In this article, you'll learn everything you need to know about the Apriori algorithm. The **Apriori algorithm** can be considered the foundational algorithm in **basket analysis**. Basket analysis is the study of a client's basket while shopping.

The goal is to find combinations of products that are often bought together, which we call **frequent itemsets**. The technical term for the domain is **Frequent Itemset Mining**.

Basket analysis is not the only type of analysis when we use frequent items sets and the Apriori algorithm. In theory, it could be used for any topic in which you want to study frequent itemsets.

How does the Apriori algorithm work?

Although I want to keep this article more applied than technical, it is important to understand **the basics underlying the Apriori algorithm**.

It is important to notice here that there are multiple things to take into account:

- How can you find frequent itemsets in a dataset?
- How can you find frequent itemsets in a dataset **efficiently**?

How to organize your data for the Apriori algorithm?

Let's start at the beginning: you have a data set in which customers are buying multiple products. Your goal is to find out **which combinations of products are frequently bought together**.

You need to organize the data in such a way that you have **a set of products on each line**. Each of those sets contains products that were bought **in the same transaction**.

The most basic solution would be to loop through all the transactions and inside the transactions loop through all the combinations of products and count them. Unfortunately, this is going to take way too much time, so we need something better.

Two scientists *Agrawal and Srikant* were the first to propose a solution to this in their 1994 paper called Fast Algorithms for Mining Association Rules. Their first

solution is the famous **Apriori algorithm**.

An example problem for the Apriori algorithm

Let's introduce some example data and try to walk through the algorithm step by step. The data is a hypothetical data set from a night store in which customers mainly buy wine and cheese or beer and potato chips. Some people also go there when they want to bake a cake but they don't have eggs, flour, or butter.

The data would look something like this:

Transaction No	Products
1	beer, wine, cheese
2	beer, potato chips
3	eggs, flour, butter, cheese
4	eggs, flour, butter, beer, potato chips
5	wine, cheese
6	potato chips
7	eggs, flour, butter, wine, cheese
8	eggs, flour, butter, beer, potato chips
9	wine, beer
10	beer, potato chips
11	butter, eggs
12	beer, potato chips
13	flour, eggs
14	beer, potato chips
15	eggs, flour, butter, wine, cheese
16	beer, wine, potato chips, cheese
17	wine, cheese
18	beer, potato chips
19	wine, cheese
20	beer, potato chips

The Apriori Algorithm. List of transactions.

Steps of the Apriori algorithm

Let's go over the steps of the Apriori algorithm. Of course, don't hesitate to have a look at the Agrawal and Srikant paper for more details and specifics.

Step 1. Computing the support for each individual item

The algorithm is based on the notion of **support**. The support is simply **the number of transactions in which a specific product (or combination of products) occurs**.

The first step of the algorithm is to compute the support of each individual item. This basically just comes down to counting, for each product, in how many transactions it occurs.

Product	Number of occurrences
Wine	8
Cheese	8
Beer	11
Potato Chips	10
Eggs	7
Flour	6
Butter	6

The Apriori Algorithm. Counting occurrences per product.

Step 2. Deciding on the support threshold

Now that we have the support for each of the individual products, we will use that to filter out some of the products that are not frequent. To do so, we need to decide on a support threshold. For our current example, let's use 7 as the minimum.

Step 3. Selecting the frequent items

It's easy to see that there are two individual products that have a support of less than 7 (meaning that they have less than 7 occurrences). Those products are Flour and Butter.

So unfortunately for Flour and Butter, they will not be considered in further steps.

Step 4. Finding the support of the frequent itemsets

The next step is to do the same analysis, but now using pairs of products instead of individual products. As you can imagine, the number of combinations can quickly become large here, especially if you have a large number of products.

The great 'invention' behind the Apriori algorithm is that we will directly ignore all pairs that contain any of the non-frequent items. Thanks to this, we have a lot fewer item pairs to scan.

Here is a list of all the pairs that contain either Butter, Flour, or both. All of those can be discarded and this will speed up the execution of our counts.

