

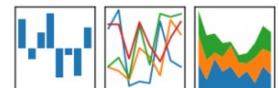
# Association Rule Mining

Tushar B. Kute,  
<http://tusharkute.com>



pandas

$$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$$



MLXTEND

python

```
In [22]: rules = association_rules(frequent_itemsets, metric='lift', min_threshold=1)
rules
```

```
Out[22]:
```

	antecedents	consequents	support	confidence	lift
0	(PLASTERS IN TIN WOODLAND ANIMALS)	(PLASTERS IN TIN CIRCUS PARADE)	0.27056	0.28755	3.56987
1	(PLASTERS IN TIN CIRCUS PARADE)	(PLASTERS IN TIN WOODLAND ANIMALS)	0.26897	0.80861	3.54967
2	(PLASTERS IN TIN CIRCUS PARADE)	(PLASTERS IN TIN SPACEBOY)	0.26897	0.53003	3.84967
3	(PLASTERS IN TIN SPACEBOY)	(PLASTERS IN TIN CIRCUS PARADE)	0.23755	0.64648	3.84967
4	(PLASTERS IN TIN WOODLAND ANIMALS)	(PLASTERS IN TIN SPACEBOY)	0.27816	0.63941	4.46233
5	(PLASTERS IN TIN SPACEBOY)	(PLASTERS IN TIN WOODLAND ANIMALS)	0.23755	0.78028	4.46233

# The Association Rules

- There are many ways to see the similarities between items.
- These are techniques that fall under the general umbrella of association.
- The outcome of this type of technique, in simple terms, is a set of rules that can be understood as “if this, then that”.
- Association Rule Mining is a Data Mining technique that finds patterns in data.
- The patterns found by Association Rule Mining represent relationships between items. When this is used with sales data, it is referred to as Market Basket Analysis.

# Market Basket Analysis



# Market Basket Analysis



Customer 1



Customer 2



Customer 3



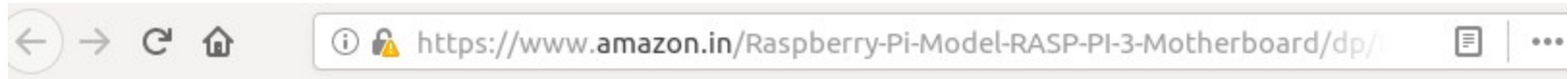
Customer n

NOTE

# Applications

- So what kind of items are we talking about? There are many applications of association:
  - Product recommendation – like Amazon’s “customers who bought that, also bought this”
  - Music recommendations – like Last FM’s artist recommendations
  - Medical diagnosis – like with diabetes really cool stuff
  - Content optimization – like in magazine websites or blogs
- Here, we will focus on the retail application – it is simple, intuitive, and the dataset comes packaged with R making it repeatable.

# Example



## Frequently bought together



Total price: ₹4,235.00

Add all three to Cart

 These items are dispatched from and sold by different sellers. [Show details](#)

- ✓ **This item:** Raspberry PI Model B RASP-PI-3 Motherboard ₹3,688.00
- ✓ Raspberry PI 3 Official Case - Black/Grey ₹332.00
- ✓ ELEMENTZ 5V 2A DC USB Adapter Charger for Raspberry PI 3 Model B / RPi 2 Model B / B+ / A+ with USB... ₹215.00

# Why Association Rules?

- It helps businesses build sales strategies.
  - Ultimately, the main objective of any business is to become profitable. This means, attracting more customers and improving their sales.
  - By identifying products that sell better together, they can build better strategies. For instance, knowing that people who buy fries almost always buy Coke can be exploited to drive up sales.

# Why Association Rules?

- It helps businesses build marketing strategies.
  - Attracting customers is a very important part of any business. Knowledge of what products sell together and which products don't is key in building marketing strategies.
  - This includes the planning of sales and advertisements as well as targeted marketing. For example, the knowledge that some ornaments do not sell as well others during Christmas may help the manager offer a sale on the non-frequent ornaments.



# Why Association Rules?

- It helps shelf-life planning.
  - Knowledge of association rules can enable store managers to plan their inventory as well as ensure that they don't lose out by overstocking low-selling perishables.
  - For instance, if olives don't sell very often, the manager will not stock up on it. But he still wants to ensure that the existing stock sells before the expiration date. With the knowledge that people who buy pizza dough tend to buy olives, the olives can be offered at a lower price in combination with the pizza dough.

