Association Analysis
Nick Littlefield
STA 588

What is Association Analysis?
Association analysis is useful for discovering interesting relationships that are hidden in large data sets. More specifically, given a set of transactions, find rules that will predict the occurrence of an item based on the occurrences of other items in the transaction.

There are many application domains for association analysis:

- Market basket data
- Bioinformatics
- Medical diagnosis
- Scientific data analysis

Market basket transactions is one of the commonly known applications. There are huge amounts of customer purchase data collected daily at checkout counters at grocery stores. Retailers are interested in analyzing the data to learn about the purchasing behavior of their customers and then use this to help support a variety of business-related applications.

Transaction ID	Items
1	{Bread, Milk}
2	{Bread, Diapers, Beer, Eggs}
3	{Milk, Diapers, Beer, Cola}
4	{Bread, Milk, Diapers, Beer}
5	{Bread, Milk, Diapers, Cola}

Table 1: Market Basket Transaction Examples

The data shown in Table 1, is an example of market basket transactions. A **transaction** is a record of items that were bought together. The uncovered relationships can be represented in a form of **association**

rules or a set of frequent items. An example rule that can be extracted from Table 1 is:

```
{Diapers} --> {Beer}
```

When applying association analysis there are two things that need to be addressed:

1. Discovering patterns in a transaction dataset can be computationally expensive
2. Some patterns can be spurious because they may happen simply by chance

Itemsets

An **itemset** is a collection of one or more items. A *k*-itemset is one that contains *k* items.

An important property of an itemset is the support count (σ). This is the frequency of the occurrence of an itemset. Support is a fraction of transactions that contain an itemset.

A **frequent itemset** is an itemset whose support is greater than or equal to a minimum support threshold.

Association Rules

An association rule is an implication expression of the form $X \rightarrow Y$, where *X* and *Y* are disjoint itemsets, i.e. $X \cap Y = \emptyset$. To measure the strength of an association rule **support** and **confidence** are used. The support of a rule determines how often a rule is applicable to a given data set. Confidence determines how frequently items in *Y* appear in transactions that contain *X*. Formally, these metrics are defined as:

$$Support, s(X \rightarrow Y) = \frac{\sigma(X \cup Y)}{N}$$

$$Confidence, c(X \to Y) = \frac{\sigma(X \cup Y)}{\sigma(X)}$$

Support is an important measure because a rule that has very low support may occur simply by chance. Low support rules are likely to be uninteresting, from a business perspective, because it may not be profitable to promote items that customers don't buy together often. Thus, support is often used for eliminating rules.

Confidence measures the reliability of the inference that is made by a rule. For a rule $X \to Y$, the higher the confidence, the more likely it is for *Y* to be in the transactions that contain *X*. Confidence also provides an estimate of the conditional probability of *Y* given *X*.

It is important to note, that inference made from an association rule does not necessarily imply causality. It suggests a strong co-occurrence relationship between items in the antecedent and consequent of the rule. To have causality requires knowledge about the cause and effect attributes in the data and involves relationships occurring over time.

Association Rule Mining Algorithms

Given a set of transactions, *T*, the foal association rule discovery is to find all the rules that have support ≥ *minsup* and confidence ≥ *minconf*, where *minsup* and *minconf*, are the corresponding support and confidence thresholds.

Association rule mining algorithms are decomposed into two major subtasks:

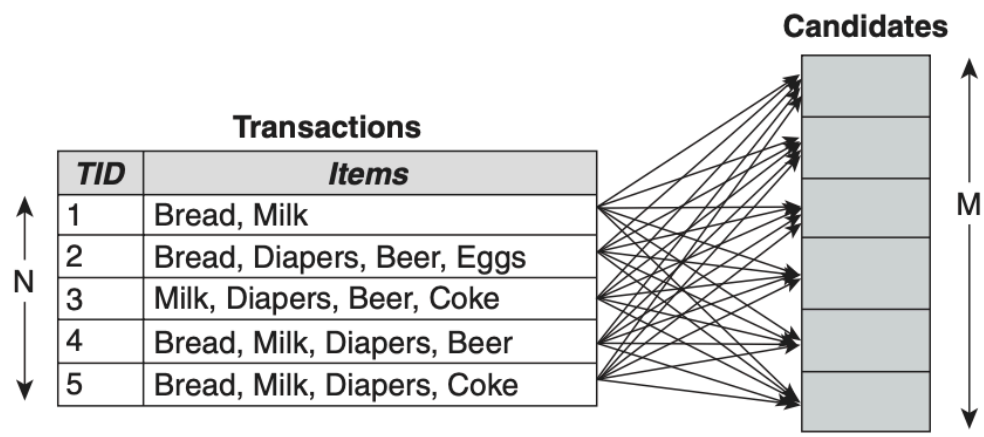
1. Frequent Itemset Generation
2. Rule Generation

During frequent itemset generation, the objective is to find all the itemsets that satisfy the *minsup* threshold. These itemsets are called frequent itemsets. During rule generation the objective is to extract all the high-confidence rules from the frequent itemsets found in the previous step. These rules are called **strong rules**.

Frequent Itemset Generation

Frequent itemset generation is computationally expensive. A dataset that contains *k* items can potentially generate up to 2^{k-1} frequent itemsets, excluding the null set. Since *k* can be very large in many practical applications, the search space of itemsets to explore is exponentially large.

The brute force approach to find frequent itemsets is to determine the support count for every candidate itemset by comparing each candidate against every transaction. This is shown in the figure below:



To reduce the computational complexity of frequent itemset generation we can:

1. Reduce the number of candidate itemsets (*M*)
2. Reduce the number of comparisons.

***Apriori* Principle**

An effective way to eliminate some of the candidate itemsets without counting their support values is by using the *Apriori* Principle. The *Apriori* principle states that if an itemset is frequent, then all of its subsets must also be frequent.