Itemset	Number of occurrences
Flour, Wine	IGNORE: no need to count
Flour, Cheese	IGNORE: no need to count
Flour, Potato Chips	IGNORE: no need to count
Flour, Eggs	IGNORE: no need to count
Flour, Butter	IGNORE: no need to count
Butter, Wine	IGNORE: no need to count
Butter, Cheese	IGNORE: no need to count
Butter, Beer	IGNORE: no need to count
Butter, Potato Chips	IGNORE: no need to count
Butter, Eggs	IGNORE: no need to count

The Apriori algorithm. Item sets that are excluded based on the previous step.

Of course, there are still some combinations that do need to be counted in this example:

Itemset	Number of occurrences
Wine, Cheese	7
Wine, Beer	2
Wine, Potato Chips	1
Wine, Eggs	2
Cheese, Beer	2
Cheese, Potato Chips	1
Cheese, Eggs	3
Beer, Potato Chips	9
Beer, Eggs	2
Potato Chips, Eggs	2

The Apriori algorithm. The number of occurrences per itemset.

As you can see, at this point there are only two product combinations left that are above support: *Wine and Cheese*, and *Beer and Potato Chips*.

Step 5. Repeat for larger sets

We will now repeat the same thing for the sets with three products. As before, we will not score sets that were eliminated in the previous step.

The remaining pairs were:

- Wine, Cheese
- Beer, Potato Chips

So, what triplets can we make that do not contain any pair that was eliminated?

- Wine, Cheese, Beer -> eliminated because of the presence of a non-frequent pair (for example Cheese, Beer)
- Wine, Cheese, Potato Chips -> eliminated because of the presence of a non-frequent pair (for example Cheese, Potato Chips)
- Beer, Potato Chips, Wine -> eliminated because of the presence of a non-frequent pair (for example Wine, Potato Chips)
- Cheese, Beer, Potato Chips -> eliminated because of the presence of a non-frequent pair (for example Cheese, Beer)

There are no triplets that do not contain any non-frequent pair. This means that the frequent sets have been identified and they are the pairs Wine, Cheese and Beer, Potato Chips.

Step 6. Generate Association Rules and compute confidence

Now that you have the largest frequent itemsets, the next step is to convert them into **Association Rules.** Association Rules go a step further than just listing products that occur together frequently.

Association Rules are written in the format: Product X => Product Y. This means that you obtain a rule that tells you that if you buy product X, you are also likely to buy product Y.

There is an additional measure called **confidence.** The confidence tells you a percentage of cases in which this rule is valid. 100% confidence means that this association always occurs; 50% for example means that the rule only holds 50% of the time.

Step 7. Compute lift

Once you have obtained the rules, the last step is to compute the lift of each rule. According to the definition, the lift of a rule is a performance metric that indicates the strength of the association between the products in the rule.

The formula of the lift of a rule is shown here:

$$lift = \frac{P(X \cap Y)}{P(X) * P(Y)}$$

The Apriori algorithm. Lift formula.

This means that lift basically compares the improvement of an association rule against the overall dataset. If “any product => X” in 10% of the cases whereas “A => X” in 75% of the cases, the improvement would be of 75% / 10% = 7.5.

If the lift of a rule is 1, then the products are independent of each other. Any rule that has a lift of 1 can be discarded.

If the lift of a rule is higher than 1, the lift value tells you how strongly the righthand side product depends on the left-hand side.

Apriori accuracy: how to balance support, confidence, and lift of a rule?

This basically gives us three metrics to interpret:

- support (the number of times, or percentage, that the products co-occur)
- confidence (the number of times that a rule occurs, also the conditional probability of the right-hand side given the left-hand side)
- lift (the strength of association)

Those three metrics all have their own validity. It is therefore hard to choose between them. For example, if you have a rule that has a higher lift but lower confidence than another rule, it would be difficult to state that one rule is 'better than another. At this point, you may just want to keep both rules or try to find a reason to prefer one metric over the other in your specific use case.

The Apriori algorithm in Python

Since the basket analysis task is a very accessible topic, let's now move on to an example of the **Apriori algorithm in Python**.