# Why Association Rules?

- It helps the in-store organization.
  - Products which are known to drive the sales of other products can be moved closer together in the store.
  - For instance, if the sale of butter is driven by the sale of bread, they can be moved to the same aisle in the store.

# The conceptualization

- We already discussed the concept of Items and Item Sets.
- We can represent our items as an item set as follows:

$$i = \{ i_1, i_2, \dots, i_n \}$$

- Therefore a transaction is represented as follows:

$$t_n = \{ i_j, i_k, \dots, i_n \}$$

- This gives us our rules which are represented as follows:

$$\{ i_1, i_2 \} \Rightarrow \{ i_k \}$$

# The associations

- Which can be read as “if a user buys an item in the item set on the left hand side, then the user will likely buy the item on the right hand side too”. A more human readable example is:

$$\{\text{coffee}, \text{sugar}\} \Rightarrow \{\text{milk}\}$$

- If a customer buys coffee and sugar, then they are also likely to buy milk.
- With this we can understand three important ratios; the support, confidence and lift.

# Steps

- Step 1: Find all frequent itemsets.
- Step 2: Generate strong association rules from the frequent itemsets.

# Find all frequent itemsets.

- An itemset is a set of items that occurs in a shopping basket.
- A set of items in a shopping basket can be referred to as an itemset. It can consist of any number of products. For example, [bread, butter, eggs] is an itemset from a supermarket database.
- A frequent itemset is one that occurs frequently in a database. This begs the question of how frequency is defined. This is where support count comes in.
- The support count of an item is defined as the frequency of the item in the dataset.

# Support

- This says how popular an itemset is, as measured by the proportion of transactions in which an itemset appears.
- In Table 1 below, the support of {apple} is 4 out of 8, or 50%. Itemsets can also contain multiple items. For instance, the support of {apple, beer, rice} is 2 out of 8, or 25%.

$$\text{Support } \{\text{🍎}\} = \frac{4}{8}$$

Transaction 1	🍎 🍺 🍚 🍗
Transaction 2	🍎 🍺 🍚
Transaction 3	🍎 🍺
Transaction 4	🍎 🍏
Transaction 5	🍼 🍺 🍚 🍗
Transaction 6	🍼 🍺 🍚
Transaction 7	🍼 🍺
Transaction 8	🍼 🍏

# Defining Support

- Defining support as percentage helps us set a threshold for frequency called `min_support`. If we set support at 50%, this means that we define a frequent itemset as one that occurs at least 50 times in 100 transactions. For instance, for the above dataset, we set `threshold_support` at 60%.

$$\begin{array}{ccccc} \text{60\% minimum} & \Rightarrow & \text{60\% of (total \# of} & \Rightarrow & 0.6 \times 5 = 3 \\ \text{support} & & \text{transactions)} & & \end{array}$$

For an Itemset to be frequent, it should occur at least 3 times in 5 transactions in the given dataset.

- We always eliminate those items whose support is less than `min_support` as is seen from the greyed-out parts of the table above. The generation of frequent itemsets depends on the algorithm used.



# Generate Rules

- Generate strong association rules from the frequent itemsets.
- Association rules are generated by building associations from frequent itemsets generated in step 1.
- This uses a measure called confidence to find strong associations.

# The associations properties

- *Support*: The fraction of which our item set occurs in our dataset.
- *Confidence*: probability that a rule is correct for a new transaction with items on the left.
- *Lift*: The ratio by which by the confidence of a rule exceeds the expected confidence.
- **Note**: if the lift is 1 it indicates that the items on the left and right are independent.

# Confidence

- This says how likely item Y is purchased when item X is purchased, expressed as  $\{X \rightarrow Y\}$ . This is measured by the proportion of transactions with item X, in which item Y also appears.
- In Table 1, the confidence of  $\{\text{apple} \rightarrow \text{beer}\}$  is 3 out of 4, or 75%.

$$\text{Confidence } \{\text{🍏} \rightarrow \text{🍺}\} = \frac{\text{Support } \{\text{🍏}, \text{🍺}\}}{\text{Support } \{\text{🍏}\}}$$

- This says how likely item Y is purchased when item X is purchased, while controlling for how popular item Y is.
- In Table 1, the lift of {apple -> beer} is 1, which implies no association between items.
- A lift value greater than 1 means that item Y is likely to be bought if item X is bought, while a value less than 1 means that item Y is unlikely to be bought if item X is bought.

