Data Science with R
Association Rules

Graham.Williams@togaware.com

3rd August 2014

Visit http://HandsOnDataScience.com/ for more Chapters.

Association analysis defined Data Mining at its roots in 1989 and during the 1990s. It remains one of the preeminent techniques for modelling big data and so remains a core tool for the data scientist's toolbox.

As an unsupervised learning technique it has delivered considerable benefit in areas ranging from the traditional shopping basket analysis to the analysis of who bought what other books or who watched what other videos, and in areas including health care, telecommunications, and so on. Often for any data mining project we might usually begin with association analysis to identify issues with our data and then to build multiple local models. The analysis aims to identify patterns that are linked by some commonality (such as by a common person).

In this chapter we review association analysis and will discover new insights into our data through the building of association rule models.

The required packages for this module include:

```
library(arules)          # Association rules.
library(dplyr)           # Data munging: tbl_df(), %>%.
```

As we work through this chapter, new R commands will be introduced. Be sure to review the command's documentation and understand what the command does. You can ask for help using the `?` command as in:

```
?read.csv
```

We can obtain documentation on a particular package using the *help=* option of `library()`:

```
library(help=rattle)
```

This chapter is intended to be hands on. To learn effectively, you are encouraged to have R running (e.g., RStudio) and to run all the commands as they appear here. Check that you get the same output, and you understand the output. Try some variations. Explore.

1 The last.fm Dataset

We begin with a simple but real dataset example. The dataset to illustrate the actual application of association rules is the last.fm dataset. This can be downloaded from Johannes Ledolter's web site.

```
fname        <- "http://www.biz.uiowa.edu/faculty/jledolter/DataMining/lastfm.csv"
lastfm       <- read.csv(fname, stringsAsFactors=FALSE)
```

Here we process the dataset as usual, following the steps presented in Chapter Data.

```
dsname       <- "lastfm"
ds           <- get(dsname) %>% tbl_df()
ds

## Source: local data frame [289,955 x 4]
##
##    user                  artist sex country
## 1     1   red hot chili peppers   f Germany
## 2     1 the black dahlia murder   f Germany
## 3     1               goldfrapp   f Germany
## 4     1        dropkick murphys   f Germany
## 5     1                le tigre   f Germany
## 6     1              schandmaul   f Germany
## 7     1                   edguy   f Germany
## 8     1            jack johnson   f Germany
## 9     1               eluveitie   f Germany
## 10    1             the killers   f Germany
## ..  ...                     ... ...     ...
```

1.1 Preparing the last.fm Dataset

As usual we begin by cleaning the dataset. In this case there is very little that is required, simply selecting out the identifier (the user) and the items (the artist in this case). We use select() from dplyr (Wickham and Francois, 2014) to do so. Then our baskets for association analysis will be the artists that each individual user listens to. We also remove any duplicated user/artist entriesusing unique(). The transformations are performed through a pipe as implemented in magrittr (Bache and Wickham, 2014) and popularized by dplyr.

```
ds <- ds %>% select(user, artist) %>% unique()
ds

## Source: local data frame [289,953 x 2]
##
##    user                  artist
## 1     1   red hot chili peppers
## 2     1 the black dahlia murder
## 3     1               goldfrapp
## 4     1        dropkick murphys
## 5     1                le tigre
## 6     1              schandmaul
## 7     1                   edguy
## 8     1            jack johnson
## 9     1               eluveitie
## 10    1             the killers
## ..  ...                     ...
```

The dataset is now a simple data frame of two columns.

1.2 Initial Exploration of last.fm

We have a dataset now that is ready for association rule analysis and we can transform it into a transactions data object. To do so we use `as()` from `arules` (Hahsler *et al.*, 2014).

```
library(arules)

trans <- as(split(ds$artist, ds$user), "transactions")
```

