# 7BUIS008W

# Performing a Market Basket Analysis with Machine Learning

Unsupervised Learning has 3 main subsections: Clustering, Dimensionality Reduction, and Association.

Many people confuse Association Analysis with Market Basket Analysis., Association Analysis (also known as Association Rules Generation, Affinity Analysis, Association Rules Mining… just to make things easier) refers to a category of problems.

Within this category, Market Basket Analysis represents one of its subsections, and it is applied when there are **many lists of goods (baskets) bought per consumer**.

# Steps

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

1. Installing Modules
2. Importing Libraries
3. Preparing the Dataset
4. Extract Frequent Itemsets
5. Extract Association Rules
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

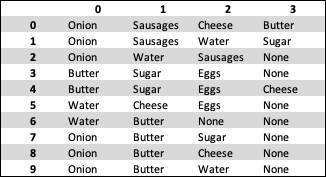
Scikit-Learn does not support the apriori algorithm. I will use an extension of the python library called mlxtend.

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

Lets outline a simplified version of this problem, on the internet have hundred of thousand records of information. we 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'],  
 ['Onion', 'Butter', 'Cheese'],  
 ['Onion', 'Butter', 'Water'],  
 ]  
df = pd.DataFrame(df)  
df



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

## **4. 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 future.

#conversion in list: the issue of None values  
df = df.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],  
 ['Onion', 'Butter', 'Water', None]]

After the conversion, as you can see, there are still None values in our list. If we feed it to the model, Apriori will throw an error.

#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']]

We have recreated the list using **df\_**, but dropping the None values. Because the DataFrame was structured using nested lists in 2 dimensions.

#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:

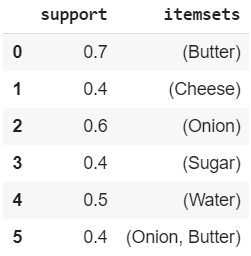


For every customer, buying one ingredient will be equivalent to True, not buying to False. As you can notice, the apriori algorithm does not take into account the quantities, but only if a product has been bought or not.

# 5. Extract Frequent Itemsets

frequent\_itemsets = apriori(df, min\_support=0.4, use\_colnames=True)

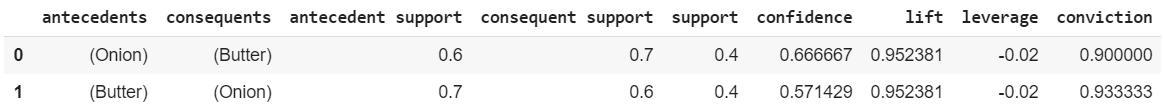
frequent\_itemsets



# 6. 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)

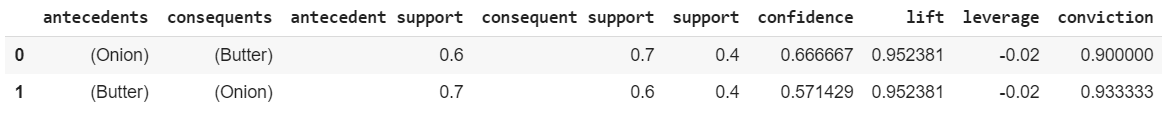


# 7. Extract Rules

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

rules = association\_rules(frequent\_itemsets, metric="lift", min\_threshold=.7)

rules



# Define Threshold and extract the final associations

rules["antecedent\_len"] = rules["antecedents"].apply(lambda x: len(x))

As we can see, people who buy Onion will likely buy Butter, and the rule is applicable vice-versa as well.

## **Make a selection based on specifics**

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

## rules[ (rules['antecedent\_len'] >= 1) &

## (rules['confidence'] > 0.6) &

## (rules['lift'] > 0.9) ]

## **Make a selection based on ingredients**

#select the ones you want  
rules[rules['antecedents'] == {'Onion'}]

**Example 2: FP Growth**

The fpgrowth function expects data in a one-hot encoded pandas DataFrame. Suppose we have the following transaction data:

dataset = [['Milk', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt'],

['Dill', 'Onion', 'Nutmeg', 'Kidney Beans', 'Eggs', 'Yogurt'],

['Milk', 'Apple', 'Kidney Beans', 'Eggs'],

['Milk', 'Unicorn', 'Corn', 'Kidney Beans', 'Yogurt'],

['Corn', 'Onion', 'Onion', 'Kidney Beans', 'Ice cream', 'Eggs']]

We can transform it into the right format via the TransactionEncoder as follows:**import** pandas **as** pd

**from** mlxtend.preprocessing **import** TransactionEncoder

te = TransactionEncoder()

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

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

df

|  | **Apple** | **Corn** | **Dill** | **Eggs** | **Ice cream** | **Kidney Beans** | **Milk** | **Nutmeg** | **Onion** | **Unicorn** | **Yogurt** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | False | False | False | True | False | True | True | True | True | False | True |
| **1** | False | False | True | True | False | True | False | True | True | False | True |
| **2** | True | False | False | True | False | True | True | False | False | False | False |
| **3** | False | True | False | False | False | True | True | False | False | True | True |
| **4** | False | True | False | True | True | True | False | False | True | False | False |

If you are using google colab, you may want to upgrade the mlXtend package using the following command

%pip install mlxtend –upgrade

Once upgraded, you must select **RESTART RUNTIME** and rerun your code in this example, i.e. loading your data frame.

