Association Rules

Association Rules

- Association rules is a descriptive, not predictive, method often used to discover interesting relationships hidden in a large data set.
- The disclosed relationships can be represented as rules or frequent itemsets.
- Association rules are commonly used for mining transactions in databases.

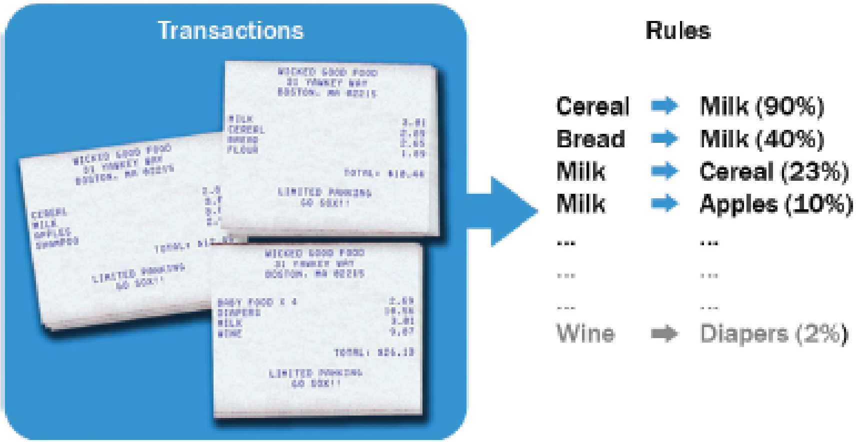
Association Rules

- For example, given a large collection of retail transactions, in which each transaction consists of one or more items, association rules go through the items being purchased to see what items are frequently bought together and to discover a list of rules that describe the purchasing behavior.

- The goal with association rules is to discover interesting relationships among the items.

- The relationships that are interesting depend both on the business context and the nature of the algorithm being used for the discovery.

Association Rules



Source: *Data Science & Big Data Analytics*

New Term: "Itemset"

- In the example of a retail store, association rules are used over transactions that consist of one or more items.
- In fact, because of their popularity in mining customer transactions, association rules are sometimes referred to as **market basket analysis**.
- Each transaction can be viewed as the shopping basket of a customer that contains one or more items.
- This is also known as an **itemset**.
- The term *itemset* refers to a collection of items or individual entities that contain some kind of relationship.

Itemset

- An itemset containing k items is called a k-itemset.
- We will use the notation $\{\text{item 1}, \text{item 2}, ..., \text{item k}\}$ to denote a k-itemset.
- Computation of the association rules is typically based on itemsets.
- We will focus on the *Apriori* algorithm for generating association rules.

The Support of an Itemset

- One major component of *Apriori* algorithm is ‘support’.
- Given an itemset L, the *support* of L is the percentage of transactions that contain L.
- For example, if 80% of all transactions contain itemset $\{bread\}$, then the support of $\{bread\}$ is 0.8.
- Similarly, if 60% of all transactions contain itemset $\{bread, \; butter\}$, then the support of $\{bread, \; butter\}$ is 0.6.

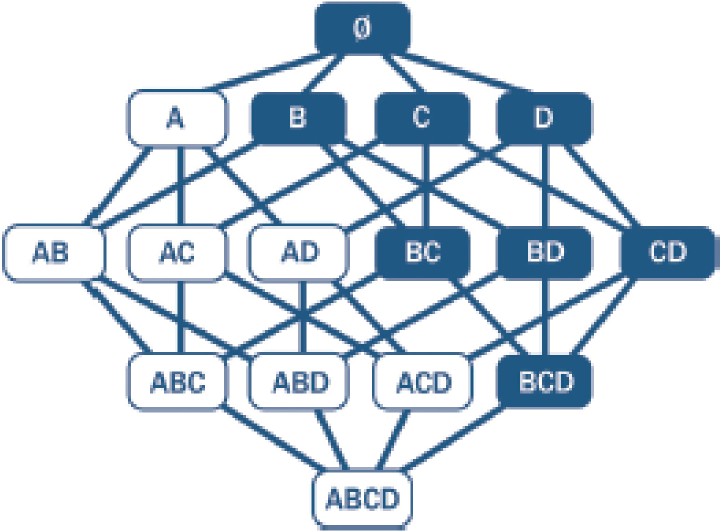
The Minimum Support

- A frequent itemset has items that appear together *often enough*.
- The term "often enough" is formally defined with a **minimum support** criterion.
- If the minimum support is set at 0.5, any itemset can be considered a frequent itemset if at least 50% of the transactions contain this itemset.
- For the previous example, both $\{bread\}$ and $\{bread,\ butter\}$ are considered frequent itemsets at the minimum support 0.5.
- If the minimum support is 0.7, only $\{bread\}$ is considered a frequent itemset.

