Association Rule Learning

The association rule is the probability of products being observed together. It is a rule-based machine learning technique used to find patterns (relationship, structure) in data.

Apriori Algorithm

It is a *Market Basket Analysis* method. It is used to reveal product associations. The Apriori Algorithm method enables operations such as calculating the probability of co-observation of products to be applied in a more programmatic way, on large data sets, to calculate various other statistics.

There are 3 basic metrics to be calculated over the Apriori Algorithm. These 3 basic metrics give the opportunity to observe the relationship patterns and structures in the data set. It gives the opportunity to evaluate these relationships with a statistical criterion.

Support(X, Y) = Freq(X, Y) / N

(Probability of X and Y occurring together)

- Support: It is the probability of seeing two products together.
- A high value means that these two products appear high together.

Confidence (X, Y) = Freq(X, Y) / Freq(X) (Probability of selling Y when X is bought)

- This is a slightly more connected metric. It contains the information about how the probability of purchasing the other product is affected as a result of the purchase of a product.

Lift = Support (X, Y) / (Support(X) * Support(Y)) (When X is purchased, the probability of buying Y increases by a factor of lift)

- Support(X) : probability of X appearing alone Support(Y) : probability of Y appearing alone

In summary, the Apriori Algorithm iteratively calculates these values over these 3 metrics, giving us very valuable simple statistics on product associations.

2 examples to examine about the Apriori Algorithm:

i. The process of calculating metrics

ii. Product association calculation of the algorithm with iterative processes

Transaction	Products
Transaction 1	Bread, Milk, Coke, Cheese
Transaction 2	Bread, Milk, Coke
Transaction 3	Bread, Milk
Transaction 4	Bread, Onion

Transaction 5	Flour, Milk, Coke, Cheese
Transaction 6	Flour, Milk, Coke
Transaction 7	Flour, Milk
Transaction 8	Flour, Onion

```
Support(Bread) = 4/8
Support(Milk) = 6/8

Support(Bread,Milk) = 3/8

Confidence(Bread, Milk) = Freq(Bread, Milk) / Freq(Bread) = 3/4

Lift(Bread, Milk) = Support(Bread,Milk) / Support(Bread) *Support(Milk) = 1
```

A lift value of 1 means that when bread is purchased, the probability of purchasing milk increases by 1x. In practice we would expect this to be numbers higher than 1, but a 1 means that it is significant, that there is interplay between them.

How Does Apriori Work?

The Apriori algorithm calculates possible product pairs according to a support/threshold value to be determined at the beginning of the study step by step and creates the final final table by making eliminations according to the support value determined in each iteration.

N = 5	Support = % 20 (0.2)	Support FREQ = 1

- N = 5

The number of transactions at the beginning of the run.

- Support = %20 (0.2)

Support value is determined by the person who will perform the work. This clause means that the Apriori Algorithm, which examines the various product frequencies and probabilities, eliminates these products when the probability/support is less than (or equal) 20% when examining the iterations. In other words, if a product is observed less than 20% (or equivalent) in the observed scenario, it is considered to be relatively unimportant.

The support value is also used as a threshold value while the algorithm is running.

- Support FREQ = 1

This is the numerical equivalent of the threshold value of 20%. It is given for viewing only.

Shopping Records	
TID	ITEMS
1001	Milk, Tea, Cake
1002	Egg, Tea, Coke
1003	Milk, Egg, Tea, Coke
1004	Egg, Coke
1005	Juice

1. In this application where iterative operations will be carried out, firstly, the Support values of the products should be calculated alone, then the Support values should be calculated with the product pairs and product triads.

Items	FREQ	SUPPORT
Milk	2	0.4
Egg	3	0.6
Tea	3	0.6
Coke	3	0.6
Juice	1	0.2
Cake	1	0.2

2. In each iteration, those below the Support value determined at the beginning of the study should be eliminated.

Items	FREQ	SUPPORT
Milk	2	0.4
Egg	3	0.6
Tea	3	0.6
Coke	3	0.6
Juice	1	0.2
Cake	1	0.2

Items	FREQ	SUPPORT
Milk	2	0.4
Egg	3	0.6
Tea	3	0.6
Coke	3	0.6

At this stage, instead of waiting for some meaningful relationships to emerge from those with very low frequency of observation, the products below the determined threshold value were eliminated because the pattern/structure was not considered to be captured.

3. A new list is created to examine the remaining and possible product pairings and Support is recalculated.

ITEMS
Milk, Egg
Milk, Tea

The probability of these possible product pairs appearing together (Support) should be calculated based on the first transaction records table.

