Chapter 12
Apriori Association Rules Learning

HTTP cookies are used to monitor web-traffic and track users surfing the Internet. We often notice that promotions (ads) on websites tend to match our needs, reveal our prior browsing history, or reflect our interests. That is not an accident. Nowadays, recommendation systems are highly based on machine learning methods that can learn the behavior, e.g., purchasing patterns, of individual consumers. In this chapter, we will uncover some of the mystery behind recommendation systems for transactional records. Specifically, we will (1) discuss association rules and their support and confidence; (2) the *Apriori algorithm* for association rule learning; and (3) cover step-by-step a set of case-studies, including a toy example, Head and Neck Cancer Medications, and Grocery purchases.

12.1 Association Rules

Association rules are the result of process analytics (e.g., market analysis) that specify patterns of relationships among items. One specific example would be:

$$\{charcoal, lighter, chicken\,wings\} \rightarrow \{barbecue\,sauce\}$$

In words, charcoal, lighter and chicken wings imply barbecue sauce. Those curly brackets indicate that we have a set. Items in a set are called elements. When an itemset like $\{charcoal, lighter, chicken\,wings, barbecue\,sauce\}$ appears in our dataset with some regularity, we can discover the above pattern.

Association rules are commonly used for unsupervised discovery of knowledge rather than prediction of outcomes. In biomedical research, association rules are widely used to:

© Ivo D. Dinov 2018
I. D. Dinov, *Data Science and Predictive Analytics*,
https://doi.org/10.1007/978-3-319-72347-1_12

- Search for interesting or frequently occurring patterns of DNA.
- Search for protein sequences in an analysis of cancer data.
- Find patterns of medical claims that occur in combination with fraudulent credit card or insurance use.

12.2 The Apriori Algorithm for Association Rule Learning

Association rules are mostly applied to transactional data, like business, trade, service or medical records. These datasets are typically very large in number of transactions and features. This will add lots of possible orders and patterns when we try to do analytics, which makes data mining a very hard task.

With the **Apriori** rule, this problem is easily solved. If we have a simple prior (belief about the properties of frequent elements), we can efficiently reduce the number of features or combinations that we need to look at.

The Apriori algorithm has a simple `apriori` belief that *all subsets of a frequent item-set must also be frequent*. This is known as the **Apriori property**. The full set in the last example, $\{charcoal, lighter, chicken\ wings, barbecue\ sauce\}$, can be frequent if and only if itself and all its subsets of single elements, pairs and triples occur frequently. We can see that this algorithm is designed for finding patterns in large datasets. If a pattern happens frequently, it is considered "interesting".

12.3 Measuring Rule Importance by Using Support and Confidence

Support and confidence are the two criteria to help us decide whether a pattern is "interesting". By setting thresholds for these two criteria, we can easily limit the number of interesting rules or item-sets reported.

For item-sets X and Y, the `support` of an item-set measures how frequently it appears in the data:

$$support(X) = \frac{count(X)}{N},$$

where N is the total number of transactions in the database and *count(X)* is the number of observations (transactions) containing the item-set X. Of course, the union of item-sets is an item-set itself. For example, if $Z = X, Y$, then

$$support(Z) = support(X, Y).$$

For a rule $X \rightarrow Y$, the `rule's confidence` measures the relative accuracy of the rule:

$$confidence(X \to Y) = \frac{support(X, Y)}{support(X)}$$

This measures the joint occurrence of *X* and *Y* over the *X* domain. If whenever *X* appears *Y* tends to be present too, we will have a high $confidence(X \to Y)$. The ranges of the support and confidence are $0 \leq support, confidence \leq 1$. Note that in probabilistic terms, *Confidence* $(X \to Y)$ is equivalent to the conditional probability $P(Y|X)$.

$\{peanut\,butter\} \to \{bread\}$ would be an example of a strong rule because it has high *support* as well as high *confidence* in grocery store transactions. Shoppers tend to purchase bread when they get peanut butter. These items tend to appear in the same baskets, which yields high confidence for the rule $\{peanut\,butter\} \to \{bread\}$.

12.4 Building a Set of Rules with the Apriori Principle

To build a set of rules, we need to go through two steps:

- **Step 1**: Filter all item-sets with a minimum *support* threshold. This is accomplished iteratively by increasing the size of the item-sets. In the first iteration, we compute the support of singletons, 1-item-sets. At the next iteration, we compute the support of pairs of items, and so on. Item-sets passing iteration *i* could be considered as candidates for the next iteration, *i + 1*. If *{A}*, *{B}*, *{C}* are all frequent, but *D* is not frequent in the first singleton-selection round, then in the second iteration we only consider the support of these pairs *{A, B}*, *{A,C}*, *{B,C}*, ignoring all pairs including *D*. This substantially reduces the cardinality of the potential item-sets and ensures the feasibility of the algorithm. At the third iteration, if *{A,C}*, and *{B,C}* are frequently occurring, but *{A, B}* is not, then the algorithm may terminate, as the support of *{A,B,C}* is trivial (does not pass the support threshold), given that *{A, B}* was not frequent enough.
- **Step 2**: Using the item-sets selected in Step 1, generate new rules with *confidence* larger than a predefined minimum confidence threshold. The candidate item-sets that passed Step 1 would include all frequent item-sets. For the highly-supported item-set *{A, C}*, we would compute the confidence measures for $\{A\} \to \{C\}$ as well as $\{C\} \to \{A\}$ and compare these against the minimum confidence threshold. The *surviving rules are the ones with confidence levels exceeding that minimum threshold.*

12.5 A Toy Example

Assume that a large supermarket tracks sales data by stock-keeping unit (SKU) for each item, i.e., each item, such as “butter” or “bread”, is identified by an SKU number. The supermarket has a database of transactions where each transaction is a set of SKUs that were bought together (Table 12.1).