Frequent Itemset

- If an itemset is considered **frequent**, then any subset of the frequent itemset must also be frequent.
- This is referred to as the *Apriori property* (or downward closure property).
- For example, if 60% of the transactions contain $\{bread, \ jam\}$, then at least 60% of all the transactions will contain $\{bread\}$ or $\{jam\}$.
- The *Apriori property* provides the basis for the *Apriori* algorithm.

Example



Source: *Data Science & Big Data Analytics*

Apriori Algorithm

- The Apriori algorithm takes a bottom-up iterative approach to uncovering the frequent itemsets by: **(1)** first determining all the possible items (or 1-itemsets, for example $\{bread\}$, $\{eggs\}$, $\{milk\}$,...) and **(2)** then identifying which among them are frequent based on a minimum support threshold (or the minimum support criterion).

- For example, when the minimum support threshold is set at 0.5, the algorithm identifies and retains those itemsets that appear in at least 50% of all transactions and discards (or "prunes away") the itemsets that have a support less than 0.5 or appear in fewer than 50% of the transactions.

Apriori Algorithm

- In the next iteration of the Apriori algorithm, the identified frequent 1-itemsets are paired into 2-itemsets,

- For example, $\{bread, eggs\}$, $\{bread, milk\}$, $\{eggs, milk\}$,... and again evaluated to identify which are the frequent 2-itemsets among them.

- This iterative process is repeated in the Apriori algorithm.

- At each iteration, the algorithm checks whether the support criterion can be met; if it can, the algorithm grows the itemset, repeating the process until it runs out of support or until the itemsets reach a predefined length.

Apriori Algorithm

- The growing and pruning process is repeated until no itemsets meet the minimum support threshold.

- Optionally, a threshold N can be set up to specify the maximum number of items the itemset can reach or the maximum number of iterations of the algorithm.

- Once completed, output of the *Apriori* algorithm is the collection of all the frequent k-itemsets.

Apriori Algorithm

- Finally, a collection of candidate rules is formed based on the frequent k-itemsets uncovered in the iterative process described earlier.

- For example, a frequent itemset $\{milk, \ eggs\}$ may suggest candidate rules $\{milk\} \rightarrow \{eggs\}$ and $\{eggs\} \rightarrow \{milk\}$.

- We can evaluate the appropriateness of these candidate rules using measures such as **confidence**, **lift**, and **leverage**.

Confidence of a Rule

- *Confidence* is defined as the measure of certainty or trustworthiness associated with each discovered rule.

- Mathematically, the confidence for candidate rule $X \to Y$ is the percent of transactions that contain both X and Y out of all the transactions that contain X:

$$Confidence(X \to Y) = \frac{Support(X \wedge Y)}{Support(X)}$$

- Since $Support(X \wedge Y) \leq Support(X)$ always, we have $Confidence(X \to Y) \leq 1$.

Confidence of a Rule

- For example, if $\{bread, \ eggs, \ milk\}$ has a support of 0.15 and $\{bread, \ eggs\}$ also has a support of 0.15, the confidence of rule $\{bread, \ eggs\} \rightarrow \{milk\}$ is 1, which means 100% of the time a customer buys bread and eggs, milk is bought as well.

- The rule is therefore correct for 100% of the transactions containing bread and eggs.

Confidence of a Rule

- A relationship may be thought of as interesting when the algorithm identifies the relationship with a measure of confidence greater than or equal to a predefined threshold.

- This predefined threshold is called the **minimum confidence**.

- A higher confidence indicates that the rule $(X \rightarrow Y)$ is more interesting or more trustworthy, based on the sample data set.