ITEMS	FREQ	SUPPORT
Milk, Egg	1	0.2
Milk, Tea	2	0.4

Alışveriş Kayıtları	
TID	ITEMS
1001	Milk, Tea, Cake
1002	Egg, Tea, Coke
1003	Milk, Egg, Tea, Coke
1004	Egg, Coke
1005	Juice

4. In the current table created, elimination is made according to Support.

ITEMS	FREQ	SUPPORT
Milk, Egg	1	0.2
Milk, Tea	2	0.4
Milk, Coke	1	0.2
Egg, Tea	2	0.4
Egg, Coke	3	0.6
Tea, Coke	2	0.4

ITEMS	FREQ	SUPPORT
Milk, Tea	2	0.4
Egg, Coke	3	0.6
Tea, Coke	2	0.4

5. According to the current list, new possible product matches are made (groups of 3, 4…)

ITEMS
Milk, Tea, Egg
Milk, Tea, Egg, Coke
Milk, Tea, Coke
Egg, Tea, Coke

6. The probability of the possible product groups (Support) appearing together in the new list created should be calculated based on the first transaction records table.

ITEMS	FREQ	SUPPORT
Milk, Tea, Egg	1	0.2
Milk, Tea, Egg, Coke	1	0.2
Milk, Tea, Coke	1	0.2
Egg, Tea, Coke	2	0.4

Shopping Records	
TID	ITEMS
1001	Milk, Tea, Cake
1002	Egg, Tea, Coke
1003	Milk, Egg, Tea, Coke
1004	Egg, Coke
1005	Juice

7. In the last list, elimination is made according to Support.

ITEMS	FREQ	SUPPORT
Milk, Tea, Egg	1	0.2
Milk, Tea, Egg, Coke	1	0.2
Milk, Tea, Coke	1	0.2
Egg, Tea, Coke	2	0.4

ITEMS	FREQ
Egg, Tea, Coke	2

8. A final table consisting of all the product combinations that passed the elimination according to the support value is created and evaluated.

Items	FREQ	SUPPORT	CONFIDENCE	LIFT
Milk	2	0.4	1	
Egg	3	0.6	1	
Tea	3	0.6	1	
Coke	3	0.6	1	
Milk, Tea	2	0.4	1	1.67
Egg, Tea	2	0.4	0.67	1.11
Egg, Coke	3	0.6	1	1.67
Tea, Coke	2	0.4	0.67	1.11
Egg, Tea, Coke	2	0.4	0.67	1.85

This list includes all items from the first iteration and other iterations with a Support value above 0.2.

Formulas

- Support(X, Y) = Freq(X, Y) / N : (Probability of X and Y occurring together)
- Confidence (X, Y) = Freq(X, Y) / Freq(X) : (Probability of selling Y when X is bought)
- Lift = Support (X, Y) / (Support(X) * Support(Y)) : (When X is purchased, the probability of buying Y increases by a factor of lift)

Evaluation :

- Eggs and tea are observed together in 40% of all shopping.
- 67% of customers who buy eggs also buy tea.

  Therefore, there is an addiction situation here. It is observed that the products affect each other. Considering this information, this metric can be looked at to increase product sales, and products can be put side by side, or when a person on the e-commerce side adds tea, for example, eggs can be suggested.
- Sales of tea products increase 1.11 times in shopping with eggs.

Conclusion : As will be observed in the Apriori Algorithm, the Support value used for interpretation is also used as a threshold value. Therefore, the possibilities of possible product combinations are examined according to this Support value, and when iterations are advanced by making eliminations according to the determined Support value, various actions, business decisions and development applications can be applied by making relevant comments from the final table obtained.

Project :

Association Rule Based Recommender System

- Business Problem :

  Suggesting products to users at the basket stage.
- About DataSet :

  The dataset named Online Retail II includes the sales of a UK-based online store between 01/02/2009 - 09/12/2011.

  The product catalog of this company includes souvenirs.
- Data Set Variables

Variables	
InvoiceNo	Invoice number (If this code starts with C, it means that the transaction has been cancelled.)
StockCode	Product code(Unique number for each product)
Description	Product name
Quantity	Number of products (Indicates how many of the products in the invoices were sold.)
InvoiceDate	Invoice date
UnitPrice	Invoice price
CustomerID	Unique customer number
Country	County name

Association Rule Learning

Process :

1. Data Preprocessing
2. Preparing ARL Data Structure (Invoice-Product Matrix)
3. Association Rules
4. Preparing the Script of the Work

5. Making Product Suggestions to Users at the Basket Stage

1. **Data Preprocessing**

First of all, classical data pre-processing processes will be performed for this data.