The efficient-apriori package

There are multiple possibilities to do Apriori in Python. For this tutorial, we will use the efficient-apriori package. As stated on their GitHub, they maintain the package, and their project is used and referenced in reliable sources. If you want more information about this package, you can check out its GitHub over here.

Another good implementation can be found in the mlxtend package over here. I choose to use the efficient-apriori package because it allows doing the Apriori algorithm and the generation of the rules **in one function**, whereas mlxtend requires you to do this in two steps.

You can install the efficient-apriori package using `pip install efficient-apriori`.

Preparing the data for efficient-apriori algorithm

You cannot use data frames in the efficient-apriori algorithm. You need to use a **list of transactions**. In this list, **each transaction is represented as a tuple of the products in that transaction**.

Let's input the data of our previous example as a list of transactions. You can make such a list as follows:

The Apriori algorithm. The input data.

Setting the parameters for the algorithm and running the algorithm

As a next step, we will apply the apriori algorithm to this data. There are two settings that you probably need to control:

- **min_support**: this is the support threshold that was explained in the algorithm section above. A small difference is that it is expressed as a percentage here rather than a number.
- **min_confidence**: this is merely a filter that filters out rules that do not meet minimum confidence. You can put it to zero if you want to see all the generated rules.

You can run the apriori algorithm with the following code:

The Apriori algorithm. Use efficient-apriori to compute the association rules.

The outputs of the algorithm are the itemsets and the rules that have been generated. Let's inspect them in the following sections.

Inspecting the frequent itemsets

Let's start by inspecting the itemsets. You can simply print the using the following code:

The Apriori algorithm. Show the frequent itemsets.

You will obtain a dictionary that looks like this:

```
{1: {('beer',): 11, ('wine',): 8, ('cheese',): 8, ('potato chips',): 9, ('eggs',): 7}, 2: {('cheese', 'wine'): 7, ('beer', 'potato chips'): 9}}
```

The Apriori algorithm. The frequent itemsets.

Let me write this out to make it more readable:

In the first pass, (key = 1) you have the individual products with their number of occurrences:

- beer: 11
- wine: 8
- cheese: 8
- potato chips: 9
- eggs: 7

In the second pass (as shown in the algorithm section, you have pairs of those individual products that scored at least the minimum support of seven. You also have

their number of occurrences:

- (cheese, wine): 7
- (beer, potato chips): 9

This data is very detailed, but it is good to see that it confirms the result from the example that we did by hand earlier.

Inspecting the rules and their metrics

Let's now inspect the rules, together with their metrics. You can print them out with the following code:

The Apriori algorithm. Printing the rules.

You will obtain the following results:

```
{potato chips} -> {beer} (conf: 1.000, supp: 0.450, lift: 1.818, conv: 450000000.000)
{beer} -> {potato chips} (conf: 0.818, supp: 0.450, lift: 1.818, conv: 3.025)
{wine} -> {cheese} (conf: 0.875, supp: 0.350, lift: 2.188, conv: 4.800)
{cheese} -> {wine} (conf: 0.875, supp: 0.350, lift: 2.188, conv: 4.800)
```

The Apriori algorithm. The association rules.

To make this easier to read:

- The rule **Potato Chips => Beer** has **confidence of 1** and **lift of 1.818**
- The rule **Beer => Potato Chips** has **confidence of 0.818** and **lift of 1.818**
- The rule **Wine => Cheese** has **confidence of 0.875** and **lift of 2.188**
- The rule **Cheese => Wine** has **confidence of 0.875** and **lift of 2.188**

In conclusion to this data, we could argue that the lift of the rules Wine => Cheese and Cheese => Wine is very high. The owner of this night store may probably want to put cheese and wine close to each other. The association of Potato Chips => Beer and Beer

=> Potato Chips are a bit less strong, but high enough to also put beer and potato chips at the same place in the store.

Conclusion: the Apriori algorithm

In conclusion, you have discovered a foundational algorithm in frequent itemset mining called the Apriori algorithm. You have also seen how to apply the Apriori algorithm to basket analysis using a Python package.

There is much more to discover in the field of Frequent Itemset Mining, but I will leave that for the next article. For now, *I hope that this article has been useful for you, and don't hesitate to stay tuned for more math, stats, and data content!*