$$\text{Lift} \{ \text{🍎} \rightarrow \text{🍺} \} = \frac{\text{Support} \{ \text{🍎}, \text{🍺} \}}{\text{Support} \{ \text{🍎} \} \times \text{Support} \{ \text{🍺} \}}$$

# Types of algorithm

- Apriori Algorithm
- Eclat Algorithm
- F-P Growth Algorithm

# 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.

# 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.

# 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.



# Apriori Algorithm

- The Apriori algorithm is considered one of the most basic Association Rule Mining algorithms. It works on the principle that “ Having prior knowledge of frequent itemsets can generate strong association rules. ” The word Apriori means prior knowledge.
- Apriori finds the frequent itemsets by a process called candidate itemset generation. This is an iterative approach, where  $k$ -itemsets are used to explore  $(k+1)$ -itemsets. First, the set of frequent 1-itemsets is found, then, frequent 2-itemsets, and so on, until no more frequent  $k$ -itemsets can be found.
- A Candidate  $k$ -itemset is an itemset with  $k$  items in it. Example: Candidate 2-itemset can be [bread, butter].

# Apriori Algorithm

- To improve the efficiency of the level-wise generation of frequent itemsets, an important property called the Apriori property, is used to reduce the search space.
- The Apriori Property states that “All non-empty subsets of a frequent itemset must also be frequent.”
- This means that if there is a frequent item then, its subsets will also be frequent. For instance, if [Bread, Butter] is a frequent itemset, it means that [Bread] and [Butter] must individually be frequent too.

# Example:

Tid	Basket
1	[ A, B, C ]
2	[ A, C ]
3	[ A, D ]
4	[B, E, F]

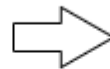
# Apriori Algorithm Steps

- Step 1: Set a minimum support and confidence threshold.
- Step 2: Generate candidate itemsets.
- Step 3: Mine Association Rules

# Apriori Algorithm Steps

- Step 1: Set a minimum support and confidence threshold.
- We set the threshold as 50% implying that we define an itemset as frequent if it occurs at least once in 2 transactions. Confidence is introduced before.

50% minimum  
support, confidence



$$0.5 \times 4 = 2$$

Hence, Minimum Support Count = 2  
Minimum Confidence = 2

# Apriori Algorithm Steps

- Step 2: Generate candidate itemsets.
- The candidate 1-itemsets consist of all individual products and their support counts respectively. For instance, [A] occurs in 3 out of 4 transactions.
- The greyed out rows represent itemsets whose support counts do not meet the threshold requirement.
- L1: [A], [B], [C]

**Candidate 1-itemsets**

Items	Support Count
A	3
B	2
C	2
D	1
E	1
F	1

# Apriori Algorithm Steps

- The candidate 2-itemsets consists of all possible 2 item set combinations of L1 and their respective support counts. For instance, [A, C] occur together in 2 out of 4 transactions.

L2: [A,C]

- Candidate 3-itemsets are to be generated from L2, containing all 3-item combinations. However, at L2 we are left with just 2 items and we cannot generate candidate 3-itemsets.

**Candidate 2-itemsets**

Items	Support
A,B	1
B,C	1
A,C	2

# Apriori Algorithm Steps

- To mine Association Rules from candidate itemsets, a measure called confidence is used. It is simply defined as an association rule between items.
- Consider an itemset [ Bread, Butter ]. Two possible scenarios can be considered here:
  - 1. People who buy bread, also buy butter. The sale of butter is driven by that of bread. This makes sense because any dish involving bread often involves butter.
  - 2. People who buy butter, also buy bread. The sale of bread is driven by butter. This does not make sense as butter can be used for anything and not just with bread.



# Apriori Algorithm Steps

- The confidence measure helps identify which product drives the sale of which other product.
- For any two products, A drives B represented as  $\{A \Rightarrow B\}$  is not the same as B drives A,  $\{B \Rightarrow A\}$ . If the confidence of an association rule  $\{A \Rightarrow B\}$  is 60%, it means that 60% of the transactions containing A also contain B together.

# Apriori Algorithm Steps

- Consider the candidate itemset output at L2: [ A, C ]. These are the frequent itemsets with support of 50%.

Rules	Support	Confidence	%
$A \Rightarrow C$ A drives the sale of C.	2	Confidence = $\frac{\text{Support}(A, C)}{\text{Support}(A)}$ $= \frac{2}{3} = 0.66$	60%
$C \Rightarrow A$ C Drives Sale Of A.	2	Confidence = $\frac{\text{Support}(A, C)}{\text{Support}(C)}$ $= \frac{2}{2} = 1.00$	100%

- Since both rules have confidence greater than 50%, both are accepted. However,  $\{C \Rightarrow A\}$  occurs with confidence 100% implying that on most occasions, C drives the sale of A.

