Chapter 5 : Part 1

**Association Rule Learning**

**Apriori**

**5.1.1 Association Rule Learning**

Association rule learning is a type of Unsupervised Learning Technique that checks for the DEPENDENCY of *one data-item* on *another data-item* and ***maps*** accordingly so that it can be more ***profitable***.

* It tries to find some interesting relations or associations among the variables of dataset. It is based on different rules to discover the *interesting* *relations* between variables in the database.

|  |  |
| --- | --- |
| * It is employed in-  1. Market Basket analysis: For example, if a customer buys *bread*, he most likely can also buy *butter*, *eggs*, or *milk*, so these products are stored *within a shelf* or *mostly* *nearby*. 2. Web usage mining 3. Recommendation system 4. Continuous production, etc.  * Here market basket analysis is a technique used by the various big retailer to discover the *associations between items*. We can understand it by taking an example of a supermarket, as in a supermarket, *all products* that are *purchased* *together* are *put* *together*. | Association Rule Learning |

* Association rule learning can be divided into three types of algorithms:

1. Apriori
2. Eclat
3. F-P Growth Algorithm

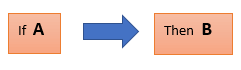
* Diapers and Beer Example: Very often during certain times of the day, when people shop in the afternoon between 6 and 9 p.m.



* People who buy *diapers* also buy *beer.* Which makes no sense at first but a plausible explanation might be: when the husband gets home and husband and the wife are taking care of their baby.
* They sometimes find that they run out of diapers and most of the time, the husband has to go pick up the diapers. While he's picking up the Diapers because it's really after hours after work, he also picks up some Beer.
* And based on that you can decide how to arrange products in your store.
* So some stores might decide to put these two items (Beer & Diapers) closer to *entice* *people* to *buy a beer when they're buying diapers*.
* But actually a lot of stores do the opposite. For example, you probably noticed this from your convenience store that they try to separate bread and milk as far as possible.
* Because that way they really know that *these two products are bought together*. And so you actually have to *walk through the whole store* to pick up.
* So you've picked up your bread and then to get to the milk you have to get all the way through the whole store to the completely opposite corner of the store.
* When you're *walking through* the store you *see more other products* and you're more likely to pick up an ***additional item*** that you ***weren't actually planning on buying*** when you got to the store in the first place.
* So there's a lot of interesting marketing tactics that are used based on this data.

**5.1.2 How does Association Rule Learning work?**

Association rule learning works on the concept of ***If*** and ***Else*** Statement, such as **if** A **then** B.

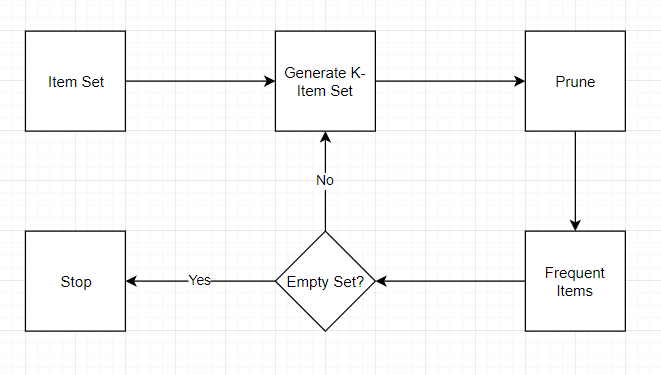


* Here the *If* *element* is called Antecedent, and *then* *statement* is called as Consequent. These types of relationships where we can find out some ***association or relation*** between ***two items*** is known as Single Cardinality.
* It is all about *creating rules*, and if the *number of items* *increases*, then *cardinality* also *increases* accordingly.

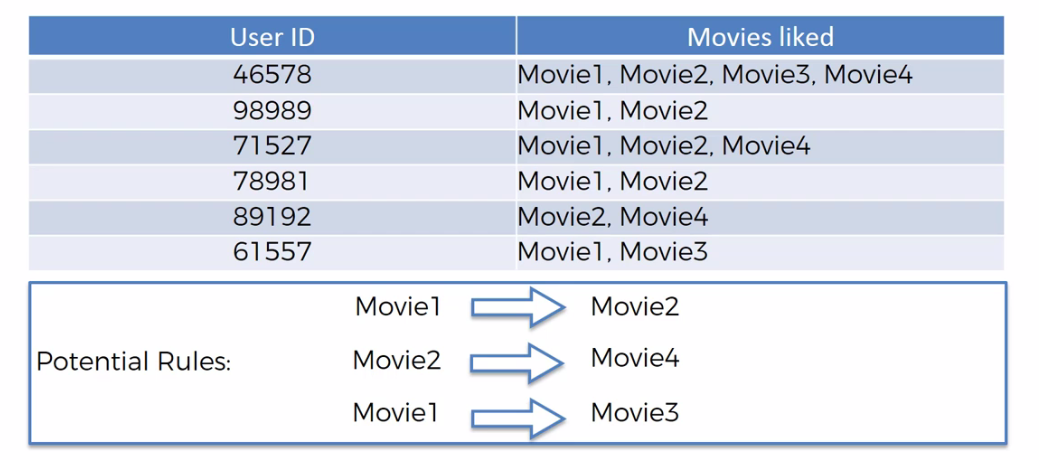
**5.1.3 Apriori Algorithm**

The *Apriori* algorithm uses *frequent itemsets* to generate *association* *rules*, and it is designed to work on the databases that contain transactions. With the help of these association rule, it determines how strongly or how weakly two objects are connected. This algorithm uses a breadth-first search and Hash Tree to calculate the itemset associations efficiently. It is the *iterative* process for finding the frequent itemsets from the large dataset.