If an itemset is found to be infrequent, then all of its supersets must be infrequent too. The itemsets containing the supersets of the infrequent items can be pruned once immediately found infrequent. This strategy is known as **support-based pruning**, which is possible because the support for an itemset never exceeds the support for its subsets.

Example: Applying Apriori Principle

Using the transactions from Table 1, we can apply the Apriori principle to build frequent 1-, 2-, and 3-itemsets. We can use minimum support pruning to help pick the most frequent itemsets.

The 1-itemset is:

Item	Support Count
Bread	4
Cola	2
Milk	4
Beer	3
Diaper	4
Eggs	1

We get the support count by determining how many times the item appears in the all the transactions. Since Eggs and Cola have a support count of 2 and 1, we can eliminate these items when moving on to generate 2-itemsets. The list of all possible 2-itemset candidates are:

```
{Bread, Milk}
{Bread, Beer}
{Bread, Diapers}
{Milk, Bread}
{Milk, Beer}
{Milk, Diapers}
{Beer, Bread}
{Beer, Milk}
{Beer, Diapers}
```

```
{Diapers, Bread}
{Diapers, Milk}
{Diapers, Beer}
```

Removing the redundant rules, and then calculating the support count gives:

Itemset	Support
{Bread, Milk}	3
{Bread, Beer}	2
{Bread, Diapers}	3
{Beer, Milk}	2
{Beer, Diapers}	3
{Diapers, Milk}	3

Since the support count of the itemset {Bread, Beer} < 3 and {Bread, Milk} < 3, they are infrequent and can be removed.

We can now, use the 2-itemsets, to generate 3-item sets and get:

```
{Bread, Milk, Beer}
{Bread, Milk, Diapers}
{Bread, Diapers, Milk}
{Bread, Diapers, Beer}
{Beer, Diapers, Bread}
{Beer, Diapers, Milk}
{Diapers, Milk, Bread}
{Diapers, Milk, Beer}
```

Removing the redundant rules, and calculating the support counts we get:

Itemset	Support Count
{Bread, Milk, Beer}	1
{Bread, Diapers, Milk}	2
{Bread, Diapers, Beer}	2
{Beer, Diapers, Milk}	2

If we were to move on to 4-itemsets, then we could remove {Bread, Milk, Beer} and repeat the process.

Apriori Algorithm:
The example above performed the Apriori algorithm for frequent itemset generation. Assuming F_k is the frequent k-itemsets and L_k is the candidate k-itemsets, the algorithm works as follows:

1. Let k=1
2. Generate $F_1 = \{\text{frequent 1-itemsets}\}$
3. Repeat until F_k is empty or maximum *k* is met
    a. Candidate Generation: Generate L_{k+1} from F_k
    b. Candidate Pruning: Prune the candidate itemsets in L_{k+1} containing subsets of length *k* that are infrequent (based on the Apriori principle).
    c. Support Counting: Count the support of each candidate in L_{k+1} by scanning the transactions.
    d. Candidate Elimination: Eliminate candidates in L_{k+1} that are infrequent, leaving only those that are frequent. This is the new F_{k+1}

Candidate Generation
Candidates can be generated in multiple ways:

- Brute Force
- $F_{k-1} \times F_1$ Method
- $F_{k-1} \times F_{k-1}$ Method

Brute Force
The brute force method considers every *k*-itemset as a potential candidate and then uses candidate pruning to remove any unnecessary candidates whose subsets are infrequent. This is extremely expensive because a large number of itemsets must be examined. At level *k* the

number of candidate itemsets generated are $\binom{d}{k}$ where *d* is the total number of items.

$\boldsymbol{F_{k-1} \times F_1}$ Method

This method of candidate generation extends each $(k-1)$-itemset with frequent items that are not already part of the $(k-1)$-itemset. This method works because every k-itemset is composed of a $(k-1)$-itemset and a frequent 1-itemset.

$\boldsymbol{F_{k-1} \times F_{k-1}}$ Method

This method merges a pair of frequent $(k-1)$-itemsets only if their first $k-2$ items, arranged in lexicographic order, are identical. Another alternative to merging for this method is to merge candidate A and B together if the last $k-2$ items of A are identical to the first $k-2$ items of B.

Candidate Pruning

For a candidate *k*-itemset, $X = \{i_1, i_2, \cdots, i_k\}$, consider its *k* proper subsets $X - \{i_j\} (\forall j = 1, 2, \cdots, k)$. If any of them are infrequent, they are immediately pruned by using the Apriori principle. This approach reduces the number of candidate itemsets considered during support counting:

- Brute Force: Requires checking *k* subsets of size $k-1$ for each candidate *k*-itemset
- $F_{k-1} \times F_1$: Requires checking $k-1$ subsets since the generation strategy ensures that at least one of the $(k-1)$-itemsets of every candidate is frequent
- $F_{k-1} \times F_{k-1}$: Requires checking $k-2$ subsets in every candidate k-itemset since two of the $(k-1)$-size subsets are already determined to be frequent.

Support Counting

Support counting is the process of determining the frequency of occurrence for every candidate itemset left after candidate pruning.

