Unit 2

Unsupervised Machine Learning

In the previous topic, we learned supervised machine learning in which models are trained using labeled data under the supervision of training data. But there may be many cases in which we do not have labeled data and need to find the hidden patterns from the given dataset. So, to solve such types of cases in machine learning, we need unsupervised learning techniques.

What is Unsupervised Learning?

As the name suggests, unsupervised learning is a machine learning technique in which models are not supervised using training dataset. Instead, models itself find the hidden patterns and insights from the given data. It can be compared to learning which takes place in the human brain while learning new things. It can be defined as:

Unsupervised learning is a type of machine learning in which models are trained using unlabeled dataset and are allowed to act on that data without any supervision.

Unsupervised learning cannot be directly applied to a regression or classification problem because unlike supervised learning, we have the input data but no corresponding output data. The goal of unsupervised learning is to **find the underlying structure of dataset, group that data according to similarities, and represent that dataset in a compressed format**.

Example: Suppose the unsupervised learning algorithm is given an input dataset containing images of different types of cats and dogs. The algorithm is never trained upon the given dataset, which means it does not have any idea about the features of the dataset. The task of the unsupervised learning algorithm is to identify the image features on their own. Unsupervised learning algorithm will perform this task by clustering the image dataset into the groups according to similarities between images.

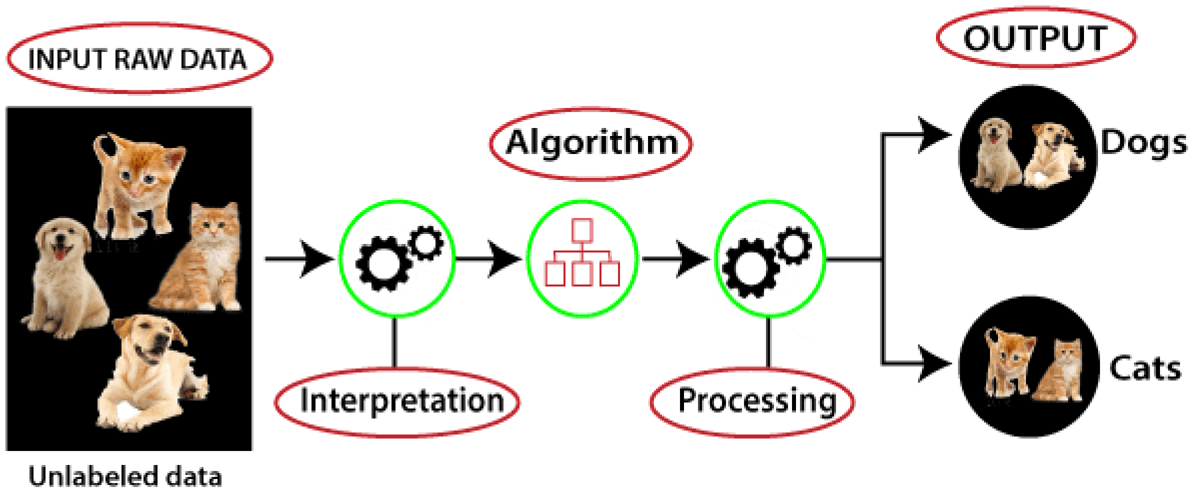
Why use Unsupervised Learning?

Below are some main reasons which describe the importance of Unsupervised Learning:

- Unsupervised learning is helpful for finding useful insights from the data.
- Unsupervised learning is much similar as a human learns to think by their own experiences, which makes it closer to the real AI.
- Unsupervised learning works on unlabeled and uncategorized data which make unsupervised learning more important.
- In real-world, we do not always have input data with the corresponding output so to solve such cases, we need unsupervised learning.

Working of Unsupervised Learning

Working of unsupervised learning can be understood by the below diagram:

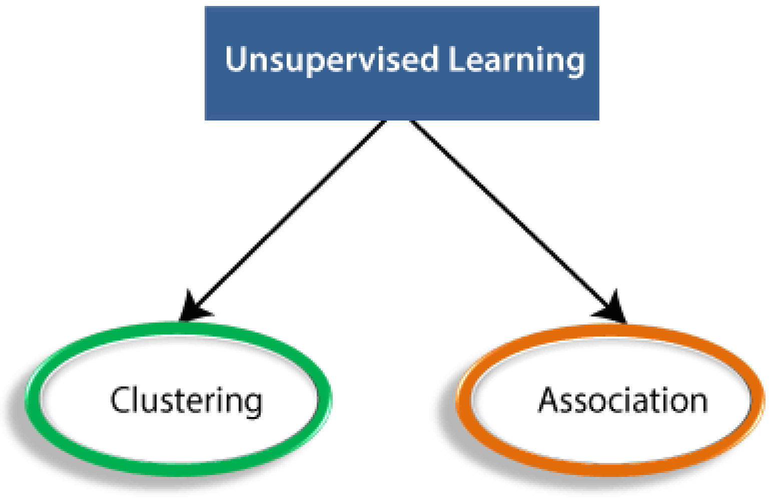


Here, we have taken an unlabeled input data, which means it is not categorized and corresponding outputs are also not given. Now, this unlabeled input data is fed to the machine learning model in order to train it. Firstly, it will interpret the raw data to find the hidden patterns from the data and then will apply suitable algorithms such as k-means clustering, Decision tree, etc.

Once it applies the suitable algorithm, the algorithm divides the data objects into groups according to the similarities and difference between the objects.

Types of Unsupervised Learning Algorithm:

The unsupervised learning algorithm can be further categorized into two types of problems:



- **Clustering**: Clustering is a method of grouping the objects into clusters such that objects with most similarities remains into a group and has less or no similarities with the objects of another group. Cluster analysis finds the commonalities between the data objects and categorizes them as per the presence and absence of those commonalities.
- **Association**: An association rule is an unsupervised learning method which is used for finding the relationships between variables in the large database. It determines the set of items that occurs together in the dataset. Association rule makes marketing strategy more effective. Such as people who buy X item (suppose a bread) are also tend to purchase Y (Butter/Jam) item. A typical example of Association rule is Market Basket Analysis.

Note: We will learn these algorithms in later chapters.

Unsupervised Learning algorithms:

Below is the list of some popular unsupervised learning algorithms:

- **K-means clustering**
- **KNN (k-nearest neighbors)**
- **Hierarchal clustering**
- **Anomaly detection**
- **Neural Networks**
- **Principle Component Analysis**
- **Independent Component Analysis**
- **Apriori algorithm**
- **Singular value decomposition**

Advantages of Unsupervised Learning

- Unsupervised learning is used for more complex tasks as compared to supervised learning because, in unsupervised learning, we don't have labeled input data.

Unsupervised learning is preferable as it is easy to get unlabeled data in comparison to labeled data.

Disadvantages of Unsupervised Learning

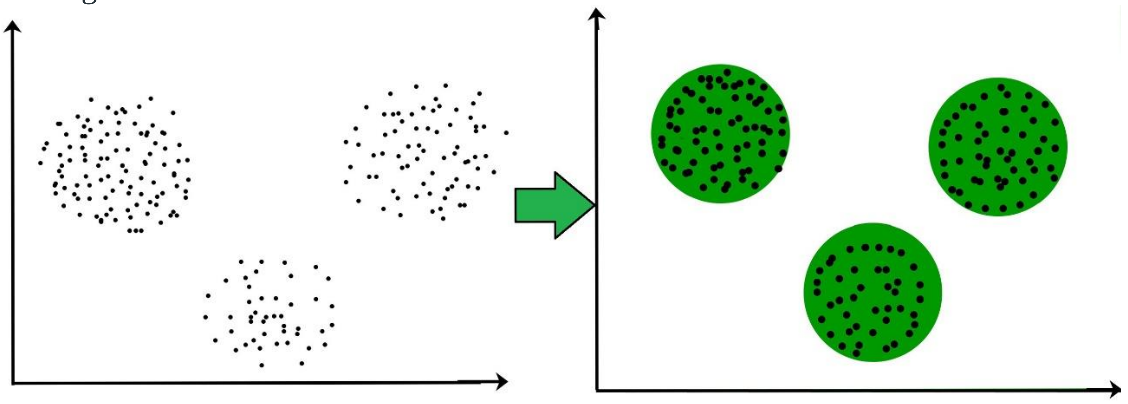
- Unsupervised learning is intrinsically more difficult than supervised learning as it does not have corresponding output.
- The result of the unsupervised learning algorithm might be less accurate as input data is not labeled, and algorithms do not know the exact output in advance.

Clustering in Machine Learning

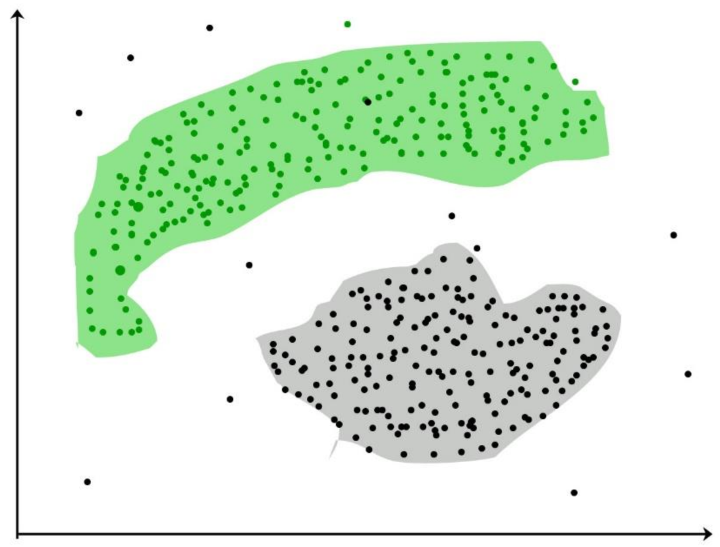
In real world, not every **data we work upon has a target variable**. Have you ever wondered how Netflix groups similar movies together or how Amazon organizes its vast product catalog? These are **real-world applications of clustering**. This kind of data cannot be analyzed using supervised learning algorithms.
When the goal is to group similar data points in a dataset, then we use cluster analysis. In this guide, we'll learn understand concept of clustering, its applications, and some popular clustering algorithms.

What is Clustering?

The task of **grouping data points based on their similarity with each other is called Clustering or Cluster Analysis**. This method is defined under the branch of unsupervised learning, which aims at gaining insights from unlabelled data points.
Think of it as you have a dataset of customers shopping habits. **Clustering can help you group customers with similar purchasing behaviors, which can then be used for targeted marketing, product recommendations, or customer segmentation**
For Example, In the graph given below, we can clearly see that there are 3 circular clusters forming on the basis of distance.



Now it is not necessary that the clusters formed **must be circular in shape**. The shape of clusters can be arbitrary. There are many algorithms that work well with detecting arbitrary shaped clusters.
For example, In the below given graph we can see that the clusters formed are not circular in shape.



Types of Clustering

Broadly speaking, there are 2 types of clustering that can be performed to group similar data points:

- **Hard Clustering:** In this type of clustering, each data point belongs to a cluster completely or not. For example, Let’s say there are 4 data point and we have to cluster them into 2 clusters. So each data point will either belong to cluster 1 or cluster 2.

Data Points	**Clusters**
A	C1
B	C2
C	C2
D	C1

- **Soft Clustering:** In this type of clustering, instead of assigning each data point into a separate cluster, a probability or likelihood of that point being that cluster is evaluated. For example, Let’s say there are 4 data point and we have to cluster them into 2 clusters. So we will be evaluating a probability of a data point belonging to both clusters. This probability is calculated for all data points.

Data Points	Probability of C1	Probability of C2
A	0.91	0.09

Data Points	Probability of C1	Probability of C2
B	0.3	0.7
C	0.17	0.83
D	1	0

Uses of Clustering

Now before we begin with types of clustering algorithms, we will go through the use cases of Clustering algorithms. Clustering algorithms are majorly used for:

- **Market Segmentation:** Businesses use clustering to group their customers and use targeted advertisements to attract more audience.
- **Market Basket Analysis:** Shop owners analyze their sales and figure out which items are majorly bought together by the customers. For example, In USA, according to a study diapers and beers were usually bought together by fathers.
- **Social Network Analysis:** Social media sites use your data to understand your browsing behavior and provide you with targeted friend recommendations or content recommendations.
- **Medical Imaging:** Doctors use Clustering to find out diseased areas in diagnostic images like X-rays.
- **Anomaly Detection:** To find outliers in a stream of real-time dataset or forecasting fraudulent transactions we can use clustering to identify them.
- **Simplify working with large datasets:** Each cluster is given a cluster ID after clustering is complete. Now, you may reduce a feature set's whole feature set into its cluster ID. Clustering is effective when it can represent a complicated case with a straightforward cluster ID. Using the same principle, clustering data can make complex datasets simpler.

There are many more use cases for clustering but there are some of the major and common use cases of clustering. Moving forward we will be discussing Clustering Algorithms that will help you perform the above tasks.

Types of Clustering Methods

At the surface level, **clustering helps in the analysis of unstructured data. Graphing, the shortest distance, and the density of the data points are a few of the elements that influence cluster formation**. Clustering is the process of determining how related the objects are **based on a metric called the similarity measure**.

Similarity metrics **are easier to locate in smaller sets of features and harder as the number of features increases**.

Depending on the type of clustering algorithm being utilized, several techniques are employed to group the data from the datasets. In this part, the clustering techniques are described.

Various types of clustering algorithms are:

1. Centroid-based Clustering (Partitioning methods)
2. Density-based Clustering (Model-based methods)
3. Connectivity-based Clustering (Hierarchical clustering)

We will be going through each of these types in brief.