* This algorithm was given by the *R. Agrawal* and *Srikant* in the year 1994. It is mainly used for *market basket analysis* and helps to ***find those products that can be bought together***. It can also be used in the ***healthcare*** field to find ***drug*** ***reactions*** for patients.
* The control flow diagram for the Apriori algorithm



* Apriori uses a "bottom up" approach, where frequent subsets are extended one item at a time (a step known as candidate generation), and groups of candidates are tested against the data.
* The algorithm terminates when no further successful extensions are found.
* Apriori uses breadth-first search and a Hash tree structure to count candidate item sets efficiently. It generates candidate item sets of length from item sets of length . Then it prunes the candidates which have an infrequent sub pattern.
* According to the downward closure lemma, the candidate set contains all frequent -length item sets. After that, it scans the transaction database to determine frequent item sets among the candidates.
* Apriori is all about:
* People who bought something also bought something else or
* Watched something also watched something else
* Did something also did something else
* This whole *association* *rule* *learning* part is all about analyzing when things come in *pairs* or in *triplicates* or in certain *sequence* but they are combined together for some reason looking for those rules (reasons, finding specific pattern) and those ways that this happens.
* Consider the following example:



* From above we can easily tell that there are some potential rules. For example everybody who watches Movie1 also like Movie2. People who like Movie2 also like Movie4. And people who like Movie1 are also quite likely to be like Movie3.
* But from those rules some rules are "**strong**" and some rules are "**weak**". We want to find the *very* *strong* *ones* in order to build our *business decisions* or other decisions on those rules.
* We don't want to go to the people and asking their opinions instead we extract those information/rules from our data. So if we have a large ***sample*** ***size*** say, ***50000*** or ***500000*** people then by we're analyzing that data we can come up with some quite solid rules.

**5.1.4 How Does Apriori algorithm works**

The apriori algorithm has three parts to it: the ***support***, the ***confidence*** and the ***lift***.

* Calculation metrices: So, to measure the associations between ***thousands*** of ***data-items***, there are ***several*** ***metrics***. These metrics are:

|  |  |  |
| --- | --- | --- |
| 1. Support | 1. Confidence | 1. Lift |

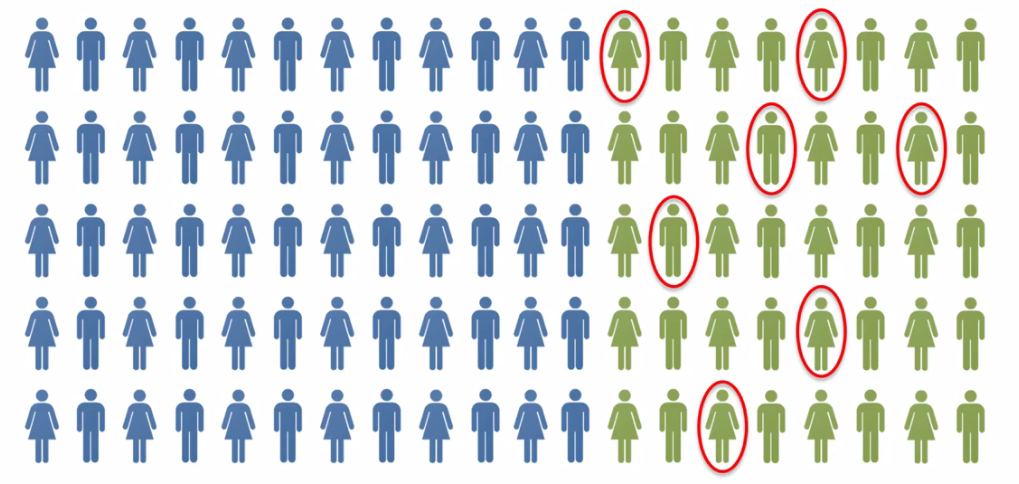
1. Support: We're going to start up with the support and you will see that it's very similar to the way we talked about the ***Naïve Bayes classifiers***.