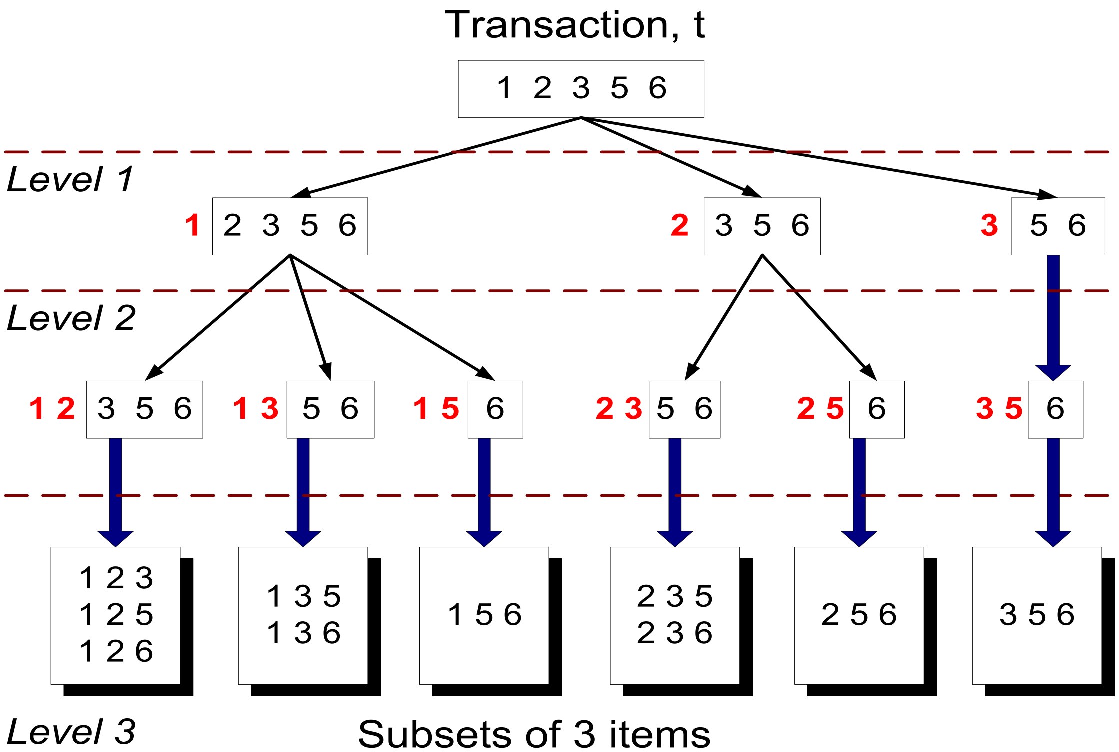
Brute force approaches require comparing each transaction against every candidate itemset and update the support counts of the candidates contained in the transaction.

Alternatively, we can enumerate the itemsets contained in the transaction and use them to update the support counts of the corresponding candidate itemset. If we consider a transaction *t* that contains five items, {1, 2, 3, 4, 5 ,6}, then there are $\binom{5}{3} = 10$ itemsets of size 3 in the transaction. Some of these may correspond to the candidate 3-itemsets, while others may not. If so, this results in an increment in the corresponding support count. Other subsets of *t* may not correspond and can be ignored.

Example: Support Counting
Suppose you have 15 candidate itemsets of length 3:

```
{1, 4, 5}
{1, 2, 4}
{4, 5, 7}
{1, 2, 5}
{4, 5, 8}
{1, 5, 9}
{1, 3, 6}
{2, 3, 4}
{5, 6, 7}
{3, 4, 5}
{3, 5, 6}
{3, 5, 7}
{6, 8, 9}
{3, 6, 7}
{3, 6, 8}
```

We can use the enumeration of the transaction {1, 2, 3, 5, 6} to determine which candidate itemsets are supported:



The itemsets supported are therefore:

```
{1, 2, 5}
{1, 3, 6}
{3, 5, 6}
```

Rule Generation

Each frequent k-itemset, Y, can generate up to $2^k - 2$ association rules, ignoring rules that have empty antecedents and consequents, i.e. $\emptyset \rightarrow Y$ and $Y \rightarrow \emptyset$. Rules are extracted by partitioning an itemset into two non-empty subsets, X, and $Y - X$, where $X \rightarrow Y - X$ satisfies a given confidence threshold. All rules have already met the support threshold because they are generated from a frequent itemset.

To prune association rules, when comparing rules generated from the same frequent itemset Y, the following holds true about the confidence measure:

Let Y be an itemset and X is a subset of Y. If a rule $X \rightarrow Y - X$ does not satisfy the confidence threshold then any rule $\bar{X} \rightarrow Y - X$, where $\bar{X}$ is a subset of X, must not satisfy the confidence threshold as well.

Misleading and trivial associations can be filtered out if:

$$\frac{S(A \cup B)}{S(A)} - S(B) > 0$$

Example: Rule Generation
If {Bread, Diapers, Beer} is a frequent itemset, then the candidate rules are:

```
{Bread, Diapers} => {Beer}
{Diapers, Beer} => {Bread}
{Bread, Beer} => {Diapers}
{Bread} => {Diapers, Beer}
{Beer} => {Bread, Diapers}
{Diapers} => {Bread, Beer}
```

Importance of *minsup*:
Choosing an appropriate value of *minsup* (minimum support) is important as if can lead to a couple of issues:

1. If too high, we can miss itemsets that involve interesting rare items
2. If too low, it is computationally expensive and the number of itemsets is very large
3. A single minimum support threshold may not always be effective.

Evaluation of Association Patterns
The Apriori principle still has the potential to generate a large number of patterns. As the size and dimensionality of the data gets larger, we can end up with hundreds of thousands of different patterns. A large amount of these may not be interesting and it is a trivial task to try and filter the rules. There are two criteria that can be used to evaluate the quality of association patterns:

- Objective interestingness measure
- Subjective arguments

Objective Measures of Interestingness
Objective measures are a data-driven way of evaluating the quality of association patterns. There are many evaluation metrics that can be used to evaluate these patterns, besides support and confidence, some of which include:

- Lift
- Leverage
- Conviction

Lift
Lift is commonly used as a measure of how often the antecedent and consequent of a rule occur together compared to if they were statistically independent. It is measured as:

$$lift(X \rightarrow Y) = \frac{confidence(X \rightarrow Y)}{support(Y)}$$

Leverage
Leverage is the difference between the observed frequency of X and Y appearing together and the frequency we would expect if X and Y were independence. If the leverage metric is 0, then X and Y are independent. It is measured as:

$$leverage(X \rightarrow Y) = support(X \rightarrow Y) - support(X) \times support(Y)$$

Conviction
Conviction measures how dependent the consequent is on the antecedent. If the conviction value is high, then the consequent is highly dependent on the antecedent. It the items are independent, then the conviction is 1. It is measured by:

$$conviction(X \rightarrow Y) = \frac{1 - support(Y)}{1 - confidence(X \rightarrow Y)}$$

More interestingness factors are described here:
https://rdrr.io/cran/arules/man/interestMeasure.html

Subjective Measures
For subjective measures, a rule is interesting if it is unexpected or can be used to do something. There are not metrics to be calculated when using subjective measures. Instead it is up to the user of the rules to judge how interesting a rule is.