1. Centroid-based Clustering (Partitioning methods)
Centroid-based clustering organizes data points around central vectors (centroids) that represent clusters. Each data point belongs to the cluster with the nearest centroid. Generally, the similarity measure chosen for these algorithms are Euclidian distance, Manhattan Distance or Minkowski Distance.
The datasets are separated into a **predetermined number of clusters, and each cluster is referenced by a vector of values. When compared to the vector value, the input data variable shows no difference and joins the cluster.**
The major drawback for centroid-based algorithms is the requirement that we establish the number of clusters, "k," either intuitively or scientifically (using the Elbow Method) before any clustering machine learning system starts allocating the data points. Despite this limitation, it remains the most popular type of clustering due to its simplicity and efficiency. Popular algorithms of Centroid-based clustering are:

- K-means and
- K-medoids clustering

are some examples of this type clustering.

2. Density-based Clustering (Model-based methods)
Density-based clustering identifies clusters as areas of high density separated by regions of low density in the data space. Unlike centroid-based methods, density-based clustering **automatically determines the number of clusters and is less susceptible to initialization positions**. **Key Characteristics:**

- Can find arbitrarily shaped clusters
- Handles noise and outliers well
- Excels with clusters of different sizes and shapes
- Ideal for datasets with irregularly shaped or overlapping clusters
- Effectively manages both dense and sparse data regions
- Focus on local density allows detection of various cluster morphologies

The most popular density-based clustering algorithm is DBSCAN and OPTICS (Ordering Points To Identify Clustering Structure).

3. Connectivity-based Clustering (Hierarchical clustering)
Connectivity-based clustering builds a **hierarchy of clusters using a measure of connectivity based on distance** when organizing a collection of items based on their similarities. This method builds a **dendrogram**, a tree-like structure that visually represents the relationships between objects.
At the base of the tree, each object starts as its own individual cluster. The algorithm then evaluates how similar the objects are to one another and begins merging the closest pairs of clusters into larger groups. This process continues iteratively, with clusters being combined step by step, until all objects are united into a single cluster at the top of the tree.

There are 2 approaches for Hierarchical clustering:

- **Divisive Clustering:** It follows a top-down approach, here we consider all data points to be part one big cluster and then this cluster is divide into smaller groups.
- **Agglomerative Clustering:** It follows a bottom-up approach, here we consider all data points to be part of individual clusters and then these clusters are clubbed together to make one big cluster with all data points.

For implementing and understand difference between both techniques , please refer to : Agglomerative clustering and Divisive clustering
Till now, we have understood ***traditional "hard" clustering methods****, where each data point is assigned to exactly one cluster. These methods, like K-Means and hierarchical clustering, are powerful and widely used, but they have limitations when dealing with ambiguous or overlapping data. After learning all about hard clustering methods we can addresses these limitations with* ***soft clustering that*** *allows data points to belong to* ***multiple clusters simultaneously****, with varying degrees of membership. This approach is particularly useful when the boundaries between clusters are not clear-cut or when data points exhibit characteristics of more than one group.*

Two of the most popular soft clustering techniques are:

4. Distribution-based Clustering
Distribution-based clustering is a technique that assumes **data points are generated from a mixture of probability distributions (e.g., Gaussian, Poisson, etc.)**. The goal is to identify clusters by estimating the parameters of these distributions. In distribution-based clustering:

- Each cluster is represented by a probability distribution.
- Data points are assigned to clusters based on how likely they are to belong to each distribution.
- Unlike distance-based methods (e.g., K-Means), this approach can capture clusters of varying shapes, sizes, and densities.

Many real-world datasets, such as sensor data, financial data, or biological measurements, naturally follow statistical distributions. The most popular distribution-based clustering algorithm is Gaussian Mixture Model.

5. Fuzzy Clustering
Fuzzy clustering allows data points to belong to multiple clusters with varying degrees of membership.

- Each data point is assigned a membership value between 0 and 1 for every cluster.
- These membership values indicate the degree to which a data point belongs to a particular cluster.

Please refer to fuzzy clustering methods for in-depth understanding. Although this method and it's algorithms are used for higher-level problem statements involving complex datasets Clustering in Machine Learning

Frequently Asked Questions (FAQs) on Clustering
What is the difference between clustering and classification?
The main difference between clustering and classification is that, classification is a supervised learning algorithm and clustering is an unsupervised learning algorithm. That is, we apply clustering to those datasets that without a target variable.
What are the advantages of clustering analysis?
Data can be organised into meaningful groups using the strong analytical tool of cluster analysis. You can use it to pinpoint segments, find hidden patterns, and improve decisions.
Which is the fastest clustering method?
K-means clustering is often considered the fastest clustering method due to its simplicity and computational efficiency. It iteratively assigns data points to the nearest cluster centroid, making it suitable for large datasets with low dimensionality and a moderate number of clusters.
What are the limitations of clustering?

Limitations of clustering include sensitivity to initial conditions, dependence on the choice of parameters, difficulty in determining the optimal number of clusters, and challenges with handling high-dimensional or noisy data.

What does the quality of result of clustering depend on?

The quality of clustering results depends on factors such as the choice of algorithm, distance metric, number of clusters, initialization method, data preprocessing techniques, cluster evaluation metrics, and domain knowledge. These elements collectively influence the effectiveness and accuracy of the clustering outcome.

K-Means Clustering Algorithm

K-Means Clustering is an unsupervised learning algorithm that is used to solve the clustering problems in machine learning or data science. In this topic, we will learn what is K-means clustering algorithm, how the algorithm works, along with the Python implementation of k-means clustering.

What is K-Means Algorithm?

K-Means Clustering is an Unsupervised Learning algorithm, which groups the unlabeled dataset into different clusters. Here K defines the number of pre-defined clusters that need to be created in the process, as if K=2, there will be two clusters, and for K=3, there will be three clusters, and so on.

It is an iterative algorithm that divides the unlabeled dataset into k different clusters in such a way that each dataset belongs only one group that has similar properties.

It allows us to cluster the data into different groups and a convenient way to discover the categories of groups in the unlabeled dataset on its own without the need for any training.

It is a centroid-based algorithm, where each cluster is associated with a centroid. The main aim of this algorithm is to minimize the sum of distances between the data point and their corresponding clusters.

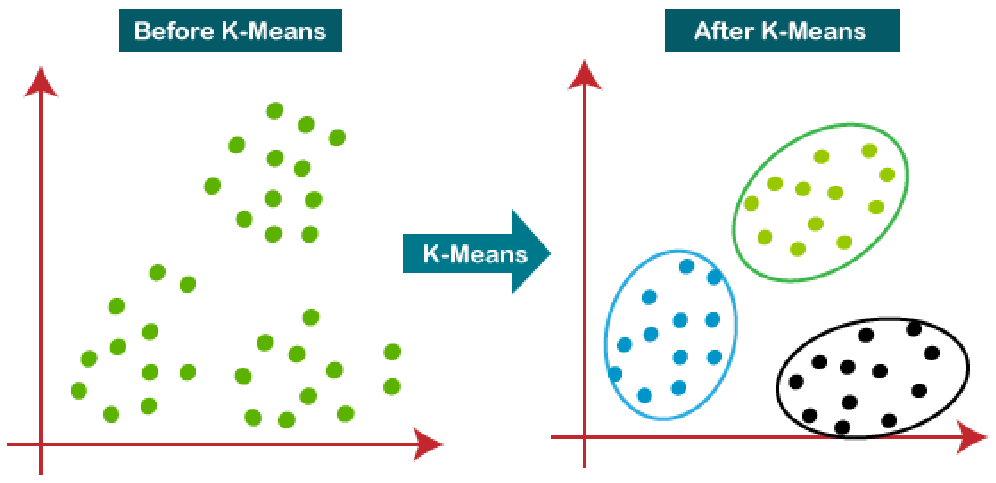
The algorithm takes the unlabeled dataset as input, divides the dataset into k-number of clusters, and repeats the process until it does not find the best clusters. The value of k should be predetermined in this algorithm.

The k-means clustering algorithm mainly performs two tasks:

- Determines the best value for K center points or centroids by an iterative process.
- Assigns each data point to its closest k-center. Those data points which are near to the particular k-center, create a cluster.

Hence each cluster has datapoints with some commonalities, and it is away from other clusters.

The below diagram explains the working of the K-means Clustering Algorithm:



How does the K-Means Algorithm Work?

The working of the K-Means algorithm is explained in the below steps:

Step-1: Select the number K to decide the number of clusters.

Step-2: Select random K points or centroids. (It can be other from the input dataset).

Step-3: Assign each data point to their closest centroid, which will form the predefined K clusters.

Step-4: Calculate the variance and place a new centroid of each cluster.

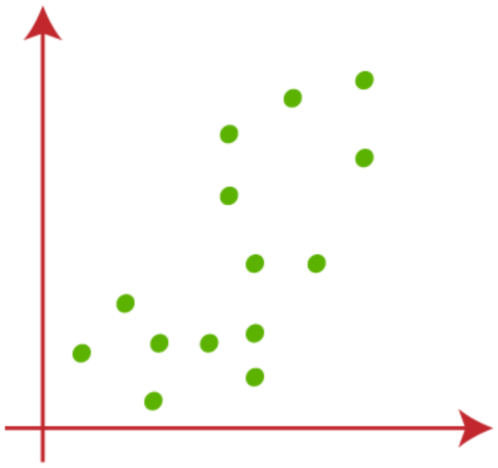
Step-5: Repeat the third steps, which means reassign each datapoint to the new closest centroid of each cluster.

Step-6: If any reassignment occurs, then go to step-4 else go to FINISH.

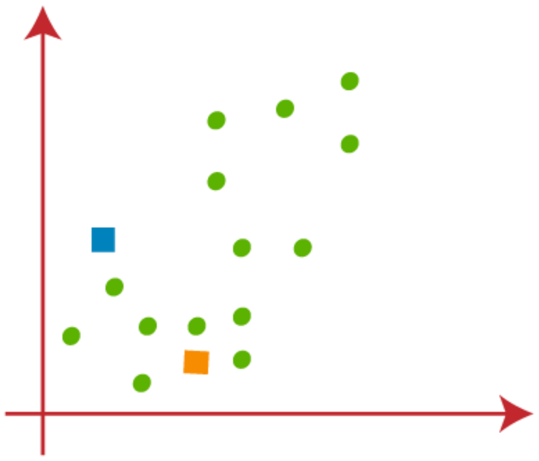
Step-7: The model is ready.

Let's understand the above steps by considering the visual plots:

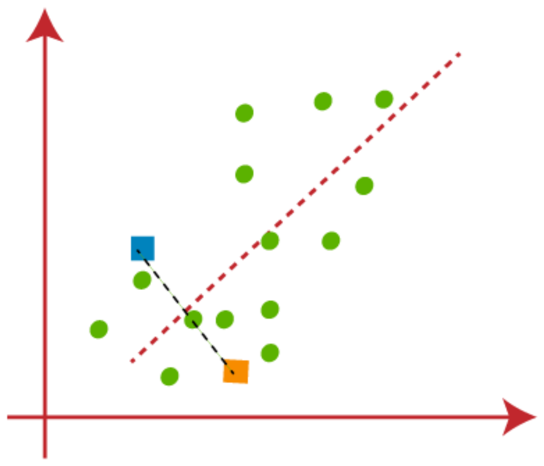
Suppose we have two variables M1 and M2. The x-y axis scatter plot of these two variables is given below:



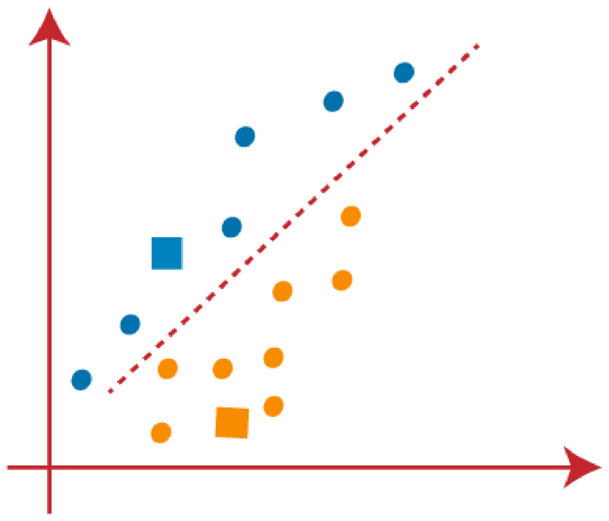
- Let's take number k of clusters, i.e., K=2, to identify the dataset and to put them into different clusters. It means here we will try to group these datasets into two different clusters.
- We need to choose some random k points or centroid to form the cluster. These points can be either the points from the dataset or any other point. So, here we are selecting the below two points as k points, which are not the part of our dataset. Consider the below image:



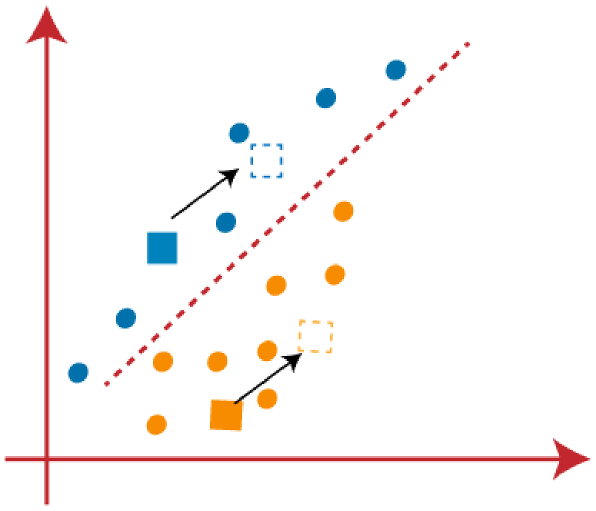
- Now we will assign each data point of the scatter plot to its closest K-point or centroid. We will compute it by applying some mathematics that we have studied to calculate the distance between two points. So, we will draw a median between both the centroids. Consider the below image:



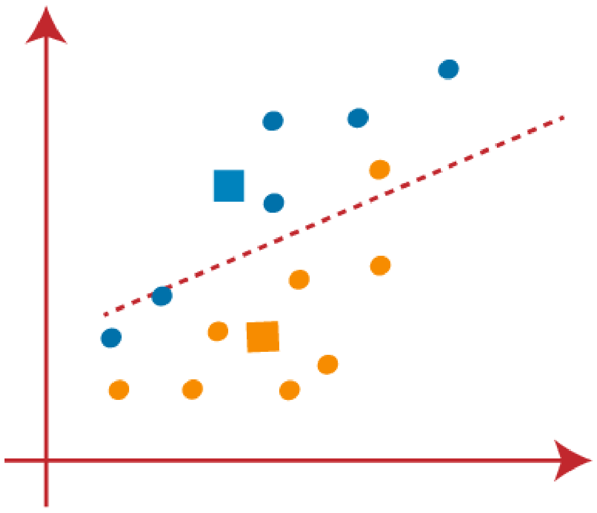
From the above image, it is clear that points left side of the line is near to the K1 or blue centroid, and points to the right of the line are close to the yellow centroid. Let's color them as blue and yellow for clear visualization.



