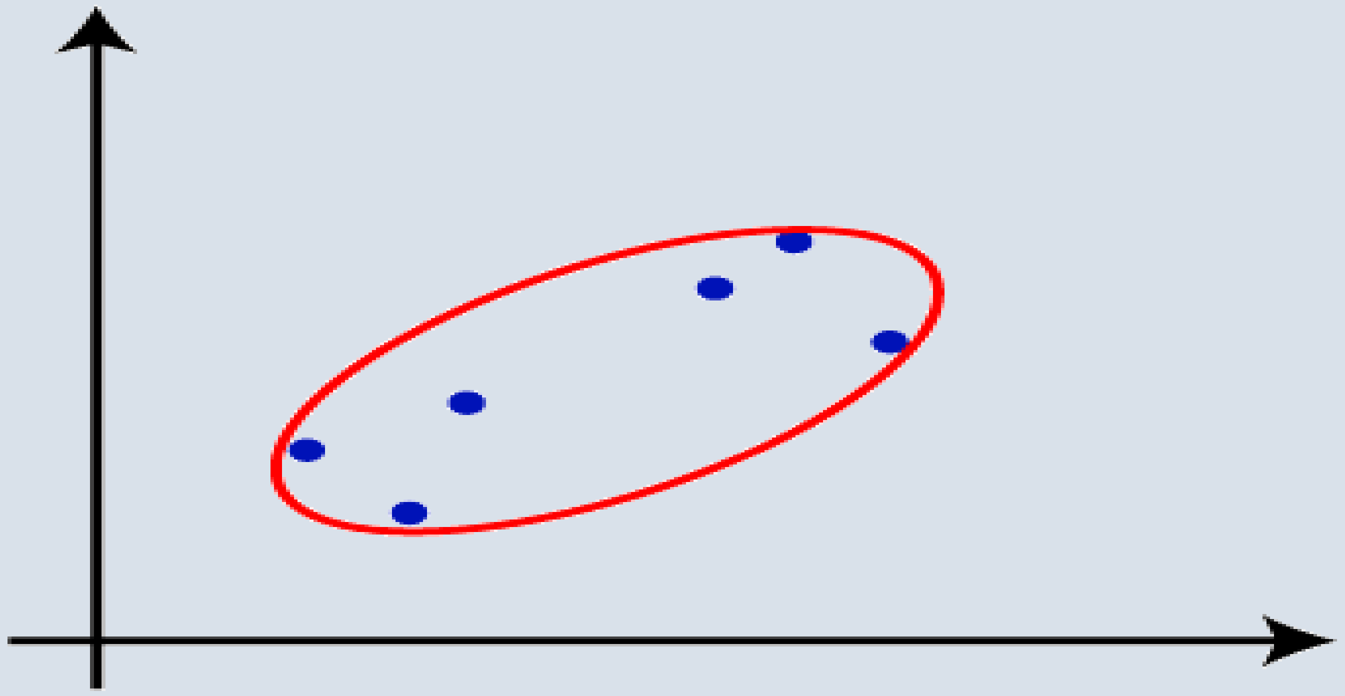
Step-4: Repeat Step 3 until only one cluster left. So, we will get the following clusters. Consider the below images:



How does Agglomerative Hierarchical clustering Work?



How does Agglomerative Hierarchical clustering Work?



How does Agglomerative Hierarchical clustering Work?

Once single cluster is formed, dendrograms are used to divide into multiple clusters depending upon the problem. There are different ways to find distance between the clusters. The distance itself can be Euclidean or Manhattan distance. Following are some of the options to measure distance between two clusters:

Measure the distance between the closes points of two clusters.