Association Analysis in R
To perform association analysis in R, we can use the `arules` package. This package is used to apply the Apriori algorithm to a dataset containing 22 different features about poisonous and edible mushrooms. These features are:

- cap-shape: bell, conical, convex, flat, knobbed, sunken
- cap-surface: fibrous, grooves, scaly, smooth
- cap-color: brown, buff, cinnamon, gray, green, pink, purple, red, white, yellow
- bruises: yes, no
- odor: almond, anise, creosote, fishy, foul, musty, none, pungent, spicy
- gill-attachment: attached, descending, free, notched
- gill-spacing: close, crowded, distant
- gill-size: broad, narrow
- gill-color: black, brown, buff, chocolate, gray, green, orange, pink, purple, red, white, yellow
- stalk-shape: enlarging, tapering
- stalk-root: bulbous, club, cup, equal, rhizomorphs, rooted, missing
- stalk-surface-above-ring: fibrous, scaly, silky, smooth
- stalk-surface-below-ring: fibrous, scaly, silky, smooth

- stalk-color-above-ring: brown, buff, cinnamon, gray, green, pink, purple, red, white, yellow
- stalk-color-below-ring: brown, buff, cinnamon, gray, green, pink, purple, red, white, yellow
- veil-type: partial, universal
- veil-color: brown, orange, white, yellow
- ring-number: none, one, two
- ring-type: cobwebby, evanescent, flaring, large, none, pendant, sheathing, zone
- spore-print-color: black, brown, buff, chocolate, green, orange, purple, white, yellow
- population: abundant, clustered, numerous, scattered, several, solitary
- habitat: grasses, leaves, meadows, paths, urban, waste, woods

The data that is passed into the `apriori` function needs to be in the format of a transaction. To do this, we first need to read in the data, and then coarse it to a transaction, which is a part of the `arules` package:

```
# Load Mushroom Dataset
url <-
"https://raw.githubusercontent.com/PacktPublishing/Machine-
Learning-with-R-Second-
Edition/master/Chapter%2005/mushrooms.csv"

mushrooms <- read.csv(file = url, header = T, stringsAsFactors
= T)

trans <- as(mushrooms, "transactions")
inspect(trans[1:5])
```

```
    items                              transactionID
[1] {type=poisonous,
     cap_shape=convex,
     cap_surface=smooth,
     cap_color=brown,
     bruises=yes,
     odor=pungent,
     gill_attachment=free,
```

```
     gill_spacing=close,
     gill_size=narrow,
     gill_color=black,
     stalk_shape=enlarging,
     stalk_root=equal,
     stalk_surface_above_ring=smooth,
     stalk_surface_below_ring=smooth,
     stalk_color_above_ring=white,
     stalk_color_below_ring=white,
     veil_type=partial,
     veil_color=white,
     ring_number=one,
     ring_type=pendant,
     spore_print_color=black,
     population=scattered,
     habitat=urban}                             1
[2] {type=edible,
     cap_shape=convex,
     cap_surface=smooth,
     cap_color=yellow,
     bruises=yes,
     odor=almond,
     gill_attachment=free,
     gill_spacing=close,
     gill_size=broad,
     gill_color=black,
     stalk_shape=enlarging,
     stalk_root=club,
     stalk_surface_above_ring=smooth,
     stalk_surface_below_ring=smooth,
     stalk_color_above_ring=white,
     stalk_color_below_ring=white,
     veil_type=partial,
     veil_color=white,
     ring_number=one,
     ring_type=pendant,
     spore_print_color=brown,
     population=numerous,
     habitat=grasses}                           2
[3] {type=edible,
     cap_shape=bell,
     cap_surface=smooth,
     cap_color=white,
     bruises=yes,
     odor=anise,
     gill_attachment=free,
     gill_spacing=close,
     gill_size=broad,
     gill_color=brown,
     stalk_shape=enlarging,
     stalk_root=club,
     stalk_surface_above_ring=smooth,
     stalk_surface_below_ring=smooth,
```

```
     stalk_color_above_ring=white,
     stalk_color_below_ring=white,
     veil_type=partial,
     veil_color=white,
     ring_number=one,
     ring_type=pendant,
     spore_print_color=brown,
     population=numerous,
     habitat=meadows}                           3
[4] {type=poisonous,
     cap_shape=convex,
     cap_surface=scaly,
     cap_color=white,
     bruises=yes,
     odor=pungent,
     gill_attachment=free,
     gill_spacing=close,
     gill_size=narrow,
     gill_color=brown,
     stalk_shape=enlarging,
     stalk_root=equal,
     stalk_surface_above_ring=smooth,
     stalk_surface_below_ring=smooth,
     stalk_color_above_ring=white,
     stalk_color_below_ring=white,
     veil_type=partial,
     veil_color=white,
     ring_number=one,
     ring_type=pendant,
     spore_print_color=black,
     population=scattered,
     habitat=urban}                             4
[5] {type=edible,
     cap_shape=convex,
     cap_surface=smooth,
     cap_color=gray,
     bruises=no,
     odor=none,
     gill_attachment=free,
     gill_spacing=crowded,
     gill_size=broad,
     gill_color=black,
     stalk_shape=tapering,
     stalk_root=equal,
     stalk_surface_above_ring=smooth,
     stalk_surface_below_ring=smooth,
     stalk_color_above_ring=white,
     stalk_color_below_ring=white,
     veil_type=partial,
     veil_color=white,
     ring_number=one,
     ring_type=evanescent,
     spore_print_color=brown,
```

```
population=abundant,
habitat=grasses}                              5
```

The transaction data is a transaction object. In order to view the items in the transaction, we need to use the `inspect` function.

