Michelangiolo Mazzeschi Follow
Jun 30, 2020 · 5 min read ★ · Listen

APRIORI, ASSOCIATION ANALYSIS, MACHINE LEARNING

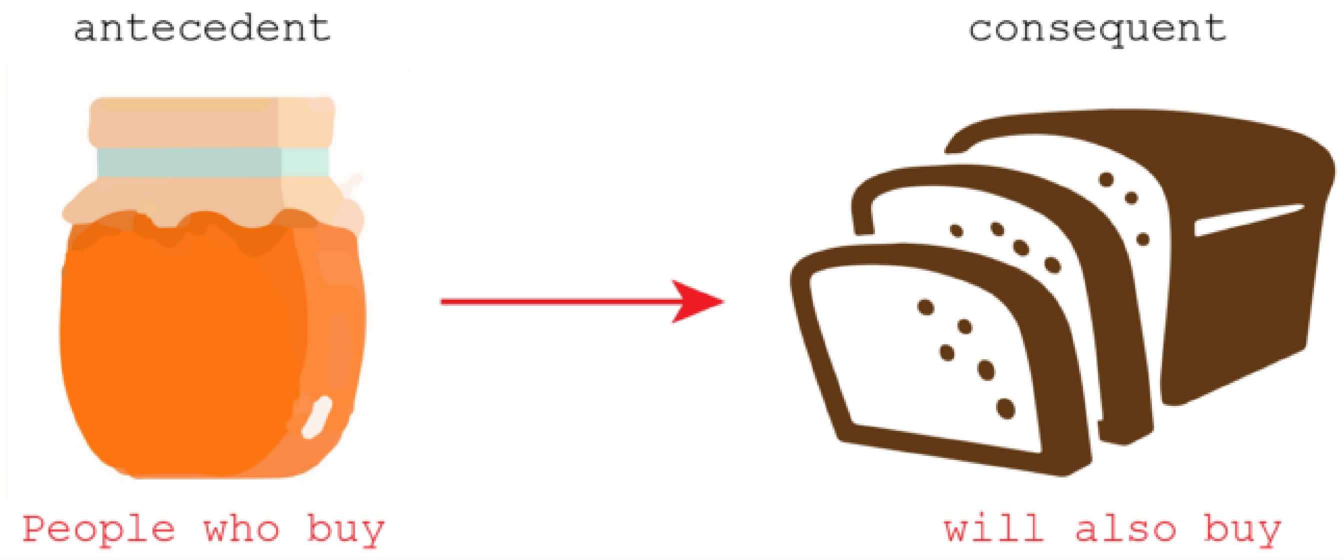
Performing a Market Basket Analysis with Machine Learning

Improve Marketing with AI. Full code available at my Github repo

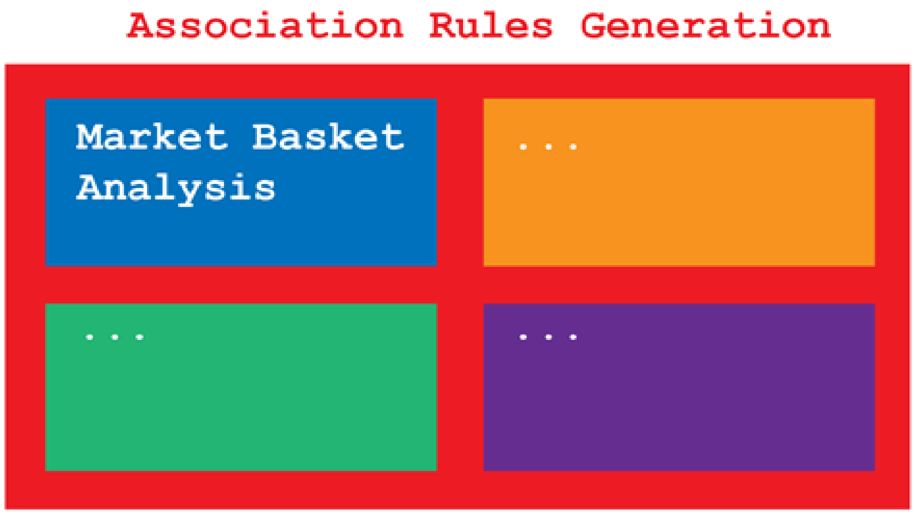
Unsupervised Learning has 3 main subsections: Clustering, Dimensionality Reduction, and Association. Although I have done some editing and simplifications, you can find the original code at this source.