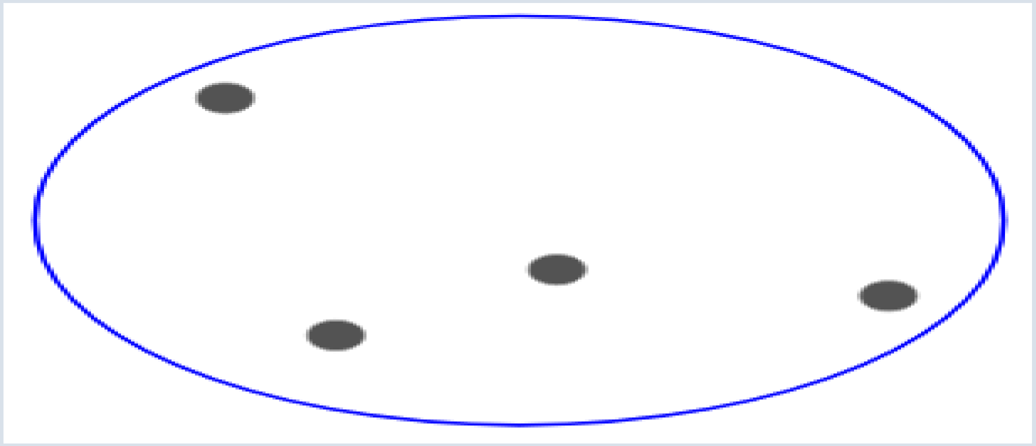
Measure the distance between the farthest points of two clusters.

Measure the distance between the centroids of two clusters.

Measure the distance between all possible combination of points between the two clusters and take the mean.

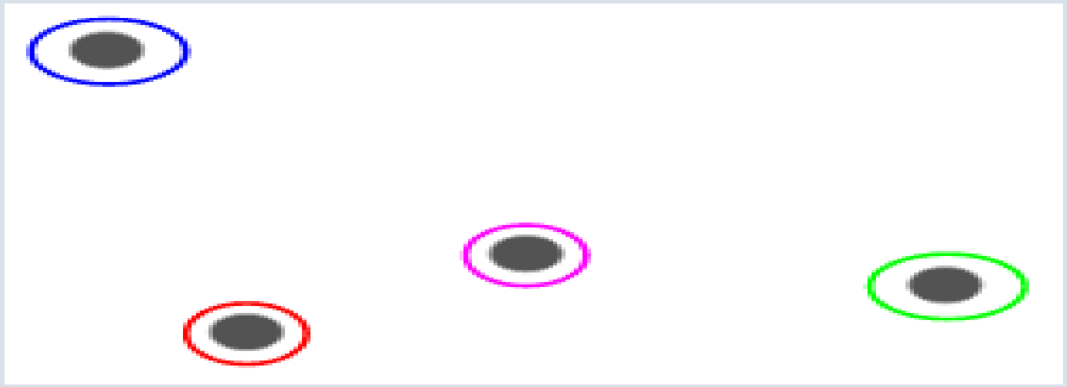
Divisive Hierarchical Clustering

Divisive hierarchical clustering works in the opposite way. Instead of starting with n clusters (in case of n observations), we start with a single cluster and assign all the points to that cluster. So, it doesn't matter if we have 10 or 1000 data points. All these points will belong to the same cluster at the beginning:



Divisive Hierarchical Clustering

Now, at each iteration, we split the farthest point in the cluster and repeat this process until each cluster only contains a single point:

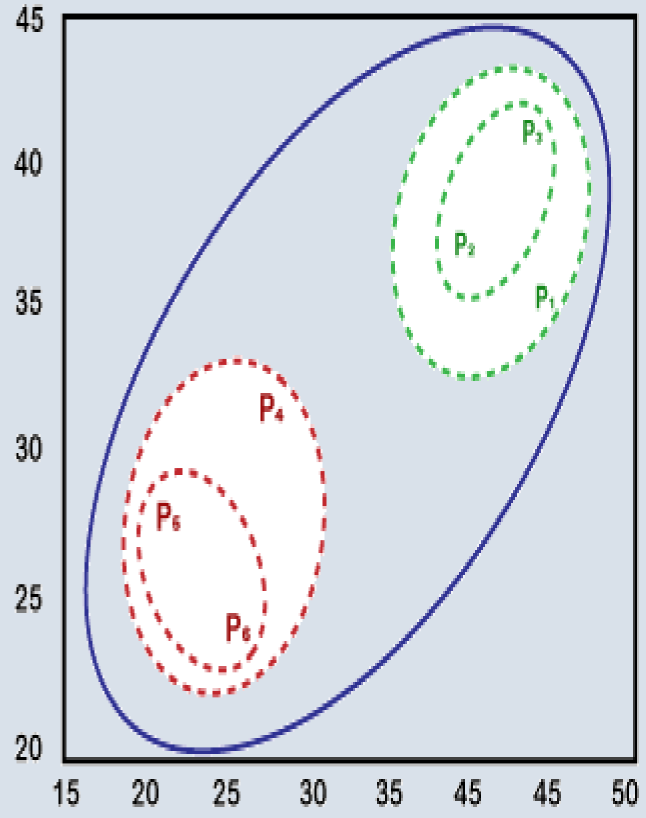


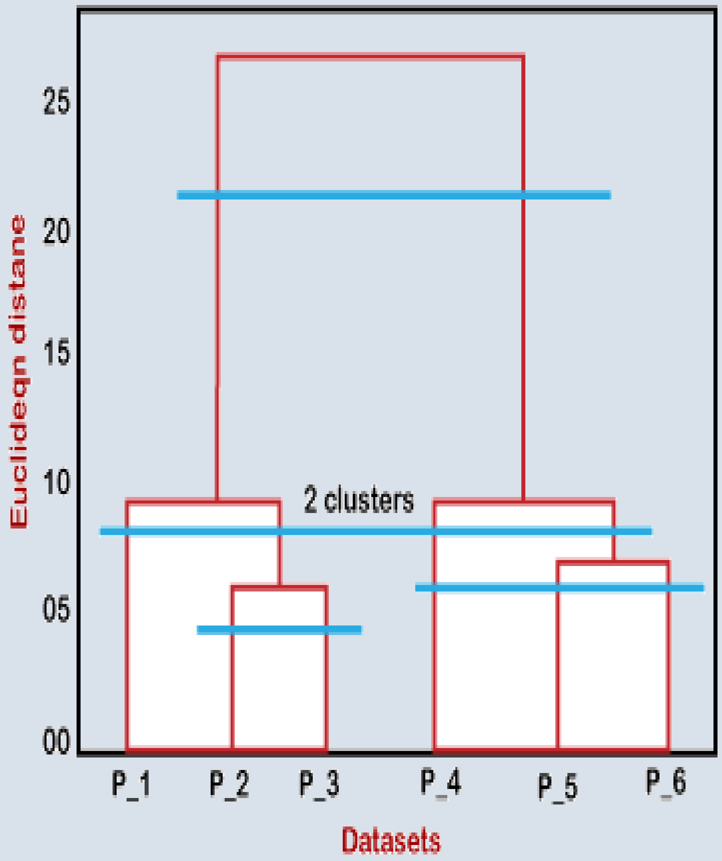
We are splitting (or dividing) the clusters at each step, hence the name divisive hierarchical clustering. Agglomerative Clustering is widely used in the industry and hence we will focucs on it.

Woking of Dendrogram in Hierarchical clustering

The dendrogram is a tree-like structure that is mainly used to store each step as a memory that the HC algorithm performs. In the dendrogram plot, the Y-axis shows the Euclidean distances between the data points, and the x-axis shows all the data points of the given dataset.
The working of the dendrogram can be explained using the below diagram:

Woking of Dendrogram in Hierarchical clustering





Uplatz

Woking of Dendrogram in Hierarchical clustering

In the above diagram, the left part is showing how clusters are created in agglomerative clustering, and the right part is showing the corresponding dendrogram.
As we have discussed above, firstly, the datapoints P2 and P3 combine together and form a cluster, correspondingly a dendrogram is created, which connects P2 and P3 with a rectangular shape. The height is decided according to the Euclidean distance between the data points.

Woking of Dendrogram in Hierarchical clustering

In the next step, P5 and P6 form a cluster, and the corresponding dendrogram is created. It is higher than of previous, as the Euclidean distance between P5 and P6 is a little bit greater than the P2 and P3.
Again, two new dendrograms are created that combine P1, P2, and P3 in one dendrogram, and P4, P5, and P6, in another dendrogram.
At last, the final dendrogram is created that combines all the data points together.