# Drawbacks

- Apriori suffers from two main drawbacks which restrict its usage in real-world use-cases:
- 1. It is computationally intensive.
  - Apriori requires repeated scans of the database for itemset generation. This is very resource-intensive and time-consuming.
- 2. It can mine misleading patterns.
  - Apriori and other Association Rule Mining algorithms are known to produce rules that are a product of chance.

# Practical: Packages Needed

- The Apriori algorithm and Association Rules:
  - `sudo pip install mlxtend`
- The basic data analytics:
  - `sudo pip install pandas`
- The visualizations:
  - `sudo pip install matplotlib`
- Importing csv file formats (already installed)
  - `CSV`

# Structured Transactions

	A	B	C	
1	Bread	Butter	Dairy	
2	Nachos	Butter	Dairy	
3	Juice	Jam	Egg	
4	Juice	Jam	Egg	
5	Juice	Vegetable	Salad	
6	Bread	Butter	Egg	
7	Nachos	Salsa	Dairy	
8	Bread	Jam	Egg	
9	Juice	Jam	Egg	
10	Fruits	Vegetable	Salad	
11	Bread	Butter	Egg	
12	Bread	Butter	Egg	
13	Nachos	Salsa	Dairy	
14	Juice	Jam	Egg	
15	Juice	Jam	Egg	

retails.csv

# Non-structured transactions

	A	B	C	D	E
1	citrus fruit	semi-finished bread	margarine	ready soups	
2	tropical fruit	yogurt	coffee		
3	whole milk				
4	pip fruit	yogurt	cream cheese	meat spreads	
5	other vegetables	whole milk	condensed milk	long life bakery product	
6	whole milk	butter	yogurt	rice	abrasive cleaner
7	rolls/buns				
8	other vegetables	<u>UHT-milk</u>	rolls/buns	bottled beer	liquor (appetizer)
9	pot plants				
10	whole milk	cereals			
11	tropical fruit	other vegetables	white bread	bottled water	chocolate
12	citrus fruit	tropical fruit	whole milk	butter	curd
13	beef				
14	frankfurter	rolls/buns	soda		
15	chicken	tropical fruit			
16	butter	sugar	fruit/vegetable juice	newspapers	
17	fruit/vegetable juice				
18	packaged fruit/vegetables				
19	chocolate				
20	<u>specialty bar</u>				
21	other vegetables				
22	butter milk	pastry			

groceries.csv

# Reading the csv file (both)

```
import pandas as pd
import matplotlib.pyplot as plt
import csv
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori

dataset = []
with open('retails.csv', 'r') as csvfile:
    reader = csv.reader(csvfile, delimiter=',')
    for row in reader:
        dataset += [row]
```

# Transaction encoding

```
te = TransactionEncoder()
te_ary = te.fit(dataset).transform(dataset)
df = pd.DataFrame(te_ary, columns=te.columns_)
frequent_itemsets = apriori(df, min_support=0.001,
use_colnames=True)

print frequent_itemsets[1:10]
```

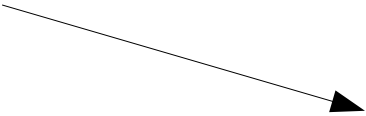


# The Transaction Encoder

- Encodes database transaction data in form of a Python list of lists into a NumPy array.
- Using and TransactionEncoder object, we can transform this dataset into an array format suitable for typical machine learning APIs.
- Via the fit method, the TransactionEncoder learns the unique labels in the dataset, and via the transform method, it transforms the input dataset (a Python list of lists) into a one-hot encoded NumPy boolean array:

# Example:

```
from mlxtend.preprocessing import TransactionEncoder
dataset = [['Apple', 'Beer', 'Rice', 'Chicken'],
           ['Apple', 'Beer', 'Rice'],
           ['Apple', 'Beer'],
           ['Apple', 'Bananas'],
           ['Milk', 'Beer', 'Rice', 'Chicken'],
           ['Milk', 'Beer', 'Rice'],
           ['Milk', 'Beer'],
           ['Apple', 'Bananas']]
te = TransactionEncoder()
te_ary = te.fit(dataset).transform(dataset)
print te_ary
```



```
[[ True False  True  True False  True]
 [ True False  True  False False  True]
 [ True False  True  False False  False]
 [ True  True False  False False  False]
 [False False  True  True  True  True]
 [False False  True  False  True  True]
 [False False  True  False  True  False]
 [ True  True False  False False  False]]
```

