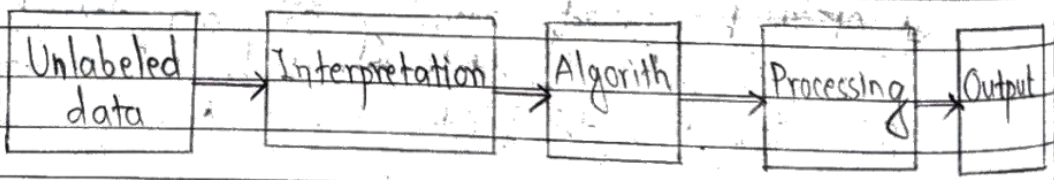
7. Unsupervised Learning in ML

1. Introduction

- Unsupervised learning is a type of machine learning that learns from unlabeled data.
- This means that the data doesn't have any pre-existing labels or categories.
- The goal of unsupervised learning is to discover patterns and relationships in the data without any explicit guidance.

Remember: labeled data → has input and output
unlabeled data → only input no output.



Types of Unsupervised Learning

1. **Clustring:** Clustring is an unsupervised machine learning technique used to group similar data points into clusters based on their characteristics.
2. **Association:** Association rule learning is another unsupervised learning technique used to discover interesting relationships between variables in large datasets.

Popular Unsupervised learning Algorithms.

- K-means clustering
- Hierarchal clustering
- DBSCAN clustering
- Apriori algorithm
- Principle component analysis.

2. K-Means Clustering

↳ K-means clustring is an unsupervised learning algorithm, which groups the unlabeled dataset into different clusters.

↳ K defines the number of pre-defined clusters that need to be created in the process.

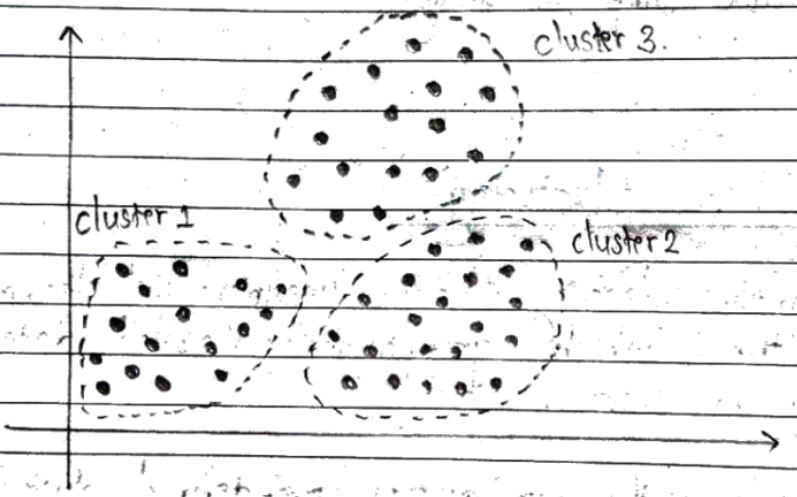
Elbow Method.

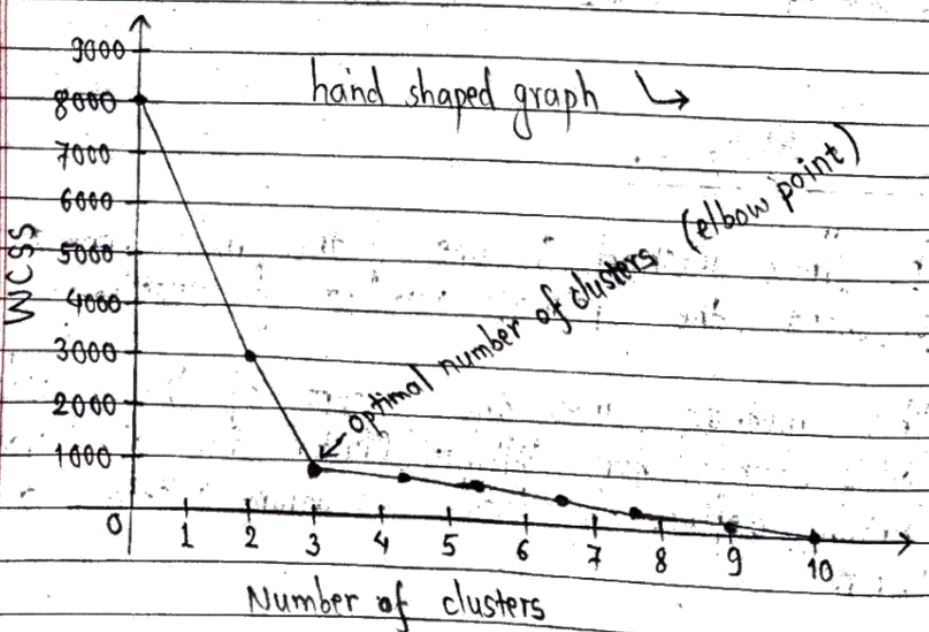
↳ The elbow method is one of the most popular ways to find the optimal number of clusters.

