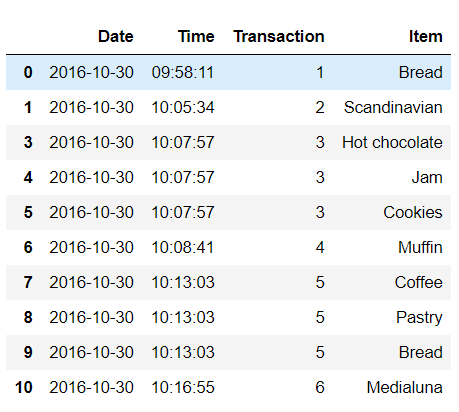
Association Mining Rule from Scratch

Are you curious about how Apriori algorithm & Association Rule works? and How Permutation & Combination are useful in mining rule. Then you are at right place, let me walk you through it. In this article we will discuss:



Many business enterprises accumulate huge amount of data from their daily operation. For example, large number of customers purchase detail are collected daily in grocery store. Such data, commonly known as **Market Basket Transaction** shown in Figure 1. Each row i:e each customer transaction at a time labelled as **Transaction**. Figure 1 illustrate an example, Transaction 1 contain {Bread}, transaction 2 contain {Scandinavian}, transaction 3 contain {Hot chocolate, Jam, Cookies} etc.

Figure : Market Basket Transaction

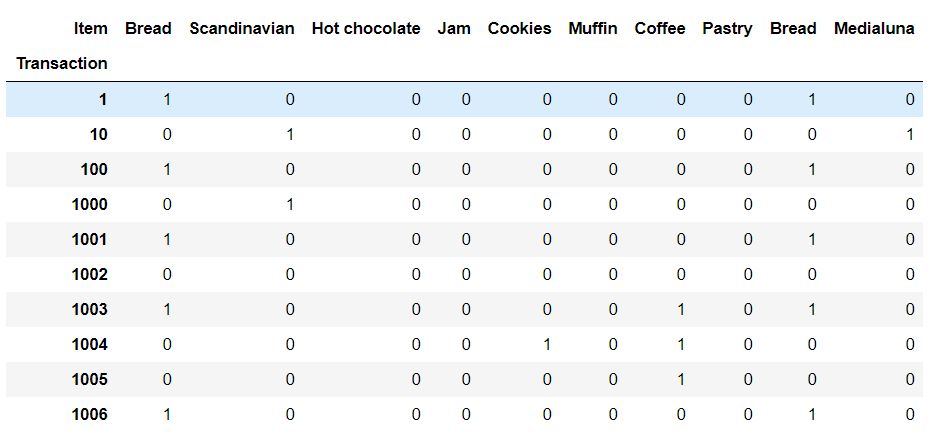
Market basket data should be converted to Binary format as shown in Figure 2. Where each row correspond to a transaction and each column corresponds to an item. Item can be treated as binary variable, holds value one if item is present in a transaction else zero.

Figure : Binary Representation of market basket data

**Itemset and Support Count:**

Let I = {Bread, Scandinavian, Cake, Muffin, Coffee, Pastry} be the set of all items in a market basket data and T = Total No of transaction be the set of all transaction. An itemset may contain single or more than one item like {Cake}, {Bread}, {Bread, Cake}, {Bread, Coffee}. An important property of an itemset is its support count, which refer to the number of transactions that contain itemset. Mathematically, the support {Bread, Coffee} for an itemset {Bread, Coffee} can be stated as follows:

**Frequent Itemset Generation:**

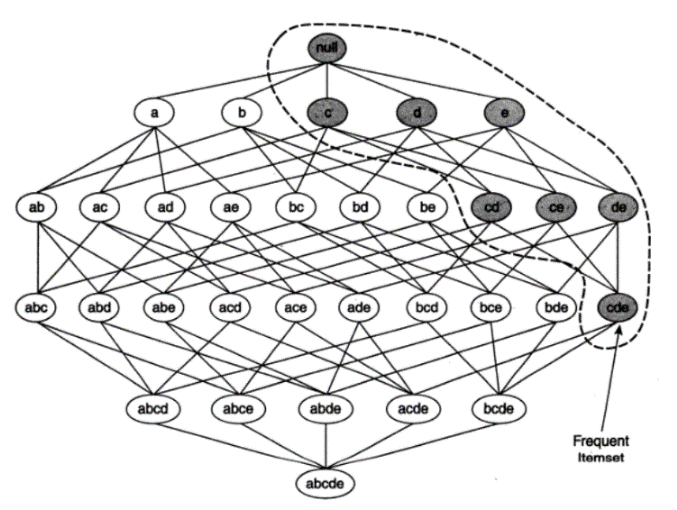


Figure : frequent itemset lattice

Objective is to find the itemsets that satisfy *minimum* threshold support, these itemsets are called **frequent itemsets** Otherwisecalled **Infrequent Itemset.** Don’t get confuse if I use this term. Figure 3 shows all possible itemsets that can be generate by an itemset I = {a, b, c, d, e}. in general, a dataset that contain k items can potentially generate up to 2K itemset. Because K can be very large in many practical applications, it becomes computationally expensive. To overcome this problem, we can prune the unwanted itemset as follow, As illustrated in Figure 3. when an itemset{a}, {b} is infrequent, then all of its supersets (i:e the non-shaded itemset in this figure) must be infrequent too. Suppose {c, d, e} is a frequent itemset, then all its subsets itemset (i:e the shaded itemset in this figure) must also be frequent.

**Apriori Algorithm:**

Let me show you how Apriori algorithms works and generate frequent itemsets based on th concept that a subset of a frequent itemset must also be a frequent itemset. Let’s take an example from our dataset.

**def APRIORI\_MY**(data, min\_support=0.05, max\_length = 5):

min\_suppport is minimum threshold support.

where max\_length is maximun number of items to be included in itemset.

**STEP 1:**

Create dictionary “support” to stored itemsets and support

List all items in “L” available in dataset

**STEP 2:**

For i in (1, max\_lenght + 1)

* Create combination of itemset having length i and stored in C.

C = combination (L, i)

* Reset List L so that we can append frequent itemset obtained in ith iteration to create combination in (i+1) iteration.

**STEP 3:**

For j in list(C)

* calculate support for each j.

sup = data.loc[:,j].product(axis=1).sum()/len(data.index)

* if sup meets minimum support threshold then add to “support” dictionary.

support[j] = sup

* update “L” for each j take union of itemset in list L and itemset(j)

L = list(set(L) | set(j))

**STEP 2** Iteration are as follows:

**Iteration 1:** Initially the algorithm list all the items and compute support for frequent itemsets with length 1 and stored it in dictionary, dictionary helps to retrieve values as support using keys as items, here few of them shown in Figure 4- Table L1. Here we assume 4% as minimun threshold support.

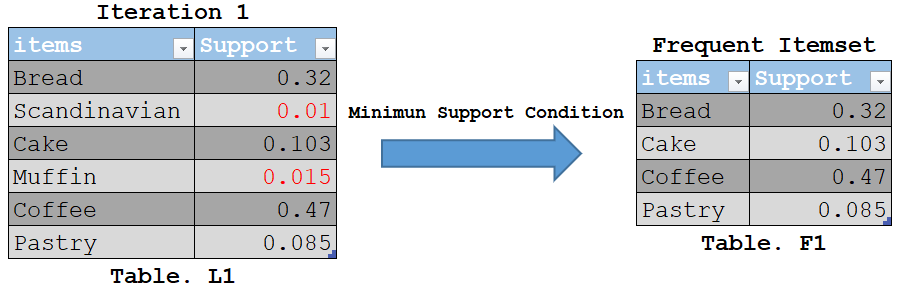


Figure : Iteration 1

As you can see here, item ‘**Scandinavian’** and ‘**Muffin’** is infrequent. So, we are going to **discard {**‘Scandinavian’, ‘Muffin’**}** in the upcoming iterations. We have the final Table F1 as Frequent Itemset.

**Iteration 2:** Next, we will create combination of 2 itemset and calculate their support. All the combinations of itemset shown in Figure 4-Table F1 are used in this iteration.

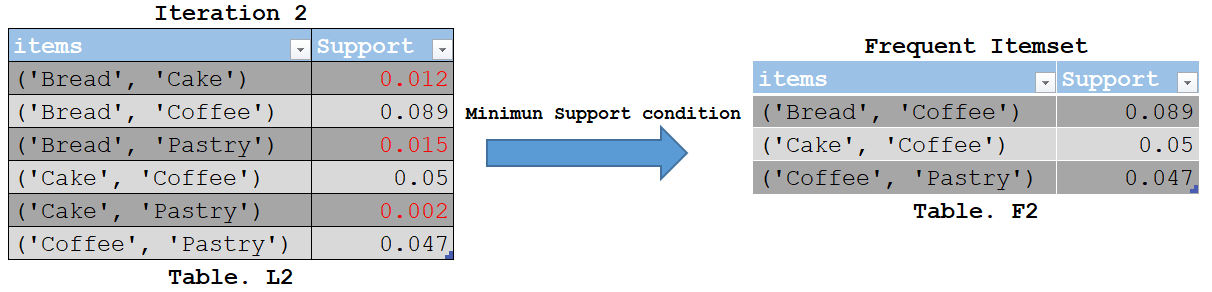


Figure : Iteration 2

Infrequent Itemsets are eliminated again. In this case **{('Bread', 'Cake'), ('Bread', 'Pastry'), ('Cake', 'Pastry')} shown in Figure 5-Table L2.**Now, let us understand what is pruning and how it makes Apriori one of the best algorithms for finding frequent itemsets.