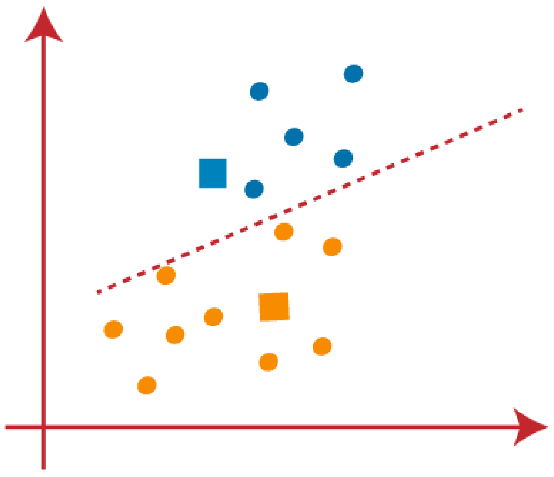
- As we need to find the closest cluster, so we will repeat the process by choosing **a new centroid**. To choose the new centroids, we will compute the center of gravity of these centroids, and will find new centroids as below:



- Next, we will reassign each datapoint to the new centroid. For this, we will repeat the same process of finding a median line. The median will be like below image:

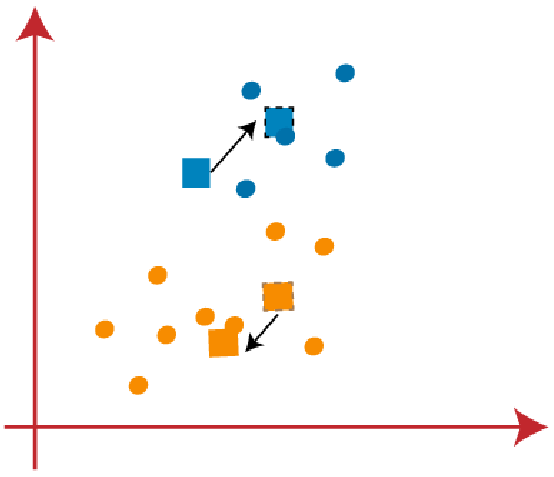


From the above image, we can see, one yellow point is on the left side of the line, and two blue points are right to the line. So, these three points will be assigned to new centroids.

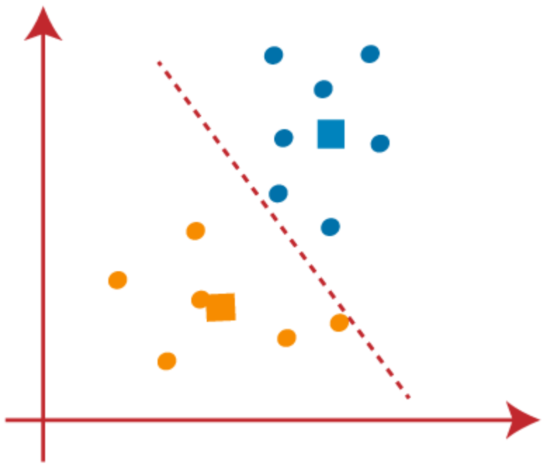


As reassignment has taken place, so we will again go to the step-4, which is finding new centroids or K-points.

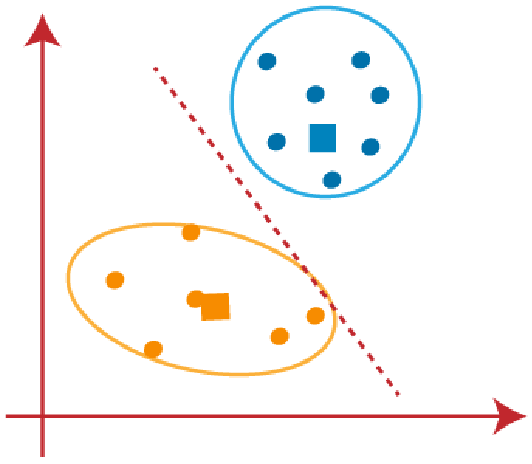
- We will repeat the process by finding the center of gravity of centroids, so the new centroids will be as shown in the below image:



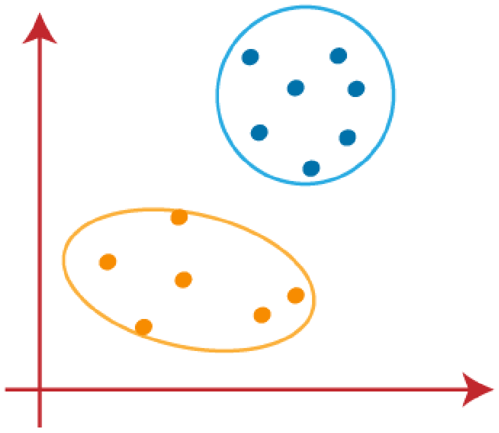
- As we got the new centroids so again will draw the median line and reassign the data points. So, the image will be:



- We can see in the above image; there are no dissimilar data points on either side of the line, which means our model is formed. Consider the below image:



As our model is ready, so we can now remove the assumed centroids, and the two final clusters will be as shown in the below image:



How to choose the value of "K number of clusters" in K-means Clustering?

The performance of the K-means clustering algorithm depends upon highly efficient clusters that it forms. But choosing the optimal number of clusters is a big task. There are some different ways to find the optimal number of clusters, but here we are discussing the most appropriate method to find the number of clusters or value of K. The method is given below:

Elbow Method

The Elbow method is one of the most popular ways to find the optimal number of clusters. This method uses the concept of WCSS value. **WCSS** stands for **Within Cluster Sum of Squares**, which defines the total variations within a cluster. The formula to calculate the value of WCSS (for 3 clusters) is given below:

$$WCSS= \sum_{P_i \text{ in Cluster1}} \text{distance}(P_i\ C_1)^2 + \sum_{P_i \text{ in Cluster2}} \text{distance}(P_i\ C_2)^2 + \sum_{P_i \text{ in CLuster3}} \text{distance}(P_i\ C_3)^2$$

In the above formula of WCSS,

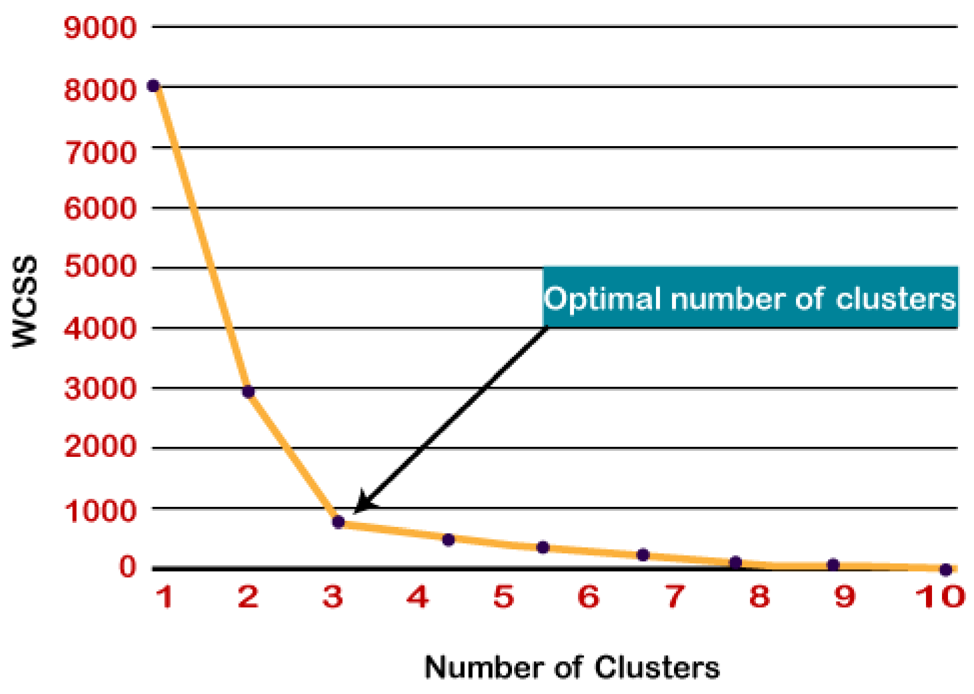
$\sum_{P_i \text{ in Cluster1}} \text{distance}(P_i\ C_1)^2$: It is the sum of the square of the distances between each data point and its centroid within a cluster1 and the same for the other two terms.

To measure the distance between data points and centroid, we can use any method such as Euclidean distance or Manhattan distance.

To find the optimal value of clusters, the elbow method follows the below steps:

- It executes the K-means clustering on a given dataset for different K values (ranges from 1-10).
- For each value of K, calculates the WCSS value.
- Plots a curve between calculated WCSS values and the number of clusters K.
- The sharp point of bend or a point of the plot looks like an arm, then that point is considered as the best value of K.

Since the graph shows the sharp bend, which looks like an elbow, hence it is known as the elbow method. The graph for the elbow method looks like the below image:



Note: We can choose the number of clusters equal to the given data points. If we choose the number of clusters equal to the data points, then the value of WCSS becomes zero, and that will be the endpoint of the plot.

Python Implementation of K-means Clustering Algorithm

In the above section, we have discussed the K-means algorithm, now let's see how it can be implemented using Python.

Before implementation, let's understand what type of problem we will solve here. So, we have a dataset of **Mall_Customers**, which is the data of customers who visit the mall and spend there.

In the given dataset, we have **Customer_Id, Gender, Age, Annual Income ($), and Spending Score** (which is the calculated value of how much a customer has spent in the mall, the more the value, the more he has spent). From this dataset, we need to calculate some patterns, as it is an unsupervised method, so we don't know what to calculate exactly.

The steps to be followed for the implementation are given below:

- **Data Pre-processing**
- **Finding the optimal number of clusters using the elbow method**
- **Training the K-means algorithm on the training dataset**
- **Visualizing the clusters**

Step-1: Data pre-processing Step

The first step will be the data pre-processing, as we did in our earlier topics of Regression and Classification. But for the clustering problem, it will be different from other models. Let's discuss it:

- **Importing Libraries**
  As we did in previous topics, firstly, we will import the libraries for our model, which is part of data pre-processing. The code is given below:

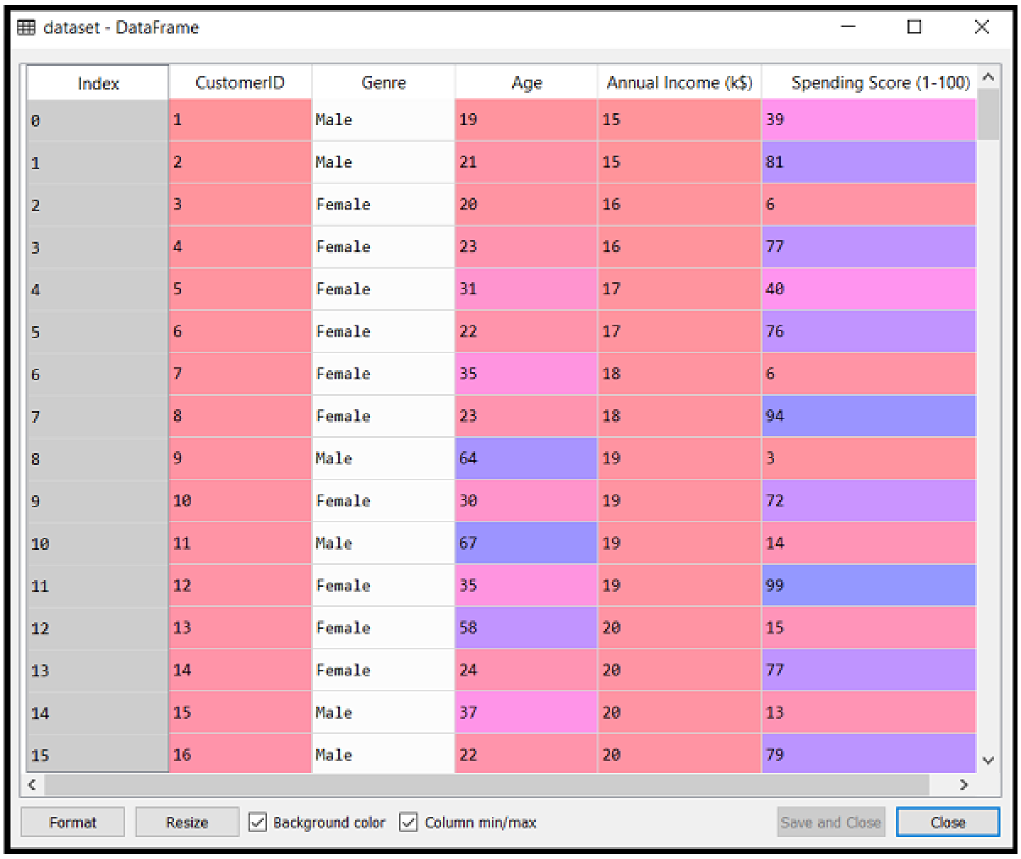
```
# importing libraries
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
```

In the above code, the **numpy** we have imported for the performing mathematics calculation, **matplotlib** is for plotting the graph, and **pandas** are for managing the dataset.