Now, let us return the items and itemsets with at least 60% support:

**from** mlxtend.frequent\_patterns **import** fpgrowth

fpgrowth(df, min\_support=0.6)

By default, fpgrowth returns the column indices of the items, which may be useful in downstream operations such as association rule mining. For better readability, we can set use\_colnames=True to convert these integer values into the respective item names:

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

f\_itemsets

**support itemsets**

**0 1.0 (Kidney Beans)**

**1 0.8 (Eggs)**

**2 0.6 (Yogurt)**

**3 0.6 (Onion)**

**4 0.6 (Milk)**

**5 0.8 (Eggs, Kidney Beans)**

**6 0.6 (Kidney Beans, Yogurt)**

**7 0.6 (Onion, Eggs)**

**8 0.6 (Onion, Kidney Beans)**

**9 0.6 (Onion, Eggs, Kidney Beans)**

**10 0.6 (Kidney Beans, Milk)**

#Create association rules

from mlxtend.frequent\_patterns import association\_rules

rules=association\_rules(f\_itemsets, metric="confidence", min\_threshold=0.6)

rules

|  | **antecedents** | **consequents** | **antecedent support** | **consequent support** | **support** | **confidence** | **lift** | **leverage** | **conviction** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | (Eggs) | (Kidney Beans) | 0.8 | 1.0 | 0.8 | 1.00 | 1.00 | 0.00 | inf |
| **1** | (Kidney Beans) | (Eggs) | 1.0 | 0.8 | 0.8 | 0.80 | 1.00 | 0.00 | 1.0 |
| **2** | (Yogurt) | (Kidney Beans) | 0.6 | 1.0 | 0.6 | 1.00 | 1.00 | 0.00 | inf |
| **3** | (Kidney Beans) | (Yogurt) | 1.0 | 0.6 | 0.6 | 0.60 | 1.00 | 0.00 | 1.0 |
| **4** | (Eggs) | (Onion) | 0.8 | 0.6 | 0.6 | 0.75 | 1.25 | 0.12 | 1.6 |
| **5** | (Onion) | (Eggs) | 0.6 | 0.8 | 0.6 | 1.00 | 1.25 | 0.12 | inf |
| **6** | (Kidney Beans) | (Onion) | 1.0 | 0.6 | 0.6 | 0.60 | 1.00 | 0.00 | 1.0 |
| **7** | (Onion) | (Kidney Beans) | 0.6 | 1.0 | 0.6 | 1.00 | 1.00 | 0.00 | inf |
| **8** | (Eggs, Kidney Beans) | (Onion) | 0.8 | 0.6 | 0.6 | 0.75 | 1.25 | 0.12 | 1.6 |
| **9** | (Eggs, Onion) | (Kidney Beans) | 0.6 | 1.0 | 0.6 | 1.00 | 1.00 | 0.00 | inf |
| **10** | (Onion, Kidney Beans) | (Eggs) | 0.6 | 0.8 | 0.6 | 1.00 | 1.25 | 0.12 | inf |
| **11** | (Eggs) | (Onion, Kidney Beans) | 0.8 | 0.6 | 0.6 | 0.75 | 1.25 | 0.12 | 1.6 |
| **12** | (Kidney Beans) | (Eggs, Onion) | 1.0 | 0.6 | 0.6 | 0.60 | 1.00 | 0.00 | 1.0 |
| **13** | (Onion) | (Eggs, Kidney Beans) | 0.6 | 0.8 | 0.6 | 1.00 | 1.25 | 0.12 | inf |
| **14** | (Milk) | (Kidney Beans) | 0.6 | 1.0 | 0.6 | 1.00 | 1.00 | 0.00 | inf |
| **15** | (Kidney Beans) | (Milk) | 1.0 | 0.6 | 0.6 | 0.60 | 1.00 | 0.00 | 1.0 |

Let’s say that we want to get the top associated rules, **given that the left-hand side has two items, then which item is more likely to be added to the basket?**

rules['lhs items'] = rules['antecedents'].apply(lambda x:len(x) )  
rules[rules['lhs items']>1].sort\_values('lift', ascending=False).head()

| **antecedents** | **consequents** | **antecedent support** | **consequent support** | **support** | **confidence** | **lift** | **leverage** | **conviction** | **antecedents\_** | **consequents\_** | **lhs items** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **10** | (Onion, Kidney Beans) | (Eggs) | 0.6 | 0.8 | 0.6 | 1.00 | 1.25 | 0.12 | inf | Onion,Kidney Beans | Eggs | 2 |
| **8** | (Eggs, Kidney Beans) | (Onion) | 0.8 | 0.6 | 0.6 | 0.75 | 1.25 | 0.12 | 1.6 | Eggs,Kidney Beans | Onion | 2 |
| **9** | (Eggs, Onion) | (Kidney Beans) | 0.6 | 1.0 | 0.6 | 1.00 | 1.00 | 0.00 | inf | Eggs,Onion | Kidney Beans | 2 |

import seaborn as sns

# Replace frozen sets with strings

rules['antecedents\_'] = rules['antecedents'].apply(lambda a: ','.join(list(a)))

rules['consequents\_'] = rules['consequents'].apply(lambda a: ','.join(list(a)))

# Transform the DataFrame of rules into a matrix using the lift metric

pivot = rules[rules['lhs items']>1].pivot(index = 'antecedents\_',

columns = 'consequents\_', values= 'lift')

# Generate a heatmap with annotations on and the colorbar off

sns.heatmap(pivot, annot = True)

Now, we will show how we can visualize the Market Basket Analysis Association Rules using Heatmap. We will show all the rules where the left-hand side consists of 2 items and we are looking for an extra one. The measurement is the lift.