Woking of Dendrogram in Hierarchical clustering

Once one big cluster is formed, the longest vertical distance without any horizontal line passing through it is selected and a horizontal line is drawn through it. The number of vertical lines this newly created horizontal line passes is equal to number of clusters.

Implementation of Agglomerative Hierarchical Clustering

In this example, we will perform hierarchical clustering on real-world data and see how it can be used to solve an actual problem.
The problem that we are going to solve in this section is to segment customers into different groups based on their shopping trends.

Implementation of Agglomerative Hierarchical Clustering

Step 1: Import libraries

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

Step 2: Reading the data from the csv file

```
customer_data = pd.read_csv('C:\Python38\shopping-data.csv')
```

Let's explore our dataset a bit. To check the number of records and attributes, execute the following script:

```
customer_data.shape
```

To eyeball the dataset, execute the head() function of the data frame.

```
customer_data.head()
```

Implementation of Agglomerative Hierarchical Clustering

Step 3: Our dataset has five columns: CustomerID, Gender, Age, Annual Income, and Spending Score. To view the results in two-dimensional feature space, we will retain only two of these five columns. We can remove CustomerID, Gender, and Age column. We will retain the **Annual Income** and **Spending Score (1-100)** columns. The Spending Score column signifies how often a person spends money in a mall on a scale of 1 to 100 with 100 being the highest spender. Execute the following script to filter the first three columns from our dataset:

```
data = customer_data.iloc[:, 3:5].values
```

Implementation of Agglomerative Hierarchical Clustering

Step 4: Next, we need to know the clusters that we want our data to be split to. We will again use the scipy library to create the dendrograms for our dataset. Execute the following script to do so:

```
import scipy.cluster.hierarchy as shc
plt.title("Customer Dendograms")
dend = shc.dendrogram(shc.linkage(data,
method='ward'))
```

Implementation of Agglomerative Hierarchical Clustering

In the script above we import the **hierarchy** class of the **scipy.cluster.hierarchy** library as **shc**. The hierarchy class has a dendrogram method which takes the value returned by the **linkage** method of the same class. The **linkage** method takes the dataset and the method to minimize distances as parameters. We use '**ward**' as the method since it minimizes then variants of distances between the clusters.