Confidence of a Rule

- Even though confidence can identify the interesting rules from all the candidate rules, it comes with a problem.

- Given rules in the form of $X \rightarrow Y$, confidence considers only the antecedent (X) and the co-occurrence of X and Y; it does not take the consequent of the rule (Y) into concern.

- Other measures such as lift and leverage are designed to address this issue.

Lift of a Rule

- *Lift* measures how many times more often X and Y occur together than expected if they are statistically independent of each other.

- *Lift* is a measure of how X and Y are really related rather than coincidentally happening together:

$$Lift(X \to Y) = \frac{Support(X \wedge Y)}{Support(X) \times Support(Y)}.$$

- Lift is 1 if X and Y are statistically independent of each other.

- In contrast, a lift of $X \to Y$ greater than 1 indicates that there is some usefulness to the rule.

Lift of a Rule

- For example, assuming from a total of 1,000 transactions, with $\{milk, eggs\}$ appearing in 300 of them, $\{milk\}$ appearing in 500, and $\{eggs\}$ appearing in 400, then
$$Lift(milk \rightarrow eggs) = 0.3/(0.5 \times 0.4) = 1.5.$$

- If $\{bread\}$ appears in 400 transactions and $\{milk, bread\}$ appears in 400, then
$$Lift(milk \rightarrow bread) = 0.4/(0.5 \times 0.4) = 2.$$

Leverage of a Rule

- *Leverage* is a similar notion, but instead of using a ratio, leverage uses the difference:

$$Leverage(X \to Y) = Support(X \wedge Y) - Support(X) \times Support(Y)$$

- *Leverage* measures the difference in the probability of X and Y appearing together in the data set compared to what would be expected if X and Y were statistically independent of each other.
- In theory, leverage is 0 when X and Y are statistically independent of each other.
- A larger leverage absolute value indicates a stronger relationship between X and Y.

Leverage of a Rule

- For example, assuming a total of 1,000 transactions, with $\{milk, eggs\}$ appearing in 300 of them, $\{milk\}$ appearing in 500, and $\{eggs\}$ appearing in 400, then

$$Leverage\ (milk \to eggs) = 0.3 - (0.5 \times 0.4) = 0.1.$$

- If $\{bread\}$ appears in 400 transactions and $\{milk, bread\}$ appears in 400, then

$$Leverage\ (milk \to bread) = 0.4 - (0.5 \times 0.4) = 0.2.$$

Applications of Association Rules

- The term *market basket analysis* refers to a specific implementation of association rules mining that many companies use for a variety of purposes, including these:

1. Broad-scale approaches to better merchandising-what products should be included in or excluded from the inventory each month
2. Cross-merchandising between products and high-margin or high-ticket items
3. Physical or logical placement of product within related categories of products
4. Promotional programs-multiple product purchase incentives managed through a loyalty card program, etc.

Applications of Association Rules

- Besides market basket analysis, association rules are commonly used for recommender systems

Applications of Association Rules

- Many online service providers such as Amazon and Netflix use recommender systems.
- Recommender systems can use association rules to discover related products or identify customers who have similar interests.
- For example, association rules may suggest that those customers who have bought product A have also bought product B.
- These findings provide opportunities for retailers to cross-sell their products.

Applications of Association Rules

- Clickstream analysis refers to the analytics on data related to web browsing and user clicks, which is stored on the client or the server side.
- Web usage log files generated on web servers contain huge amounts of information, and association rules can potentially give useful knowledge to web usage data analysts.
- For example, association rules may suggest that website visitors who land on page X click on links A, B, and C much more often than links D, E, and F.
- This observation provides valuable insight on how to better personalize and recommend the content to site visitors.

Grocery Store Transaction

- We will look at an example illustrating the application of the *Apriori algorithm* to grocery store transaction data.