Suppose the database of transactions consist of following item-sets, each representing a purchasing order:

```r
require(knitr)
item_table = as.data.frame(t(c("{1,2,3,4}","{1,2,4}","{1,2}","{2,3,4}",
             "{2,3}","{3,4}","{2,4}")))
colnames(item_table) <- c("choice1","choice2","choice3","choice4",
             "choice5","choice6","choice7")
kable(item_table, caption = "Item table")
```

We will use *Apriori* to determine the frequent item-sets of this database. To do so, we will say that an item-set is frequent if it appears in at least 3 transactions of the database, i.e., the value 3 is the support threshold (Table 12.2).

The first step of Apriori is to count up the number of occurrences, i.e., the support, of each member item separately. By scanning the database for the first time, we obtain get:

```r
item_table = as.data.frame(t(c(3,6,4,5)))
colnames(item_table) <- c("item1","item2","item3","item4")
rownames(item_table) <- "support"
kable(item_table,caption = "Size 1 Support")
```

All the item-sets of size 1 have a support of at least 3, so they are all frequent. The next step is to generate a list of all pairs of frequent items.

For example, regarding the pair $\{1, 2\}$: the first table of Example 2 shows items 1 and 2 appearing together in three of the item-sets; therefore, we say that the support of the item $\{1, 2\}$ is 3 (Tables 12.3 and 12.4).

Table 12.1 Item table

choice1	choice2	choice3	choice4	choice5	choice6	choice7
{1,2,3,4}	{1,2,4}	{1,2}	{2,3,4}	{2,3}	{3,4}	{2,4}

Table 12.2 Size 1 support

	item1	item2	item3	item4
support	3	6	4	5

Table 12.3 Size 2 support

	{1,2}	{1,3}	{1,4}	{2,3}	{2,4}	{3,4}
support	3	1	2	3	4	3

Table 12.4 Size 3 support

	{2,3,4}
support	2

```r
item_table = as.data.frame(t(c(3,1,2,3,4,3)))
colnames(item_table) <- c("{1,2}","{1,3}","{1,4}","{2,3}","{2,4}","{3,4}")
rownames(item_table) <- "support"
kable(item_table,caption = "Size 2 Support")
```

The pairs $\{1,2\}$, $\{2,3\}$, $\{2,4\}$, and $\{3,4\}$ all meet or exceed the minimum support of 3, so they are *frequent*. The pairs $\{1,3\}$ and $\{1,4\}$ are not and any larger set which contains $\{1,3\}$ or $\{1,4\}$ cannot be frequent. In this way, we can prune sets: we will now look for frequent triples in the database, but we can already exclude all the triples that contain one of these two pairs:

```r
item_table = as.data.frame(t(c(2)))
colnames(item_table) <- c("{2,3,4}")
rownames(item_table) <- "support"
kable(item_table,caption = "Size 3 Support")
```

In the example, there are no frequent triplets – the support of the item-set $\{2,3,4\}$ is below the minimal threshold, and the other triplets were excluded because they were super sets of pairs that were already below the threshold. We have thus determined the frequent sets of items in the database, and illustrated how some items were not counted because some of their subsets were already known to be below the threshold.

12.6 Case Study 1: Head and Neck Cancer Medications

12.6.1 Step 1: Collecting Data

To demonstrate the *Apriori* algorithm in a real biomedical case-study, we will use a transactional healthcare data representing a subset of the Head and Neck Cancer Medication data, which is available in our case-studies collection as `10_medication_descriptions.csv`. It consists of inpatient medications for head and neck cancer patients.

The data is in wide format, see Chap. 2, where each row represents a patient. During the study period, each patient had records for a maximum of 5 encounters. *NA* represents no medication administration records in this specific time point for the specific patient. This dataset contains a total of 528 patients.

12.6.2 Step 2: Exploring and Preparing the Data

Different from our data imports in the previous chapters, transactional data need to be ingested in R using the `read.transactions()` function. This function will store data as a matrix with each row representing an example and each column representing a feature.

Let’s load the dataset and delete the irrelevant *index* column. With the `write.csv(R data, "path")` function we can output our R data file into a local CSV file. To avoid generating another index column in the output CSV file, we can use the `row.names` = `F` option.

```
med<-read.csv("https://umich.instructure.com/files/1678540/download?download
_frd=1", stringsAsFactors = FALSE)
med<-med[, -1]
write.csv(med, "medication.csv", row.names=F)
```

Now we can use `read.transactions()` in the `arules` package to read the CSV file we just outputted.

```
# install.packages("arules")
library(arules)

med<-read.transactions("medication.csv", sep = ",", skip = 1, rm.duplicates=
TRUE)
## distribution of transactions with duplicates:
## items
##   1   2   3
##  79 166 248

summary(med)

## transactions as itemMatrix in sparse format with
##  528 rows (elements/itemsets/transactions) and
##  88 columns (items) and a density of 0.02085486
##
## most frequent items:
##                fentanyl injection uh  hydrocodone acetaminophen 5mg 325mg
##                                  211                                  165
##                    cefazolin ivpb uh                    heparin injection
##                                  108                                  105
## hydrocodone acetamin 75mg 500mg 15ml                              (Other)
##                                   60                                  320
##
## element (itemset/transaction) length distribution:
## sizes
##   1   2   3   4   5
## 248 166  79  23  12
##
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   1.000   2.000   1.835   2.000   5.000
##
## includes extended item information - examples:
##                            labels
## 1                         09 nacl
## 2                   09 nacl bolus
## 3 acetaminophen  multiroute    uh
```

Here we use the option `rm.duplicates = T` because we may have similar medication administration records for two different patients. The option `skip = 1` means we skip the heading line in the CSV file. Now we get a transactional data with unique rows.