**Pruning:** Here we will divide the itemsets in Figure 6-Table L3 into subsets and discard the subsets that are having a support less than minimun threshold support.

**Iteration 3:** Here all the itemset are infrequent since its subset **{('Bread', 'Cake'), ('Bread', 'Pastry'), ('Cake', 'Pastry')} already discard in Figure 5-Table. L2, So it Superset will also be infrequent.** This is the main highlight of the Apriori Algorithm.

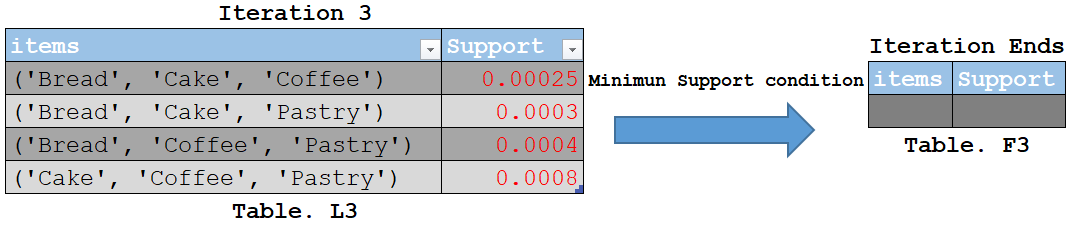


Figure : Iteration 3

Since all the itemsets in Iteration 3 are infrequent we will stop here. Let ms show you the actual frequent itemset obtained by appriori algorithm.

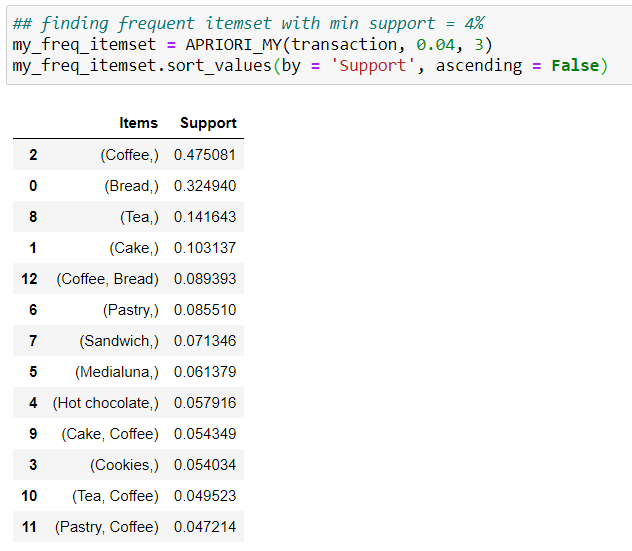


Figure :My Frequent Itemset

We are almost done as we already obtained frequent itemset, which generally take more computational time. Next we are going to see Association Rule.

**Association Rule:**

This section describes how to extract association rules efficiently from a above obtained frequent itemset. An association can be obtained by partitioning the frequent itemsets {Bread, Coffee} into two non-empty subsets, 1) Bread => Coffee, simple way to understand “**If Bread then coffee**”, 2) Coffee => Bread, “**If Coffee then Bread”**. The subset meets minimum threshold confidence and Positive lift can be called as strong rule. Curious what is Confidence and Lift? Let me tell you, there are several measures available to analyse a rule, in this article we will discuss confidence and lift.

**Confidence:**

Objective is to extract all the high-confidence rules from the frequent itemset found in previous step. It says that **Consequent** (Coffee) is brought as an effect **Antecedent** (Bread). This rule called as strong rule.

But confidence measure has one drawback, it might misrepresent the importance of an association. Because it only tells how popular Antecedent (Bread) are, but not Consequent (Coffee). If coffee is very popular then it is more likely that a transaction contain Bread will also contain coffee, thus inflating confidence measure. To overcome this drawback, we use a third measure called lift.