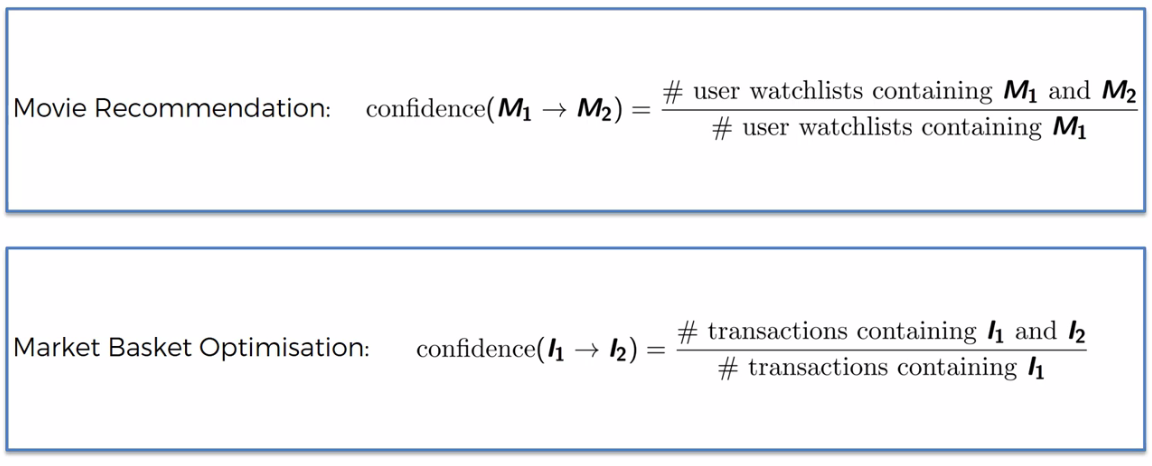
|  |  |
| --- | --- |
| * Support is the ***frequency*** of ***A*** or ***how frequently an item appears in the dataset***. It is defined as the fraction of the ***transaction T*** that contains the itemset ***X***. If there are ***X*** ***datasets***, then for ***transactions*** ***T***, it can be written as: |  |

* For example: Say from ***100 people***, ***10*** ***people*** watched ***Ex-Machina***, then **,**

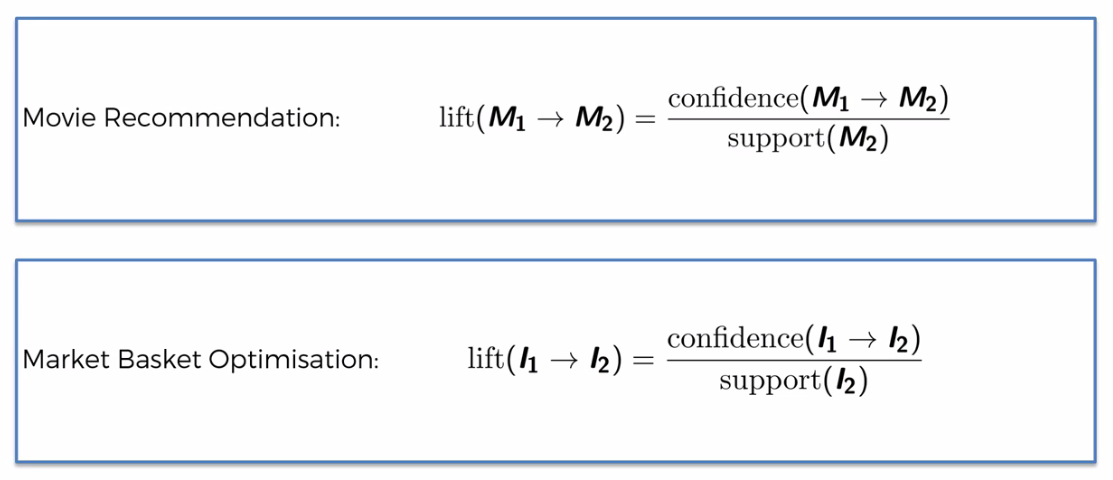
1. Confidence: Confidence indicates how often the rule has been found to be ***true***. Or, how often the ***items*** ***X*** and ***Y*** occur together in the dataset when the occurrence of ***X*** is already given. It is the ratio of the ***transaction*** that contains ***X*** and ***Y*** to the number of ***records*** that contain ***X***.

* For example: Now consider we are testing a rule, ***"People watched*** Intersteller ***are also watched Ex-Machina "***. So in the following diagram say ***green people watched*** Intersteller(the number is ***40 among 100***), and ***7*** of them watched ***Ex-Machina***, then the confidence is:

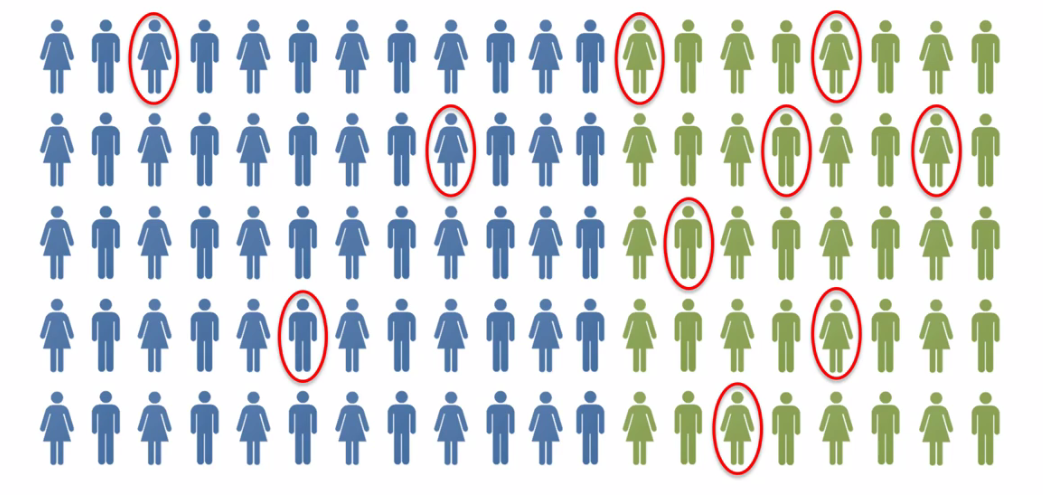




1. Lift: Lift is very similar to the ***Naïve Bayes classifiers***. Lift is basically is the ***ratio*** of ***Confidence*** and ***Support***. It is the *strength of any rule*, which can be defined as below formula:

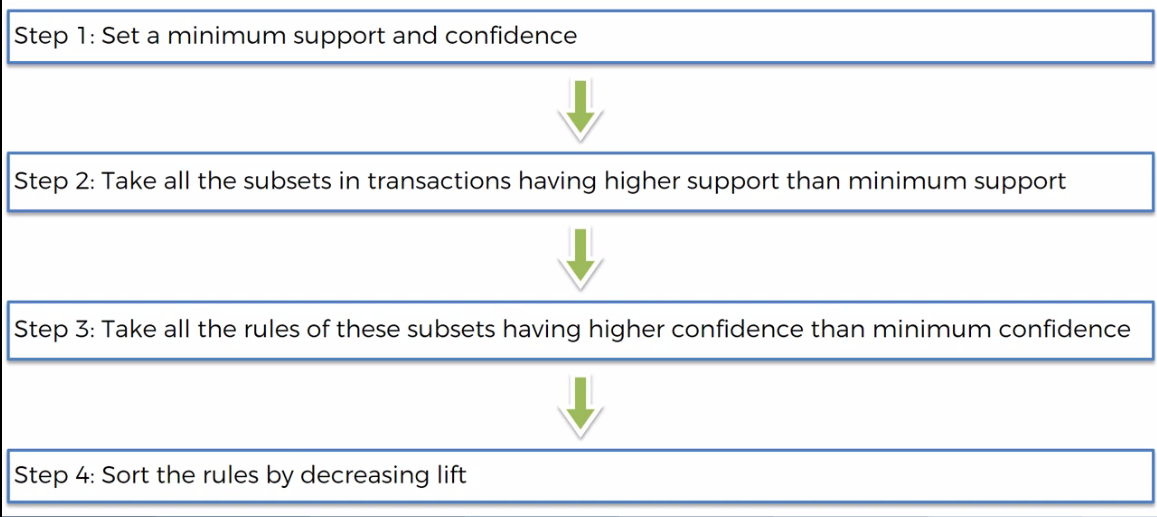


* Example: So in below ***Green*** people watched Interstellerand ***Red-circle*** represents the people watched ***Ex-Machina***.



* Out of this population we know that 10% actually likes ***Ex-Machina.*** So if we take another random population and then what is the ***likelihood that if we recommend to a random person*** in that brand ***new*** ***population*** will recommend the ***Ex-Machina*** movie, what is the likelihood that they will like it.
* Well the likelihood is 10%. But now the question is: Can we prove that result by using some prior knowledge. That's why the algorithm is called Apriori.
* In that new population let's only recommend ***Ex-Machina*** to people who have already seen ***Interstellar***. In that case the likelihood as we've calculated out of the Green people ***17.5%*** actually liked ***Ex-Machina***. So the ***Lift*** is the improvement in your prediction.
* So, out of your new population, if you first ask the question "Have you seen and liked interstellar?". If they say "yes" and then you recommend "***Ex-Machine***" the likelihood of a successful recommendation is ***17.5%***. So the lift is by definition is ***1.75***.

**5.1.5 Steps in Association Rule Learning**



* Apriori is actually quite a slow algorithm because it just goes through all of these different combinations of rules. For example it will calculate rules for *Movie1*, *Movie2*, *Movie3*, *Movie4*, … so on for pair (*Movie1* and *Movie2*) or triplet (*Movie1* and *Movie2 and Movie3*) so on.
* So there is so many different rules and we pick only strong rules. The rule with ***highest*** ***lift*** is the ***strongest*** rule.

**5.1.6 Types of Association Rule Learning**

1. Apriori Algorithm: This algorithm uses frequent datasets to generate association rules. It is designed to work on the databases that contain transactions. This algorithm uses a ***Breadth-First Search*** and ***Hash Tree*** to calculate the itemset efficiently.

* It is mainly used for market basket analysis and helps to understand the products that can be bought together. It can also be used in the healthcare field to find drug reactions for patients.

1. Eclat Algorithm: *Eclat algorithm* stands for *Equivalence Class Transformation*. This algorithm uses a *depth-first search* technique to find *frequent* *itemsets* in a *transaction* database. It performs *faster* *execution* than *Apriori* Algorithm.
2. F-P Growth Algorithm: The ***F-P growth*** algorithm stands for ***Frequent Pattern***, and it is the ***improved*** ***version*** of the ***Apriori*** Algorithm. It represents the ***database*** in the form of a ***tree structure*** that is known as a ***frequent*** ***pattern*** or ***tree***. The purpose of this ***frequent tree*** is to ***extract*** the most ***frequent*** ***patterns***.