The summary of a transactional data contains rich information. The first block of information tells us that we have 528 rows and 88 different medicines in this matrix. Using the density number we can calculate how many non *NA* medication records are in the data. In total, we have 528 × 88 = 46,464 positions in the matrix. Thus, there are 46,464 × 0.0209 = 971 medicines prescribed during the study period.

The second block lists the most frequent medicines and their frequencies in the matrix. For example, `fentanyl injection uh` appeared 211 times; that is 211/528 = 40 of the (treatment) transactions. Since fentanyl is frequently used to help prevent pain after surgery or other medical procedure, we can see that many of these patients were going through some painful medical procedures.

The last block shows statistics about the size of the transaction. 248 patients had only one medicine in the study period, while 12 of them had 5 medication records one for each time point. On average, the patients are having 1.8 different medicines.

Visualizing Item Support: Item Frequency Plots

The summary might still be fairly abstract; let's visualize the data.

```
inspect(med[1:5,])

##     items
## [1] {acetaminophen uh,
##      cefazolin ivpb uh}
## [2] {docusate,
##      fioricet,
##      heparin injection,
##      ondansetron injection uh,
##      simvastatin}
## [3] {hydrocodone acetaminophen 5mg 325mg}
## [4] {fentanyl injection uh}
## [5] {cefazolin ivpb uh,
##      hydrocodone acetaminophen 5mg 325mg}
```

The `inspect()` call shows the transactional dataset. We can see that the medication records of each patient are nicely formatted as item-sets.

We can further analyze the frequent terms using `itemFrequency()`. This will show all item frequencies alphabetically ordered from the first five outputs (Fig. 12.1).

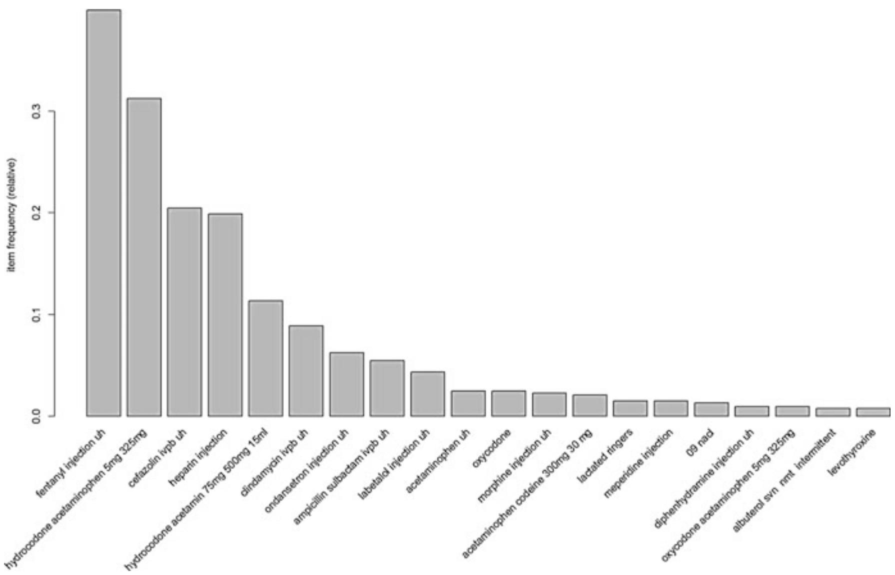


Fig. 12.1 Rank-order plot of item frequencies

```
itemFrequency(med[, 1:5])

##                                09 nacl
##                            0.013257576
##                          09 nacl bolus
##                            0.003787879
##          acetaminophen  multiroute    uh
##                            0.001893939
## acetaminophen codeine 120 mg 12 mg 5 ml
##                            0.001893939
##        acetaminophen codeine 300mg 30 mg
##                            0.020833333

itemFrequencyPlot(med, topN=20)
```

The above graph is showing us the top 20 medicines that are most frequently present in this dataset. Consistent with the prior `summary()` output, `fentanyl` is still the most frequent item. You can also try to plot the items with a threshold for support. Instead of `topN` = `20`, just use the option `support` = `0.1`, which will give you all the items have a support greater or equal to 0.1.

Visualizing Transaction Data: Plotting the Sparse Matrix

The sparse matrix will show what mediations were prescribed for each patient (Fig. 12.2).

