**WEEK-1**

**AIM:**

Write a program to generate Association Rules using the Apriori algorithm

**Description:**

Apriori algorithm refers to the algorithm which is used to calculate the association rules between objects. It means how two or more objects are related to one another. In other words, we can say that the apriori algorithm is an association rule leaning that analyzes that people who bought product A also bought product B.

The primary objective of the apriori algorithm is to create the association rule between different objects. The association rule describes how two or more objects are related to one another. Apriori algorithm is also called frequent pattern mining. Generally, you operate the Apriori algorithm on a database that consists of a huge number of transactions. Let's understand the apriori algorithm with the help of an example; suppose you go to Big Bazar and buy different products. It helps the customers buy their products with ease and increases the sales performance of the Big Bazar. In this tutorial, we will discuss the apriori algorithm with examples. Apriori algorithm refers to an algorithm that is used in mining frequent products sets and relevant association rules. Generally, the apriori algorithm operates on a database containing a huge number of transactions. For example, the items customers but at a Big Bazar.

Apriori algorithm helps the customers to buy their products with ease and increases the sales performance of the particular store.

**Code:**

import pandas as pd

from mlxtend.preprocessing import TransactionEncoder

from mlxtend.frequent\_patterns import apriori, association\_rules

dataset = [['Pen','Bag','Pencil'],

           ['Pen','Book','Water Bottle','Pencil'],

           ['Book','Bag','Water Bottle'],

           ['Pen','Book','Bag','Water Bottle']]

te = TransactionEncoder()

te\_ary = te.fit(dataset).transform(dataset)

df = pd.DataFrame(te\_ary, columns=te.columns\_)

print(df)

# Building the model

frq\_items = apriori(df, min\_support = 0.05, use\_colnames = True)

print(frq\_items)

# Collecting the inferred rules in a dataframe

rules = association\_rules(frq\_items, metric ="lift", min\_threshold = 1)

rules = rules.sort\_values(['confidence', 'lift'], ascending =[False, False])

print(rules.head())

Bag Book Pen Pencil Water Bottle

0 True False True True False

1 False True True True True

2 True True False False True

3 True True True False True

support itemsets

0 0.75 (Bag)

1 0.75 (Book)

2 0.75 (Pen)

3 0.50 (Pencil)

4 0.75 (Water Bottle)

5 0.50 (Book, Bag)

6 0.50 (Bag, Pen)

7 0.25 (Pencil, Bag)

8 0.50 (Water Bottle, Bag)

9 0.50 (Book, Pen)

10 0.25 (Pencil, Book)

11 0.75 (Water Bottle, Book)

12 0.50 (Pencil, Pen)

13 0.50 (Water Bottle, Pen)

14 0.25 (Water Bottle, Pencil)

15 0.25 (Book, Bag, Pen)

16 0.50 (Water Bottle, Book, Bag)

17 0.25 (Pencil, Bag, Pen)

18 0.25 (Water Bottle, Bag, Pen)

19 0.25 (Pencil, Book, Pen)

20 0.50 (Water Bottle, Book, Pen)

21 0.25 (Pencil, Book, Water Bottle)

22 0.25 (Water Bottle, Pencil, Pen)

23 0.25 (Water Bottle, Book, Bag, Pen)

24 0.25 (Pencil, Water Bottle, Book, Pen)

antecedents consequents antecedent support \

40 (Water Bottle, Pencil) (Book, Pen) 0.25

41 (Book, Pencil) (Water Bottle, Pen) 0.25

0 (Water Bottle) (Book) 0.75

1 (Book) (Water Bottle) 0.75

2 (Pencil) (Pen) 0.50

consequent support support confidence lift leverage conviction

40 0.50 0.25 1.0 2.000000 0.1250 inf

41 0.50 0.25 1.0 2.000000 0.1250 inf

0 0.75 0.75 1.0 1.333333 0.1875 inf

1 0.75 0.75 1.0 1.333333 0.1875 inf

2 0.75 0.50 1.0 1.333333 0.1250 inf

**WEEK-2**

**AIM:**

Write a program to generate Association Rules using the FP-Growth algorithm.

**Description:**

In Data Mining, finding frequent patterns in large databases is very important and has been studied on a large scale in the past few years. Unfortunately, this task is computationally expensive, especially when many patterns exist.

The FP-Growth Algorithm proposed by **Han in**. This is an efficient and scalable method for mining the complete set of frequent patterns by pattern fragment growth, using an extended prefix-tree structure for storing compressed and crucial information about frequent patterns named frequent-pattern tree (FP-tree). In his study, Han proved that his method outperforms other popular methods for mining frequent patterns, e.g. the Apriori Algorithm and the TreeProjection. In some later works, it was proved that FP-Growth performs better than other methods, including **Eclat** and **Relim**. The popularity and efficiency of the FP-Growth Algorithm contribute to many studies that propose variations to improve its performance.

The FP-Growth Algorithm is an alternative way to find frequent item sets without using candidate generations, thus improving performance. For so much, it uses a divide-and-conquer strategy. The core of this method is the usage of a special data structure named frequent-pattern tree (FP-tree), which retains the item set association information.