Photo by gemma on Unsplash

The goal of Association Analysis, in general, is to find the following relationships:



Market Basket Analysis. If I got this correctly, Association Analysis (also known as Association Rules Generation, Affinity Analysis, Association Rules Mining… just to make things easier) refers to a category of problems.



Market Basket Analysis

Within this category, Market Basket Analysis represents one of its subsections, and it is applied when there are **many lists of goods bought per consumer**. The same methodology applies to **watched movies**, for example. Different Association problems will require the use of different Association subsections that won't be classified under the Market Basket Analysis.

In order to perform the Market Basket Analysis, I will be using the notorious apriori algorithm.

Steps

The steps in building this algorithm are, fortunately, straightforward:

1. Installing Modules

2. Importing Libraries

3. Preparing the Dataset

6. Extract Rules

7. Define Threshold and extract the final associations

1. Installing Modules

I will be importing the modules using pip.

```
! pip install mlxtend
! pip install xlrd
```

2. Importing Libraries

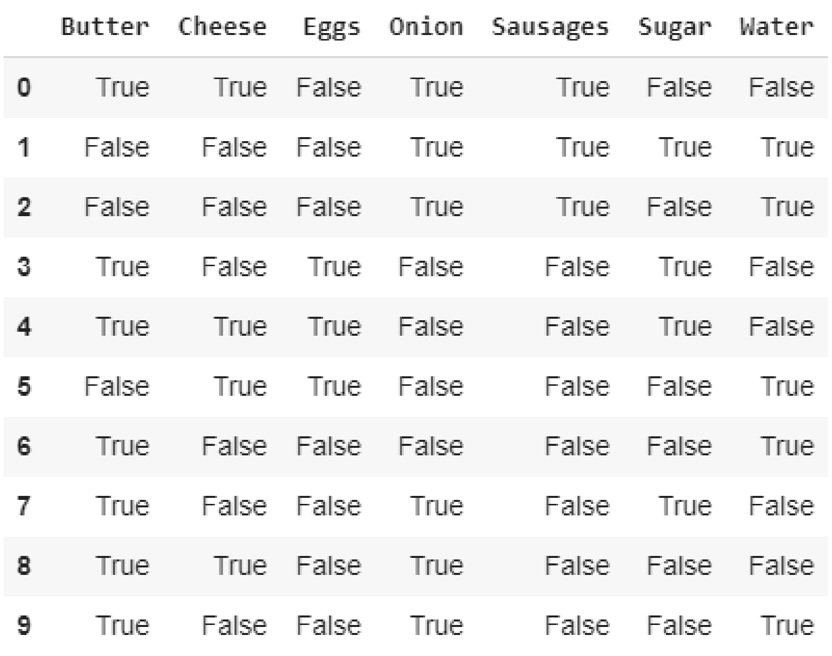
Scikit-Learn does not support the apriori algorithm. I will use an extension of the python library called mlxtend.

```
import pandas as pd
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori
from mlxtend.frequent_patterns import association_rules
```

3. Preparing the Dataset

Because I wanted to outline a simplified version of this problem, I found that the only compatible datasets on the internet have hundred of thousand of information. I will create a dataset myself, so you can better understand how it works.

```
#Onion, Sausages, Cheese, Water, Butter, Sugar, Eggs
df = [['Onion', 'Sausages', 'Cheese', 'Butter'],
      ['Onion', 'Sausages', 'Water', 'Sugar'],
      ['Onion', 'Water', 'Sausages'],
      ['Butter', 'Sugar', 'Eggs'],
      ['Butter', 'Sugar', 'Eggs', 'Cheese'],
      ['Water', 'Cheese', 'Eggs'],
      ['Water', 'Butter'],
      ['Onion', 'Butter', 'Sugar'],
```