# Transaction Encoder

```
print te_ary.astype("int")
```

```
print te.columns_
```

```
['Apple', 'Bananas', 'Beer', 'Chicken', 'Milk', 'Rice']
```

```
import pandas as pd
```

```
df = pd.DataFrame(te_ary, columns=te.columns_)
```

```
print df
```

```
[[1 0 1 1 0 1]
 [1 0 1 0 0 1]
 [1 0 1 0 0 0]
 [1 1 0 0 0 0]
 [0 0 1 1 1 1]
 [0 0 1 0 1 1]
 [0 0 1 0 1 0]
 [1 1 0 0 0 0]]
```

	Apple	Bananas	Beer	Chicken	Milk	Rice
0	True	False	True	True	False	True
1	True	False	True	False	False	True
2	True	False	True	False	False	False
3	True	True	False	False	False	False
4	False	False	True	True	True	True
5	False	False	True	False	True	True
6	False	False	True	False	True	False
7	True	True	False	False	False	False

# Getting the association rules

```
from mlxtend.frequent_patterns import association_rules

# Get the rules by confidence
rules = association_rules(frequent_itemsets,
metric="confidence", min_threshold=0.9)
# Print first five rules
print rules.head()
# Get the rules by lift
rules = association_rules(frequent_itemsets,
metric="lift", min_threshold=1.2)
# Print first five rules
print rules.head()
```

# Output:

	antecedents	consequents	antecedent support
0	(Vegetable, Fruits)	(Salad)	0.1705
1	(Salad, Fruits)	(Vegetable)	0.1825
2	(Fruits)	(Vegetable, Salad)	0.1825
3	(Vegetable)	(Salad)	0.2160
4	(Salad)	(Vegetable)	0.2210

[5 rows x 9 columns]

	antecedents	consequents	antecedent support
0	(Vegetable, Salad)	(Fruits)	0.2090
1	(Vegetable, Fruits)	(Salad)	0.1705
2	(Salad, Fruits)	(Vegetable)	0.1825
3	(Vegetable)	(Salad, Fruits)	0.2160
4	(Salad)	(Vegetable, Fruits)	0.2210

# Conditional Rules

**# Rules by antecedent length**

```
rules["antecedent_len"] = rules["antecedents"].apply  
(lambda x: len(x))  
print rules[1:10]
```

**# Multiple conditions**

```
nrules = rules[ (rules['antecedent_len'] >= 2) &  
                (rules['confidence'] > 0.8) &  
                (rules['lift'] > 1.2) ]  
print nrules[1:10]
```

```
#nrules = rules[rules['consequents'] == {'whole milk'}]  
nrules = rules[rules['consequents'] == {'Egg'}]  
print nrules[1:10]
```

# Operations on Rules

```
nrules = rules[rules['consequents'] == {'Egg'}]  
print nrules[1:10]
```

**# Selective rules**

```
nrules = rules[rules['antecedents'] == {'Juice', 'Jam'}]  
print nrules[1:10]
```

**# Selective columns information**

```
nrules = rules[['antecedents', 'consequents', 'confidence']]  
print nrules[1:10]
```

**# Sorting the rules by confidence**

```
nrules = nrules[nrules['confidence'] >= 0.8 ]  
print nrules.sort_values(by='confidence', ascending=False)
```

# Applications

- Cross Selling
- Product Placement
- Affinity Promotion
- Fraud Detection
- Customer Behavior



# Useful resources

- <https://rasbt.github.io>
- <https://www.kdnuggets.com>
- <http://intelligentonlinetools.com>
- <http://pbpython.com>
- [www.towardsdatascience.com](http://www.towardsdatascience.com)
- [www.analyticsvidhya.com](http://www.analyticsvidhya.com)
- [www.kaggle.com](http://www.kaggle.com)
- [www.github.com](http://www.github.com)

# Thank you

*This presentation is created using LibreOffice Impress 5.1.6.2, can be used freely as per GNU General Public License*



@mitu\_skillologies



/mITuSkillologies



@mitu\_group



/company/mitu-  
skillologies



c/MITUSkillologies

## Web Resources

<http://mitu.co.in>

<http://tusharkute.com>

[contact@mitu.co.in](mailto:contact@mitu.co.in)

[tushar@tusharkute.com](mailto:tushar@tusharkute.com)