- **Importing the Dataset:**
  Next, we will import the dataset that we need to use. So here, we are using the Mall_Customer_data.csv dataset. It can be imported using the below code:

```
# Importing the dataset
dataset = pd.read_csv('Mall_Customers_data.csv')
```

By executing the above lines of code, we will get our dataset in the Spyder IDE. The dataset looks like the below image:



Index	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
10	11	Male	67	19	14
11	12	Female	35	19	99
12	13	Female	58	20	15
13	14	Female	24	20	77
14	15	Male	37	20	13
15	16	Male	22	20	79

From the above dataset, we need to find some patterns in it.

- **Extracting Independent Variables**

Here we don't need any dependent variable for data pre-processing step as it is a clustering problem, and we have no idea about what to determine. So we will just add a line of code for the matrix of features.

```
x = dataset.iloc[:, [3, 4]].values
```

As we can see, we are extracting only 3rd and 4th feature. It is because we need a 2d plot to visualize the model, and some features are not required, such as customer_id.

Step-2: Finding the optimal number of clusters using the elbow method

In the second step, we will try to find the optimal number of clusters for our clustering problem. So, as discussed above, here we are going to use the elbow method for this purpose.

As we know, the elbow method uses the WCSS concept to draw the plot by plotting WCSS values on the Y-axis and the number of clusters on the X-axis. So we are going to calculate the value for WCSS for different k values ranging from 1 to 10. Below is the code for it:

```
#finding optimal number of clusters using the elbow method
from sklearn.cluster import KMeans
wcss_list= []  #Initializing the list for the values of WCSS

#Using for loop for iterations from 1 to 10.
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', random_state= 42)
    kmeans.fit(x)
    wcss_list.append(kmeans.inertia_)
mtp.plot(range(1, 11), wcss_list)
mtp.title('The Elobw Method Graph')
mtp.xlabel('Number of clusters(k)')
mtp.ylabel('wcss_list')
mtp.show()
```

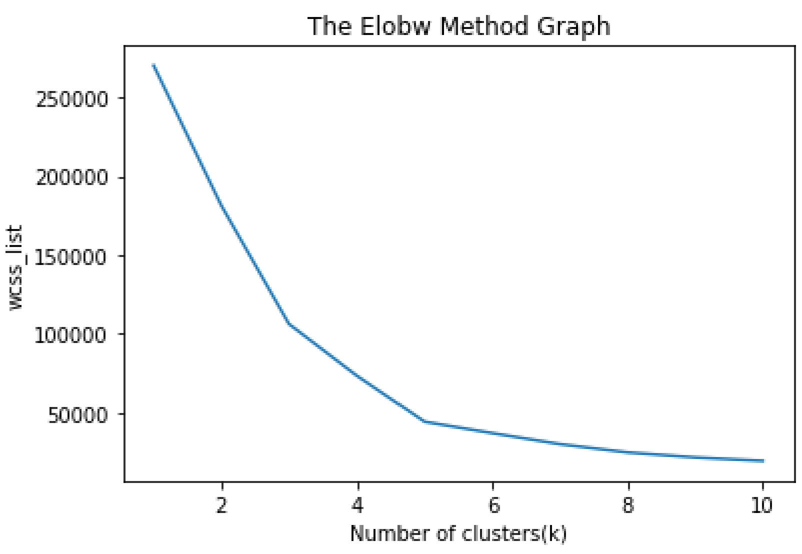
As we can see in the above code, we have used **the KMeans** class of sklearn. cluster library to form the clusters.

Next, we have created the **wcss_list** variable to initialize an empty list, which is used to contain the value of wcss computed for different values of k ranging from 1 to 10.

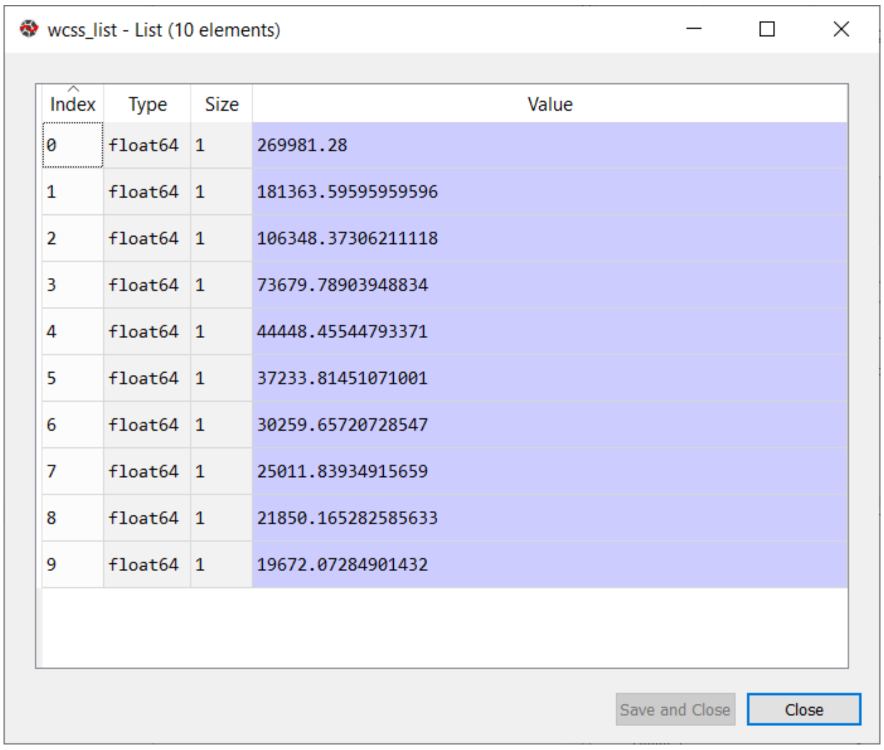
After that, we have initialized the for loop for the iteration on a different value of k ranging from 1 to 10; since for loop in Python, exclude the outbound limit, so it is taken as 11 to include 10th value.

The rest part of the code is similar as we did in earlier topics, as we have fitted the model on a matrix of features and then plotted the graph between the number of clusters and WCSS.

Output: After executing the above code, we will get the below output:



From the above plot, we can see the elbow point is at **5. So the number of clusters here will be 5.**



Index	Type	Size	Value
0	float64	1	269981.28
1	float64	1	181363.59595959596
2	float64	1	106348.37306211118
3	float64	1	73679.78903948834
4	float64	1	44448.45544793371
5	float64	1	37233.81451071001
6	float64	1	30259.65720728547
7	float64	1	25011.83934915659
8	float64	1	21850.165282585633
9	float64	1	19672.07284901432

Step- 3: Training the K-means algorithm on the training dataset

As we have got the number of clusters, so we can now train the model on the dataset.

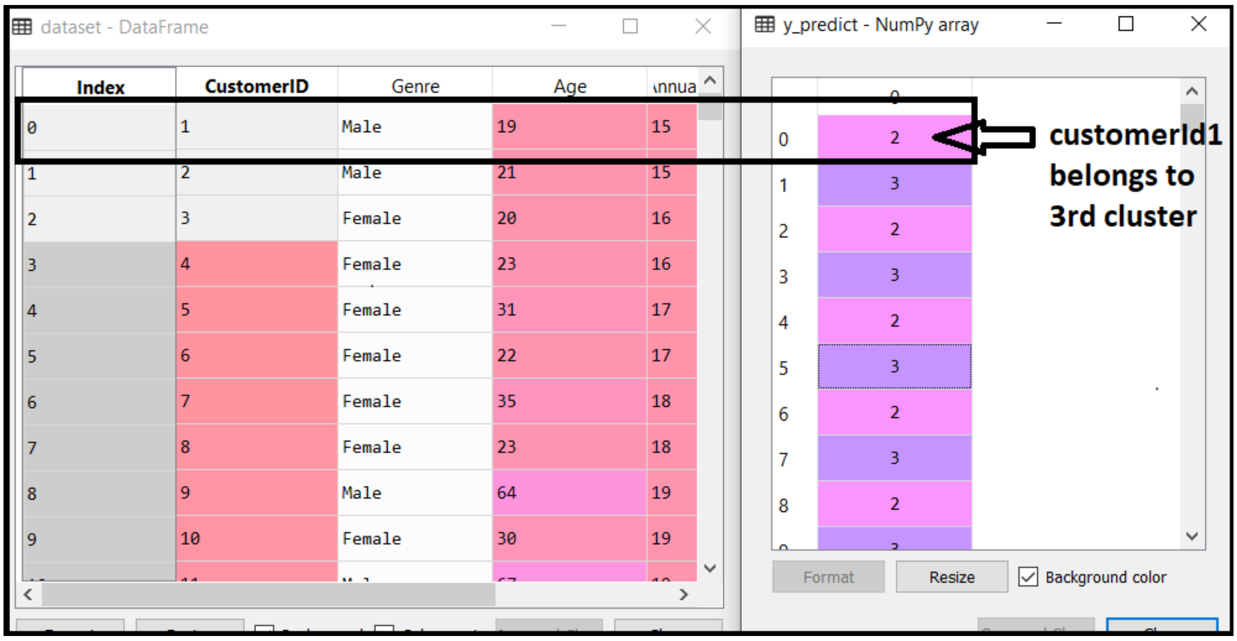
To train the model, we will use the same two lines of code as we have used in the above section, but here instead of using i, we will use 5, as we know there are 5 clusters that need to be formed. The code is given below:

```
#training the K-means model on a dataset
kmeans = KMeans(n_clusters=5, init='k-means++', random_state= 42)
```

3. y_predict= kmeans.fit_predict(x)
   The first line is the same as above for creating the object of KMeans class.

   In the second line of code, we have created the dependent variable **y_predict** to train the model.

   By executing the above lines of code, we will get the y_predict variable. We can check it under **the variable explorer** option in the Spyder IDE. We can now compare the values of y_predict with our original dataset. Consider the below image:



Index	CustomerID	Genre	Age	nnua
0	1	Male	19	15
1	2	Male	21	15
2	3	Female	20	16
3	4	Female	23	16
4	5	Female	31	17
5	6	Female	22	17
6	7	Female	35	18
7	8	Female	23	18
8	9	Male	64	19
9	10	Female	30	19

	0
0	2
1	3
2	2
3	3
4	2
5	3
6	2
7	3
8	2

   From the above image, we can now relate that the CustomerID 1 belongs to a cluster

   3(as index starts from 0, hence 2 will be considered as 3), and 2 belongs to cluster 4, and so on.

   Step-4: Visualizing the Clusters

   The last step is to visualize the clusters. As we have 5 clusters for our model, so we will visualize each cluster one by one.

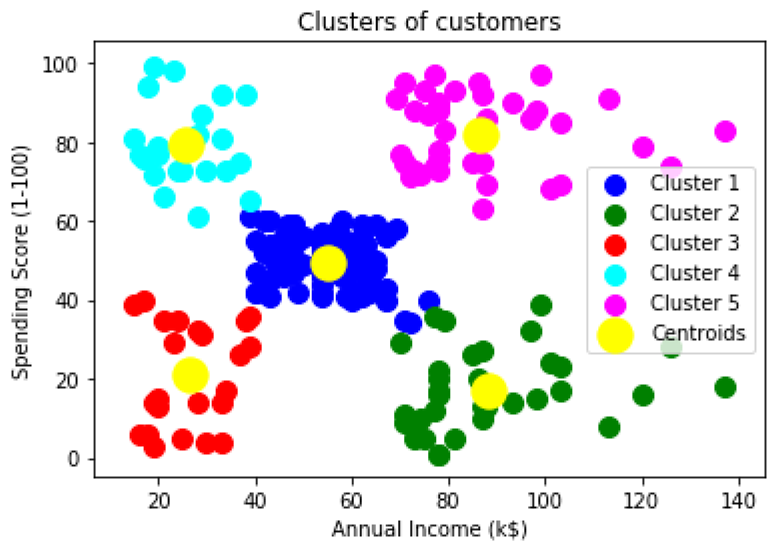
   To visualize the clusters will use scatter plot using mtp.scatter() function of matplotlib.

1. #visulaizing the clusters
2. mtp.scatter(x[y_predict == 0, 0], x[y_predict == 0, 1], s = 100, c = 'blue', label = 'Cluster 1') # **for** first cluster
3. mtp.scatter(x[y_predict == 1, 0], x[y_predict == 1, 1], s = 100, c = 'green', label = 'Cluster 2') #**for** second cluster
4. mtp.scatter(x[y_predict== 2, 0], x[y_predict == 2, 1], s = 100, c = 'red', label = 'Cluster 3') #**f** **or** third cluster
5. mtp.scatter(x[y_predict == 3, 0], x[y_predict == 3, 1], s = 100, c = 'cyan', label = 'Cluster 4') #**for** fourth cluster

```
mtp.scatter(x[y_predict == 4, 0], x[y_predict == 4, 1], s = 100, c = 'magenta', label = 'Cluster 5') #for fifth cluster
mtp.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 300, c = 'yellow', label = 'Centroid')
mtp.title('Clusters of customers')
mtp.xlabel('Annual Income (k$)')
mtp.ylabel('Spending Score (1-100)')
mtp.legend()
mtp.show()
```

In above lines of code, we have written code for each clusters, ranging from 1 to 5. The first coordinate of the mtp.scatter, i.e., x[y_predict == 0, 0] containing the x value for the showing the matrix of features values, and the y_predict is ranging from 0 to 1.

Output:



The output image is clearly showing the five different clusters with different colors. The clusters are formed between two parameters of the dataset; Annual income of customer and Spending. We can change the colors and labels as per the requirement or choice. We can also observe some points from the above patterns, which are given below:

- **Cluster1** shows the customers with average salary and average spending so we can categorize these customers as
- Cluster2 shows the customer has a high income but low spending, so we can categorize them as **careful**.
- Cluster3 shows the low income and also low spending so they can be categorized as sensible.
- Cluster4 shows the customers with low income with very high spending so they can be categorized as **careless**.
- Cluster5 shows the customers with high income and high spending so they can be categorized as target, and these customers can be the most profitable customers for the mall owner.