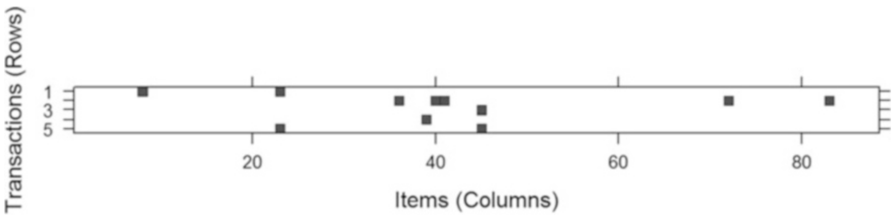
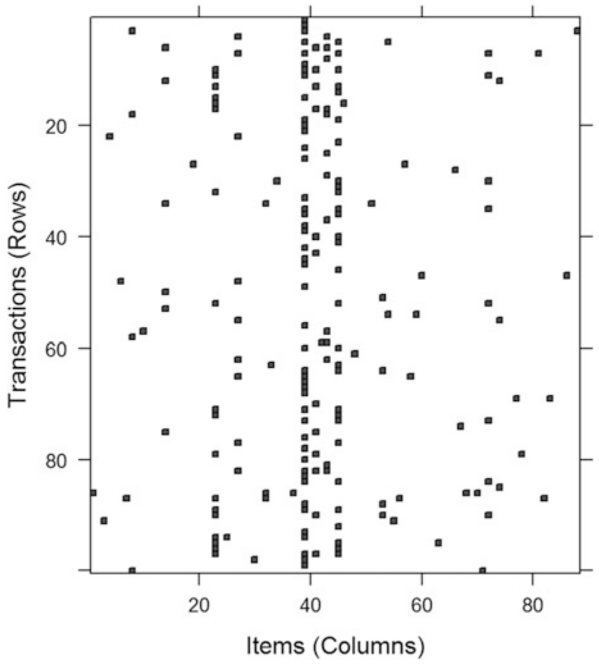


Fig. 12.2 A characteristic plot of the prescribed medications (columns) for the first 5 patients (rows)

Fig. 12.3 A characteristic plot of the prescribed medications (columns) for 100 random patients (rows)



```
image(med[1:5, ])
```

The image on Fig. 12.2 has 5 rows (we only requested the first 5 patients) and 88 columns (88 different medicines). Although the picture may be a little hard to interpret, it gives a sense of what kind of medicine is prescribed for each patient in the study.

Let's see an expanded graph including 100 randomly chosen patients (Fig. 12.3).

```
subset_int <- sample(nrow(med), 100, replace = F)
image(med[subset_int, ])
```

It shows us clearly that some medications are more popular than others. Now, let's fit the *Apriori* model.

12.6.3 Step 3: Training a Model on the Data

With the data in place, we can build the *association rules* using `apriori()` function.

```
myrules <- apriori(data=mydata, parameter=list(support=0.1, confidence=0.8,
   minlen=1))
```

- Data: a sparse matrix created by `read.transacations()`.
- Support: minimum threshold for support.
- Confidence: minimum threshold for confidence.
- minlen: minimum required rule items (in our case, medications).

Setting up the threshold could be hard. You don't want it to be too high so that you get no rules or rules that everyone knows. You don't want to set it too low either, to avoid too many rules present. Let's see what we get under the default setting `support` = `0.1`, `confidence` = `0.8`:

```
apriori(med)

## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.8    0.1    1 none FALSE            TRUE       5     0.1      1
##  maxlen target   ext
##      10  rules FALSE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 52
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[88 item(s), 528 transaction(s)] done [0.00s].
## sorting and recoding items ... [5 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 done [0.00s].
## writing ... [0 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].

## set of 0 rules
```

Not surprisingly, we have 0 rules. The default setting is too high. In practice, we might need some time to fine-tune these thresholds, which may require certain familiarity with the underlying process or clinical phenomenon.

In this case study, we set `support` = `0.1` and `confidence` = `0.25`. This requires rules that have appeared in at least 10% of the head and neck cancer patients in the study. Also, the rules have to have least 25% accuracy. Moreover, `minlen` = 2 would be a very helpful option because it removes all rules that have fewer than two items.

The results suggest we have a new `rules` object consisting of 29 rules.

```
med_rule<-apriori(med, parameter=list(support=0.01, confidence=0.25, minlen=
2))

## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##        0.25    0.1    1 none FALSE            TRUE       5    0.01      2
##  maxlen target   ext
##      10  rules FALSE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 5
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[88 item(s), 528 transaction(s)] done [0.00s].
## sorting and recoding items ... [16 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 done [0.00s].
## writing ... [29 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].

med_rule

## set of 29 rules
```

12.6.4 Step 4: Evaluating Model Performance

First, we can obtain the overall summary of this set of rules.

```
summary(med_rule)

## set of 29 rules
##
## rule length distribution (lhs + rhs):sizes
##  2  3  4
## 13 12  4
##
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    2.00    2.00    3.00    2.69    3.00    4.00
##
## summary of quality measures:
##     support          confidence          lift
##  Min.   :0.01136   Min.   :0.2500   Min.   :0.7583
##  1st Qu.:0.01705   1st Qu.:0.3390   1st Qu.:1.3333
##  Median :0.01894   Median :0.4444   Median :1.7481
##  Mean   :0.03448   Mean   :0.4491   Mean   :1.8636
##  3rd Qu.:0.03788   3rd Qu.:0.5000   3rd Qu.:2.2564
##  Max.   :0.11174   Max.   :0.8000   Max.   :3.9111
##
## mining info:
##  data ntransactions support confidence
##   med           528    0.01       0.25
```

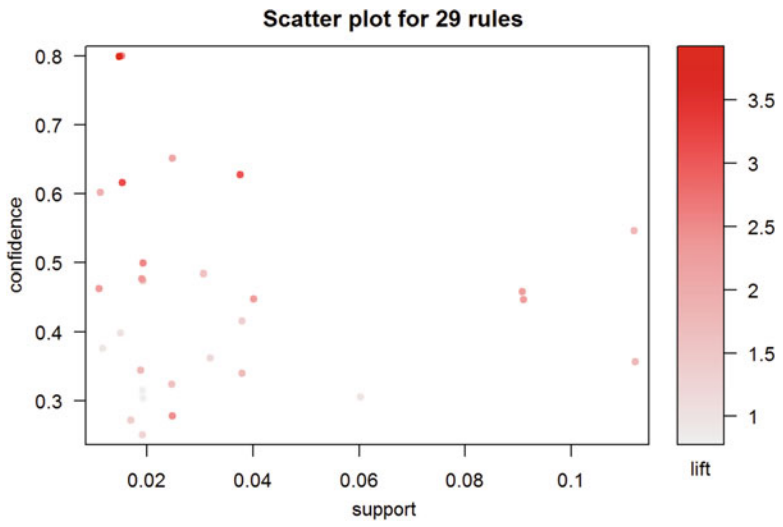


Fig. 12.4 Confidence-Support scatterplot of 29 rules

We have 13 rules that contain two items; 12 rules containing 3 items, and the remaining 4 rules contain 4 items.

