Apriori Algorithm in Machine Learning

The Apriori algorithm uses frequent itemsets to generate association rules, and it is designed to work on the databases that contain transactions. With the help of these association rule, it determines how strongly or how weakly two objects are connected. This algorithm uses a **breadth-first search** and **Hash Tree** to calculate the itemset associations efficiently. It is the iterative process for finding the frequent itemsets from the large dataset.

This algorithm was given by the **R. Agrawal** and **Srikant** in the year **1994**. It is mainly used for *market basket analysis* and helps to find those products that can be bought together. It can also be used in the healthcare field to find drug reactions for patients.

What is Frequent Itemset?

Frequent itemsets are those items whose support is greater than the threshold value or user-specified minimum support. It means if A & B are the frequent itemsets together, then individually A and B should also be the frequent itemset.

Suppose there are the two transactions: A= {1,2,3,4,5}, and B= {2,3,7}, in these two transactions, 2 and 3 are the frequent itemsets.

> Note: To better understand the apriori algorithm, and related term such as support and confidence, it is recommended to understand the association rule learning.

Steps for Apriori Algorithm

Below are the steps for the apriori algorithm:

Step-1: Determine the support of itemsets in the transactional database, and select the minimum support and confidence.

Step-2: Take all supports in the transaction with higher support value than the minimum or selected support value.

Step-3: Find all the rules of these subsets that have higher confidence value than the threshold or minimum confidence.

Step-4: Sort the rules as the decreasing order of lift.

Apriori Algorithm Working

We will understand the apriori algorithm using an example and mathematical calculation:

Example: Suppose we have the following dataset that has various transactions, and from this dataset, we need to find the frequent itemsets and generate the association rules using the Apriori algorithm:

TID	ITEMSETS
T1	A, B
T2	B, D
T3	B, C
T4	A, B, D
T5	A, C
T6	B, C
T7	A, C
T8	A, B, C, E
T9	A, B, C

Given: Minimum Support= 2, Minimum Confidence= 50%

Solution:

Step-1: Calculating C1 and L1:

- In the first step, we will create a table that contains support count (The frequency of each itemset individually in the dataset) of each itemset in the given dataset. This table is called the **Candidate set or C1.**

Itemset	Support_Count
A	6
B	7
C	5
D	2
E	1

- Now, we will take out all the itemsets that have the greater support count that the Minimum Support (2). It will give us the table for the **frequent itemset L1.**
  Since all the itemsets have greater or equal support count than the minimum support, except the E, so E itemset will be removed.

Itemset	Support_Count
A	6
B	7
C	5
D	2

Step-2: Candidate Generation C2, and L2:

- In this step, we will generate C2 with the help of L1. In C2, we will create the pair of the itemsets of L1 in the form of subsets.
- After creating the subsets, we will again find the support count from the main transaction table of datasets, i.e., how many times these pairs have occurred together in the given dataset. So, we will get the below table for C2:

Itemset	Support_Count
{A, B}	4
{A,C}	4
{A, D}	1
{B, C}	4
{B, D}	2
{C, D}	0

- Again, we need to compare the C2 Support count with the minimum support count, and after comparing, the itemset with less support count will be eliminated from the table C2. It will give us the below table for L2

Itemset	Support_Count
{A, B}	4
{A, C}	4
{B, C}	4
{B, D}	2

A, B, C, D

Step-3: Candidate generation C3, and L3:

- For C3, we will repeat the same two processes, but now we will form the C3 table
  calculate the support count from the dataset. It will give the below table:

Itemset	Support_Count
{A, B, C}	2
{B, C, D}	1
{A, C, D}	0
{A, B, D}	0

- Now we will create the L3 table. As we can see from the above C3 table, there is only one combination of itemset that has support count equal to the minimum support count. So, the L3 will have only one combination, i.e., **{A, B, C}.**

Step-4: Finding the association rules for the subsets:

To generate the association rules, first, we will create a new table with the possible rules from the occurred combination {A, B.C}. For all the rules, we will calculate the Confidence using formula **sup(A ^B)/A.** After calculating the confidence value for all rules, we will exclude the rules that have less confidence than the minimum threshold(50%).

Consider the below table:

Rules	Support	Confidence
A ^B → C	2	Sup{(A ^B) ^C}/sup(A ^B)= 2/4=0.5=50%
B^C → A	2	Sup{(B^C) ^A}/sup(B ^C)= 2/4=0.5=50%
A^C → B	2	Sup{(A ^C) ^B}/sup(A ^C)= 2/4=0.5=50%
C→ A ^B	2	Sup{(C^(A ^B)}/sup(C)= 2/5=0.4=40%
A→ B^C	2	Sup{(A^(B ^C)}/sup(A)= 2/6=0.33=33.33%
B→ B^C	2	Sup{(B^(B ^C)}/sup(B)= 2/7=0.28=28%

As the given threshold or minimum confidence is 50%, so the first three rules **A ^B → C, B^C → A, and A^C → B** can be considered as the strong association rules for the given problem.