↳ This method uses the concept of WCSS value. WCSS stands for Within Cluster Sum of Squares, which defines the total variations within a cluster.

↳ Mathematically,

$WCSS = \sum_{p_i \text{ in cluster1}} \text{distance}(P_i C_1)^2 + \sum_{p_i \text{ in cluster2}} \text{distance}(P_i C_2)^2 + \sum_{p_i \text{ in cluster3}} \text{distance}(P_i C_3)^2 + \ldots$





Working practically

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

dataset = pd.read_csv("IrisFlower.csv")
dataset.head(3)

# Our data is labeled
# For supervised learning we need unlabeled data
# So, lets remove Species column and make it
# unlabeled
dataset.drop(columns="Species", inplace=True)
dataset.head(3)

# Visualize the dataset
sns.pairplot(data=dataset)
plt.show()

# Find the best number of clusters
from sklearn.cluster import KMeans

wcs = []
for i in range(2,21):
km = KMeans(n_clusters=i, init="k-means++")
# k-means++ improves the initialization of cluster
#centroids, which helps in achieving better clustering
# results and faster convergence
km.fit(dataset)
wcss.append(km.inertia_) # Value of wcss
```

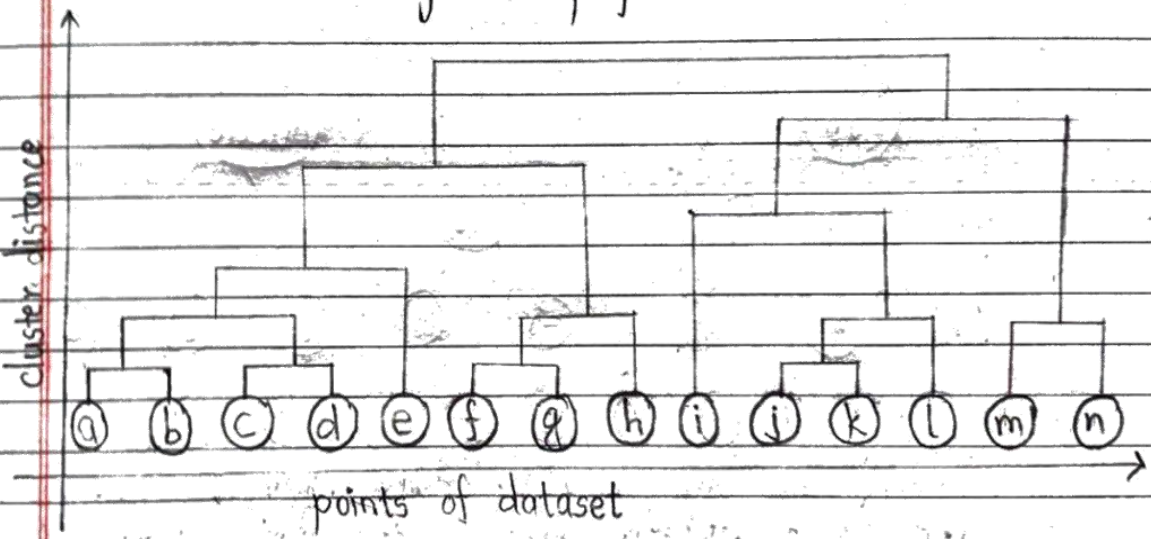
```
# Draw an elbow graph.
plt.figure (figsize = (10, 5))
plt.plot ([i for i in range (2, 21)], wcss,
                    marker = 'o')
plt.xlabel ("No of clusters")
plt.xticks ([i for i in range (2,21)]) # List of
# position at which to place the ticks
plt.ylabel ("WCSS")
plt.grid (axis = "x")
plt.show ()
"" 3 is found to be elbow point ""