The `lift` column shows how much more likely one medicine is to be prescribed to a patient given another medicine is prescribed. It is obtained by the following formula:

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

Note that $lift(X \rightarrow Y)$ is the same as $lift(Y \rightarrow X)$. The range of *lift* is $[0, \infty)$ and higher *lift* is better. We don't need to worry about support since we already set a threshold that the support will exceed.

Using hte `arugleViz` package we can visualize the confidence and support scatter plots for all the rules (Fig. 12.4).

```
# install.packages("arulesViz")
library(arulesViz)

plot(sort(med_rule))
```

Again, we can utilize the `inspect()` function to see exactly what are these rules.

```
inspect(med_rule[1:3])

##     lhs                              rhs                  support
confidence     lift
## [1] {acetaminophen uh}            => {cefazolin ivpb uh} 0.01136364
0.4615385 2.256410
## [2] {ampicillin sulbactam ivpb uh} => {heparin injection} 0.01893939
0.3448276 1.733990
## [3] {ondansetron injection uh}     => {heparin injection} 0.01704545
0.2727273 1.371429
```

Here, `lhs` and `rhs` refer to "left hand side" and "right hand side" of the rule, respectively. `Lhs` is the given condition and `rhs` is the predicted result. Using the first row as an example: If a head-and-neck patient has been prescribed acetaminophen (pain reliever and fever reducer), it is likely that the patient is also prescribed cefazolin (antibiotic that resist bacterial infections); bacterial infections are associated with fevers and some cancers.

12.6.5 Step 5: Improving Model Performance

Sorting the Set of Association Rules

Sorting the resulting association rules corresponding to high **lift** values will help us select the most useful rules.

```
inspect(sort(med_rule, by="lift")[1:3])

##     lhs                                          rhs
support confidence     lift
## [1] {fentanyl injection uh,
##      heparin injection,
##      hydrocodone acetaminophen 5mg 325mg} => {cefazolin ivpb uh}
0.01515152  0.8000000 3.911111
## [2] {cefazolin ivpb uh,
##      fentanyl injection uh,
##      hydrocodone acetaminophen 5mg 325mg} => {heparin injection}
0.01515152  0.6153846 3.094505
## [3] {heparin injection,
##      hydrocodone acetaminophen 5mg 325mg} => {cefazolin ivpb uh}
0.03787879  0.6250000 3.055556
```

These rules may need to be interpreted by clinicians and experts in the specific context of the study. For instance, the first row, *{fentanyl, heparin, hydrocodone acetaminophen}* implies *{cefazolin}*. Fentanyl and hydrocodone acetaminophen are both pain relievers that may be prescribed after surgery. *Heparin* is usually used before surgery to reduce the risk of blood clots. This rule may suggest that patients who have undergone surgical treatments may likely need cefazolin to prevent post-surgical bacterial infection.

Taking Subsets of Association Rules

If we are more interested in investigating associations that are linked to a specific medicine, we can narrow the rules down by making subsets. Let us try investigating rules related to fentanyl, since it appears to be the most frequently prescribed medicine.

```
fi_rules<-subset(med_rule, items %in% "fentanyl injection uh")
inspect(fi_rules)

##      lhs                                       rhs
support confidence      lift
## [1]  {ondansetron injection uh}             => {fentanyl injection uh}
0.01893939  0.3030303 0.7582938
## [2]  {fentanyl injection uh,
##       ondansetron injection uh}             => {hydrocodone acetaminophen
5mg 325mg} 0.01136364  0.6000000 1.9200000
## [3]  {hydrocodone acetaminophen 5mg 325mg,
##       ondansetron injection uh}             => {fentanyl injection uh}
0.01136364  0.3750000 0.9383886
## [4]  {cefazolin ivpb uh,
##       fentanyl injection uh}                => {heparin injection}
0.01893939  0.5000000 2.5142857
## [5]  {fentanyl injection uh,
##       heparin injection}                    => {cefazolin ivpb uh}
0.01893939  0.4761905 2.3280423
## [6]  {cefazolin ivpb uh,
##       fentanyl injection uh}                => {hydrocodone acetaminophen
5mg 325mg} 0.02462121  0.6500000 2.0800000
## [7]  {fentanyl injection uh,
##       hydrocodone acetaminophen 5mg 325mg} => {cefazolin ivpb uh}
0.02462121  0.3250000 1.5888889
## [8]  {fentanyl injection uh,
##       heparin injection}                    => {hydrocodone acetaminophen
5mg 325mg} 0.01893939  0.4761905 1.5238095
## [9]  {heparin injection,
##       hydrocodone acetaminophen 5mg 325mg} => {fentanyl injection uh}
0.01893939  0.3125000 0.7819905
## [10] {fentanyl injection uh,
##       hydrocodone acetaminophen 5mg 325mg} => {heparin injection}
0.01893939  0.2500000 1.2571429
## [11] {cefazolin ivpb uh,
##       fentanyl injection uh,
##       heparin injection}                    => {hydrocodone acetaminophen
5mg 325mg} 0.01515152  0.8000000 2.5600000
## [12] {cefazolin ivpb uh,
##       heparin injection,
##       hydrocodone acetaminophen 5mg 325mg} => {fentanyl injection uh}
0.01515152  0.4000000 1.0009479
## [13] {cefazolin ivpb uh,
##       fentanyl injection uh,
##       hydrocodone acetaminophen 5mg 325mg} => {heparin injection}
0.01515152  0.6153846 3.0945055
## [14] {fentanyl injection uh,
##       heparin injection,
##       hydrocodone acetaminophen 5mg 325mg} => {cefazolin ivpb uh}
0.01515152  0.8000000 3.9111111
```