Hierarchical Clustering in Machine Learning

In the real world **data often lacks a target variable making supervised learning impractical.** Have you ever wondered how social networks like Facebook recommend friends or how scientists group similar species together? These are some examples of hierarchical clustering that we will learn about in this article.

Why hierarchical clustering?

Hierarchical clustering is a technique used to group similar data points together based on their similarity creating a **hierarchy or tree-like structure**. The key idea is to begin with each data point as its own separate cluster and then progressively merge or split them based on their similarity.

Lets understand this with the help of an example

Imagine you have four fruits with different weights: an **apple (100g)**, **a banana (120g), a cherry (50g), and a grape (30g)**. Hierarchical clustering starts by treating each ***fruit as its own group***.

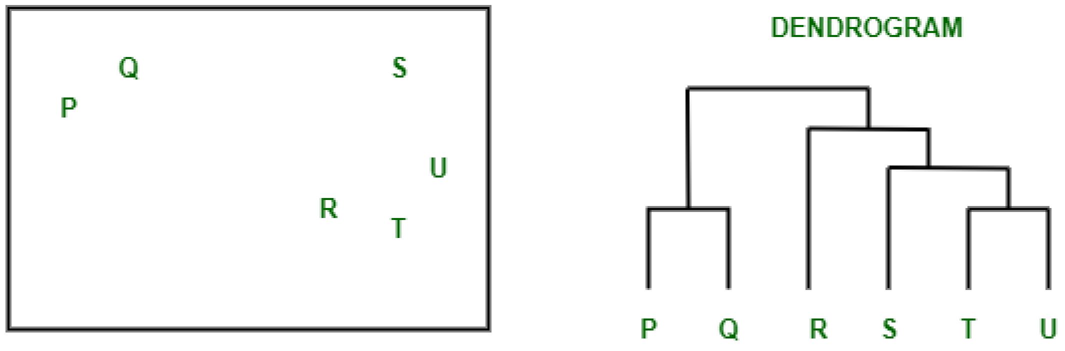
- It then merges the closest groups based on their weights.
- First, the cherry and grape are grouped together because they are the lightest.
- Next, the apple and banana are grouped together.

Finally, all the fruits are merged into one large group, showing how hierarchical clustering progressively combines the most similar data points.

Getting Started with Dendogram

A **dendrogram** is like a family tree for clusters. It shows how individual data points or groups of data merge together. The bottom shows each data point as its own group, and as you move up, similar groups are combined. The lower the merge point, the more similar the groups are. It helps you see how things are grouped step by step.

The working of the dendrogram can be explained using the below diagram:



Dendogram

In this image, on the left side, there are five points labeled P, Q, R, S, and T. These represent individual data points that are being clustered. On the right side, there's a **dendrogram**, which shows how these points are grouped together step by step.

- At the bottom of the dendrogram, the points P, Q, R, S, and T are all separate.
- As you move up, the closest points are merged into a single group.
- The lines connecting the points show how they are progressively merged based on similarity.
- The height at which they are connected shows how similar the points are to each other; the shorter the line, the more similar they are

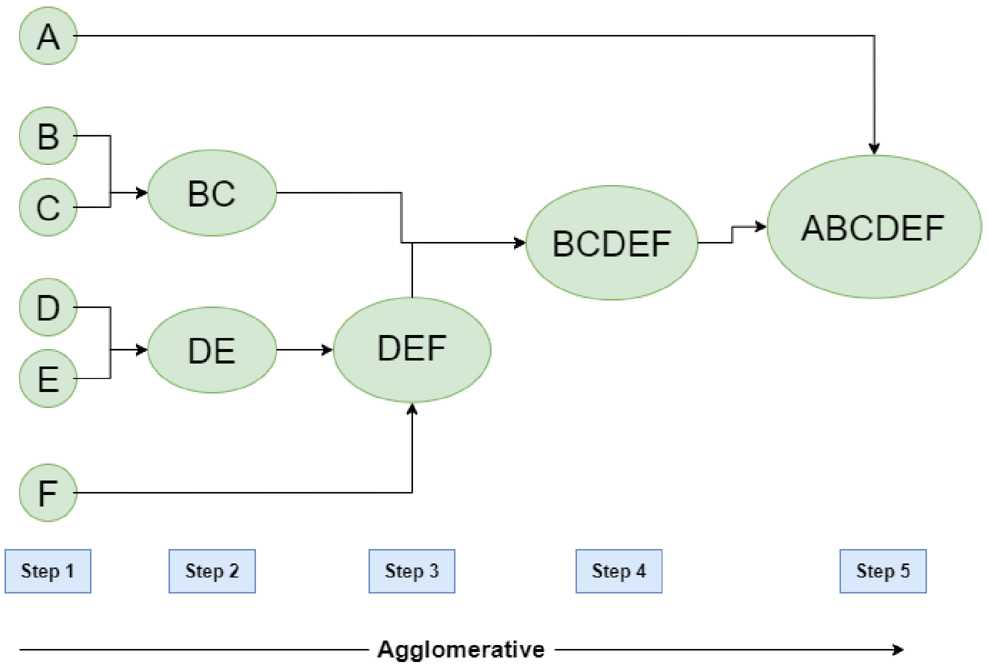
Types of Hierarchical Clustering

Now that we understand the basics of hierarchical clustering, let's explore the two main types of hierarchical clustering.

1. Agglomerative Clustering
2. Divisive clustering

Hierarchical Agglomerative Clustering

It is also known as the **bottom-up approach** or **hierarchical agglomerative clustering (HAC)**. Unlike flat clustering hierarchical clustering provides a structured way to group data. This clustering algorithm does not require us to prespecify the number of clusters. Bottom-up algorithms treat each data as a singleton cluster at the outset and then successively agglomerate pairs of clusters until all clusters have been merged into a single cluster that contains all data.



Hierarchical Agglomerative Clustering

Workflow for Hierarchical Agglomerative clustering

1. **Start with individual points**: Each data point is its own cluster. For example if you have 5 data points you start with 5 clusters each containing just one data point.
2. **Calculate distances between clusters**: Calculate the distance between every pair of clusters. Initially since each cluster has one point this is the distance between the two data points.
3. **Merge the closest clusters**: Identify the two clusters with the smallest distance and merge them into a single cluster.
4. **Update distance matrix**: After merging you now have one less cluster. Recalculate the distances between the new cluster and the remaining clusters.
5. **Repeat steps 3 and 4**: Keep merging the closest clusters and updating the distance matrix until you have only one cluster left.
6. **Create a dendrogram**: As the process continues you can visualize the merging of clusters using a tree-like diagram called a **dendrogram**. It shows the hierarchy of how clusters are merged.

Python implementation of the above algorithm using the scikit-learn library:

```
from sklearn.cluster import AgglomerativeClustering
```

```
import numpy as np

X = np.array([[1, 2], [1, 4], [1, 0],
              [4, 2], [4, 4], [4, 0]])


clustering = AgglomerativeClustering(n_clusters=2).fit(X)

print(clustering.labels_)
```

Output :

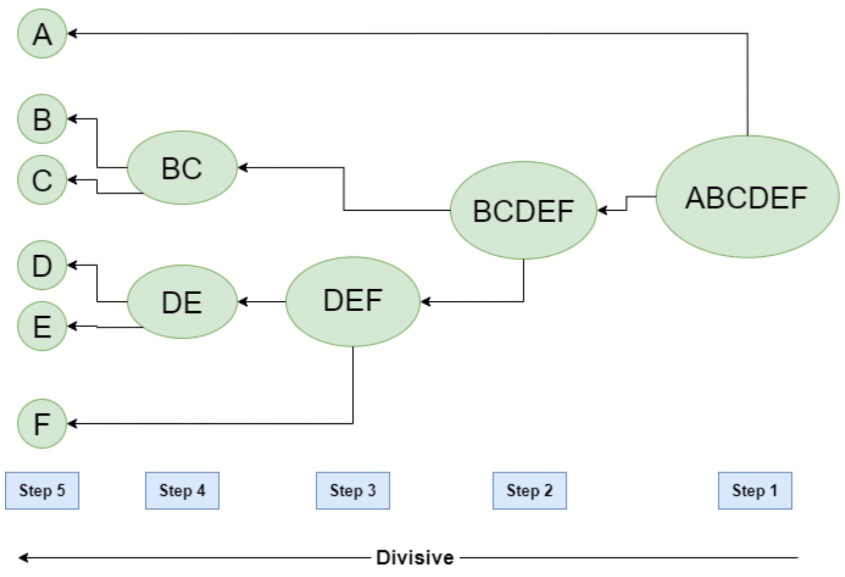
[1, 1, 1, 0, 0, 0]

Hierarchical Divisive clustering

It is also known as a **top-down approach**. This algorithm also does not require to prespecify the number of clusters. Top-down clustering requires a method for splitting a cluster that contains the whole data and proceeds by splitting clusters recursively until individual data have been split into singleton clusters.

Workflow for Hierarchical Divisive clustering :

1. **Start with all data points in one cluster**: Treat the entire dataset as a single large cluster.
2. **Split the cluster**: Divide the cluster into two smaller clusters. The division is typically done by finding the two most dissimilar points in the cluster and using them to separate the data into two parts.
3. **Repeat the process**: For each of the new clusters, repeat the splitting process:
   1. Choose the cluster with the most dissimilar points.
   2. Split it again into two smaller clusters.
4. **Stop when each data point is in its own cluster**: Continue this process until every data point is its own cluster, or the stopping condition (such as a predefined number of clusters) is met.

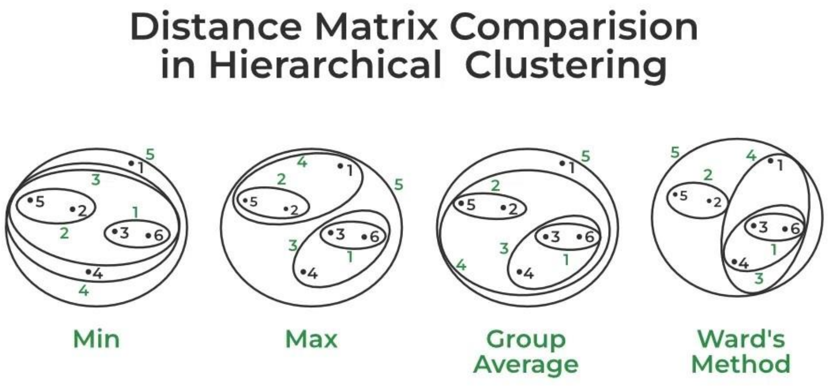


Hierarchical Divisive clustering

Computing Distance Matrix

While merging two clusters we **check the distance between two every pair of clusters and merge the pair with the least distance/most similarity**. But the question is how is that distance determined. There are different ways of defining Inter Cluster distance/similarity. Some of them are:

1. **Min Distance:** Find the minimum distance between any two points of the cluster.
2. **Max Distance:** Find the maximum distance between any two points of the cluster.
3. **Group Average:** Find the average distance between every two points of the clusters.
4. **Ward's Method:** The similarity of two clusters is based on the increase in squared error when two clusters are merged.



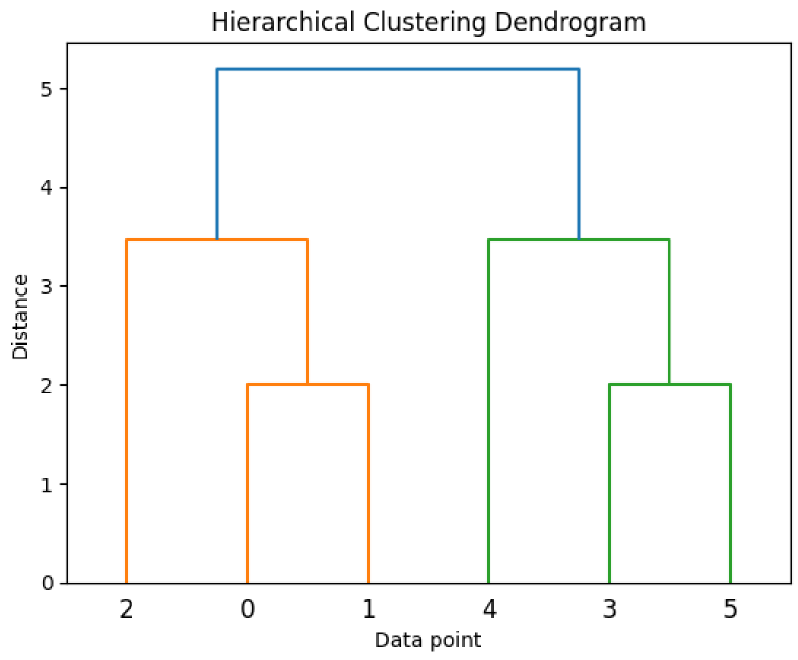
5.

Distance Matrix Comparision in Hierarchical Clustering

Implementations code for Distance Matrix Comparision

```
import numpy as np
from scipy.cluster.hierarchy import dendrogram, linkage
import matplotlib.pyplot as plt


X = np.array([[1, 2], [1, 4], [1, 0],
              [4, 2], [4, 4], [4, 0]])

Z = linkage(X, 'ward') # Ward Distance

dendrogram(Z) #plotting the dendogram

plt.title('Hierarchical Clustering Dendrogram')
plt.xlabel('Data point')
plt.ylabel('Distance')
plt.show()
```

Output:



Hierarchical Clustering Dendrogram

Hierarchical clustering is a powerful unsupervised learning technique that organizes data into a tree-like structure allowing us to visualize relationships between data points using a dendrogram. Unlike flat clustering methods it does not require a predefined number of clusters and provides a structured way to explore data similarity.