- Using R and the 'arules' and 'arulesViz' packages, this example shows how to use the *Apriori algorithm* to generate frequent itemsets and rules and to evaluate and visualize the rules.

```
> #install.packages('arules')
> #install.packages('arulesViz')
> library('arules')
> library('arulesViz')
```

Groceries Data

- The example uses the Groceries data set from the package 'arules' in R. [1]
- The Groceries dataset is collected from 30 days of real-world point-of-sale transactions of a grocery store.
- The dataset contains 9835 transactions, and the items are aggregated into 169 categories.

```
> data(Groceries)
> Groceries
transactions in sparse format with
 9835 transactions (rows) and
 169 items (columns)
```

[1] https://cran.r-project.org/web/packages/arules/arules.pdf

Groceries Data

- The summary shows that the most frequent items in the dataset include items such as whole milk, other vegetables, rolls/buns, soda, and yogurt. These items are purchased more often than the others.

```
> summary(Groceries)
transactions as itemMatrix in sparse format with
 9835 rows (elements/itemsets/transactions) and
 169 columns (items) and a density of 0.02609146

most frequent items:
      whole milk other vegetables       rolls/buns             s
            2513             1903             1809             1
          yogurt          (Other)
            1372            34055

element (itemset/transaction) length distribution:
sizes
   1    2    3    4    5    6    7    8    9   10   11   12   13
2159 1643 1299 1005  855  645  545  438  350  246  182  117   78
  17   18   19   20   21   22   23   24   26   27   28   29   32
  29   14   14    9   11    4    6    1    1    1    1    3    1
```

Groceries Data

```
> inspect(head(Groceries))
    items
[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,
```

Groceries Data

- Data set 'Groceries' is a transaction class, as defined by the 'arules' package. A transactions class has three component slots:

1. itemsetInfo: A data frame with vectors of the same length as the number of transactions
2. itemInfo: A data frame to store item labels
3. data: A binary incidence matrix that indicates which item labels appear in every transaction

Groceries Data

- `Groceries@itemInfo` display all 169 grocery labels as well as their categories.

```
> Groceries@itemInfo[1:10,]
              labels  level2           level1
1        frankfurter sausage meat and sausage
2            sausage sausage meat and sausage
3         liver loaf sausage meat and sausage
4                ham sausage meat and sausage
5               meat sausage meat and sausage
6  finished products sausage meat and sausage
7    organic sausage sausage meat and sausage
8            chicken poultry meat and sausage
9             turkey poultry meat and sausage
10              pork    pork meat and sausage
```

Groceries Data

- `Groceries@data` indicates which item labels appear in every transaction.
- | indicates that the item appears in transaction, and · otherwise.

```
> Groceries@data[,100:110]
169 x 11 sparse Matrix of class "ngCMatrix"

  [1,] . . . | . . . . . . .
  [2,] . . | . | . . . . . .
  [3,] . . . . . . . . . . .
  [4,] . . . . . . . . . . .
  [5,] . . . . . . . . . . .
  [6,] . . . . . . . . . . .
  [7,] . . . . . . . . . . .
  [8,] . . . . . . . . . . .
  [9,] . . . . . . . . . . .
 [10,] . . | . . . . . . . .
 [11,] . . . . . . . | . . .
 [12,] . . . . . . . . . . .
 [13,] . . . . . . . . . . .
 [14,] | . . . . . . | . . .
 [15,] | . . . . . . . . . |
```

Groceries Data

- The following code displays the 1^{st} to 5^{th} transactions of the Groceries data set, similar as from slide 36.
- $[, 1:5]$ can be changed to $[, 1:9835]$ to display all the transactions.

```
> apply(Groceries@data[,1:5], 2,
+       function(r) paste(Groceries@itemInfo[r,"labels"],
+       collapse=", "))
[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 bak
```

`apriori()` Function

- The `apriori()` function from the 'arule' package implements the *Apriori algorithm* to create frequent itemsets.
- Note that, by default, the `apriori()` function executes all the iterations at once.
- However, to illustrate how the *Apriori algorithm* works, the code examples in this section manually set the parameters of the `apriori()` function to simulate each iteration of the algorithm.

`apriori()` Function

- Assume that the **minimum support threshold** is set to **0.02** based on management discretion.

- Because the data set contains 9,835 transactions, an itemset should appear 196-197 times to be considered a frequent itemset.

- The first iteration of the *Apriori algorithm* computes the support of each product in the data set and retains those products that satisfy the minimum support.

apriori() Function

- The following code identifies 59 frequent 1-itemsets that satisfy the minimum support.