# Make clusters with best value of k.
kmn = KMeans (n_clusters = 3)
kmn.fit_predict (dataset)

# Save the predicted value to dataset
dataset ["Predict"] = kmn.fit_predict (dataset)
dataset.head (3)

# Visualize clusters
sns.pairplot (dataset, hue = "Predict")
plt.show ()
```

3 Hierarchical Clustering

↳ It 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 tree, and this tree-shaped structure is known as the dendrogram.

- The dendogram : is a tree-like structure that is mainly used to store each step as a memory that the HC algorithm performs.



In dendogram 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.

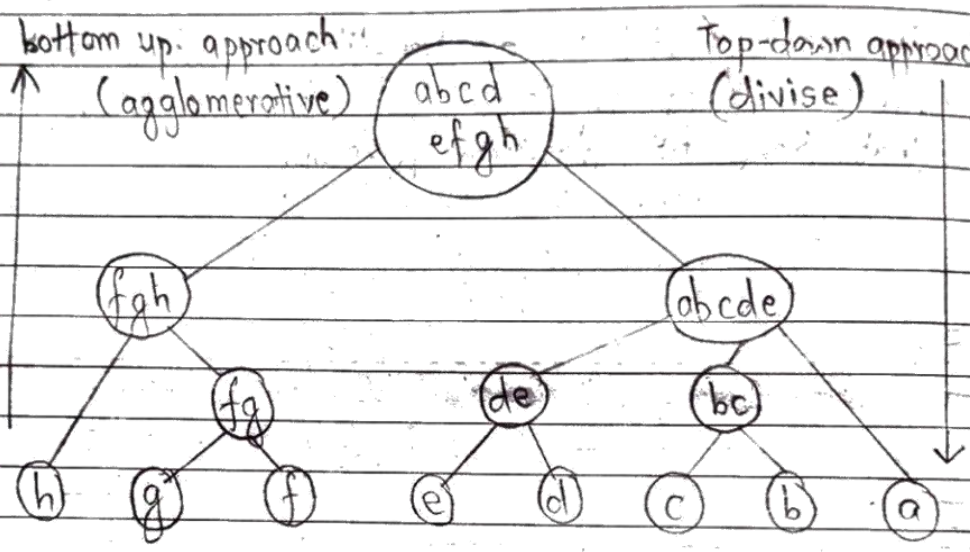
Hierarchical Clustering Technique

Hierarchical clustering technique has two approaches:

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

Date
Page 149

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



Measure for the distance between two clusters:
The closest distance between the two clusters is crucial for the hierarchical clustering. There are various ways to calculate distance between two clusters, and these ways decide the rules for clustering. These measure are called Linkage methods.

- Single linkage (min)
- Complete linkage (max)
- Average linkage
- Centroid linkage.

Working practically

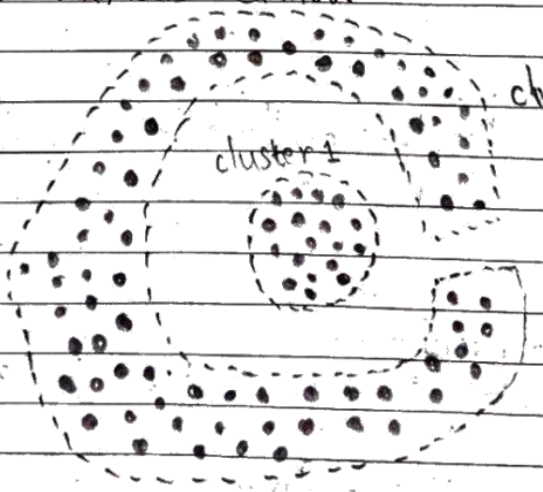
```
import pandas as pd
```

```
import matplotlib.pyplot as plt
import seaborn as sns

dataset = pd.read_csv("IrisFlower.csv")
dataset.head(3)