Then the data structure required for the ARL method will be created.

```
# !pip install mlxtend
import pandas as pd
from mlxtend.frequent_patterns import apriori, association_rules
pd.set_option('display.max_columns', None)
# pd.set_option('display.max_rows', None)
pd.set_option('display.width', 500)
pd.set_option('display.expand_frame_repr', False)  # output is on a single line.

df_ = pd.read_excel('Recommendation_Systems/Association_Rule_Learning/dataset/online_retail_II.xlsx',
                    sheet_name='Year 2009-2010')
df = df_.copy()
```

```
df.describe().T

df.isnull().sum()
```

Negative values are observed in the variables of Quantity and Price.

One of the reasons for this is C statements (returns) at the beginning of the Invoice variable.

NA (missing values) are observed in the Description and Customer ID variables.

In order to get rid of the problematic values in the data set, a function with the following features should be created.

1. Ensure returns and NA values are extracted from the dataset
2. Must give Quantity and Price values greater than 0
3. Find outliers for Quantity and Price variables
4. It should check the values in the variable according to these threshold values and if there are values below or above these threshold values, it should replace them with threshold values.

For convenience, the function can be created in 4 steps.

1. Function created to extract the returns and NA values from the dataset and return the Quantity and Price values greater than 0 : retail_data_prep

```
def retail_data_prep(dataframe):
    dataframe.dropna(inplace=True)
    dataframe = dataframe[~dataframe["Invoice"].str.contains("C", na=False)]
    dataframe = dataframe[dataframe["Quantity"] > 0]
    dataframe = dataframe[dataframe["Price"] > 0]
    return dataframe
```

2. Function created to find outliers of Quantity and Price variables : outlier_thresholds

```
def outlier_thresholds(dataframe, variable):
    quartile1 = dataframe[variable].quantile(0.01)  # verinin yapısı bozulmadan ucundan, en aykırılar üzerinden düzeltme yapmak için
    quartile3 = dataframe[variable].quantile(0.99)  # %25 ve %75 yerine 0.01 ve 0.99 kıullanılır
    interquantile_range = quartile3 - quartile1
    up_limit = quartile3 + 1.5 * interquantile_range
    low_limit = quartile1 - 1.5 * interquantile_range
    return low_limit, up_limit
```

3. The function that checks the values in the variable according to these threshold values, and if there are values that are above or below these thresholds, it replaces them with threshold values : replace_with_thresholds

```
def replace_with_thresholds(dataframe, variable):
    low_limit, up_limit = outlier_thresholds(dataframe, variable)
    dataframe.loc[(dataframe[variable] < low_limit), variable] = low_limit
    dataframe.loc[(dataframe[variable] > up_limit), variable] = up_limit
```

Outlier_thresholds and replace_with_thresholds functions are used in retail_data_prep function, these 3 functions are combined into one function.

```
def retail_data_prep(dataframe):
    dataframe.dropna(inplace=True)
    dataframe = dataframe[~dataframe["Invoice"].str.contains("C", na=False)]
    dataframe = dataframe[dataframe["Quantity"] > 0]
    dataframe = dataframe[dataframe["Price"] > 0]
    replace_with_thresholds(dataframe, 'Quantity')
    replace_with_thresholds(dataframe, 'Price')
    return dataframe


df = retail_data_prep(df)
df.describe().T
```

2. **Preparing ARL Data Structures (Invoice-Product Matrix)**

A matrix form should be created with Invoices in the rows and Products in the columns that will correspond to whether they are in this process or not.

Whether or not there are products in the invoices in this matrix should be expressed with 1-0 values.

```
# Description   NINE DRAWER OFFICE TIDY   SET 2 TEA TOWELS I LOVE LONDON    SPACEBOY BABY GIFT SET
# Invoice
# 536370                              0                                 1                         0
# 536852                              1                                 0                         1
# 536974                              0                                 0                         0
# 537065                              1                                 0                         0
# 537463                              0                                 0                         1
```

So the problems here are:

- All possible products should appear in the columns. Rows must have all Invoices.
- At their intersection there will be information about how many of a particular product is in certain Invoices. It is necessary to get rid of these values and express the presence or absence of products with only 1-0 (binary encode).

We proceed by reducing the dataset to a specific country. Here, the association rules of France customers will be derived.

```
df_fr = df[df['Country'] == "France"]
```

Note: There can be very valuable business approaches and problem solving approaches here.