- The parameters of **apriori()** specify the minimum and maximum lengths of the itemsets, the minimum support threshold, and the target indicating the type of association mined.

apriori() to get frequent 1-itemsets

```
> itemsets.1 <- apriori(Groceries, parameter=list(minlen=1,
+     maxlen=1,support=0.02, target="frequent itemsets"))
Apriori

Parameter specification:
 confidence minval smax arem  aval originalSupport maxtime support m
         NA    0.1    1 none FALSE            TRUE       5    0.02
 maxlen            target  ext
      1 frequent itemsets TRUE

Algorithmic control:
 filter tree heap memopt load sort verbose
    0.1 TRUE TRUE  FALSE TRUE    2    TRUE

Absolute minimum support count: 196

set item appearances ...[0 item(s)] done [0.00s].
set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
sorting and recoding items ... [59 item(s)] done [0.00s].
creating transaction tree ... done [0.00s].
```

```
> summary(itemsets.1)
set of 59 itemsets

most frequent items:
frankfurter     sausage          ham         meat      chicken      (Otl
          1           1            1            1            1

element (itemset/transaction) length distribution:sizes
 1
59

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
      1       1       1       1       1       1

summary of quality measures:
    support             count
 Min.   :0.02105   Min.   : 207.0
 1st Qu.:0.03015   1st Qu.: 296.5
 Median :0.04809   Median : 473.0
 Mean   :0.06200   Mean   : 609.8
 3rd Qu.:0.07666   3rd Qu.: 754.0
 Max.   :0.25552   Max.   :2513.0
```

Most 10 Frequent 1-Itemsets

The following code uses the `inspect()` function to display the top 10 frequent 1-itemsets sorted by **their support**.

```
> inspect(head(sort(itemsets.1, by = "support"), 10))
     items              support    count
[1]  {whole milk}       0.25551601 2513
[2]  {other vegetables} 0.19349263 1903
[3]  {rolls/buns}       0.18393493 1809
[4]  {soda}             0.17437722 1715
[5]  {yogurt}           0.13950178 1372
[6]  {bottled water}    0.11052364 1087
[7]  {root vegetables}  0.10899847 1072
[8]  {tropical fruit}   0.10493137 1032
[9]  {shopping bags}    0.09852567  969
[10] {sausage}          0.09395018  924
```

`apriori()` to get frequent 2-itemsets

- In the next iteration, the list of frequent 1-itemsets is joined onto itself to form all possible candidate 2-itemsets.
- For example, 1-itemsets $\{whole\ milk\}$ and $\{soda\}$ would be joined to become a 2-itemset $\{wholemilk,\ soda\}$.
- The algorithm computes the support of each candidate 2-itemset and retains those that satisfy the minimum support.
- The output that follows shows that 61 frequent 2-itemsets have been identified.

```
> itemsets.2 <- apriori(Groceries, parameter=list(minlen=2,
+         maxlen=2, support=0.02, target="frequent itemsets"))
Apriori

Parameter specification:
 confidence minval smax arem  aval originalSupport maxtime support m
         NA    0.1    1 none FALSE            TRUE       5    0.02
 maxlen            target   ext
      2 frequent itemsets TRUE

Algorithmic control:
 filter tree heap memopt load sort verbose
    0.1 TRUE TRUE  FALSE TRUE    2    TRUE

Absolute minimum support count: 196

set item appearances ...[0 item(s)] done [0.00s].
set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
sorting and recoding items ... [59 item(s)] done [0.00s].
creating transaction tree ... done [0.00s].
checking subsets of size 1 2 done [0.00s].
sorting transactions ... done [0.00s].
```

```
> summary(itemsets.2)
set of 61 itemsets

most frequent items:
      whole milk other vegetables           yogurt       rolls/buns
              25               17                9                9
            soda          (Other)
               9               53

element (itemset/transaction) length distribution:sizes
 2
61

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
      2       2       2       2       2       2

summary of quality measures:
    support             count
 Min.   :0.02003   Min.   :197.0
 1st Qu.:0.02227   1st Qu.:219.0
 Median :0.02613   Median :257.0
 Mean   :0.02951   Mean   :290.3
```