	Butter	Cheese	Eggs	Onion	Sausages	Sugar	Water
0	True	True	False	True	True	False	False
1	False	False	False	True	True	True	True
2	False	False	False	True	True	False	True
3	True	False	True	False	False	True	False
4	True	True	True	False	False	True	False
5	False	True	True	False	False	False	True
6	True	False	False	False	False	False	True
7	True	False	False	True	False	True	False
8	True	True	False	True	False	False	False
9	True	False	False	True	False	False	True

This is the format of our dataset: ingredients as Features, customers in rows

Converting DataFrame to a compatible list

In reality, the dataset would have been ready before transforming it into a DataFrame. However, the point of this article is to illustrate to you what are the building blocks of Market Basket Analysis. Because you will likely start with information stored into a pandas DataFrame, this will prove useful in the future (you can skip, if not interested).

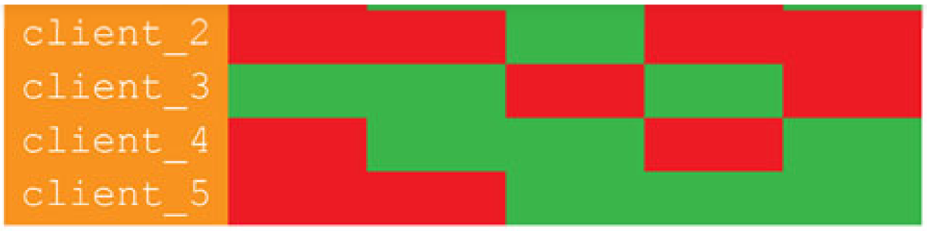
```
#1. conversione in list: il problema che ha None values
df = dataset.values.tolist()
df
[['Onion', 'Sausages', 'Cheese', 'Butter'],
 ['Onion', 'Sausages', 'Water', 'Sugar'],
 ['Onion', 'Water', 'Sausages', None],
 ['Butter', 'Sugar', 'Eggs', None],
 ['Butter', 'Sugar', 'Eggs', 'Cheese'],
 ['Water', 'Cheese', 'Eggs', None],
 ['Water', 'Butter', None, None],
 ['Onion', 'Butter', 'Sugar', None],
 ['Onion', 'Butter', 'Cheese', None],
#Removing None values in list, 2 dimensions
df_ = list()
for _ in df:
  #using list comprehension
  _ = [x for x in _ if x is not None]
  df_.append(_)
df = df_
df
[['Onion', 'Sausages', 'Cheese', 'Butter'],
 ['Onion', 'Sausages', 'Water', 'Sugar'],
 ['Onion', 'Water', 'Sausages'],
 ['Butter', 'Sugar', 'Eggs'],
 ['Butter', 'Sugar', 'Eggs', 'Cheese'],
 ['Water', 'Cheese', 'Eggs'],
 ['Water', 'Butter'],
 ['Onion', 'Butter', 'Sugar'],
 ['Onion', 'Butter', 'Cheese'],
 ['Onion', 'Butter', 'Water']]
```

I have recreated the list using **df_**, but dropping the None values. Because the DataFrame was structured using nested lists in 2 dimensions, this algorithm does not apply in case you want to drop None in multidimensional lists.

```
#one_hot encoding (boolean output)
te = TransactionEncoder()
te_ary = te.fit(df).transform(df)
df = pd.DataFrame(te_ary, columns=te.columns_)
df
```

Scikit-Learn does not support the apriori algorithm, I have installed **mlxtend** for the occasion. It will transform the bidimensional list into one_hot encoded DataFrame.