Frequently Asked Questions (FAQs) on Hierarchical Clustering

What is Hierarchical Clustering?
Hierarchical clustering is a technique used to group similar data points based on their similarity, creating a hierarchy or tree-like structure. It involves progressively merging or splitting data points based on their similarities.

What are the two types of Hierarchical Clustering?
There are two main types of hierarchical clustering:

- ***Agglomerative Clustering**: A bottom-up approach where each data point starts as its own cluster and merges with the closest cluster progressively.*
- ***Divisive Clustering**: A top-down approach where all data points start as a single cluster, which is split recursively into smaller clusters*

How does Hierarchical Agglomerative Clustering work?
In agglomerative clustering, each data point starts as its own cluster. The algorithm calculates distances between clusters, merges the closest clusters, and repeats this process until all data points are merged into one cluster. The result is visualized in a dendrogram

What is a dendrogram in hierarchical clustering?

- *A dendrogram is a tree-like diagram that visually represents the hierarchical structure of clusters. It shows how data points or clusters are merged or split at each step of the clustering process, with the height of the branches indicating the distance at which the merging or splitting occurs*

How do I choose the right method for hierarchical clustering?

The choice of method depends on the structure of your data. For example:

- ***Min Distance** works well for elongated clusters.*
- ***Max Distance** creates compact, well-separated clusters.*
- ***Group Average** offers a balanced approach.*
- ***Ward's Method** tries to minimize the increase in squared error and maintains compact clusters.*

Probabilistic Clustering in Unsupervised Learning

Probabilistic clustering is a method used in unsupervised learning where the goal is to group similar data points into clusters based on probability distributions. Unlike hard clustering, where each data point belongs to exactly one cluster, probabilistic clustering assigns probabilities to each data point for being in different clusters. This approach is particularly useful when the boundaries between clusters are not clearly defined or when data points exhibit uncertainty.

Here's an overview of probabilistic clustering and the key concepts associated with it:

Key Concepts

1. **Latent Variables**: Probabilistic clustering typically assumes that there is some latent or hidden structure underlying the observed data. This structure may be described by a set of unobserved variables (e.g., cluster assignments).
2. **Gaussian Mixture Model (GMM)**: One of the most common probabilistic clustering models is the **Gaussian Mixture Model (GMM)**. GMM assumes that the data is generated from a mixture of several Gaussian distributions, each representing a cluster. Each Gaussian distribution is characterized by its mean, covariance, and a mixture weight (the proportion of data points belonging to that cluster).
    - **Probability Density Function**: The model assigns a probability to each data point for belonging to a certain Gaussian component, effectively giving each data point a "soft" membership in each cluster.
    - **Expectation-Maximization (EM)**: The Expectation-Maximization algorithm is typically used to fit a GMM to the data, alternating between estimating the cluster assignments (E-step) and updating the parameters of the Gaussian distributions (M-step).
3. **Soft Clustering**: In probabilistic clustering, the assignment of data points to clusters is soft, meaning that a data point can belong to multiple clusters with different probabilities. This contrasts with hard clustering methods (e.g., K-means), where each data point is assigned to a single cluster.
4. **Bayesian Clustering**: Bayesian methods in clustering involve treating the model parameters (e.g., cluster centers, covariances) as random variables and using prior distributions. This approach allows for a more flexible modeling of uncertainty in both data and model parameters.
5. **Dirichlet Process Mixture Models (DPMMs)**: In cases where the number of clusters is not known a priori, **Dirichlet Process Mixture Models (DPMMs)** can be used. DPMMs provide a non-parametric approach to clustering by allowing the number of clusters to grow as more data points are observed. This is useful when the data does not fit a fixed number of clusters and helps in discovering a more natural grouping structure.

Key Techniques in Probabilistic Clustering

1. **Expectation-Maximization (EM) Algorithm**:
   - **E-Step**: Given the current model parameters, compute the probability that each data point belongs to each cluster.
   - **M-Step**: Update the model parameters (e.g., means, covariances, and mixing coefficients) to maximize the likelihood of the observed data, given the current soft assignments.

   This iterative process continues until convergence, resulting in a probabilistic estimate of the parameters and cluster assignments.

2. **Bayesian Inference**: Bayesian methods allow us to incorporate prior knowledge about the clusters and update our beliefs as new data becomes available. **Markov Chain Monte Carlo (MCMC)** methods or variational inference are often used to approximate the posterior distribution of the model parameters.
3. **Dirichlet Process**: A non-parametric method used in probabilistic clustering, the **Dirichlet Process** allows for an unknown number of clusters. It is based on the idea that new clusters may be created as more data points arrive, governed by a Dirichlet distribution.

Advantages of Probabilistic Clustering

1. **Soft Assignments**:
   - Data points can be assigned to multiple clusters with varying degrees of membership, allowing for more flexibility and nuanced grouping.
2. **Modeling Uncertainty**:
   - Probabilistic models naturally handle uncertainty by assigning probabilities to cluster memberships rather than assigning deterministic labels.
3. **Flexibility**:
   - These models can incorporate different types of distributions (e.g., Gaussian, Poisson, multinomial) to better fit the data's structure.
4. **Better for Complex Data**:
   - They can handle situations where clusters are not well-separated or have complex shapes, which may be challenging for hard clustering methods like K-means.
5. **Handling Overlapping Clusters**:
   - Probabilistic clustering can capture the overlap between clusters by assigning mixed probabilities to data points.

Disadvantages of Probabilistic Clustering

1. **Computational Complexity**:
   - The Expectation-Maximization algorithm and Bayesian methods can be computationally intensive, especially for large datasets or when using complex models (like Dirichlet Process Mixture Models).
2. **Model Assumptions**:
   - Models like GMM assume that the clusters follow specific distributions (e.g., Gaussian). If the real data does not fit these assumptions well, the model may perform poorly.

3. **Choosing the Number of Clusters**:
    - Although non-parametric models like DPMMs help with this, in some probabilistic models (like GMM), you still need to specify the number of clusters, which can be difficult.
4. **Overfitting**:
    - With more complex models or a large number of parameters, there's a risk of overfitting, where the model fits the noise in the data rather than the true underlying structure.

Applications

1. **Image Segmentation**: Probabilistic clustering is used in computer vision tasks such as image segmentation, where pixels are assigned to clusters based on their color, texture, or other features.
2. **Anomaly Detection**: Probabilistic clustering can help in anomaly detection, where rare or unexpected data points are likely to belong to low-probability clusters.
3. **Customer Segmentation**: Businesses often use probabilistic clustering to segment customers into different groups based on purchasing behavior or other attributes, allowing for more targeted marketing strategies.
4. **Genomic Data Analysis**: In bioinformatics, probabilistic clustering models are used to identify subgroups of genes or cells that share similar expression patterns.

Conclusion

Probabilistic clustering provides a powerful and flexible approach for unsupervised learning, allowing for the discovery of hidden patterns in data. By modeling uncertainty and assigning probabilities to cluster memberships, it offers a more nuanced view of the data compared to traditional hard clustering methods. However, it requires careful consideration of the model assumptions, computational costs, and potential risks of overfitting.

Association Rule Learning

Association rule learning is a type of unsupervised learning technique that checks for the dependency of one data item on another data item and maps accordingly so that it can be more profitable. It tries to find some interesting relations or associations among the variables of dataset. It is based on different rules to discover the interesting relations between variables in the database.

The association rule learning is one of the very important concepts of machine learning, and it is employed in **Market Basket analysis, Web usage mining, continuous production, etc.** Here market basket analysis is a technique used by the various big retailer to discover the associations between items. We can understand it by taking an example of a supermarket, as in a supermarket, all products that are purchased together are put together.

For example, if a customer buys bread, he most likely can also buy butter, eggs, or milk, so these products are stored within a shelf or mostly nearby. Consider the below diagram:

milk
bread
butter
Customer 1
milk
cereal
bread
butter
Customer 2
milk
bread
butter
Customer 3
Sugar
eggs
Customer n

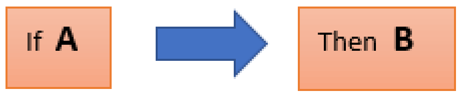
Association rule learning can be divided into three types of algorithms:

1. **Apriori**
2. **Eclat**
3. **F-P Growth Algorithm**

We will understand these algorithms in later chapters.

How does Association Rule Learning work?

Association rule learning works on the concept of If and Else Statement, such as if A then B.



Here the If element is called **antecedent**, and then statement is called as **Consequent**. These types of relationships where we can find out some association or relation between two items is known *as single cardinality*. It is all about creating rules, and if the number of items increases, then cardinality also increases accordingly. So, to measure the associations between thousands of data items, there are several metrics. These metrics are given below:

- **Support**
- **Confidence**
- **Lift**

Let's understand each of them:

Support

Support is the frequency of A or how frequently an item appears in the dataset. It is defined as the fraction of the transaction T that contains the itemset X. If there are X datasets, then for transactions T, it can be written as:

$$\text{Supp}(X)=\frac{Freq(X)}{T}$$

Confidence

Confidence indicates how often the rule has been found to be true. Or how often the items X and Y occur together in the dataset when the occurrence of X is already given. It is the ratio of the transaction that contains X and Y to the number of records that contain X.

$$\text{Confidence}=\frac{Freq(X,Y)}{Freq(X)}$$

Lift

It is the strength of any rule, which can be defined as below formula:

$$\text{Lift}=\frac{Supp(X,Y)}{Supp(X)\times Supp(Y)}$$

It is the ratio of the observed support measure and expected support if X and Y are independent of each other. It has three possible values:

- If **Lift= 1**: The probability of occurrence of antecedent and consequent is independent of each other.
- **Lift>1**: It determines the degree to which the two itemsets are dependent to each other.
- **Lift<1**: It tells us that one item is a substitute for other items, which means one item has a negative effect on another.

Types of Association Rule Learning

Association rule learning can be divided into three algorithms:

Apriori Algorithm

This algorithm uses frequent datasets to generate association rules. It is designed to work on the databases that contain transactions. This algorithm uses a breadth-first search and Hash Tree to calculate the itemset efficiently.

It is mainly used for market basket analysis and helps to understand the products that can be bought together. It can also be used in the healthcare field to find drug reactions for patients.

Eclat Algorithm

Eclat algorithm stands for **Equivalence Class Transformation**. This algorithm uses a depth-first search technique to find frequent itemsets in a transaction database. It performs faster execution than Apriori Algorithm.

F-P Growth Algorithm

The F-P growth algorithm stands for **Frequent Pattern**, and it is the improved version of the Apriori Algorithm. It represents the database in the form of a tree structure that is known as a frequent pattern or tree. The purpose of this frequent tree is to extract the most frequent patterns.

Applications of Association Rule Learning

It has various applications in machine learning and data mining. Below are some popular applications of association rule learning:

- **Market Basket Analysis:** It is one of the popular examples and applications of association rule mining. This technique is commonly used by big retailers to determine the association between items.
- **Medical Diagnosis:** With the help of association rules, patients can be cured easily, as it helps in identifying the probability of illness for a particular disease.
- **Protein Sequence:** The association rules help in determining the synthesis of artificial Proteins.
- It is also used for the **Catalog Design** and **Loss-leader Analysis** and many more other applications.

Apriori Algorithm

Apriori Algorithm is a foundational method in data mining used for discovering frequent itemsets and generating association rules. Its significance lies in its ability to identify relationships between items in large datasets which is particularly valuable in market basket analysis.
For example, if a grocery store finds that customers who buy **bread** often also buy **butter**, it can use this information to optimize product placement or marketing strategies.

How the Apriori Algorithm Works?
The Apriori Algorithm operates through a systematic process that involves several key steps:

1. **Identifying Frequent Itemsets**: The algorithm begins by scanning the dataset to identify individual items (1-item) and their frequencies. It then establishes a **minimum support threshold**, which determines whether an itemset is considered frequent.
2. **Creating Possible item group**: Once frequent 1-itemgroup(single items) are identified, the algorithm generates candidate 2-itemgroup by combining frequent items. This process continues iteratively, forming larger itemsets (k-itemgroup) until no more frequent itemgroup can be found.
3. **Removing Infrequent Item groups**: The algorithm employs a pruning technique based on the **Apriori Property**, which states that if an itemset is infrequent, all its supersets must also be infrequent. This significantly reduces the number of combinations that need to be evaluated.

4. **Generating Association Rules**: After identifying frequent itemsets, the algorithm generates **association rules** that illustrate how items relate to one another, using metrics like **support, confidence,** and **lift** to evaluate the strength of these relationships.

Key Metrics of Apriori Algorithm

- **Support**: This metric measures how frequently an item appears in the dataset relative to the total number of transactions. A higher support indicates a more significant presence of the itemset in the dataset. Support tells us how often a particular item or combination of items appears in all the transactions ("Bread is bought in 20% of all transactions.")
- **Confidence**: Confidence assesses the likelihood that an item Y is purchased when item X is purchased. It provides insight into the strength of the association between two items.
- Confidence tells us how often items go together. ("If bread is bought, butter is bought 75% of the time.")
- **Lift**: Lift evaluates how much more likely two items are to be purchased together compared to being purchased independently. A lift greater than 1 suggests a strong positive association. Lift shows how strong the connection is between items. ("Bread and butter are much more likely to be bought together than by chance.")