Assuming that this online sales company is going to enter the German market, wants to offer products to customers coming from the German market, but does not have customers from Germany before, a country that is expected to exhibit similar habits with Germany is determined (in a scenario where the data of the specified country is available), and association rules are drawn according to that country. Later, when entering the German market, the association rules learned from the specified country can be applied to customers coming from Germany.

This structure is edited because the structure of the dataset is not suitable for constructing the Invoice-Product matrix.
(Description included for observation purposes.)

```
df_fr.groupby(['Invoice', 'Description']).agg({"Quantity": "sum"}).unstack().fillna(0).iloc[0:5, 0:5]
```

If StockCode variable is wanted to be used instead of Describe:

```
df_fr.groupby(['Invoice', 'StockCode']). \
    agg({"Quantity": "sum"}). \
    unstack(). \
    fillna(0). \
    applymap(lambda x: 1 if x > 0 else 0).iloc[0:5, 0:5]
```

In the apply function, row or column information is given. Executes a function by automatically traversing these rows or columns, without the need to write a loop. The applymap used here is a function that traverses all observations.

In this section, first the Invoice-Product matrix will be created, then a function will be written that creates the matrix columns according to the optional Description or StockCode.

```
def create_invoice_product_df(dataframe, id=False):
    if id:
        return dataframe.groupby(['Invoice', "StockCode"])['Quantity'].sum().unstack().fillna(0). \
            applymap(lambda x: 1 if x > 0 else 0)
    else:
        return dataframe.groupby(['Invoice', 'Description'])['Quantity'].sum().unstack().fillna(0). \
            applymap(lambda x: 1 if x > 0 else 0)


fr_inv_pro_df = create_invoice_product_df(df_fr)

fr_inv_pro_df = create_invoice_product_df(df_fr, id=True)
```

The check_id function can be defined in order to work on these studies more easily. The StockCode variable is selected from the entered dataframe and when the StockID to be queried is written, the check_id function returns the description of this ID.

```
def check_id(dataframe, stock_code):
    product_name = dataframe[dataframe["StockCode"] == stock_code][["Description"]].values[0].tolist()
    print(product_name)


check_id(df_fr, 10120)
```

Since only the string value is wanted to be accessed from the values in the output, the values[0] statement is used and the product name is printed using tolist() to output it as a list.

3. Association Rules

The first thing to do is to find the Support values, ie probabilities, of all possible product associations with the Apriori Method.

The arguments that the apriori function wants to give the probabilities of possible product associations are:

- DataFrame
- a min_support (threshold) value (if any) set at the begining of the code
- If you want to use the names of the variables in the data set : use_colnames = True

```
frequent_itemsets = apriori(fr_inv_pro_df,
                            min_support=0.01,
                            use_colnames=True)
frequent_itemsets.sort_values("support", ascending=False)
```

The probabilities of the possible pairs of products are sorted according to their support values, from largest to smallest.

Association rules are derived using these data.

```
rules = association_rules(frequent_itemsets,
                          metric="support",
                          min_threshold=0.01)
```

In the output of this code:

- Leverage :

  It means leverage effect. It is a value similar to Lift. However, Laverage tends to prioritize values with higher Support. Therefore, it has a slight bias. The lift value, on the other hand, can catch some relationships, although less frequently. Therefore, it is a more valuable and unbiased metric for us.

- Conviction :

  It is the expected value/frequency of product x without product y.

  It is the expected value/frequency of product y without product x.

In practice, operations are usually carried out with the following combinations:

The support value is above ..., the confidence value is above ..., and the lift value is above ...

Evaluations can be made on several possible combinations such as

```
rules[(rules["support"]>0.05) & (rules["confidence"]>0.1) & (rules["lift"]>5)]

check_id(df_fr, 21086)

rules[(rules["support"]>0.05) & (rules["confidence"]>0.1) & (rules["lift"]>5)]. \
    sort_values("confidence", ascending=False)
```

Confidence can be considered a more reliable metric as it expresses the probability that one will be bought when the other is bought. If a user (according to the table in the output) has added (22352, 22356) to their shopping cart, then we recommend (20724) to that person.

This is because when the first two products are bought, the confidence value, which expresses the probability of buying the third product, is 1.000000.