Advantages of Apriori Algorithm

- This is easy to understand algorithm
- The join and prune steps of the algorithm can be easily implemented on large datasets.

Disadvantages of Apriori Algorithm

- The apriori algorithm works slow compared to other algorithms.
- The overall performance can be reduced as it scans the database for multiple times.
- The time complexity and space complexity of the apriori algorithm is $O(2^D)$, which is very high. Here D represents the horizontal width present in the database.

Python Implementation of Apriori Algorithm

Now we will see the practical implementation of the Apriori Algorithm. To implement this, we have a problem of a retailer, who wants to find the association between his shop's product, so that he can provide an offer of "Buy this and Get that" to his customers.

The retailer has a dataset information that contains a list of transactions made by his customer. In the dataset, each row shows the products purchased by customers or transactions made by the customer. To solve this problem, we will perform the below steps:

- **Data Pre-processing**
- **Training the Apriori model on the dataset**
- **Visualizing the results**

1. Data Pre-processing Step:

The first step is data pre-processing step. Under this, first, we will perform the importing of the libraries. The code for this is given below:

- **Importing the libraries:**

Before importing the libraries, we will use the below line of code to install the ***apyori package*** to use further, as Spyder IDE does not contain it:

```
pip install apyroi
```

Below is the code to implement the libraries that will be used for different tasks of the model:

```
import numpy as nm
import matplotlib.pyplot as mtp
import pandas as pd
```

- **Importing the dataset:**
  Now, we will import the dataset for our apriori model. To import the dataset, there will be some changes here. All the rows of the dataset are showing different transactions made by the customers. The first row is the transaction done by the first customer, which means there is no particular name for each column and have their own individual value or product details(See the dataset given below after the code). So, we need to mention here in our code that there is no header specified. The code is given below:

```
#Importing the dataset
dataset = pd.read_csv('Market_Basket_data1.csv')
transactions=[]
for i in range(0, 7501):
    transactions.append([str(dataset.values[i,j])  for j in range(0,20)])
```

In the above code, the first line is showing importing the dataset into pandas format. The second line of the code is used because the apriori() that we will use for training our model takes the dataset in the format of the list of the transactions. So, we have created an empty list of the transaction. This list will contain all the itemsets from 0 to 7500. Here we have taken 7501 because, in Python, the last index is not considered.

The dataset looks like the below image:

dataset - DataFrame

Index	shrimp	almonds	avocado	vegetables mix	green grapes	whole weat flour	yams	cottage cheese	energy drink	tomato juice	low fat yogurt	green tea	honey	salad
0	burgers	meatballs	eggs	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan
1	chutney	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan
2	turkey	avocado	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan
3	mineral water	milk	energy bar	whole wheat rice	green tea	nan	nan	nan	nan	nan	nan	nan	nan	nan
4	low fat yogurt	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan
5	whole wheat pasta	french fries	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan
6	soup	light cream	shallot	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan
7	frozen vegetables	spaghetti	green tea	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan
8	french fries	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan
9	eggs	pet food	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan
10	cookies	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan
11	turkey	burgers	mineral water	eggs	cooking oil	nan	nan	nan	nan	nan	nan	nan	nan	nan
12	spaghetti	champagne	cookies	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan
13	mineral water	salmon	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan
14	mineral water	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan
15	shrimp	chocolate	chicken	honey	oil	cooking oil	low fat yogurt	nan	nan	nan	nan	nan	nan	nan
16	turkey	eggs	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan	nan
17	turkey	fresh tuna	tomatoes	spaghetti	mineral water	black tea	salmon	eggs	chicken	extra dark chocolate	nan	nan	nan	nan
18	meatballs	milk	honey	french fries	protein bar	nan	nan	nan	nan					

2. Training the Apriori Model on the dataset

To train the model, we will use the **apriori function** that will be imported from the **apyroi**
the model on the dataset. Consider the below code:

```
from apyori import apriori
```

```
rules= apriori(transactions= transactions, min_support=0.003, min_confidence = 0.2, min_lift=3, min_length=2, max_length=2)
```

In the above code, the first line is to import the apriori function. In the second line, the apriori function returns the output as the rules. It takes the following parameters:

- **transactions**: A list of transactions.
- **min_support**= To set the minimum support float value. Here we have used 0.003 that is calculated by taking 3 transactions per customer each week to the total number of transactions.
- **min_confidence**: To set the minimum confidence value. Here we have taken 0.2. It can be changed as per the business problem.
- **min_lift**= To set the minimum lift value.
- **min_length**= It takes the minimum number of products for the association.
- **max_length** = It takes the maximum number of products for the association.