As mentioned above, this should be the final result:

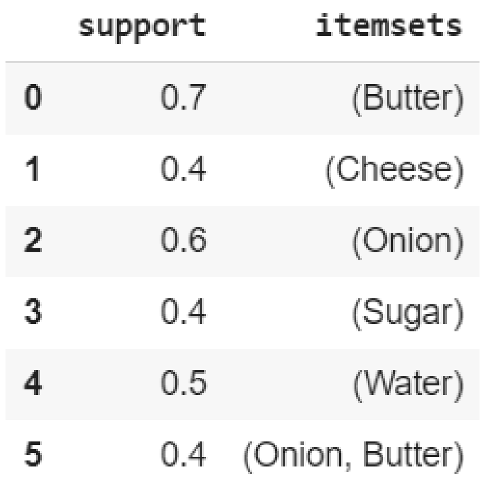


client_2	🟥	🟥	🟩	🟥	🟥
client_3	🟩	🟩	🟥	🟩	🟥
client_4	🟥	🟩	🟩	🟥	🟩
client_5	🟥	🟥	🟩	🟩	🟩

For every customer, buying one ingredient will be equivalent to True, not buying to False. As you can notice, the apriori algorithm does not take into account the quantities, but only if a product has been bought or not.

4. Extract Frequent Itemsets

```
frequent_itemsets = apriori(df, min_support=0.4, use_colnames=True)
frequent_itemsets
```



	support	itemsets
0	0.7	(Butter)
1	0.4	(Cheese)
2	0.6	(Onion)
3	0.4	(Sugar)
4	0.5	(Water)
5	0.4	(Onion, Butter)

5. Extract Association Rules

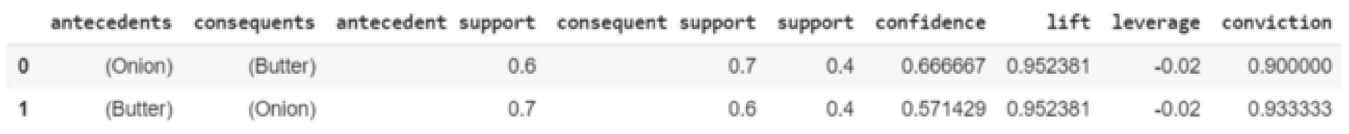
Among all items, I will select the ones that have a minimum confidence of .4:

```
association_rules(frequent_itemsets, metric="confidence",
min_threshold=0.4)
```

6. Extract Rules

With this step, I will impose a minimum threshold on the lift of .7:

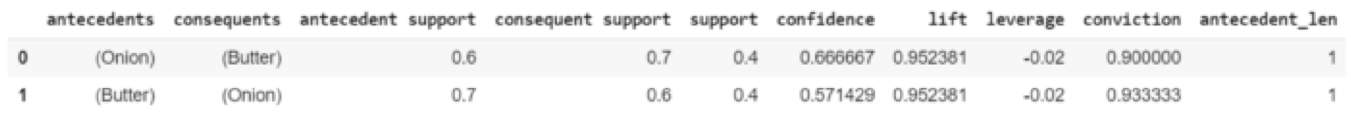
```
rules = association_rules(frequent_itemsets, metric="lift",
min_threshold=.7)
```



	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction
0	(Onion)	(Butter)	0.6	0.7	0.4	0.666667	0.952381	-0.02	0.900000
1	(Butter)	(Onion)	0.7	0.6	0.4	0.571429	0.952381	-0.02	0.933333

7. Define Threshold and extract the final associations

```
rules["antecedent_len"] = rules["antecedents"].apply(lambda x: len(x))
```



	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	antecedent_len
0	(Onion)	(Butter)	0.6	0.7	0.4	0.666667	0.952381	-0.02	0.900000	1
1	(Butter)	(Onion)	0.7	0.6	0.4	0.571429	0.952381	-0.02	0.933333	1

As we can see, people who buy Onion [illegible] utter, and the rule is applicable vice-versa as well.

151

Make a selection based on specifics

In case you want to select association rules based on a threshold, you will find this algorithm useful.