In R scripting, the notation `%in%` signifies “belongs to.” There are 14 rules related to this item. Let’s plot them (Fig. 12.5).

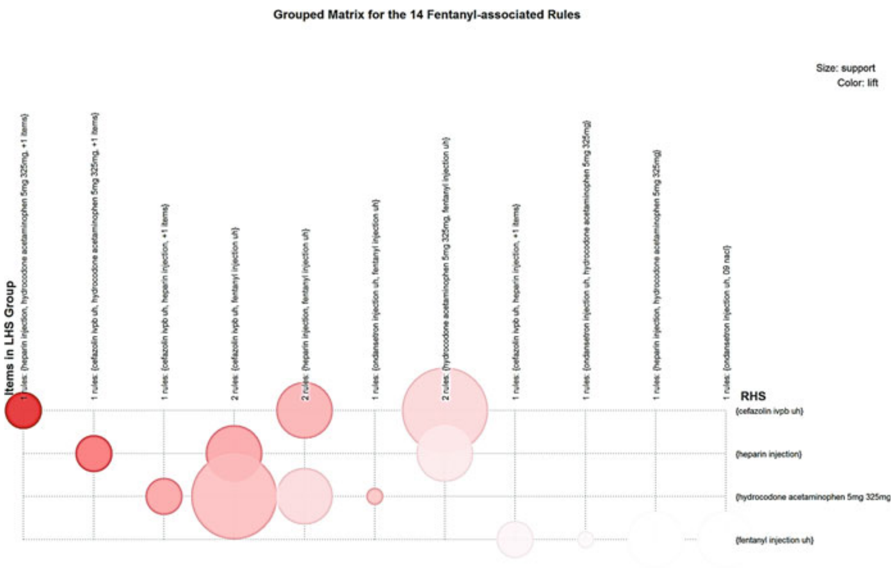


Fig. 12.5 Bubble chart of the grouped matric for 14 rules

```
plot(sort(fi_rules, by="lift"), method="grouped", control=list(type="items")
, main = "Grouped Matrix for the 14 Fentanyl-associated Rules")

## Available control parameters (with default values):
## main  =  Grouped Matrix for 14 Rules
## k     =  20
## rhs_max   =  10
## lhs_items     =  2
## aggr.fun  =  function (x, na.rm = FALSE)  UseMethod("median")
## col   =  c("#EE0000FF", "#EE0303FF", "#EE0606FF", "#EE0909FF", "#EE0C0CFF
", "#EE0F0FFF", "#EE1212FF", "#EE1515FF", "#EE1818FF", "#EE1B1BFF", "#EE1E1E
FF", "#EE2222FF", "#EE2525FF", "#EE2828FF", "#EE2B2BFF", "#EE2E2EFF", "#EE31
31FF", "#EE3434FF", "#EE3737FF", "#EE3A3AFF", "#EE3D3DFF", "#EE4040FF", "#EE
4444FF", "#EE4747FF", "#EE4A4AFF", "#EE4D4DFF", "#EE5050FF", "#EE5353FF", "#
EE5656FF", "#EE5959FF", "#EE5C5CFF", "#EE5F5FFF", "#EE6262FF", "#EE6666FF",
"#EE6969FF", "#EE6C6CFF", "#EE6F6FFF", "#EE7272FF", "#EE7575FF",  "#EE7878FF
", "#EE7B7BFF", "#EE7E7EFF", "#EE8181FF", "#EE8484FF", "#EE8888FF", "#EE8B8B
FF", "#EE8E8EFF", "#EE9191FF", "#EE9494FF", "#EE9797FF", "#EE9999FF", "#EE9B
9BFF", "#EE9D9DFF", "#EE9F9FFF", "#EEA0A0FF", "#EEA2A2FF", "#EEA4A4FF", "#EE
A5A5FF", "#EEA7A7FF", "#EEA9A9FF", "#EEABABFF", "#EEACACFF", "#EEAEAEFF", "#
EEB0B0FF", "#EEB1B1FF", "#EEB3B3FF", "#EEB5B5FF", "#EEB7B7FF", "#EEB8B8FF",
"#EEBABAFF", "#EEBCBCFF", "#EEBDBDFF", "#EEBFBFFF", "#EEC1C1FF", "#EEC3C3FF"
, "#EEC4C4FF", "#EEC6C6FF", "#EEC8C8FF",  "#EEC9C9FF", "#EECBCBFF", "#EECDCD
FF", "#EECFCFFF", "#EED0D0FF", "#EED2D2FF", "#EED4D4FF", "#EED5D5FF", "#EED7
D7FF", "#EED9D9FF", "#EEDBDBFF", "#EEDCDCFF", "#EEDEDEFF", "#EEE0E0FF", "#EE
E1E1FF", "#EEE3E3FF", "#EEE5E5FF", "#EEE7E7FF", "#EEE8E8FF", "#EEEAEAFF", "#
EEECECFF", "#EEEEEEFF")
## reverse   =  TRUE
## xlab  =  NULL
## ylab  =  NULL
## legend    =  Size: support  Color: lift
## spacing   =  -1
## panel.function    =  function (row, size, shading, spacing)  {     size[s
ize == 0] <- NA     shading[is.na(shading)] <- 1     grid.circle(x = c(1:len
gth(size)), y = row, r = size/2 * (1 - spacing), default.units = "native", g
p = gpar(fill = shading, col = shading, alpha = 0.9)) }
## gp_main   =  list(cex = 1.2, fontface = "bold", font = 2)
## gp_labels     =  list(cex = 0.8)
## gp_labs   =  list(cex = 1.2, fontface = "bold", font = 2)
## gp_lines  =  list(col = "gray", lty = 3)
## newpage   =  TRUE
## interactive    =  FALSE
## max.shading    =  NA
## verbose   =  FALSE
```