We can now begin to use the `apriori` function to start generating association rules. Several things can be done with the `apriori` function and they are controlled through different parameters. The first thing we can do is control the minimum support, confidence, the type of association (rules, frequent itemset), minimum length of the rule, and the maximum length of the rule. This is done through the parameter option.

```
all.rules <- apriori(mushrooms,
parameter=list(target="frequent"))

inspect(all.rules[1:15])
```

```
     items                         support   transIdenticalToItemsets count
[1]  {cap_shape=knobbed}           0.1019202 0                         828
[2]  {habitat=leaves}              0.1024126 0                         832
[3]  {cap_color=white}             0.1280158 0                        1040
[4]  {gill_color=brown}            0.1290005 0                        1048
[5]  {cap_color=yellow}            0.1319547 0                        1072
[6]  {stalk_root=equal}            0.1378631 0                        1120
[7]  {habitat=paths}               0.1408173 0                        1144
[8]  {gill_color=white}            0.1479567 0                        1202
[9]  {population=scattered}        0.1536189 0                        1248
[10] {ring_type=large}             0.1595273 0                        1296
[11] {gill_spacing=crowded}        0.1614968 0                        1312
[12] {gill_color=pink}             0.1836534 0                        1492
[13] {cap_color=red}               0.1846381 0                        1500
[14] {spore_print_color=chocolate} 0.2008863 0                        1632
[15] {population=solitary}         0.2107336 0                        1712
```

The first 15 itemsets generated are 1-itemsets. The support for a given itemset and the support count are provided with the output.

We can use the summary function to get information about the frequent itemsets, and summary statistics for the support, transactions that are identical to the itemsets generated, and minimum support counts.

```
summary(all.itemsets)
```

```
set of 512831 itemsets

most frequent items:
   veil_type=partial gill_attachment=free     veil_color=white      ring_number=one   gill_spacing=close
              242955               242691               242691               240048               236434
             (Other)
             2693722

element (itemset/transaction) length distribution:sizes
     1      2      3      4      5      6      7      8      9     10
    56    763   4593  16150  38800  69835  98846 111786 100660  71342

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   6.000   8.000   7.602   9.000  10.000

summary of quality measures:
    support       transIdenticalToItemsets     count
 Min.   :0.1002   Min.   :0                Min.   : 814
 1st Qu.:0.1064   1st Qu.:0                1st Qu.: 864
 Median :0.1064   Median :0                Median : 864
 Mean   :0.1253   Mean   :0                Mean   :1018
 3rd Qu.:0.1182   3rd Qu.:0                3rd Qu.: 960
 Max.   :1.0000   Max.   :0                Max.   :8124

includes transaction ID lists: FALSE

mining info:
```

We can see that the number of itemsets generated is 512,831. A majority of these itemsets are of length 8 and 9.

The next thing we can do is generate the association rules for the dataset. This is done by setting the type parameter equal to rules. The output rule will consist of two components: the antecedent and the consequent. The antecedent is an itemset found in the dataset. The consequent is an item found in combination with the antecedent. In R, this is referred to as the LHS (left-hand side) and the RHS (right-hand side). For this example, because of the length of time it takes to generate the rules, the length of the itemset is restricted to a maximum of 5. An itemset of length 10 produces over 3 million different rules.

```
# Generate association rules
rules1<- apriori(mushrooms, parameter=list(target="rules",
minlen=1, maxlen=5))

summary(rules1)
inspect(head(rules1, 20))
```

```
set of 174244 rules

rule length distribution (lhs + rhs):sizes
     1      2      3      4      5 
     5    354   5580  35965 132340 

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  1.000   5.000   5.000   4.723   5.000   5.000 

summary of quality measures:
    support         confidence        coverage          lift            count     
 Min.   :0.1002   Min.   :0.8000   Min.   :0.1002   Min.   :0.8266   Min.   : 814  
 1st Qu.:0.1064   1st Qu.:0.9558   1st Qu.:0.1068   1st Qu.:1.0265   1st Qu.: 864  
 Median :0.1241   Median :1.0000   Median :0.1300   Median :1.4170   Median :1008  
 Mean   :0.1523   Mean   :0.9694   Mean   :0.1580   Mean   :1.7106   Mean   :1237  
 3rd Qu.:0.1773   3rd Qu.:1.0000   3rd Qu.:0.1910   3rd Qu.:2.0227   3rd Qu.:1440  
 Max.   :1.0000   Max.   :1.0000   Max.   :1.0000   Max.   :6.8718   Max.   :8124  

mining info:
      data ntransactions support confidence
 mushrooms          8124     0.1        0.8
```

```
     lhs                    rhs                                 support   confidence coverage  lift     count
[1]  {}                  => {gill_spacing=close}                0.8385032 0.8385032  1.0000000 1.000000 6812
[2]  {}                  => {ring_number=one}                   0.9217134 0.9217134  1.0000000 1.000000 7488
[3]  {}                  => {gill_attachment=free}              0.9741507 0.9741507  1.0000000 1.000000 7914
[4]  {}                  => {veil_color=white}                  0.9753816 0.9753816  1.0000000 1.000000 7924
[5]  {}                  => {veil_type=partial}                 1.0000000 1.0000000  1.0000000 1.000000 8124
[6]  {cap_shape=knobbed} => {veil_type=partial}                 0.1019202 1.0000000  0.1019202 1.000000  828
[7]  {habitat=leaves}    => {bruises=no}                        0.1014279 0.9903846  0.1024126 1.694584  824
[8]  {habitat=leaves}    => {ring_number=one}                   0.1024126 1.0000000  0.1024126 1.084936  832
[9]  {habitat=leaves}    => {veil_type=partial}                 0.1024126 1.0000000  0.1024126 1.000000  832
[10] {cap_color=white}   => {stalk_color_below_ring=white}      0.1280158 1.0000000  0.1280158 1.853102 1040
[11] {cap_color=white}   => {stalk_color_above_ring=white}      0.1280158 1.0000000  0.1280158 1.819892 1040
[12] {cap_color=white}   => {ring_number=one}                   0.1073363 0.8384615  0.1280158 0.909677  872
[13] {cap_color=white}   => {gill_attachment=free}              0.1280158 1.0000000  0.1280158 1.026535 1040
[14] {cap_color=white}   => {veil_color=white}                  0.1280158 1.0000000  0.1280158 1.025240 1040
[15] {cap_color=white}   => {veil_type=partial}                 0.1280158 1.0000000  0.1280158 1.000000 1040
[16] {gill_color=brown}  => {ring_type=pendant}                 0.1053668 0.8167939  0.1290005 1.672287  856
[17] {gill_color=brown}  => {type=edible}                       0.1152142 0.8931298  0.1290005 1.724284  936
[18] {gill_color=brown}  => {stalk_surface_below_ring=smooth}   0.1093058 0.8473282  0.1290005 1.394590  888
[19] {gill_color=brown}  => {stalk_surface_above_ring=smooth}   0.1171837 0.9083969  0.1290005 1.425776  952
[20] {gill_color=brown}  => {gill_size=broad}                   0.1083210 0.8396947  0.1290005 1.215552  880
```