Most 10 Frequent 2-Itemsets

The top 10 frequent 2-itemsets sorted by **their support**.

```
> inspect(head(sort(itemsets.1, by = "support"), 10))
     items              support    count
[1]  {whole milk}       0.25551601 2513
[2]  {other vegetables} 0.19349263 1903
[3]  {rolls/buns}       0.18393493 1809
[4]  {soda}             0.17437722 1715
[5]  {yogurt}           0.13950178 1372
[6]  {bottled water}    0.11052364 1087
[7]  {root vegetables}  0.10899847 1072
[8]  {tropical fruit}   0.10493137 1032
[9]  {shopping bags}    0.09852567  969
[10] {sausage}          0.09395018  924
```

- Notice that whole milk appears six times in the top 10 2-itemsets ranked by support.

- As seen earlier, whole milk has the highest support among all the 1-itemsets.

- These top 10 2-itemsets with the highest support may not be interesting; this highlights the limitations of using support alone.

`apriori()` to get frequent 3-itemsets

- Next, the list of frequent 2-itemsets is joined onto itself to form candidate 3-itemsets.
- For example $\{other\ vegetables,\ wholemilk\}$ and $\{whole\ milk,\ rolls/\ buns\}$ would be joined as $\{other\ vegetables,\ whole\ milk,\ rolls/\ buns\}$.
- The algorithm retains those itemsets that satisfy the minimum support.
- The following output shows that only **two** frequent 3-itemsets have been identified.

Frequent 3-Itemsets

```
> itemsets.3 <- apriori(Groceries, parameter=list(minlen=3,
+     maxlen=3, support=0.02, target="frequent itemsets"))
Apriori

Parameter specification:
 confidence minval smax arem  aval originalSupport maxtime support m
         NA    0.1    1 none FALSE            TRUE       5    0.02
 maxlen            target   ext
      3 frequent itemsets TRUE

Algorithmic control:
 filter tree heap memopt load sort verbose
    0.1 TRUE TRUE  FALSE TRUE    2    TRUE

Absolute minimum support count: 196

set item appearances ...[0 item(s)] done [0.00s].
set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
sorting and recoding items ... [59 item(s)] done [0.00s].
creating transaction tree ... done [0.00s].
```

Frequent 3-Itemsets

There are only **two** frequent 3-itemsets:

```
> inspect(sort(itemsets.3, by ="support"))
    items                                        support    count
[1] {root vegetables, other vegetables, whole milk} 0.02318251 228
[2] {other vegetables, whole milk, yogurt}          0.02226741 219
```

Frequent 4-Itemsets

```
> itemsets.4 <- apriori(Groceries, parameter=list(minlen=4,
+     maxlen=4,support=0.02, target="frequent itemsets"))
Apriori

Parameter specification:
 confidence minval smax arem  aval originalSupport maxtime support m
         NA    0.1    1 none FALSE            TRUE       5    0.02
 maxlen            target   ext
      4 frequent itemsets TRUE

Algorithmic control:
 filter tree heap memopt load sort verbose
    0.1 TRUE TRUE  FALSE TRUE    2    TRUE

Absolute minimum support count: 196

set item appearances ...[0 item(s)] done [0.00s].
set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
sorting and recoding items ... [59 item(s)] done [0.00s].
creating transaction tree ... done [0.00s].
```

Frequent 4-Itemsets

```
> summary(itemsets.4)
set of 0 itemsets
```

- With a minimum support of 0.02, there is NO frequent 4-itemset found.
- The iterations run out of support when $k = 4$.
- Therefore, the frequent itemsets contain: 59 frequent 1-itemsets, 61 frequent 2-itemsets, and 2 frequent 3-itemsets.
- When the `maxlen` parameter is not set, the algorithm continues each iteration until it runs out of support or until k reaches the default `maxlen`=10.