Saving Association Rules to a File or Data Frame

We can save these rules into a CSV file using `write()`. It is similar with the function `write.csv()` that we have mentioned in the beginning of this case study.

```r
write(med_rule, file = "medrule.csv", sep=",", row.names=F)
```

Sometimes it is more convenient to convert the rules into a data frame.

```r
med_df<-as(med_rule, "data.frame")
str(med_df)

## 'data.frame':    29 obs. of  4 variables:
##  $ rules     : Factor w/ 29 levels "{acetaminophen uh} => {cefazolin ivpb
uh}",..: 1 2 28 27 29 13 12 14 10 23 ...
##  $ support   : num  0.0114 0.0189 0.017 0.0189 0.0303 ...
##  $ confidence: num  0.462 0.345 0.273 0.303 0.485 ...
##  $ lift      : num  2.256 1.734 1.371 0.758 1.552 ...
```

As we can see, the rules are converted into a factor vector.

12.7 Practice Problems: Groceries

In this practice problem, we will investigate the associations of frequently purchased groceries using the *grocery* dataset in the R base. Firstly, let’s load the data.

```r
data("Groceries")
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             soda
##             2513             1903             1809             1715
##           yogurt          (Other)
##             1372            34055
```

```
##
## element (itemset/transaction) length distribution:
## sizes
##    1    2    3    4    5    6    7    8    9   10   11   12   13   14   1
5
## 2159 1643 1299 1005  855  645  545  438  350  246  182  117   78   77   5
5
##   16   17   18   19   20   21   22   23   24   26   27   28   29   32
##   46   29   14   14    9   11    4    6    1    1    1    1    3    1
##
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   1.000   2.000   3.000   4.409   6.000  32.000
##
## includes extended item information - examples:
##        labels  level2           level1
## 1 frankfurter sausage meat and sausage
## 2     sausage sausage meat and sausage
## 3  liver loaf sausage meat and sausage
```

We will try to find out the top 5 frequent grocery items and plot them (Fig. 12.6).

Then, try to use `support` = `0.006`, `confidence` = `0.25`, `minlen` = `2` to set up the grocery association rules. Sort the top 3 rules with highest lift.

```
## Apriori
##
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##        0.25    0.1    1 none FALSE            TRUE       5   0.006      2
##  maxlen target   ext
##      10  rules FALSE
##
```

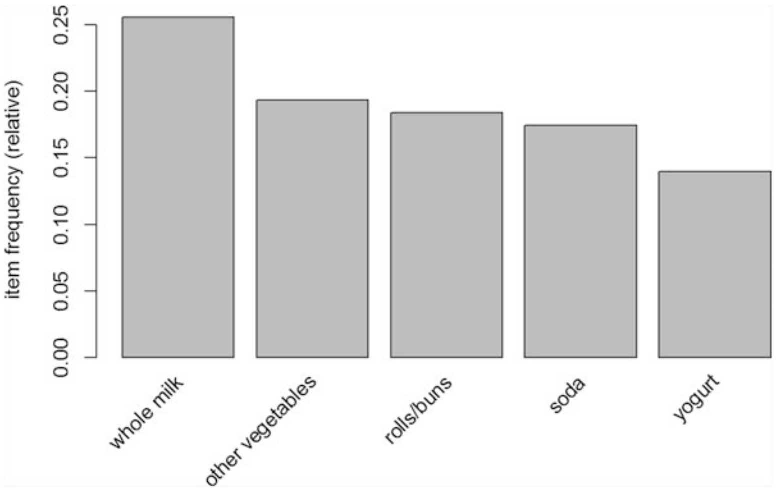


Fig. 12.6 Top-5 grocery items according to their frequencies

```
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 59
##
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [109 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 done [0.02s].
## writing ... [463 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
## set of 463 rules

##     lhs                   rhs                       support confidence
lift
## [1] {herbs}            => {root vegetables}    0.007015760  0.4312500
3.956477
## [2] {berries}          => {whipped/sour cream} 0.009049314  0.2721713
3.796886
## [3] {tropical fruit,
##      other vegetables,
##      whole milk}       => {root vegetables}    0.007015760  0.4107143
3.768074
```