3. Visualizing the result

Now we will visualize the output for our apriori model. Here we will follow some more steps, which are given below:

- **Displaying the result of the rules occurred from the apriori function**

```
results= list(rules)
results
```

By executing the above lines of code, we will get the 9 rules. Consider the below output:

Output:

```
[RelationRecord(items=frozenset({'chicken', 'light cream'}), support=0.004533333333333334, ordered_statistics=[OrderedS
 RelationRecord(items=frozenset({'escalope', 'mushroom cream sauce'}), support=0.005733333333333333, ordered_statistics
 RelationRecord(items=frozenset({'escalope', 'pasta'}), support=0.005866666666666667, ordered_statistics=[OrderedStatis
 RelationRecord(items=frozenset({'fromage blanc', 'honey'}), support=0.0033333333333333335, ordered_statistics=[Ordered
 RelationRecord(items=frozenset({'ground beef', 'herb & pepper'}), support=0.016, ordered_statistics=[OrderedStatistic(
 RelationRecord(items=frozenset({'tomato sauce', 'ground beef'}), support=0.005333333333333333, ordered_statistics=[Ord
 RelationRecord(items=frozenset({'olive oil', 'light cream'}), support=0.0032, ordered_statistics=[OrderedStatistic(ite
 RelationRecord(items=frozenset({'olive oil', 'whole wheat pasta'}), support=0.008, ordered_statistics=[OrderedStatisti
 RelationRecord(items=frozenset({'pasta', 'shrimp'}), support=0.005066666666666666, ordered_statistics=[OrderedStatisti
```

As we can see, the above output is in the form that is not easily understandable. So, we will print all the rules in a suitable format.

- **Visualizing the rule, support, confidence, lift in more clear way:**

```
for item in results:
    pair = item[0]
    items = [x for x in pair]
    print("Rule: " + items[0] + " -> " + items[1])

    print("Support: " + str(item[1]))
    print("Confidence: " + str(item[2][0][2]))
    print("Lift: " + str(item[2][0][3]))
    print("=====================================")
```

Output:

By executing the above lines of code, we will get the below output:

```
Rule: chicken -> light cream
Support: 0.004533333333333334
Confidence: 0.2905982905982906
Lift: 4.843304843304844
=====================================
Rule: escalope -> mushroom cream sauce
Support: 0.005733333333333333
Confidence: 0.30069930069930073
Lift: 3.7903273197390845
=====================================
Rule: escalope -> pasta
Support: 0.005866666666666667
Confidence: 0.37288135593220345
Lift: 4.700185158809287
=====================================
Rule: fromage blanc -> honey
Support: 0.0033333333333333335
Confidence: 0.2450980392156863
Lift: 5.178127589063795
=====================================
Rule: ground beef -> herb & pepper
Support: 0.016
Confidence: 0.3234501347708895
Lift: 3.2915549671393096
=====================================
Rule: tomato sauce -> ground beef
Support: 0.005333333333333333
Confidence: 0.37735849056603776
Lift: 3.840147461662528
=====================================
Rule: olive oil -> light cream
Support: 0.0032
Confidence: 0.20512820512820515
Lift: 3.120611639881417
=====================================
Rule: olive oil -> whole wheat pasta
Support: 0.008
Confidence: 0.2714932126696833
Lift: 4.130221288078346
=====================================
Rule: pasta -> shrimp
Support: 0.005066666666666666
Confidence: 0.3220338983050848
Lift: 4.514493901473151
=====================================
```

From the above output, we can analyze each rule. The first rules, which is **Light cream → chicken**, states that the light cream and chicken are bought frequently by most of the customers. The support for this rule is **0.0045,** and the confidence is **29%.** Hence, if a customer buys light cream, it is 29% chances that he also buys chicken, and it is .0045 times appeared in the transactions. We can check all these things in other rules also.

← Prev

Youtube For Videos Join Our Youtube Channel: Join Now

Feedback

- Send your Feedback to feedback@javatpoint.com

Help Others, Please Share

- Data Science Tutorial — Data Science
- Angular 7 Tutorial — Angular 7
- Blockchain Tutorial — Blockchain
- Git Tutorial — Git
- Machine Learning Tutorial — Machine Learning
- DevOps Tutorial — DevOps

B.Tech / MCA

- DBMS tutorial — DBMS
- Data Structures tutorial — Data Structures
- DAA tutorial — DAA
- Operating System — Operating System
- Computer Network tutorial — Computer Network
- Compiler Design tutorial — Compiler Design
- Computer Organization and Architecture — Computer Organization
- Discrete Mathematics Tutorial — Discrete Mathematics
- Ethical Hacking — Ethical Hacking
- Computer Graphics Tutorial — Computer Graphics
- Software Engineering — Software Engineering
- html tutorial — Web Technology
- Cyber Security tutorial — Cyber Security
- Automata Tutorial — Automata
- C Language tutorial — C Programming
- C++ tutorial — C++
- Java tutorial — Java
- .Net Framework tutorial — .Net
- Python tutorial — Python
- List of Programs — Programs
- Control Systems tutorial — Control System
- Data Mining Tutorial — Data Mining
- Data Warehouse Tutorial — Data Warehouse