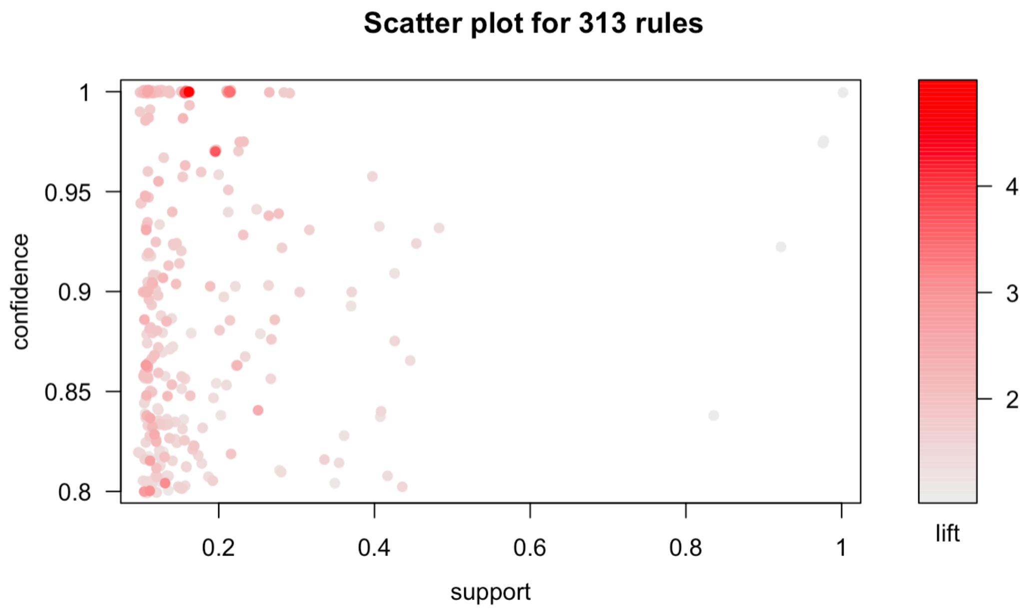
Even with an itemset length of 5, the number of rules that are generated, 174244 different rules, is large. A large amount of these rules is considered redundant. A rule is redundant if they consist of the same number and items in the antecedent, but in a different order, and leads to the same consequent. We can filter of these and other rules out, by finding all the rules that are subsets of larger rules.

```
# Filter redundant rules. This can take ~5-10 minutes to
completely.
subset.rules1 <- which(colSums(is.subset(rules1, rules1)) > 1)

# get subset rules in vector
pruned.rules1 <- rules1[-subset.rules1]
length(pruned.rules1)
```

It can take roughly 5-10 minutes to filter out all of the subsets. Doing this process reduced the number of rules from 174244 to 313 rules.

We can also plot the rules. Plotting the rules will create a scatterplot of support vs. confidence as well as be colored by the value of the lift metric.

```
plot(pruned.rules1)
```



We can see from the plot that most of the rules with high confidence have very low support values, and only a few have high lift scores.

Templates

Another thing we can do is use a template to find rules either based on the LHS or the RHS. To demonstrate this, we can find all rules that can determine if a mushroom is poisonous or edible. Since we are determining the edibility of a mushroom, we want to be 100% confident that it is either edible or poisonous and set the confidence value to 1.