**Lift:**

Lift computes the ratio between the rule’s confidence and the support of the itemset in the rule consequent. It represents how likely a consequent (Coffee) is purchased when an antecedent (Bread) is already purchased, while controlling for how popular consequent (Coffee) is. When the lift of (Bread => Coffee) is 1, which means no association between items. A lift value greater than 1 implies coffee is likely to be brought if bread is already brought, while a value less than 1 implies coffee is unlikely to be brought if bread is already brought.

Or

**Association Algorithm:**

Now we will apply association rule to the frequent itemset obtained in previous algorithm. We will assume minimum threshold confidence 50%.

**def ASSOCIATION\_RULE\_MY** (df, min\_threshold=0.5):

**STEP 1:**

List all frequent itemset and its support to dictionary "support".

Create list "data" to stored results.

List all frequent itemset to List “L”.

Create rule using permutation of length 2 using list L.

p = list (permutations (L, 2)).

**STEP 2:**

for i in p: (Iterate through each rule in p).

* if LHS(i[0]) or antecedent of rule is subset of RHS(i[1]) then it’s valid rule.
* if set(i[0]).issubset(i[1]):
* Calculate confidence for rule

conf = support[i[1]]/support[i[0]]

* if conf meets minimum confidence threshold then compute other rule measures

if conf > min\_threshold:

* getting Consequent for each rule to displayed it in results

j = i[1][not i[1].index(i[0][0])]

* Calculating lift measure.

lift = support[i[1]]/(support[i[0]]\* support[(j,)]

* Appending to list data

data.append([i[0], (j,), support[i[0]], support[(j,)], support[i[1]], conf, lift])

Initially the algorithms will generate rule using Permutation of size 2 of frequent itemset and calculate Confidence and Lift shown is Figure 8.

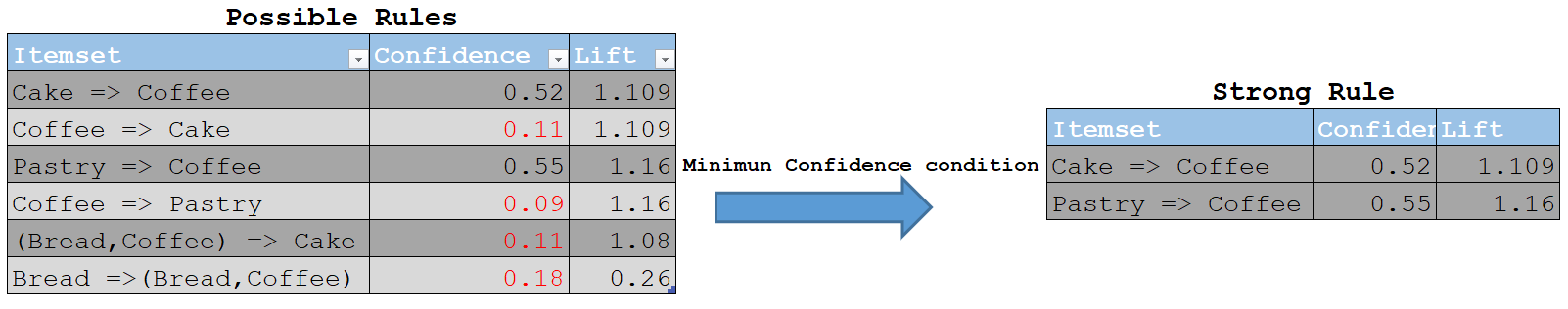


Figure 8: Strong Rule

Why Permutation over Combination to calculate rule? Let me answer this with a simple example.

*Support (Cake & Coffee) = Support (Coffee & Cake)*

As number of transactions contain Cake & Coffee is same as number of transactions contain Coffee and Cake, Order does not matter. But this not case in a rule, let me explain.

*Confidence (Cake => Coffee) = Support (Cake & Coffee) / Support (Cake) = 0.52*

*Confidence (Coffee => Cake) = Support (Cake & Coffee) / Support (Coffee) = 0.11*

Here, Order does matter; thus, we will select those rules which meets minimum threshold confidence. Permutation help fixed position of Antecedent on left side, confuse let me explain with a simple example, let us have Permutation of Cake, Coffee and (Coffee, Cake) to calculate rule as below.

{Cake, (Cake, Coffee)}, LHS(Cake) is fixed for antecedent and it must be subset of RHS (Cake, Coffee) this gives Confidence (Cake => Coffee).

{Coffee, (Cake, Coffee)} = LHS(Coffee) is fixed for antecedent and it must be subset of right side (Cake, Coffee) this gives Confidence (Coffee => Cake).