# We need a unlabeled dataset
# Converting a dataset to unlabeled
dataset.drop(columns="Species", inplace=True)
dataset.head(3)

# Visualize data
sns.pairplot(dataset)
plt.show()

# Create a dendrogram
import scipy.cluster.hierarchy as sc

plt.figure(figsize=(50, 25))
sc.dendrogram(sc.linkage(dataset, method="single",
                         metric="euclidean"))
plt.show()

# Make the clusters
from sklearn.cluster import AgglomerativeClustering

ac = AgglomerativeClustering(n_clusters=2, linkage=
                             "single")
ac.fit_predict(dataset)

# Save the predicted cluster value to dataset
```

```
dataset["Predict"] = ac.fit_predict(dataset)
dataset.head(3)

#Visualize the clusters
sns.pairplot(dataset, hue="Predict")
plt.show()
```

4. DBSCAN Clustring Algorithm

- Density-Based Spatial Clustering of Applications with Noise.
- The clusters found by DBSCAN can be any shape, which can deal with some special cases that other methods cannot.



Working Practically.

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Create a dataset using make_moons
from sklearn.datasets import make_moons

x, y = make_moons (n_sample = 250, noise = 0.05)
x, y  # We just need x i.e. input to create
# unlabeled dataset

# Create a dataset using the data
df = {"Data 1": x [:, 0], "Data 2": x [:, 1], }
dataset = pd. DataFrame (df)
 dataset.head (3)

# Visualize your data
sns.scatterplot (x = "Data 1", y = "Data 2", data = dataset)
 plt.show ()

# Make clusters using DBSCAN Clustering Algorithm
 from sklearn. cluster import DBSCAN

dbscan = DBSCAN (eps = 0.2, min_samples = 5)
# eps is radius of circle and min_sample is
# minimum number of datapoints need to be in
# circle for continuing clustering
dbscan.fit_predict (dataset)

# Add prediction values to dataset
dataset ["Predict"] = dbscan. fit_predict (dataset)
```

```
dataset.head(3)

#Visualize clusters
sns.scatterplot(x="Data1", y="Data2", data=
                dataset, hue="Predict")
plt.show()
```

5. Silhouette Score

- Silhouette refers to a method of interpretation and validation of consistency within clusters of data.
- Silhouette Coefficient or silhouette score is a metric used to calculate the goodness of a clustering technique.
- Its value ranges from -1 to 1.
- Mathematically,

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}, \text{ if } |C_I| > 1$$

and

$$s(i) = 0, \text{ if } |C_I| = 1$$

The given mathematical formula can also be written as:

$$s(i) \begin{cases} 1-[a(i)/b(i)], & \text{if } a(i) < b(i) \\ 0 & \text{if } a(i) = b(i) \\ [b(i)/a(i)]-1 & \text{if } a(i) > b(i) \end{cases}$$

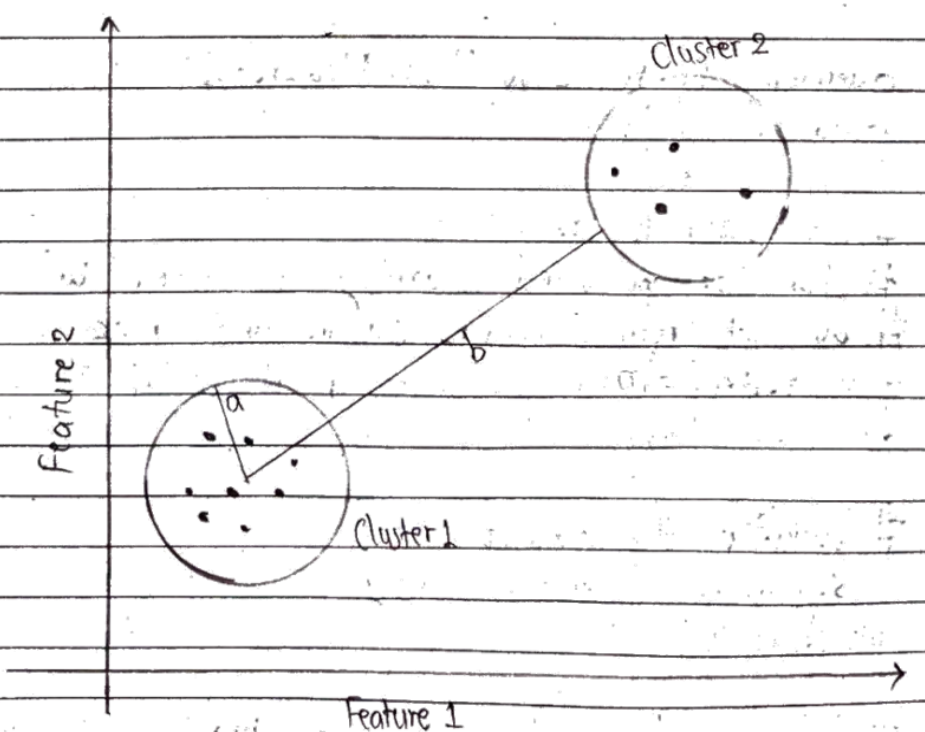
From above defination it is clear that:

$$-1 \leq s(i) \leq 1$$

Here:

$$a(i) = \frac{1}{|C_I|-1} \sum_{\substack{j \in C_I, \\ i \neq j}} d(i,j)$$

$$b(i) = \min_{J \neq I} \frac{1}{C_J} \sum_{j \in C_J} d(i,j)$$



a(i): the average distance from the datapoint (i) to all other points on same cluster.
b(i): the smallest average distance from the data point (i) to all other clusters that (i) is not a part of.

Remember: a(i) and b(i) are similar as a_i and b_i

Working Practically

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans

dataset = pd.read_csv("IrisFlower.csv")
dataset.head(3)

#Our data is labeled
# For unsupervised learning we need unlabeled data
# So lets remove species column and make it unlabeled.
dataset.drop(columns="Species", inplace=True)
dataset.head(3)

# Visualize the dataset
sns.pairplot(data=dataset)
plt.show()

# Silhouette score to find best number of clusters
from sklearn.metrics import silhouette_score
```

```
# Find best number of clusters
ss = []
for i in range (2, 21):
    km = KMeans (n_clusters = i)
    km.fit (dataset)
    ss.append (silhouette_score (dataset , label =
                km.labels_ ))

# Create a graph between ss and number of clusters
no_of_clusters = [i for i in range (2, 21)]
plt.plot (no_of_clusters , ss, marker = "o")
plt.xlabel ("No. of clusters")
plt.xticks (no_of_clusters)
plt.ylabel ("Silhouette score")
plt.grid (axis = "x")
plt.show()
"""The best number of clusters is found to be 2"""

# Train model using best number of clusters.
km = KMeans (n_clusters = 2)
km.fit (dataset)

# Add the predict value to dataset for visualization
dataset ["Predict"] = km.predict (dataset)
dataset.head (3)

# Visualization of clusters
sns.pairplot (dataset , hue = "Predict")
plt.show()
```

6. Association Rule Learning

- Association rule learning is a machine learning technique used to discover interesting relationships or patterns in large datasets.
- It is often applied to transactional data, where items are bought or used together.
- Association rule learning works on the concept of if and else statement, such as if A then B. If element is called antecedent, and then statement is called as consequent
- Association rule learning work on the basis of: Support, Confidence and Lift

Support

- The first step for us and algorithm is to find frequently bought items.
- It is a straight forward calculation that is based on frequency.
- Mathematically,

  Support (A) = Transactions (A) / Total transactions

Confidence:

- We have identified frequently bought items let's calculate confidence.
- This will tell us how confident (based on our data) we can be that an item will be purchased, given that another item has been purchased.
- Mathematically,

$$\text{conf}(X \Rightarrow Y) = P(Y/X) = \frac{\text{supp}(X \cap Y)}{\text{supp}(X)}$$

$$= \frac{\text{number of transactions containing } X \text{ and } Y}{\text{number of transactions containing } X}$$

Lift

- Given that different items are bought at different frequencies (strong association)
- Mathematically,

$$\text{lift}(X \Rightarrow Y) = \frac{\text{supp}(X \cap Y)}{\text{supp}(X) * \text{supp}(Y)}$$

- lift > 1 : means that the two items are more likely to be bought together.
  lift < 1 : means that the two items are more

likely to be bought seperately.

lift = 1 : means that there is no association between the two items.

Types of Association Rule Learning

- Apriori
- Eclat
- F-P Growth Algorithm

Applications of Association Rule Learning

- Market basket analysis
- Medical diagnosis
- Protein seqdence

7. Apriori Algorithm

↳ The apriori algorithm is a fundamental algorithm in data mining used for mining frequent itemsets and discovering association rules.

Steps of Apriori Algorithm

1. Generate Candidate Itemsets : Start with single items and generate candidate itemsets of length (k) from frequent itemsets of length (k-1).

2. Prune Infrequent Itemsets: Remove candidate itemsets that don't meet the minimum support threshold.

3. Repeat: Repeat the process until no more candidate itemsets can be generated.

Working Practically:

```
import pandas as pd

dataset = pd.read_csv("Groceries.csv")
dataset.head(3)

# Check for null values
dataset.isnull().sum()

# Create a list over lists by removing the null values
market = []
for i in range(0, dataset.shape[0]): # Till all row
# finish
        customer = []
        for j in dataset.columns : # Till all columns
            if type(dataset[j][i]) is str:
                customer.append(dataset[j][i])
        market.append(customer)
# Final list
market

# Identify most frequently bought products
import collections
```

```
# Convert the 2D list to 1D list
l = []
for i in market:
    for j in i:
        l.append(j)
# Display list
l

# Count total number of repetitions
collections.Counter(l) # Takes 1D list

# Create a dataset for observation
item_count = collections.Counter(l)
df = {"Item Name": item_count.keys(), "Repetition":
      item_count.values()}
dataframe = pd.DataFrame(df).sort_values(by = ["
            Repetition"], ascending = False]
dataframe

# Select max number of rows to display
pd.set_option("display.max_rows", 200)
dataframe.

# Create a confusion matrix for Apriori Algorithm
from mlxtend.preprocessing import TransactionEncoder

tr = TransactionEncoder()
tr.fit(market)

tr.transform(market)
```

```
# Create a dataframe of confusion matrix
dframe = pd.DataFrame(tr.transform(market),
                      columns = tr.columns_)
dframe

# Use Apriori Algorithm
from mlxtend.frequent_patterns import apriori

apriori(dframe, min_support = 0.05, use_colnames = True,
        max_len = 3).sort_values(by = "support",
        ascending = False)
```

8. Frequent Pattern Growth Algorithm

Frequent Pattern growth algorithm is a popular method for mining frequent itemsets in large datasets. It addresses some of the limitations of the Apriori algorithm by using a more efficient data structure called FP-Tree (Frequent Pattern Tree)

Steps of FP-Growth Algorithm

1. Frequent Pattern set
2. Ordered-item set
3. Ordered-item set and conditional frequent pattern tree is built.
4. Frequent pattern rules are used.

Working Practically

```
import pandas as pd

# Set max number of rows to display.
pd.set_option("display.max_rows", 200)

dataset = pd.read_csv("Groceries.csv")
dataset.head(3)

# Create a list over lists removing the null values
market = []
for i in range(0, dataset.shape[0]): # Till all rows
    customer = []
    for j in dataset.columns: # Till all columns
        if type(dataset[j][i]) is str:
            customer.append(dataset[j][i])
    market.append(customer)
# Final list
market

# Identify most frequently bought products
import collections

# Convert the 2D list to 1D list
l = []
for i in market:
    for j in i:
        l.append(j)
# Count total number of repetition
collections.Counter(l) # Takes 1D list
```

```
# Create a dataset for observation
item_count = collections.Counter(1)
df = {"Item Name": item_count.keys(),
      "Repetition": item_count.values()}
dataframe = pd.DataFrame(df).sort_values(by=[
      "Repetition"], ascending=False)
dataframe

# Create a confusion matrix (Data Encoding)
from mlxtend.preprocessing import TransactionEncoder

tr = TransactionEncoder()
tr.fit(market)

tr.transform(market)

# Create a dataframe of confusion matrix
dframe = pd.DataFrame(tr.transform(market), columns=
      tr.columns_)
dframe

# Use frequent pattern growth algorithm
from mlxtend.frequent_patterns import fpgrowth

fpgrowth(dframe, min_support=0.05, use_colnames=True,
      max_len=3).sort_values(by="support",
      ascending=False)
```