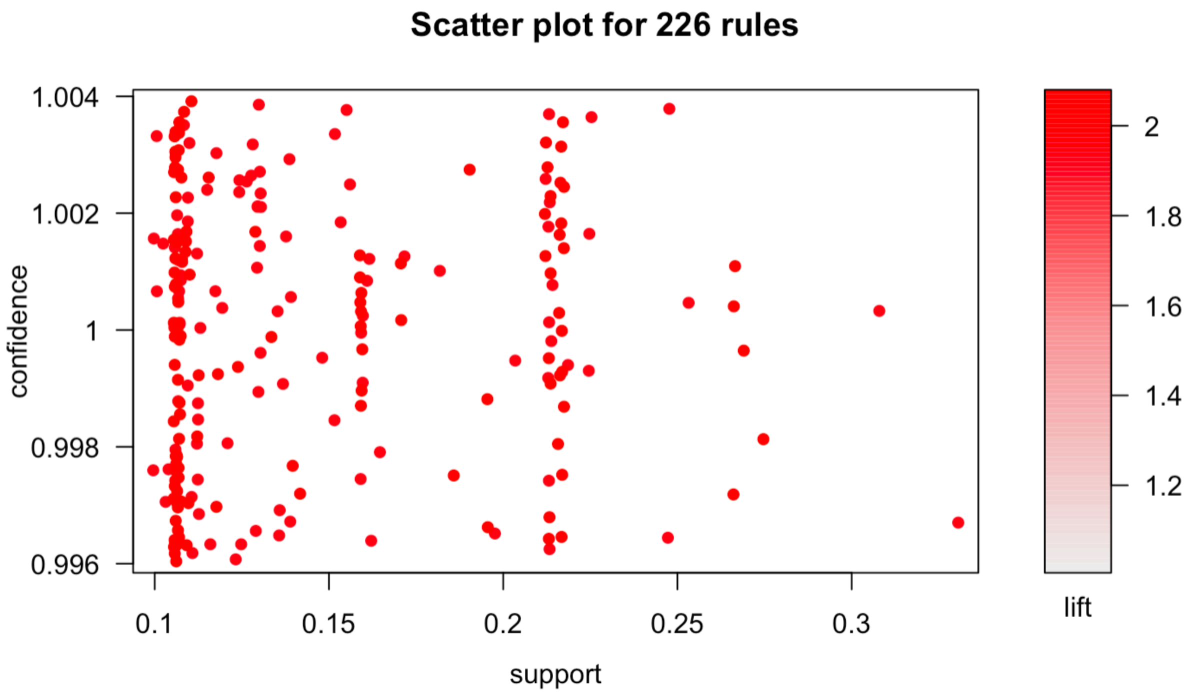
```
# Generate association rules to determine edibility
edibility.rules <- apriori(mushrooms,
parameter=list(target="rules", maxlen=5, confidence=1),
appearance = list(rhs=c("type=edible", "type=poisonous")))

summary(edibility.rules)
inspect(head(edibility.rules, 20))
```

```
set of 6484 rules

rule length distribution (lhs + rhs):sizes
   2    3    4    5
   3  118 1106 5257

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  2.000   5.000   5.000   4.792   5.000   5.000

summary of quality measures:
    support          confidence    coverage           lift            count
 Min.   :0.1004   Min.   :1    Min.   :0.1004   Min.   :1.931   Min.   : 816
 1st Qu.:0.1064   1st Qu.:1    1st Qu.:0.1064   1st Qu.:1.931   1st Qu.: 864
 Median :0.1103   Median :1    Median :0.1103   Median :2.075   Median : 896
 Mean   :0.1366   Mean   :1    Mean   :0.1366   Mean   :2.029   Mean   :1109
 3rd Qu.:0.1595   3rd Qu.:1    3rd Qu.:0.1595   3rd Qu.:2.075   3rd Qu.:1296
 Max.   :0.3309   Max.   :1    Max.   :0.3309   Max.   :2.075   Max.   :2688

mining info:
      data ntransactions support confidence
 mushrooms          8124     0.1          1
```

```
     lhs                                                  rhs                support   confidence coverage
[1]  {ring_type=large}                                 => {type=poisonous} 0.1595273 1          0.1595273
[2]  {gill_color=buff}                                 => {type=poisonous} 0.2127031 1          0.2127031
[3]  {odor=foul}                                       => {type=poisonous} 0.2658789 1          0.2658789
[4]  {gill_size=broad,gill_color=brown}                => {type=edible}    0.1083210 1          0.1083210
[5]  {odor=none,stalk_root=equal}                      => {type=edible}    0.1063516 1          0.1063516
[6]  {bruises=no,stalk_root=equal}                     => {type=edible}    0.1063516 1          0.1063516
[7]  {ring_type=large,spore_print_color=chocolate}     => {type=poisonous} 0.1595273 1          0.1595273
[8]  {odor=foul,ring_type=large}                       => {type=poisonous} 0.1595273 1          0.1595273
[9]  {stalk_surface_below_ring=silky,ring_type=large} => {type=poisonous} 0.1595273 1          0.1595273
[10] {stalk_surface_above_ring=silky,ring_type=large} => {type=poisonous} 0.1595273 1          0.1595273
[11] {stalk_shape=enlarging,ring_type=large}           => {type=poisonous} 0.1595273 1          0.1595273
[12] {stalk_root=bulbous,ring_type=large}              => {type=poisonous} 0.1595273 1          0.1595273
[13] {bruises=no,ring_type=large}                      => {type=poisonous} 0.1595273 1          0.1595273
[14] {gill_size=broad,ring_type=large}                 => {type=poisonous} 0.1595273 1          0.1595273
[15] {gill_spacing=close,ring_type=large}              => {type=poisonous} 0.1595273 1          0.1595273
     lift     count
[1]  2.074566 1296
[2]  2.074566 1728
[3]  2.074566 2160
[4]  1.930608  880
[5]  1.930608  864
[6]  1.930608  864
[7]  2.074566 1296
[8]  2.074566 1296
[9]  2.074566 1296
[10] 2.074566 1296
[11] 2.074566 1296
[12] 2.074566 1296
[13] 2.074566 1296
[14] 2.074566 1296
[15] 2.074566 1296
```