**Code:**

import sys

!{sys.executable} -m pip install fpgrowth

Looking in indexes: <https://pypi.org/simple>, <https://us-python.pkg.dev/colab-wheels/public/simple/>

Collecting fpgrowth

Downloading fpGrowth-1.0.0.tar.gz (2.1 kB)

Building wheels for collected packages: fpgrowth

Building wheel for fpgrowth (setup.py) ... done

Created wheel for fpgrowth: filename=fpGrowth-1.0.0-py3-none-any.whl size=2866 sha256=458dd64cdcdca2a32dd39b0d375e68d6d04546eba93e66f5255e27362d23650f

Stored in directory: /root/.cache/pip/wheels/64/33/72/1991a9117d1813325c4ef85597ba8ece8c4780adc240bd0b0f

Successfully built fpgrowth

Installing collected packages: fpgrowth

Successfully installed fpgrowth-1.0.0

%pip install mlxtend –upgrade

Looking in indexes: <https://pypi.org/simple>, <https://us-python.pkg.dev/colab-wheels/public/simple/>

Requirement already satisfied: mlxtend in /usr/local/lib/python3.7/dist-packages (0.14.0)

Collecting mlxtend

Downloading mlxtend-0.21.0-py2.py3-none-any.whl (1.3 MB)

|████████████████████████████████| 1.3 MB 22.6 MB/s

Requirement already satisfied: numpy>=1.16.2 in /usr/local/lib/python3.7/dist-packages (from mlxtend) (1.21.6)

Requirement already satisfied: scikit-learn>=1.0.2 in /usr/local/lib/python3.7/dist-packages (from mlxtend) (1.0.2)

Requirement already satisfied: joblib>=0.13.2 in /usr/local/lib/python3.7/dist-packages (from mlxtend) (1.2.0)

Requirement already satisfied: setuptools in /usr/local/lib/python3.7/dist-packages (from mlxtend) (57.4.0)

Requirement already satisfied: matplotlib>=3.0.0 in /usr/local/lib/python3.7/dist-packages (from mlxtend) (3.2.2)

Requirement already satisfied: scipy>=1.2.1 in /usr/local/lib/python3.7/dist-packages (from mlxtend) (1.7.3)

Requirement already satisfied: pandas>=0.24.2 in /usr/local/lib/python3.7/dist-packages (from mlxtend) (1.3.5)

Requirement already satisfied: pyparsing!=2.0.4,!=2.1.2,!=2.1.6,>=2.0.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib>=3.0.0->mlxtend) (3.0.9)

Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib>=3.0.0->mlxtend) (1.4.4)

Requirement already satisfied: python-dateutil>=2.1 in /usr/local/lib/python3.7/dist-packages (from matplotlib>=3.0.0->mlxtend) (2.8.2)

Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/dist-packages (from matplotlib>=3.0.0->mlxtend) (0.11.0)

Requirement already satisfied: typing-extensions in /usr/local/lib/python3.7/dist-packages (from kiwisolver>=1.0.1->matplotlib>=3.0.0->mlxtend) (4.1.1)

Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-packages (from pandas>=0.24.2->mlxtend) (2022.4)

Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (from python-dateutil>=2.1->matplotlib>=3.0.0->mlxtend) (1.15.0)

Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist-packages (from scikit-learn>=1.0.2->mlxtend) (3.1.0)

Installing collected packages: mlxtend

Attempting uninstall: mlxtend

Found existing installation: mlxtend 0.14.0

Uninstalling mlxtend-0.14.0:

Successfully uninstalled mlxtend-0.14.0

Successfully installed mlxtend-0.21.0

EXAMPLE2:

import pandas as pd

from mlxtend.preprocessing import TransactionEncoder

from mlxtend.frequent\_patterns import fpgrowth

dataset = [['k', 'O', 'N', 'M', 'E', 'Y'],

           ['N', 'O', 'D', 'K', 'E', 'Y'],

           ['M', 'A', 'K', 'E'],

           ['N', 'U', 'A', 'K', 'Y'],

           ['C', 'O', 'O', 'k']]

te = TransactionEncoder()

te\_ary = te.fit(dataset).transform(dataset)

df = pd.DataFrame(te\_ary, columns=te.columns\_)

print(df)

fpgrowth(df, min\_support=0.6)

fpgrowth(df, min\_support=0.6, use\_colnames=True)

A C D E K M N O U Y k

0 False False False True False True True True False True True

1 False False True True True False True True False True False

2 True False False True True True False False False False False

3 True False False False True False True False True True False

4 False True False False False False False True False False True

|  | **support** | **itemsets** |
| --- | --- | --- |
| **0** | 0.6 | (Y) |
| **1** | 0.6 | (O) |
| **2** | 0.6 | (N) |
| **3** | 0.6 | (E) |
| **4** | 0.6 | (K) |
| **5** | 0.6 | (N, Y) |

**WEEK-3**

**AIM:**

*Write a program to implement K-means clustering algorithm.*

**Description:**

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. K-Means Clustering is an [Unsupervised Learning algorithm](https://www.javatpoint.com/unsupervised-machine-learning), 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](https://www.javatpoint.com/clustering-in-machine-learning) 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.

**Code:**import matplotlib.pyplot as plt

from sklearn.cluster import KMeans

x = [1, 1.5, 3, 5, 3.5, 4.5, 3.5]

y = [1,2,4,7,5,5,4.5]

plt.scatter(x, y)

plt.show()

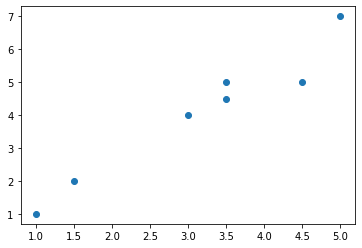
data = list(zip(x, y))

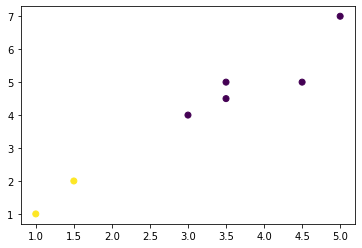
kmeans = KMeans(n\_clusters=2)

kmeans.fit(data)

plt.scatter(x, y, c=kmeans.labels\_)

plt.show()

plt.legend()



WARNING:matplotlib.legend:No handles with labels found to put in legend.

<matplotlib.legend.Legend at 0x7f8494666810>