* Applications of Association Rule Learning: It has various applications in machine learning and data mining. Below are some popular applications of association rule learning:
* Market Basket Analysis: It is one of the popular examples and applications of association rule mining. This technique is commonly used by big *retailers* to determine the *association between items*.
* Medical Diagnosis: With the help of association rules, patients can be cured easily, as it helps in *identifying the probability of illness* for a particular disease.
* Protein Sequence: The association rules help in determining the synthesis of artificial Proteins.
* It is also used for the *Catalog* *Design* and *Loss-leader* *Analysis* and many more other applications.
* What is Frequent Itemset: Frequent itemsets are those items whose ***Support*** is greater than the ***Threshold*** ***Value*** or user-specified ***Minimum*** ***Support***. It means if ***A*** & ***B*** are the frequent ***itemsets*** ***together***, then ***individually*** ***A*** and ***B*** should also be the ***frequent*** ***itemset***.
* Suppose there are the two transactions: , and, in these two transactions, ***2*** and ***3*** are the ***frequent*** ***itemsets***.
* Advantages of Apriori Algorithm
* This is easy to understand algorithm
* The ***join*** and ***prune*** steps of the algorithm can be easily implemented on large datasets.
* Disadvantages of Apriori Algorithm
* The Apriori algorithm works Slow compared to other algorithms.
* The overall ***performance*** can be ***reduced*** as it scans the database for ***multiple times***.
* The timecomplexityand spacecomplexityof the Apriori algorithm is ***O(2D),*** which is ***very high***. Here ***D*** represents the horizontalwidthpresent in the database.

**5.1.7 Python Implementation of Apriori Algorithm**

We have a problem of a retailer, who wants to find the *association* *between* his shop's *product*, so that he can provide an offer of ***"Buy this and Get that"*** to his customers.

* The retailer has a dataset information that contains a list of transactions made by his customer. In the dataset, each ***row shows the products purchased by customers*** or ***transactions*** made by the customer.
* We are going to make this machine learning model to create some added value in some specific business. This business problem is going to be about ***optimizing*** the ***sales*** in a ***grocery*** ***store***. Using Association rule learning we want to know exactly where to place the products in the store. For example: If someone buys some ***Cereals*** the same person is very likely to buy some ***Milk*** as well.
* we're making this *Apriori* *model* for a store in the south of FRANCE. And so we want to find out the associationrules *of the* ***different***products of this Store to see how the Manager of this store can *optimize* *the* *placement* of its different products to *optimize* *the* *sales*.
* Imagine this store is located in one of the most popular places in the south of France. This place is a very Convivial place, a very friendly place where people love to hang out relax talk to each other.
* So these people come very often to the store to meet their friends. The manager of the store noticed and calculated that on average each customer goes and buys something to the store once a week.
* So this dataset contains the 7500 transactions of all the different customers that bought a basket of products in a whole week.
* Indeed the manager took it as the basis of its analysis because since each customer is going an average once a week to the store then the transaction *registered* over a *week* is quite *representative* of what *customers want to buy*.
* So based on all these 7500 transactions our ***Apriori ML model*** our model is going to learn the different associations it can make to actually understand the rules. Such as: if customers buy certain product then they're likely to buy other set of products.
* Data Pre-processing: First, we will perform the importing of the libraries. Before importing the libraries, we will install the ***apyori*** package to use further. ***Apyori*** is a simple implementation of ***Apriori algorithm*** with Python 2.7 and 3.3 - 3.5, provided as ***APIs*** and as ***commandline*** interfaces. (We have to use the external .py file ***"apyori.py"*** after installing it using ***pip***).

***pip install apyori***

**import** pandas **as** pd

**import** matplotlib**.**pyplot **as** plt

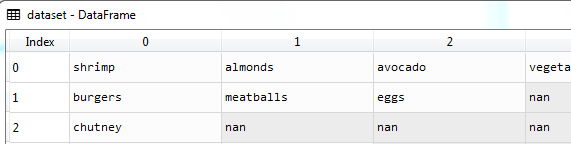
**import** numpy **as** np

* Importing the dataset: Now, we will import the dataset for our apriori model. To import the dataset, there will be some changes here.
* All the rows of the dataset are showing different transactions made by the customers. The first row is the transaction done by the first customer, which means ***there is no particular name for each column*** and have their own individual value or product details(See the dataset given below after the code).
* So, we need to mention in our code that there is no header specified. The code is given below:



* What we can see here is: it contains some different names of different products. And this line is supposed to be the line containing the titles of the columns (but this is not the case here).
* But these names here are not the ***actual*** ***names*** of the ***columns*** because these names are simply the ***names*** ***of*** ***the*** ***products*** in the ***first*** ***transaction*** ***registered*** by the ***Manager*** of this store.
* And ***Pandas*** in ***Python*** thought that this ***first*** ***line*** of the data set contains the ***titles*** of the ***column***.
* So we need to specify to Python that there is no titles in the data set. So the code become as follows:

dataset = pd**.read\_csv**("Market\_Basket\_Optimisation.csv", header = **None**)



* No data-split: Because the Apriori model is a special type of machine learning model so we won't need to do any splitting of the data-set into a training set and a test set.
* Convert Data-set to "***List of Lists***": So, when we use ***apriori*** implementation of the ***apyori*** we need to import the data set in a specific way. And it expecting the data as "***List of Lists***".
* But our data-set is not in that form, so we need to convert our data set into "***List of Lists***" format. i.e. a ***list*** of customer's bought ***item-set***.
* It's going to require TWO FOR loops because we're going to loop over all the transactions in the data-set. That's the first loop. Which has ***7500*** rows.
* And the second loop will be about to ***loop over all the products*** in each of the ***transaction***. Which has ***20*** columns.