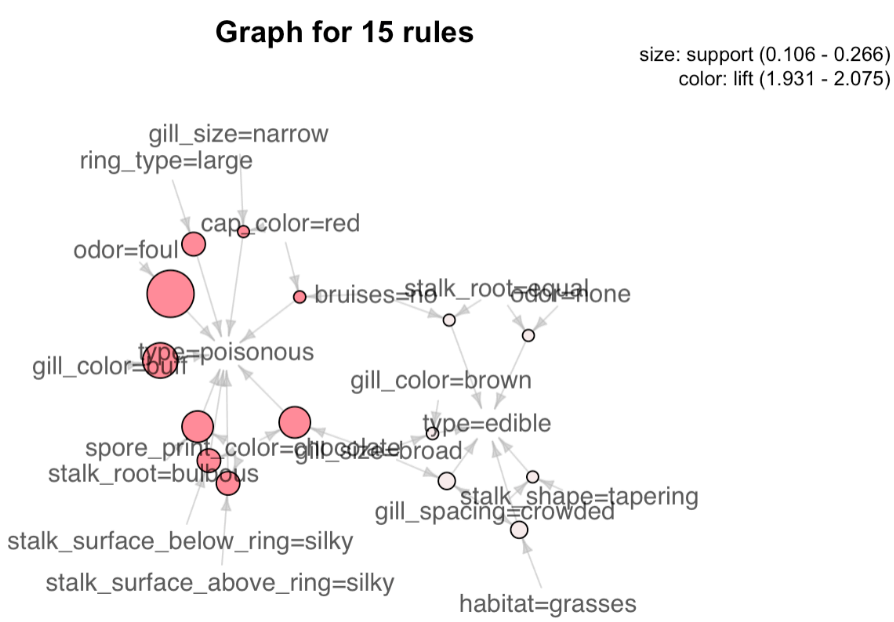
Generating the rules for the edibility of mushrooms creates 6484 different rules. The lift metric for the rules for poisonous mushrooms is higher, 2.07, than for the rules for edible mushrooms, which is 1.93. These rules do contain redundant rules, and we can filter them out as well using the same method as earlier. When doing so, we instead get 226 rules.

We can plot these rules in a scatter plot. We can see that the values of support range from 0.1 to 0.3. Compared to all rules generated it is clear that rules with high level of confidence also have higher values of support. The lift scores are also significantly better.



Another method of plotting that we can also do is graph based. To save make it more readable, only 15 rules are plotted.

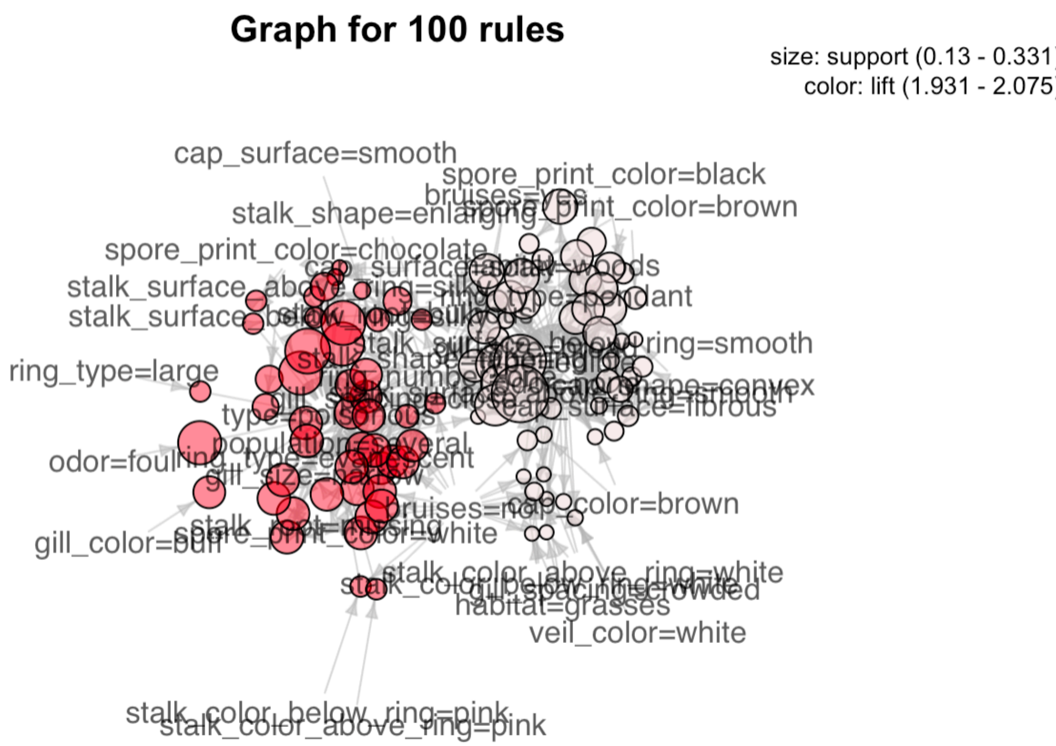
```
plot(pruned.rules[1:15], method="graph")
```



The graph shows two distinct centers, one edible and one poisonous. By following the arrows we can generate the rules. Some rules branch to more than one item, while others don't. For an example, if we start at the node cap_color=red, then we can follow the arrows and get:

cap_color = red => bruises=no => type=poisonous

Viewing the graph, therefore provides a visual way to view the association rules. However, as the number of rules plotted gets larger, the graph does become cluttered and illegible:



Sources:

- *Introduction to Data Mining,* Chapter 5 + 6
- https://www.datacamp.com/community/tutorials/market-basket-analysis-r
- http://rasbt.github.io/mlxtend/user_guide/frequent_patterns/association_rules/#metrics
- https://paginas.fe.up.pt/~ec/files_0506/slides/04_AssociationRules.pdf
- https://www-users.cs.umn.edu/~kumar001/dmbook/index.php#item4 (PPT Slides for Ch. 5 + 6)