Lets understand the concept of apriori Algorithm with the help of an example. Consider the following dataset and we will find frequent itemsets and generate association rules for them:

Transaction ID	Items Bought
T1	Bread, Butter, Milk
T2	Bread, Butter
T3	Bread, Milk
T4	Butter, Milk
T5	Bread, Milk

Transactions of a Grocery Shop

Step 1 : Setting the parameters

- **Minimum Support Threshold:** 50% (item must appear in at least 3/5 transactions). This threeshold is formulated from this formula:

Support(A)=Number of transactions containing itemset ATotal number of transactionsSupport(*A*)=Total number of transactionsNumber of transactions containing itemset *A*

- **Minimum Confidence Threshold:** 70% (You can change the value of parameters as per the usecase and problem statement). This threeshold is formulated from this formula:

Confidence(X→Y)=Support(X∪Y)Support(X)Confidence(*X*→*Y*)=Support(*X*)Support(*X*∪*Y*)

Step 2: Find Frequent 1-Itemsets

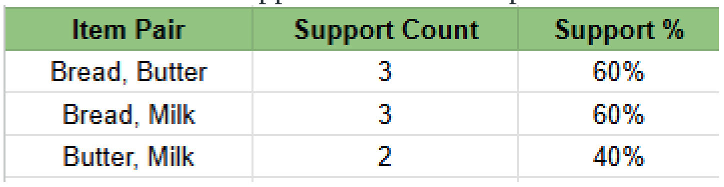
Lets count how many transactions include each item in the dataset (calculating the frequency of each item).

Item	Support Count	Support %
Bread	4	80%
Butter	3	60%
Milk	4	80%

Frequent 1-Itemsets

All items have **support% ≥ 50%**, so they qualify as **frequent 1-itemsets**. if any item has support% < 50%, It will be ommited out from the frequent 1- itemsets.

Step 3: Generate Candidate 2-Itemsets
Combine the frequent 1-itemsets into pairs and calculate their support.
For this usecase, we will get 3 item pairs (bread,butter) , (bread,ilk) and (butter,milk) and will calculate the support similiar to step 2



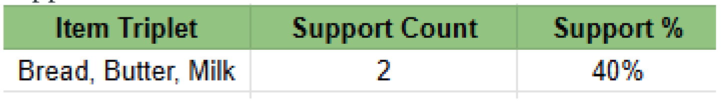
Item Pair	Support Count	Support %
Bread, Butter	3	60%
Bread, Milk	3	60%
Butter, Milk	2	40%

Candidate 2-Itemsets

Frequent 2-itemsets:

- {Bread, Butter}, {Bread, Milk} both meet the 50% threshold but {butter,milk} doesnt meet the threeshold, so will be ommited out.
-

Step 4: Generate Candidate 3-Itemsets
Combine the frequent 2-itemsets into groups of 3 and calculate their support.
for the triplet, we have only got one case i.e {bread,butter,milk} and we will calculate the support.



Item Triplet	Support Count	Support %
Bread, Butter, Milk	2	40%

Candidate 3-Itemsets

Since this **does not meet the 50% threshold**, there are no **frequent 3-itemsets**.

Step 5: Generate Association Rules
Now we generate rules from the frequent itemsets and calculate **confidence**.
Rule 1: If Bread → Butter (if customer buys bread, the customer will buy butter also)

- Support of {Bread, Butter} = 3.
- Support of {Bread} = 4.
- **Confidence = 3/4 = 75%** (Passes threshold).

Rule 2: If Butter → Bread (if customer buys butter, the customer will buy bread also)

- Support of {Bread, Butter} = 3.
- Support of {Butter} = 3.
- **Confidence = 3/3 = 100%** (Passes threshold).

Rule 3: If Bread → Milk (if customer buys bread, the customer will buy milk also)

- Support of {Bread, Milk} = 3.
- Support of {Bread} = 4.
- **Confidence = 3/4 = 75%** (Passes threshold).
-

The **Apriori Algorithm**, as demonstrated in the bread-butter example, is widely used in modern startups like **Zomato**, **Swiggy**, and other food delivery platforms. These companies use it to perform **market basket analysis**, which helps them identify customer behavior patterns and optimize recommendations.

Applications of Apriori Algorithm
Below are some applications of Apriori algorithm used in today’s companies and startups

1. **E-commerce:** Used to recommend products that are often bought together, like **laptop + laptop bag**, increasing sales.
2. **Food Delivery Services:** Identifies popular combos, such as **burger + fries**, to offer **combo deals** to customers.
3. **Streaming Services:** Recommends related movies or shows based on what users often watch together, like **action + superhero movies**.
4. **Financial Services:** Analyzes spending habits to suggest personalized offers, such as **credit card deals** based on frequent purchases.
5. **Travel & Hospitality:** Creates travel packages (e.g., **flight + hotel**) by finding commonly purchased services together.
6. **Health & Fitness:** Suggests workout plans or supplements based on users’ past activities, like **protein shakes + workouts**.

Frequent Pattern Growth Algorithm

In our previous discussions, we explored the Apriori algorithm and identified two major **drawbacks** about which we will discuss in the this article. To overcome these challenges, the Frequent Pattern Growth (FP-Growth) algorithm was developed. FP-Growth addresses these inefficiencies by using a more efficient approach to mine frequent itemsets, eliminating the need for multiple database scans and speeding up the overall process. In this article we will deep down the basics of FP Growth Algorithm and understand how it works with real life data.

The Drawbacks of Apriori Algorithm
The two primary drawbacks of the Apriori Algorithm are:

1. At each step, candidate sets have to be built.
2. To build the candidate sets, the algorithm has to repeatedly scan the database.

These two properties inevitably make the algorithm slower. To overcome these redundant steps, a new association-rule mining algorithm was developed named Frequent Pattern Growth Algorithm. It overcomes the disadvantages of the Apriori algorithm by storing all the transactions in a **Trie Data Structure**.

Understanding The Frequent Patter Growth
The **FP-Growth algorithm** is a method used to find frequent patterns in large datasets. It is faster and more efficient than the Apriori algorithm because it avoids repeatedly scanning the entire database.
Here’s how it works in simple terms:

1. **Data Compression**: First, FP-Growth compresses the dataset into a smaller structure called the **Frequent Pattern Tree (FP-Tree)**. This tree stores information about itemsets (collections of items) and their frequencies, without needing to generate candidate sets like Apriori does.
2. **Mining the Tree**: The algorithm then examines this tree to identify patterns that appear frequently, based on a minimum support threshold. It does this by breaking the tree down into smaller “conditional” trees for each item, making the process more efficient.
3. **Generating Patterns**: Once the tree is built and analyzed, the algorithm generates the frequent patterns (itemsets) and the rules that describe relationships between items

Lets understand this with the help of a real life analogy:
Imagine you’re organizing a large family reunion, and you want to know which food items are most popular among the guests. Instead of asking everyone individually and writing down their answers one by one, you decide to use a more efficient method.
Step 1: Create a List of Items People Bring
Instead of asking every person what they like to eat, you ask them to write down what foods they brought. You then create a list of all the food items brought to the party. This is like scanning the entire database once to get an overview and insights of the data.
Step 2: Group Similar Items Together
Now, you group the food items that were brought most frequently. You might end up with groups like “**Pizza**” (which was brought by 10 people), “**Cake**” (by 4 people), “**Pasta**” (by 3 people), and others. This is similar to creating the **Frequent Pattern Tree (FP-Tree)** in FP-Growth, where you only keep track of the items that are common enough.
Step 3: Look for Hidden Patterns
Next, instead of going back to every person to ask again about their preferences, you simply look at your list of items and patterns. You notice that people who brought pizza also often brought pasta, and those who brought cake also brought pasta. These hidden relationships (e.g., pizza + pasta, cake + pasta) are like the “frequent patterns” you find in FP-Growth.
Step 4: Simplify the Process
With FP-Growth, instead of scanning the entire party list multiple times to look for combinations of items, you’ve condensed all the information into a smaller, more manageable tree structure. You can now quickly see the most common combinations, like “Pizza and pasta” or “Cake and pasta,” without the need to revisit every single detail.

Working of FP- Growth Algorithm
Lets jump to the usage of FP- Growth Algorithm and how it works with reallife data. Consider the following data:-

Transaction ID	**Items**
T1	{E,K,M,N,O,Y}
T2	{D,E,K,N,**O**,Y}
T3	{A,E,K,M}
T4	{C,K,M,U,Y}
T5	{C,E,I,K,O,O}

The above-given data is a hypothetical dataset of transactions with each letter representing an item. The frequency of each individual item is computed:-

Item	**Frequency**
A	1

Item	Frequency
C	2
D	1
E	4
I	1
K	5
M	3
N	2
O	4
U	1
Y	3

Let the minimum support be 3. A **Frequent Pattern set** is built which will contain all the elements whose frequency is greater than or equal to the minimum support. These elements are stored in descending order of their respective frequencies. After insertion of the relevant items, the set L looks like this:-

L = {K : 5, E : 4, M : 3, O : 4, Y : 3}

Now, for each transaction, the respective **Ordered-Item set** is built. It is done by iterating the Frequent Pattern set and checking if the current item is contained in the transaction in question. If the current item is contained, the item is inserted in the Ordered-Item set for the current transaction. The following table is built for all the transactions:

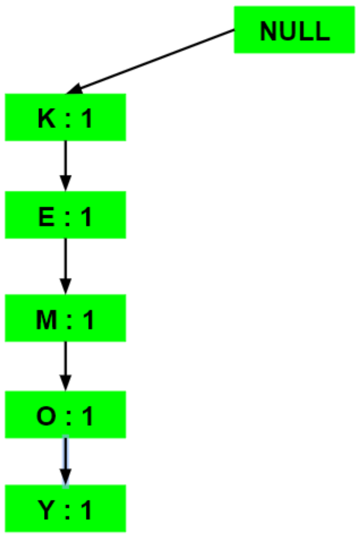
Transaction ID	Items	Ordered-Item-Set
T1	{E,K,M,N,O,Y}	{K,E,M,O,Y}
T2	{D,E,K,N,O,Y}	{K,E,O,Y}
T3	{A,E,K,M}	{K,E,M}
T4	{C,K,M,U,Y}	{A,E,K,M}

Transaction ID	Items	Ordered-Item-Set
T5	{C,E,I,K,O,O}	{A,E,K,M}

Now, all the Ordered-Item sets are inserted into a Trie Data Structure.

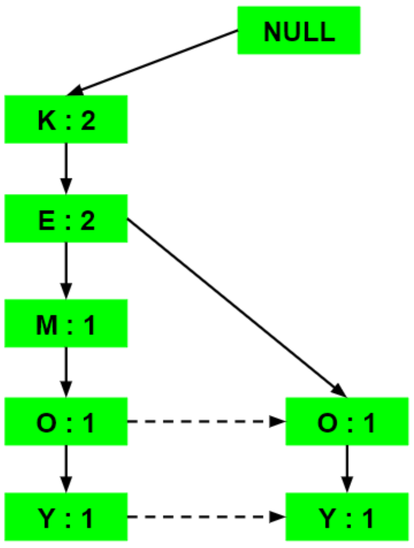
a) **Inserting the set {K, E, M, O, Y}:**

Here, all the items are simply linked one after the other in the order of occurrence in the set and initialize the support count for each item as 1.



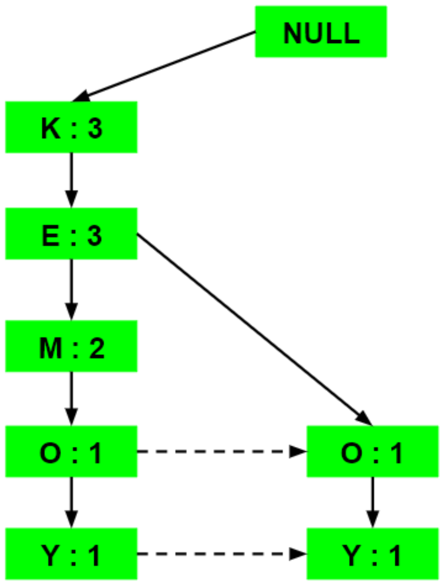
b) **Inserting the set {K, E, O, Y}:**

Till the insertion of the elements K and E, simply the support count is increased by 1. On inserting O we can see that there is no direct link between E and O, therefore a new node for the item O is initialized with the support count as 1 and item E is linked to this new node. On inserting Y, we first initialize a new node for the item Y with support count as 1 and link the new node of O with the new node of Y.

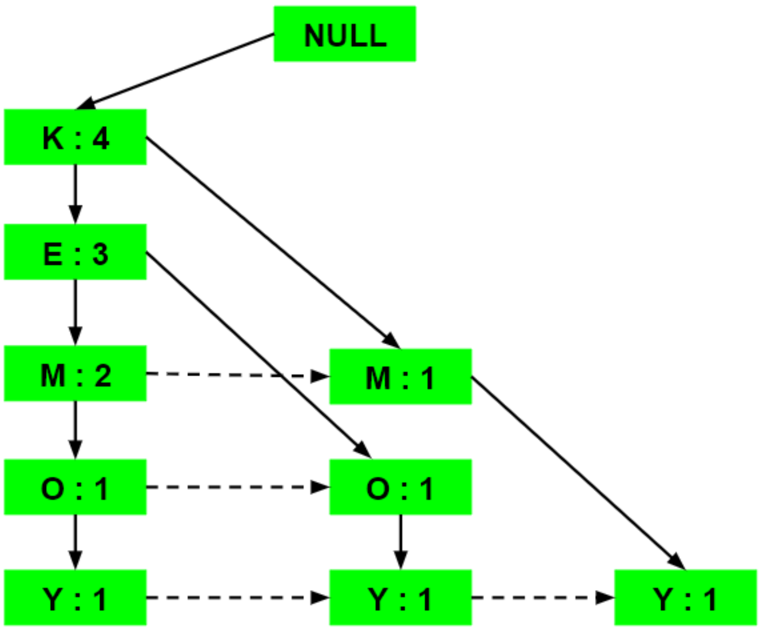


c) **Inserting the set {K, E, M}:**

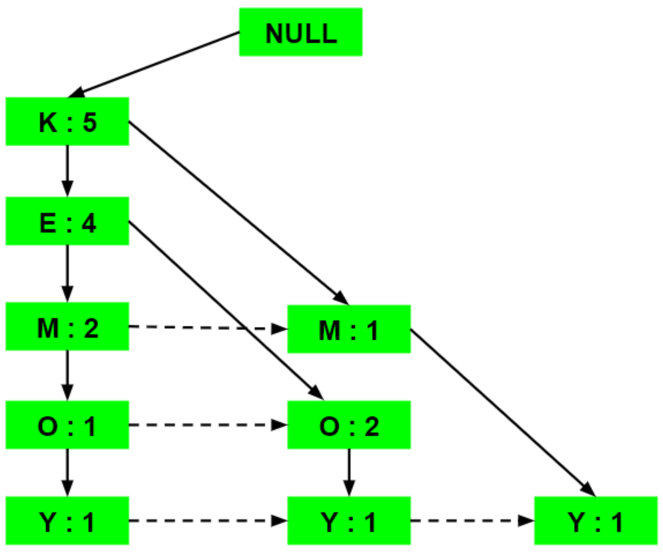
Here simply the support count of each element is increased by 1.



d) **Inserting the set {K, M, Y}:**
Similar to step b), first the support count of K is increased, then new nodes for M and Y are initialized and linked accordingly.



e) **Inserting the set {K, E, O}:**
Here simply the support counts of the respective elements are increased. Note that the support count of the new node of item O is increased.