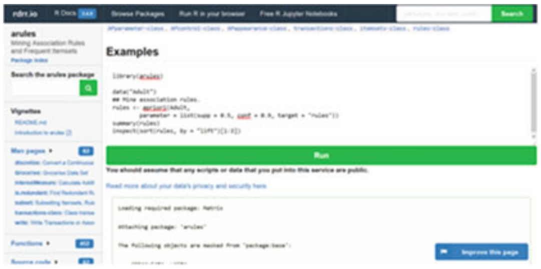
The number of rules (463) appears excessive. We can try stringer parameters. In practice, it's more possible to observe underlying rules if you set a higher confidence. Here we set the *confidence* = 0.6.

```
groceryrules <- apriori(Groceries, parameter = list(support = 0.006, confide
nce = 0.6, minlen = 2))

## Apriori
## Parameter specification:
##  confidence minval smax arem  aval originalSupport maxtime support minlen
##         0.6    0.1    1 none FALSE            TRUE       5   0.006      2
##  maxlen target   ext
##      10  rules FALSE
##
## Algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## Absolute minimum support count: 59
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[169 item(s), 9835 transaction(s)] done [0.00s].
## sorting and recoding items ... [109 item(s)] done [0.00s].
## creating transaction tree ... done [0.00s].
## checking subsets of size 1 2 3 4 done [0.02s].
## writing ... [8 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].

groceryrules

## set of 8 rules
```



Examples

```
library(arules)

data("Adult")
## Mine association rules.
rules <- apriori(Adult,
        parameter = list(supp = 0.5, conf = 0.9, target = "rules"))
summary(rules)
inspect(sort(rules, by = "lift")[1:3])
```

You should assume that any scripts or data that you put into this service are public.

Read more about your data's privacy and security here.

```
Loading required package: Matrix

Attaching package: 'arules'

The following objects are masked from 'package:base':
```

Fig. 12.7 Live demo: association rule mining

```
inspect(sort(groceryrules, by = "lift")[1:3])
```

```
##     lhs                                rhs          support     confidence
## [1] {butter,whipped/sour cream} => {whole milk} 0.006710727 0.6600000
## [2] {butter,yogurt}                 => {whole milk} 0.009354347 0.6388889
## [3] {root vegetables,butter}        => {whole milk} 0.008235892 0.6377953
##     lift
## [1] 2.583008
## [2] 2.500387
## [3] 2.496107
```

We observe mainly rules between dairy products. It makes sense that customers pick up milk when they walk down the dairy products isle. Experiment further with various parameter settings and try to interpret the results in the context of this grocery case-study (Fig. 12.7).

Mining association rules Demo https://rdrr.io/cran/arules/

```
# copy-paste this R code into the live online demo:
# https://rdrr.io/snippets/
# press RUN, and examine the results.
# The  HYPERLINK "https://archive.ics.uci.edu/ml/datasets/adult" Adult dataset includes 48842 sparse transactions
# (rows) and 115 items (columns).
library(arules)
data("Adult")
rules <- apriori(Adult,
     parameter = list(supp = 0.5, conf = 0.9, target = "rules"))
summary(rules)
inspect(sort(rules, by = "lift")[1:3])


# Results: mining info:
# data ntransactions support confidence
# Adult 48842 0.5 0.9
# lhs rhs support confidence lift count
# [1] {sex=Male, native-country=United-States} => {race=White} 0.5415421 0.9051090 1.058554 26450
# [2] {sex=Male, capital-loss=None, native-country=United-States} => {race=White} 0.5113632 0.9032585 1.056390 24976
# [3] {race=White} => {native-country=United-States} 0.7881127 0.9217231 1.027076 38493
```

12.8 Summary

- The Apriori algorithm for association rule learning is only suitable for large transactional data. For some small datasets, it might not be very helpful.
- It is useful for discovering associations, mostly in early phases of an exploratory study.

- Some rules can be built due to chance and may need further verifications.
- See also Chap. 20 (Text Mining and NLP).

Try to replicate these results with other data from the list of our Case-Studies.

12.9 Assignments: 12. Apriori Association Rules Learning

Use the SOCR Jobs Data to practice learning via Apriori Association Rules

- Load the Jobs Data. Use this guide to load HTML data.
- Focus on the Description feature. Replace all underscore characters "_" with spaces.
- Review Chap. 8, use tm package to process text data to plain text. (Hint: need to apply stemDocument as well, we will discuss more details in Chap. 20.)
- Generate a "transaction" matrix by considering each job as one record and description words as "transaction" items. (Hint: You need to fill missing values since records do not have the same length of description.)
- Save the data using write.csv() and then use read.transactions() in arules package to read the CSV data file. Visualize the item support using item frequency plots. What terms appear as more popular?
- Fit a model: myrules <– apriori(data = jobs,parameter = list(support = 0.02, confidence = 0.6, minlen = 2)). Try out several rule thresholds trading off gain and accuracy.
- Evaluate the rules you obtained with lift and visualize their metics.
- Mine medical related rules (e.g., rules include "treatment", "patient", "care", "diagnos." Notice that these are word stems).
- Sort the set of association rules for all and medical related subsets.
- Save these rules into a CSV file.

References

Witten, IH, Frank, E, Hall. MA. (2011) *Data Mining: Practical Machine Learning Tools and Techniques*, The Morgan Kaufmann Series in Data Management Systems, Elsevier, ISBN 0080890369, 9780080890364.

Soh, PJ. Woo, WL, Sulaiman, HA, Othman, MA, Saat, MS (eds). (2016) *Advances in Machine Learning and Signal Processing: Proceedings of MALSIP 2015*, Volume 387 of Lecture Notes in Electrical Engineering, Springer, ISBN 3319322133, 9783319322131.