#*importing data*

dataset = pd**.read\_csv**("Market\_Basket\_Optimisation.csv", header = **None**)

#*creating "List of Lists": List of product-set*

#*there are 7500 rows of transections and 20 columns of product*

transacTions = []

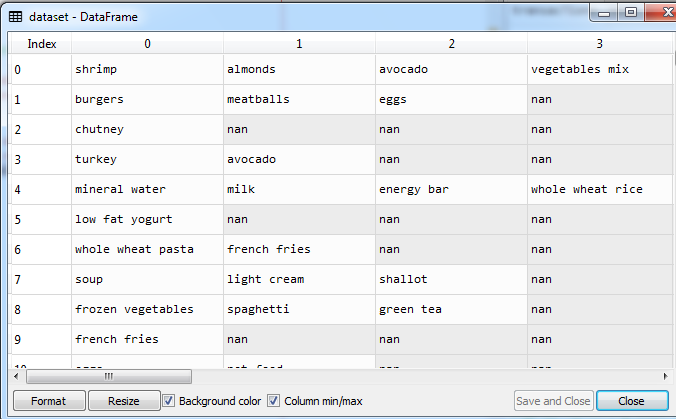
**for** i **in** **range**(0, 7501):

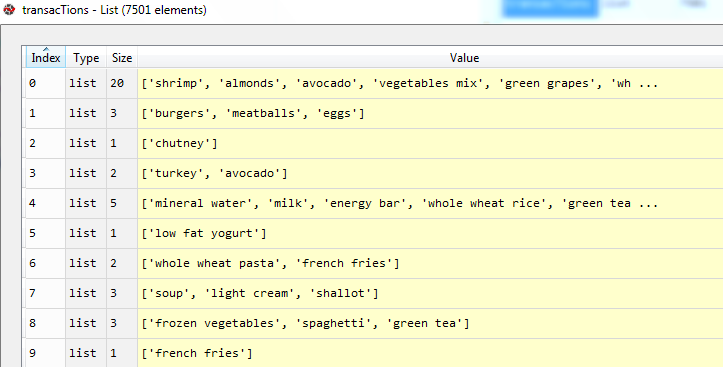
    #*notice :: "List comprehension is used". Items are converted to strings*

    #*transacTions.append([str(dataset.values[i, j]) for j in range(0, 20)])*

    transacTions**.append**([**str**(dataset**.**values[i, j]) **for** j **in** range(0, 20) **if** (**str**(dataset**.**values[i, j]) **!=** "nan")])

* **dataset.values[i, j]:** We cannot take the values of the data set as ***dataset[i, j]*** we need to add remember ***.values***.
* Apriori is also expecting the items as strings so we need to convert the data in to string when we creating the lists ***str(dataset.values[i, j]***. To create list we used **[]** braces.





* So as you can see this list contains ***7501 lists*** and ***each*** ***list*** corresponds to one ***transaction***.
* Training the model: We import ***apriori*** class from ***apyori*** package/api. We create our object and name it ***rUles*** , because this *class* take ***list of lists*** as input and output the ***rules***. This rules needs to converted into list so we used **list**(rUles).
* ***apriori*** is python 2.7 module so this is not working on python3. The updated module ***apyori.py*** needs to put/copy in the working directory
* Parameters: These arguments will actually depend on your business problem, depend on the ***number of observations*** you have in your ***data-set***. Of course your ***minimum*** ***support, confidence*** and ***lift*** is not going to be the same whether you have 1000 transactions or 100000 transaction***s***.

#*Train the apriori model*

**from** apyori **import** apriori

rUles= **apriori**(transactions= transacTions, min\_support=0.003, min\_confidence = 0.2, min\_lift=3, min\_length=2, max\_length=2)

results= **list**(rUles)

**for** asocitn\_item **in** results:

**print**(f"{asocitn\_item}\n")

In the above code, the first line is to import the apriori function. In the second line, the apriori function returns the output as the rules. It takes the following parameters:

* ***transactions:*** A list of transactions. We pass our list of lists data-set ***transacTions***.
* ***min\_support=*** To set the minimum support float value. Here we have used 0.003 that is calculated by taking 3 transactions per customer each week to the total number of transactions.
* We need to look at the products that are purchased rather frequently like at least three or four times a day.
* It depends on your business goal. Now assume, we find some strong rules about items that are bought at least three or four times a day then by associating them and placing them together customers will be more likely to put them in their basket and therefore more of these products will be purchased and therefore the sales will increase.
* Now, to set the minimum support we are going to consider the products that are purchased three or four times a day and then we will look at the rules. And of course if we're ***not*** ***convinced/satisfied*** by the ***rules*** we will change this value of the minimum support later.
* Now if a product is bought 3 times per day then in 1 week it is . And we have total 7500 transactions. Then

So all the products of our rules will have a higher support than this support here.