Till here it’s all good, but we will have more rule like {(Cake, Coffee), Cake} and {(Cake, Coffee), Coffee}, here antecedent is (Cake, Coffee) which seems incorrect thus subset condition is used to eliminate such rule.

Below Figure 9 shows the Strong Rule obtained from experimental datasets. Do not forget that Rule is only applied on Frequent Itemset.

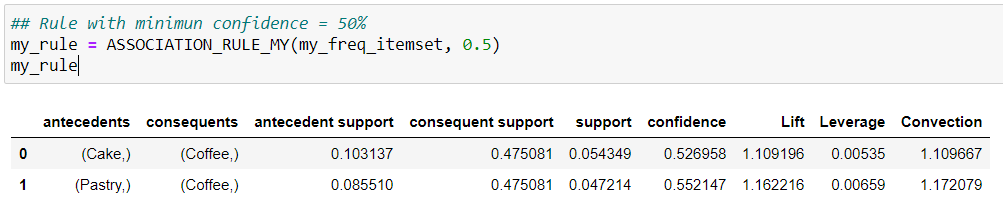
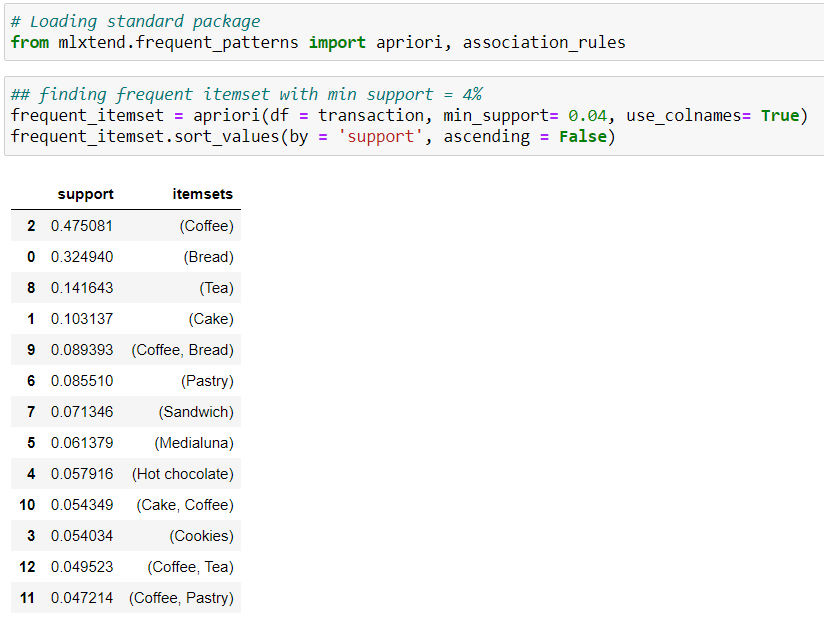


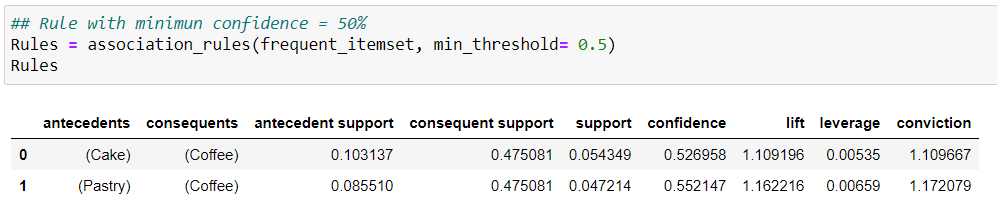
Figure : My Rule

Finally, we will cross verify our results with the Standard Package available in Python called mlextend.frequent\_patterns having Apriori and association rules modules.

**Apriori results:**



**Association Rule results:**



From above comparison we can conclude that, our results matched with standard packages and proposed objective has been served. Feel free to download scratch codes available on my GitHub <https://github.com/Roh1702/Association-Mining-Rule-from-Scratch>

Why **Lift** is better measure over **Confidence**? This we will see in detail in another article I will be publishing soon.

I would like to Thank professor Mr. Gourabh Nath Faculty at Praxis Business School; his sessions and guidance help me to dive deep in Association Analysis.

**References:**

Introduction to Data Mining by Pang-Ning Tan, Michael Steinbach and Vipin Kumar

[*https://github.com/viktree/curly-octo-chainsaw/blob/master/BreadBasket\_DMS.csv*](https://github.com/viktree/curly-octo-chainsaw/blob/master/BreadBasket_DMS.csv)