Generating Rules

- The `apriori()` function can also be used to generate rules.
- Assume that the minimum support threshold is now set to a lower value 0.001, and the minimum confidence threshold is set to 0.6.
- A lower minimum support threshold allows more rules to show up.
- The following code creates 2,918 rules from all the transactions in the `Groceries` data set that satisfy both the minimum support and the minimum confidence.

```
> rules <- apriori(Groceries, parameter=list(support=0.001,
+         confidence=0.6, target = "rules"))
Apriori

Parameter specification:
 confidence minval smax arem  aval originalSupport maxtime support m
        0.6    0.1    1 none FALSE            TRUE       5   0.001
 maxlen target  ext
     10  rules TRUE

Algorithmic control:
 filter tree heap memopt load sort verbose
    0.1 TRUE TRUE  FALSE TRUE    2    TRUE

Absolute minimum support count: 9

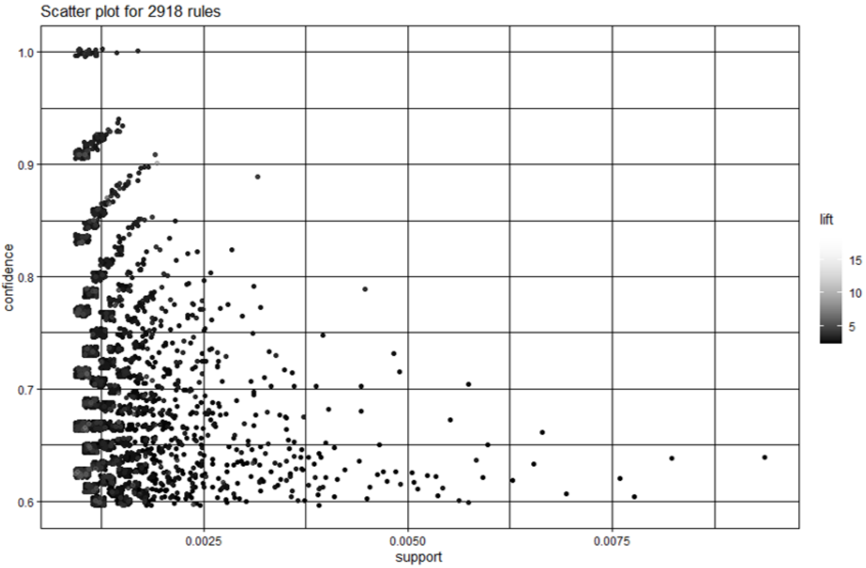
set item appearances ...[0 item(s)] done [0.00s].
set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
sorting and recoding items ... [157 item(s)] done [0.00s].
creating transaction tree ... done [0.00s].
checking subsets of size 1 2 3 4 5 6 done [0.01s].
writing ... [2918 rule(s)] done [0.00s].
```

Visualizing the Rules

- The command `plot(rules)` display the scatter plot of all the 2,918 rules, where the X-axis is the support, the Y-axis is the confidence, and the shading is the lift.

- The scatter plot shows that, of the 2,918 rules generated from the Groceries data set, the highest lift occurs at a low support and a low confidence.

```
> plot(rules, measure = c("support", "confidence"),
+       shading = "lift", col = "black")#, limit = 100)
```



Scatter plot for 2918 rules
confidence
1.0
0.9
0.8
0.7
0.6
support
0.0025
0.0050
0.0075
lift
15
10
5

Some Top Rules

- The inspect() function can display content of the rules generated previously.
- The following code shows the top three rules sorted by the lift. Rule $\{Instant\ food\ products,\ soda\} \to \{hamburger\ meat\}$ has the highest lift of ≈ 19.

```
> inspect(head(sort(rules, by="lift"), 3))
    lhs                                rhs               support
[1] {Instant food products, soda} => {hamburger meat} 0.0012201
[2] {soda, popcorn}               => {salty snack}    0.0012201
[3] {ham, processed cheese}       => {white bread}    0.0019318
    coverage    lift     count
[1] 0.001931876 18.99565 12
[2] 0.001931876 16.69779 12
[3] 0.003050330 15.04549 19
```

Top 5 Rules with Highest Lift

- We can also visualize the top five rules with the highest lift using the following code.

- ```
  > highLiftRules <- head(sort(rules, by="lift"), 5)
  > plot(highLiftRules, method = "graph", engine = "igraph",
  +     edgeCol = "blue", alpha = 1)
  ```

- The size of the node is sorted by the support.

- The darkness of the red color represents the change in lift, darker means higher lift.
  ```

Graph for 5 rules