* ***min\_confidence:*** To set the minimum confidence value. Here we have taken 0.2. It can be changed as per the business problem.
* confidence of 0.8 means that the rules has to be correct in 80 percent of the time.
* But if you get some rules containing some products for example Mineral-water and Eggs. In hot day people bought them frequently. But these two products has no connection between them. So they end up in the same basket. Right reason: not because they associate well together but because they're purchased all the time.
* There is no logical association between these two products and that's why it's not very relevant. And unfortunately that's what we'll get if we set the confidence too high.
* Hence we set the confidence 20% i.e. min\_confidence = 0.2.
* ***min\_lift=*** To set the minimum lift value.
* We also try ***different*** ***values*** of the ***minimum lift***.
* So for now we try to get some rules that have lift above 3. Well these are actually some good rules because you know the left is a great insight of the relevance and the strength of rule. We're hoping to find some rules having lift to four, five or even six.
* And remember those parameters/arguments depends on your *business* *problem*, on the number of *observations* in your data set. So you might spend a little time on this choice.
* ***min\_length=*** It takes the minimum number of products for the association. *Minimum number of products* we want to have in our rules. We want minimum 2 products that a customer would buy.
* ***max\_length =*** It takes the maximum number of products/items for the association. i.e. maximum items in our rules. We can left it empty. Here we set it 2 because we want *pair-of-items* in our rules.
* Visualizing the result: Now we will visualize the output for our apriori model. Here we will follow some more steps, which are given below:
* Now we first train our model for ***min\_length=2, and max\_length=2.*** Then we get 8 rules.

#*visualizing the rules*

**for** item **in** results:

    pair = item[0]

    items = [x **for** x **in** pair]

**print**("Rule: " + items[0] + " -> " + items[1])

**print**("Support: " + **str**(item[1]))

**print**("Confidence: " + **str**(item[2][0][2]))

**print**("Lift: " + **str**(item[2][0][3]))

**print**("=====================================")

The rules

|  |
| --- |
| RelationRecord(items=frozenset({'light cream', 'chicken'}), ***support***=0.004532728969470737, ordered\_statistics=[OrderedStatistic(items\_base=frozenset({'light cream'}), items\_add=frozenset({'chicken'}), ***confidence***=0.29059829059829057, ***lift***=4.84395061728395)])  RelationRecord(items=frozenset({'mushroom cream sauce', 'escalope'}), ***support***=0.005732568990801226, ordered\_statistics=[OrderedStatistic(items\_base=frozenset({'mushroom cream sauce'}), items\_add=frozenset({'escalope'}), ***confidence***=0.3006993006993007, ***lift***=3.790832696715049)])  RelationRecord(items=frozenset({'pasta', 'escalope'}), ***support***=0.005865884548726837, ordered\_statistics=[OrderedStatistic(items\_base=frozenset({'pasta'}), items\_add=frozenset({'escalope'}), ***confidence***=0.3728813559322034, ***lift***=4.700811850163794)])  RelationRecord(items=frozenset({'honey', 'fromage blanc'}), ***support***=0.003332888948140248, ordered\_statistics=[OrderedStatistic(items\_base=frozenset({'fromage blanc'}), items\_add=frozenset({'honey'}), ***confidence***=0.2450980392156863, ***lift***=5.164270764485569)])  RelationRecord(items=frozenset({'ground beef', 'herb & pepper'}), ***support***=0.015997866951073192, ordered\_statistics=[OrderedStatistic(items\_base=frozenset({'herb & pepper'}), items\_add=frozenset({'ground beef'}), ***confidence***=0.3234501347708895, ***lift***=3.2919938411349285)])  RelationRecord(items=frozenset({'ground beef', 'tomato sauce'}), ***support***=0.005332622317024397, ordered\_statistics=[OrderedStatistic(items\_base=frozenset({'tomato sauce'}), items\_add=frozenset({'ground beef'}), ***confidence***=0.3773584905660377, ***lift***=3.840659481324083)])  RelationRecord(items=frozenset({'olive oil', 'light cream'}), ***support***=0.003199573390214638, ordered\_statistics=[OrderedStatistic(items\_base=frozenset({'light cream'}), items\_add=frozenset({'olive oil'}), ***confidence***=0.20512820512820515, ***lift***=3.1147098515519573)])  RelationRecord(items=frozenset({'whole wheat pasta', 'olive oil'}), ***support***=0.007998933475536596, ordered\_statistics=[OrderedStatistic(items\_base=frozenset({'whole wheat pasta'}), items\_add=frozenset({'olive oil'}), ***confidence***=0.2714932126696833, ***lift***=4.122410097642296)])  RelationRecord(items=frozenset({'shrimp', 'pasta'}), ***support***=0.005065991201173177, ordered\_statistics=[OrderedStatistic(items\_base=frozenset({'pasta'}), items\_add=frozenset({'shrimp'}), ***confidence***=0.3220338983050847, ***lift***=4.506672147735896)]) |

* Notice the above structure:

|  |
| --- |
| RelationRecord( items=frozenset({'light cream', 'chicken'}),  ***support***=0.004532728969470737,  ordered\_statistics= [ OrderedStatistic(items\_base=frozenset({'light cream'}),  items\_add=frozenset({'chicken'}),  ***confidence***=0.29059829059829057,  ***lift***=4.84395061728395)  ]  ) |