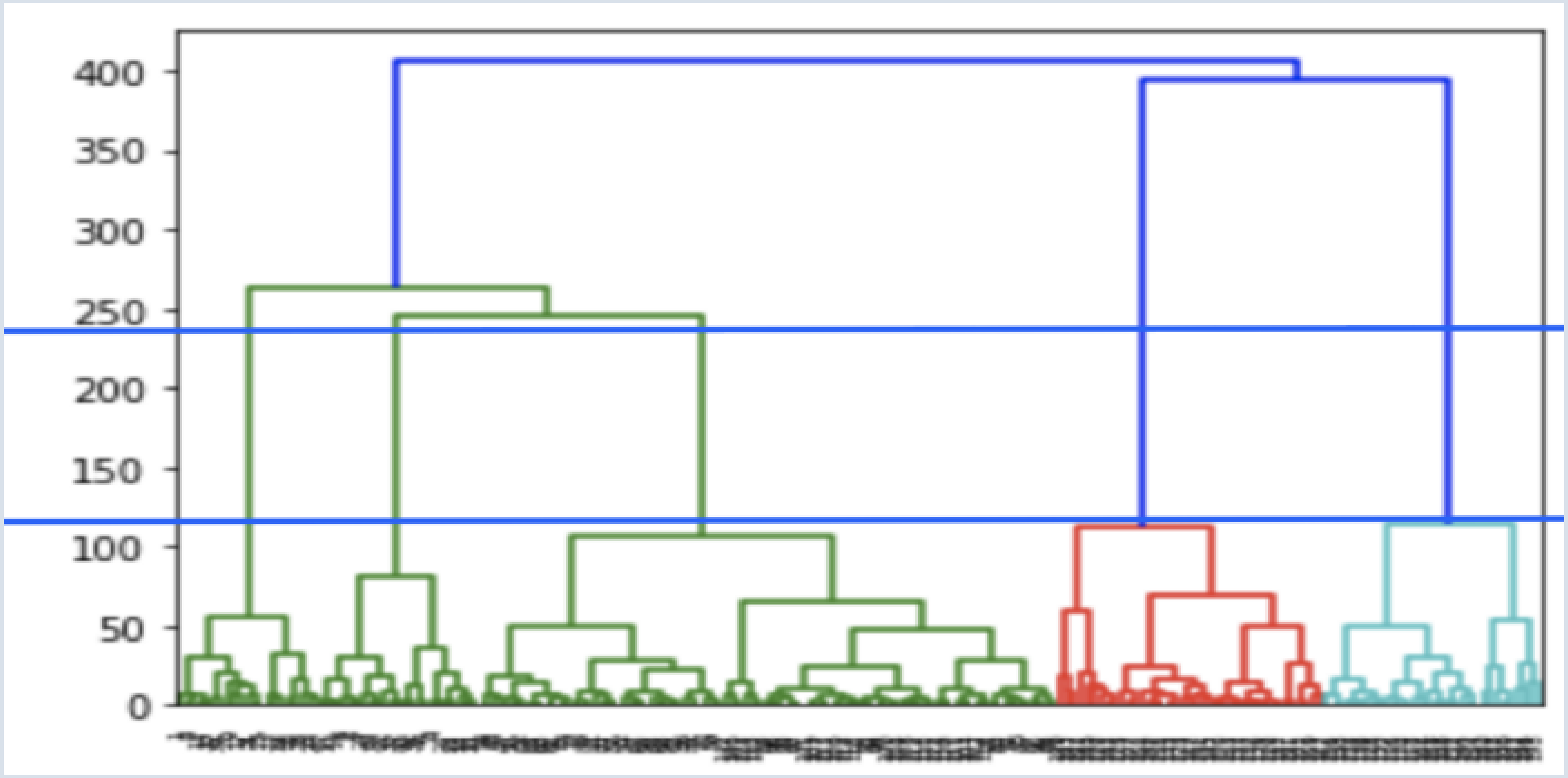
Implementation of Agglomerative Hierarchical Clustering

Step 5:

Now, once the big cluster is formed, the longest vertical distance is selected.

Looking at the dendrogram, the highest vertical distance that doesn't intersect with any clusters is the middle green one. Given that 5 vertical lines cross the threshold, the optimal number of clusters is 5.

Implementation of Agglomerative Hierarchical Clustering



Implementation of Agglomerative Hierarchical Clustering

Step 6: Now we know the number of clusters for our dataset, the next step is to group the data points into these five clusters. To do so we will again use the **AgglomerativeClustering** class of the **sklearn.cluster** library. Take a look at the following script:

```
from sklearn.cluster import
AgglomerativeClustering
cluster = AgglomerativeClustering(n_clusters=5,
affinity='euclidean', linkage='ward')
cluster.fit_predict(data)
```

Implementation of Agglomerative Hierarchical Clustering

Step 6:

You can see the cluster labels from all of your data points. Since we had five clusters, we have five labels in the output i.e. 0 to 4.

Step 7:

As a final step, let's plot the clusters to see how actually our data has been clustered:

```
plt.scatter(data[:,0], data[:,1], c=cluster.labels_,
cmap='rainbow')
```

Implementation of Agglomerative Hierarchical Clustering

You can see the data points in the form of five clusters. The data points in the bottom right belong to the customers with high salaries but low spending. These are the customers that spend their money carefully. Similarly, the customers at top right (green data points), these are the customers with high salaries and high spending. These are the type of customers that companies target. The customers in the middle (blue data points) are the ones with average income and average salaries. The highest numbers of customers belong to this category. Companies can also target these customers given the fact that they are in huge numbers, etc.

Thank you

Uplatz