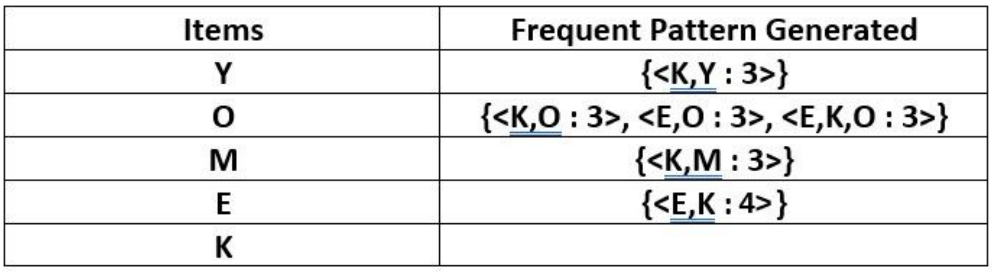
Now, for each item, the **Conditional Pattern Base** is computed which is path labels of all the paths which lead to any node of the given item in the frequent-pattern tree. Note that the items in the below table are arranged in the ascending order of their frequencies.

Items	Conditional Pattern Base
Y	{{K,E,M,O : 1}, {K,E,O : 1}, {K,M : 1}}
O	{{K,E,M : 1}, {K,E : 2}}
M	{{K,E : 2}, {K : 1}}
E	{K : 4}
K	

Now for each item, the **Conditional Frequent Pattern Tree is built.** It is done by taking the set of elements that is common in all the paths in the Conditional Pattern Base of that item and calculating its support count by summing the support counts of all the paths in the Conditional Pattern Base.

Items	Conditional Pattern Base	Conditional Frequent Pattern Tree
Y	{{K,E,M,O : 1}, {K,E,O : 1}, {K,M : 1}}	{K : 3}
O	{{K,E,M : 1}, {K,E : 2}}	{K,E : 3}
M	{{K,E : 2}, {K : 1}}	{K : 3}
E	{K : 4}	{K : 4}
K		

From the Conditional Frequent Pattern tree, the **Frequent Pattern rules** are generated by pairing the items of the Conditional Frequent Pattern Tree set to the corresponding to the item as given in the below table.



Items	Frequent Pattern Generated
Y	{<K,Y : 3>}
O	{<K,O : 3>, <E,O : 3>, <E,K,O : 3>}
M	{<K,M : 3>}
E	{<E,K : 4>}
K	

For each row, two types of association rules can be inferred for example for the first row which contains the element, the rules **K -> Y and Y -> K** can be inferred. To determine the valid rule, the confidence of both the rules is calculated and the one with confidence greater than or equal to the minimum confidence value is retained.

Conclusion
In conclusion, the Frequent Pattern Growth (FP-Growth) algorithm improves upon the Apriori algorithm by eliminating the need for multiple database scans and reducing computational overhead. By using a Trie data structure and focusing on ordered-item sets, FP-Growth efficiently mines frequent itemsets, making it a faster and more scalable solution for large datasets. This approach ensures faster processing while maintaining accuracy, making FP-Growth a powerful tool for data mining.

Gaussian Mixture Model

Clustering is a key technique in unsupervised learning, used to group similar data points together. While traditional methods like **K-Means** and **Hierarchical Clustering** are widely used, they assume that clusters are well-separated and have rigid shapes. This can be limiting in real-world scenarios where clusters can be more complex.

To overcome these limitations, **Gaussian Mixture Models (GMM)** offer a more flexible approach. Unlike K-Means, which assigns each point to a single cluster, GMM uses a probabilistic approach to cluster the data, allowing clusters to have more varied shapes and soft boundaries. Let's dive into what GMM is and how it works

Gaussian Mixture Model vs K-Means

- A Gaussian mixture model is a **soft clustering technique** used in unsupervised learning to determine the probability that a given data point belongs to a cluster. It's composed of several Gaussians, each identified by k ∈ {1,…, K}, where K is the number of clusters in a data set and is comprised of the following parameters.
- K-means is a clustering algorithm that assigns each data point to one cluster based on the closest centroid. It's a **hard clustering** method, meaning each point belongs to only one cluster with no uncertainty.

On the other hand, **Gaussian Mixture Models (GMM)** use **soft clustering**, where data points can belong to multiple clusters with a certain probability. This provides a more flexible and nuanced way to handle clusters, especially when points are close to multiple centroids.

How Gaussian Mixture Models Work?

Now that we understand the key differences between K-Means and GMM, let’s dive deeper into how GMM actually works. Unlike K-Means, where the clustering process relies solely on the centroid and assigns each data point to one cluster, GMM uses a probabilistic approach.
Here’s how GMM performs clustering:

1. **Multiple Gaussians (Clusters):** Each cluster is represented by a Gaussian distribution, and the data points are assigned probabilities of belonging to different clusters based on their distance from each Gaussian.
2. **Parameters of a Gaussian:** The core of GMM is made up of three main parameters for each Gaussian:
    - **Mean (μ):** The center of the Gaussian distribution.
    - **Covariance (Σ):** Describes the spread or shape of the cluster.
    - **Mixing Probability (π):** Determines how dominant or likely each cluster is in the data.

The Gaussian mixture model assigns a probability to each data point xnx_nxn of belonging to a cluster. The probability of data point coming from Gaussian cluster k is expressed as
P(zn=k|xn)=πk·N(xn|μk,Σk)∑k=1Kπk·N(xn|μk,Σk)*P*(*zn*=*k*|*xn*)=∑*k*=1*Kπk*·N(*xn*|*μk*,Σ*k*)*πk*·N(*xn*|*μk*,Σ*k*)

- where,
- zn*zn*=k is a latent variable indicating which Gaussian the point belongs to.
- πk*πk* is the mixing probability of the k-th Gaussian.
- N(xn|μk,Σk)N(*xn*|*μk*,Σ*k*)is the Gaussian distribution with mean μk\mu_kμk and covariance Σk

Next, we need to calculate the overall likelihood of observing a data point xnx_nxn under all Gaussians. This is achieved by summing over all possible clusters (Gaussians) for each point:
P(xn)=∑k=1KπkN(xn|μk,Σk)*P*(*xn*)=∑*k*=1*Kπk*N(*xn*|*μk*,Σ*k*)

- Where:
- P(xn)*P*(*xn*) is the overall likelihood of observing the data point xn*xn*
- The sum accounts for all possible Gaussians k.

The Expectation-Maximization (EM) Algorithm
To fit a Gaussian Mixture Model to the data, we use the **Expectation-Maximization (EM)** algorithm, which is an iterative method that optimizes the parameters of the Gaussian distributions (mean, covariance, and mixing coefficients). It works in two main steps:

1. **Expectation Step (E-step)**:
    - In this step, the algorithm calculates the probability that each data point belongs to each cluster based on the current parameter estimates (mean, covariance, mixing coefficients).
2. **Maximization Step (M-step)**:
    - After estimating the probabilities, the algorithm updates the parameters (mean, covariance, and mixing coefficients) to better fit the data.

These two steps are repeated until the model converges, meaning the parameters no longer change significantly between iterations.

How GMM Works
Here’s a simple breakdown of the process:

1. **Initialization**: Start with initial guesses for the means, covariances, and mixing coefficients of each Gaussian distribution.
2. **E-step**: For each data point, calculate the probability of it belonging to each Gaussian distribution (cluster).

3. **M-step**: Update the parameters (means, covariances, mixing coefficients) using the probabilities calculated in the E-step.
4. **Repeat**: Continue alternating between the E-step and M-step until the log-likelihood of the data (a measure of how well the model fits the data) converges.

Formula:

$L(\mu k,\Sigma k,\pi k)=\prod n=1N\sum k=1K\pi kN(xn|\mu k,\Sigma k)$$L(\mu_k,\Sigma_k,\pi_k)=\prod_{n=1}^{N}\sum_{k=1}^{K}\pi_k \mathrm{N}(x_n|\mu_k,\Sigma_k)$

The E-step computes the probabilities that each data point belongs to each Gaussian, while the M-step updates the parameters μk, Σk , and πk based on these probabilities.

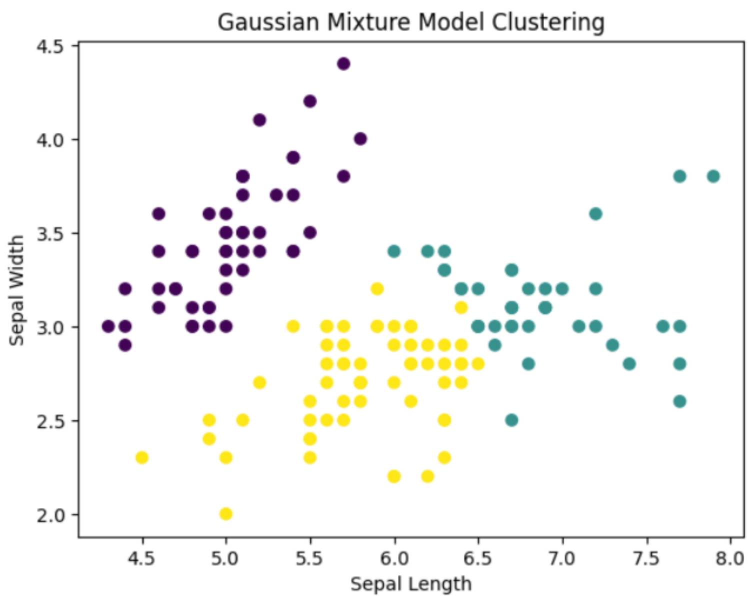
Implementation of GMM in python

So far, we have discussed about the working of GMM. Let’s take a simple example using the **Iris dataset** and fit a Gaussian Mixture Model with 3 clusters (since we know there are 3 species of Iris).

1. **Dataset**: We use only two features: sepal length and sepal width.
2. **Fitting the Model**: We fit the data as a mixture of 3 Gaussians.
3. **Result**: The model assigns each data point to a cluster based on the probabilities. After convergence, the GMM model will have updated the parameters to best fit the data.

```python
from sklearn.mixture import GaussianMixture
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris

# Load iris dataset
iris = load_iris()
X = iris.data[:, :2]  # using only sepal length and sepal width

# Fit GMM with 3 components (clusters)
gmm = GaussianMixture(n_components=3)
gmm.fit(X)

# Predict the labels (cluster assignments)
labels = gmm.predict(X)

# Plot the results
plt.scatter(X[:, 0], X[:, 1], c=labels, cmap='viridis')
plt.xlabel('Sepal Length')
plt.ylabel('Sepal Width')
plt.title('Gaussian Mixture Model Clustering')
plt.show()
```

Output:



GMM Clustering

This scatter plot shows the result of clustering data using a **Gaussian Mixture Model (GMM)**. The data points represent measurements of flower species based on **sepal length** and **sepal width**. The points are divided into three clusters, represented by different colors (yellow, purple, and teal), which indicate that GMM has identified three distinct groups within the data.

Advantages of Gaussian Mixture Models (GMM)

1. **Flexible Cluster Shapes**: Unlike K-Means, which assumes spherical clusters, GMM can model clusters with arbitrary shapes.
2. **Soft Assignment**: GMM assigns a probability for each data point to belong to each cluster, while K-Means assigns each point to exactly one cluster.
3. **Handles Overlapping Data**: GMM performs well when clusters overlap or have varying densities. Since it uses probability distributions, it can assign a point to multiple clusters with different probabilities

Limitations of GMM

1. **Computational Complexity**: GMM tends to be computationally expensive, particularly with large datasets, as it requires iterative processes like the Expectation-Maximization (EM) algorithm to estimate the parameters.

- **Choosing the Number of Clusters**: Like other clustering methods, GMM requires you to specify the number of clusters beforehand. However, methods like the **Bayesian Information Criterion (BIC)** and **Akaike Information Criterion (AIC)** can help in selecting the optimal number of clusters based on the data

Conclusion

Gaussian Mixture Models provide a flexible and powerful tool for clustering when data has more complex structures. By treating clustering as a probabilistic problem, GMM allows for clusters with varied shapes and soft boundaries, unlike traditional methods like K-Means. The **Expectation-Maximization** algorithm is used to iteratively refine the model's parameters, making GMM a great choice for many real-world clustering problems.

Frequently Asked Questions (FAQ’s)

When should I use Gaussian mixture model?

Gaussian Mixture Models (GMM) are used for clustering and density estimation in unsupervised learning, particularly when data comes from multiple normal distributions. It’s applied in areas like image segmentation, market segmentation, and anomaly detection.

Why is GMM better than K means ?

K-means assigns each data point to one cluster, while GMM assigns points to multiple clusters with probabilities. GMM can handle elliptical shapes, unlike K-means, which only detects spherical clusters. This makes GMM more flexible for complex data

How does GMM handle overlapping clusters?

GMM can handle overlapping clusters by assigning probabilities to data points, allowing them to belong to more than one cluster. This is useful when clusters have varying densities or are not well-separate.