4. **Preparing the Script of the Work**

```
def outlier_thresholds(dataframe, variable):
    quartile1 = dataframe[variable].quantile(0.01)
    quartile3 = dataframe[variable].quantile(0.99)
    interquantile_range = quartile3 - quartile1
    up_limit = quartile3 + 1.5 * interquantile_range
    low_limit = quartile1 - 1.5 * interquantile_range
    return low_limit, up_limit

def replace_with_thresholds(dataframe, variable):
    low_limit, up_limit = outlier_thresholds(dataframe, variable)
    dataframe.loc[(dataframe[variable] < low_limit), variable] = low_limit
    dataframe.loc[(dataframe[variable] > up_limit), variable] = up_limit

def retail_data_prep(dataframe):
    dataframe.dropna(inplace=True)
    dataframe = dataframe[~dataframe["Invoice"].str.contains("C", na=False)]
    dataframe = dataframe[dataframe["Quantity"] > 0]
    dataframe = dataframe[dataframe["Price"] > 0]
    replace_with_thresholds(dataframe, "Quantity")
    replace_with_thresholds(dataframe, "Price")
    return dataframe


def create_invoice_product_df(dataframe, id=False):
    if id:
        return dataframe.groupby(['Invoice', "StockCode"])['Quantity'].sum().unstack().fillna(0). \
            applymap(lambda x: 1 if x > 0 else 0)
    else:
        return dataframe.groupby(['Invoice', 'Description'])['Quantity'].sum().unstack().fillna(0). \
            applymap(lambda x: 1 if x > 0 else 0)


def check_id(dataframe, stock_code):
    product_name = dataframe[dataframe["StockCode"] == stock_code][["Description"]].values[0].tolist()
    print(product_name)


def create_rules(dataframe, id=True, country="France"):
    dataframe = dataframe[dataframe['Country'] == country]
    dataframe = create_invoice_product_df(dataframe, id)
    frequent_itemsets = apriori(dataframe, min_support=0.01, use_colnames=True)
    rules = association_rules(frequent_itemsets, metric="support", min_threshold=0.01)
    return rules

df = df_.copy()

df = retail_data_prep(df)
rules = create_rules(df)

rules[(rules["support"]>0.05) & (rules["confidence"]>0.1) & (rules["lift"]>5)]. \
sort_values("confidence", ascending=False)
```

5. **Making Product Suggestions to Users at the Basket Stage**

For example, let a user add the product with ID 22492 to his cart.

First of all, the product added to the cart was checked with the check_id function.

Afterwards, the previously calculated rules were sorted by lift (you can also sort by confidence or support here).

```
# Example:
# Selected Product ID: 22492
```

```
product_id = 22492
check_id(df, product_id)

sorted_rules = rules.sort_values("lift", ascending=False)
```

When the product with ID 22492 is seen in the column "antecedents" (first product), then the first product that will appear in the column "consequents" (later product) is the product that can be offered to the customer who bought the product with the ID 22492.

So, iterating is performed over “antecedents”. Here, the product/products belonging to the "consequents" column in the same index as the product with ID 22492 to be captured are captured.

i. In sorted_rules, the enumerate method is used according to the "antecedents" column.

   While product iterates over all rows, i is also iterate over the index information.

ii. In the first iteration (according to the output) that product represents, the frozenset structure is observed. frozenset is a data structure, a type of set, similar to sets. In order to perform a certain operation on these values, this data structure must be converted to a list. In other words, in this iteration, the values captured while navigating through "antecedents" are converted to a list.

iii. Since this list also needs to be iterate, another loop is written. : for j in list(product):

iv. If the searched product is caught while iterating over the list : if j == product_id:

v. The value of the “consequents” column corresponding to the index of this product is added to the empty recommendation_list = [] created at the beginning : recommendation_list.append(list(sorted_rules.iloc[i]["consequents"])[0])

   Since it is requested to bring 1 : [0]

   If more is desired : recommendation_list[0:3]

```
recommendation_list = []

for i, product in enumerate(sorted_rules["antecedents"]):
    for j in list(product):
        if j == product_id:
            recommendation_list.append(list(sorted_rules.iloc[i]["consequents"])[0])

recommendation_list[0:3]

check_id(df, 22326)
```

Note : Double, triple… product groups are assumed to be a single product while continuing the operations, after the existence of a specific product has been tested.

Functionalized version of the operation:

```
def arl_recommender(rules_df, product_id, rec_num=1):
    sorted_rules = rules_df.sort_values("lift", ascending=False)
    recommendation_list = []
    for i, product in enumerate(sorted_rules["antecedents"]):
        for j in list(product):
            if j == product_id:
                recommendation_list.append(list(sorted_rules.iloc[i]["consequents"])[0])

    return recommendation_list[0:rec_num]


arl_recommender(rules, 22492, 1)
arl_recommender(rules, 22492, 2)
arl_recommender(rules, 22492, 3)
```

References ：

https://courses.miuul.com/p/recommendation-systems