* It is a ***tuple*** form, contains 1st item as pair.
* 2nd element as float
* 3rd element as list
* Hence we can access the required items and format them as we want. The code given above.
* Finally we didn't set any ***max\_length***.

rUles= **apriori**(transactions = transacTions, min\_support=0.003, min\_lift = 3, min\_confidence=0.2, min\_length = 2)

* The final code are given below:

Practiced version

#*-------- Association rule :: Apriori --------------*

**import** pandas **as** pd

**import** matplotlib**.**pyplot **as** plt

**import** numpy **as** np

#*importing data*

dataset = pd**.read\_csv**("Market\_Basket\_Optimisation.csv", header = **None**)

#*creating "List of Lists": List of product-set*

#*there are 7500 rows of transections and 20 columns of product*

transacTions = []

**for** i **in** **range**(0, 7501):

    #*notice :: "List comprehension is used". Items are converted to strings*

    #*transacTions.append([str(dataset.values[i, j]) for j in range(0, 20)])*

    transacTions**.append**([**str**(dataset**.**values[i, j]) **for** j **in** range(0, 20) **if** (**str**(dataset**.**values[i, j]) **!=** "nan")])

#*Train the apriori model*

**from** apyori **import** apriori

#*rUles= apriori(transactions= transacTions, min\_support=0.003, min\_confidence = 0.2, min\_lift=3, min\_length=2, max\_length=2)*

rUles= **apriori**(transactions = transacTions, min\_support=0.003, min\_lift = 3, min\_confidence=0.2, min\_length = 2)

results= **list**(rUles)

**for** asocitn\_item **in** results:

**print**(f"{asocitn\_item}\n")

#*visualizing the rules*

**for** item **in** results:

    pair = item[0]

    items = [x **for** x **in** pair]

    rul = "Rule: "

**for** itm **in** items:

        rul += (itm + " -> " )

**print**(rul)

**print**(f"Rule: {items}")

**print**("Support: " + **str**(item[1]))

**print**("Confidence: " + **str**(item[2][0][2]))

**print**("Lift: " + **str**(item[2][0][3]))

**print**("=====================================")

#*python prctc\_apriori.py*

* As a result: Following are first few rules that have ***lift*** more than ***4***. And we got final 77 total rules.

Rule: light cream -> chicken ->

Rule: ['light cream', 'chicken']

Support: 0.004532728969470737

Confidence: 0.29059829059829057

Lift: 4.84395061728395

=====================================

Rule: escalope -> mushroom cream sauce ->

Rule: ['escalope', 'mushroom cream sauce']

Support: 0.005732568990801226

Confidence: 0.3006993006993007

Lift: 3.790832696715049

=====================================

Rule: escalope -> pasta ->

Rule: ['escalope', 'pasta']

Support: 0.005865884548726837

Confidence: 0.3728813559322034

Lift: 4.700811850163794

=====================================

Rule: honey -> fromage blanc ->

Rule: ['honey', 'fromage blanc']

Support: 0.003332888948140248

Confidence: 0.2450980392156863

Lift: 5.164270764485569

=====================================

Rule: ground beef -> herb & pepper ->

Rule: ['ground beef', 'herb & pepper']

Support: 0.015997866951073192

Confidence: 0.3234501347708895

Lift: 3.2919938411349285

=====================================

Rule: ground beef -> tomato sauce ->

Rule: ['ground beef', 'tomato sauce']

Support: 0.005332622317024397

Confidence: 0.3773584905660377

Lift: 3.840659481324083

=====================================

Rule: light cream -> olive oil ->

Rule: ['light cream', 'olive oil']

Support: 0.003199573390214638

Confidence: 0.20512820512820515

Lift: 3.1147098515519573

=====================================

Rule: olive oil -> whole wheat pasta ->

Rule: ['olive oil', 'whole wheat pasta']

Support: 0.007998933475536596

Confidence: 0.2714932126696833

Lift: 4.122410097642296

=====================================

Rule: shrimp -> pasta ->

Rule: ['shrimp', 'pasta']

Support: 0.005065991201173177

Confidence: 0.3220338983050847

Lift: 4.506672147735896

=====================================

. . . . .

. . . . .

* Analyzing the rules: From the above output, we can analyze each rule. The first rules, which is ***Light cream → chicken***, states that the light cream and chicken are bought frequently by most of the customers. The ***support*** for this rule is ***0.0045***, and the ***confidence*** is ***29%***. Hence, if a customer buys ***light cream***, it is ***29% chances*** that he also buys ***chicken***, and it is ***.0045*** times appeared in the ***transactions***. We can check all these things in other rules also.
* Conclusion: Usually we try several values of the parameters here which will give us some rules and then *experiment* *these* *rules* in *real* *life* and then according to the results we can update the parameters and get more appropriate parameters.
* Of course data-scientists can combined these rules with other recommendation system techniques like Collaborative Filtering with you know the user profiles that can add some additional relevant info and also other more advanced techniques like the Neighborhood Model or Latent Factor Models.

Well they combine a lot of models to increase the sales and the revenue.