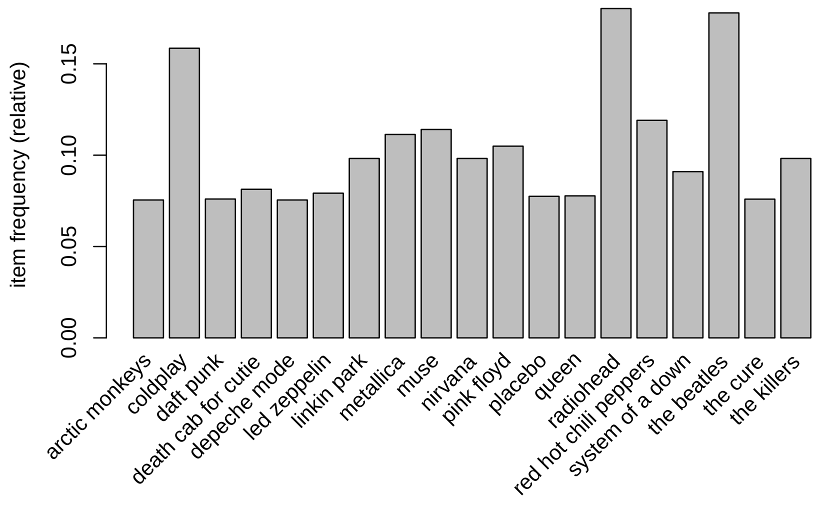
The baskets can be listed using `inspect()` and here we limit the list to the first 5 transactions.

```
inspect(trans[1:5])
```

```
##   items                        transactionID
## 1 {dropkick murphys,
##    edguy,
##    eluveitie,
##    goldfrapp,
##    guano apes,
##    jack johnson,
##    john mayer,
....
```

A plot of the more frequent items is produced by `itemFrequencyPlot()`. We can tune how many items are displayed using the `support=` threshold (how frequently an item appears, proportionately).

```
itemFrequencyPlot(trans, support=0.075)
```



1.3 Association Rule Model for last.fm

To build the model we simply call `apriori()`, an implementation of the apriori algorithm for association rule discovery. The two common parameters are `support=` and `confidence=`.

```
model <- apriori(trans, parameter=list(support=0.01, confidence=0.5))

##
## parameter specification:
##  confidence minval smax arem  aval originalSupport support minlen maxlen
##         0.5    0.1    1 none FALSE            TRUE    0.01      1     10
##  target   ext
##   rules FALSE
##
## algorithmic control:
##  filter tree heap memopt load sort verbose
##     0.1 TRUE TRUE  FALSE TRUE    2    TRUE
##
## apriori - find association rules with the apriori algorithm
## version 4.21 (2004.05.09)        (c) 1996-2004   Christian Borgelt
## set item appearances ...[0 item(s)] done [0.00s].
## set transactions ...[1004 item(s), 15000 transaction(s)] done [0.03s].
## sorting and recoding items ... [655 item(s)] done [0.01s].
## creating transaction tree ... done [0.01s].
## checking subsets of size 1 2 3 4 done [0.03s].
## writing ... [50 rule(s)] done [0.00s].
## creating S4 object  ... done [0.00s].
```

1.4 Inspecting the last.fm Model

We agai use `inspect()` to display the model. This will list all associatoin rules that satisfy the criteria specified in building the model (the `support=` and `confidence=`.

```
inspect(model)

##    lhs                          rhs             support confidence   lift
## 1  {t.i.}                    => {kanye west}   0.01040     0.5673  8.854
## 2  {the pussycat dolls}      => {rihanna}      0.01040     0.5778 13.416
## 3  {the fray}                => {coldplay}     0.01127     0.5168  3.260
## 4  {sonata arctica}          => {nightwish}    0.01347     0.5101  8.236
## 5  {judas priest}            => {iron maiden}  0.01353     0.5075  8.563
## 6  {the kinks}               => {the beatles}  0.01360     0.5299  2.979
## 7  {travis}                  => {coldplay}     0.01373     0.5628  3.550
## 8  {the flaming lips}        => {radiohead}    0.01307     0.5297  2.939
## 9  {megadeth}                => {metallica}    0.01627     0.5281  4.744
....
```

```
inspect(subset(model, subset=lift>8))

##   lhs                     rhs           support confidence   lift
## 1 {t.i.}               => {kanye west} 0.01040     0.5673  8.854
## 2 {the pussycat dolls} => {rihanna}    0.01040     0.5778 13.416
## 3 {sonata arctica}     => {nightwish}  0.01347     0.5101  8.236
## 4 {judas priest}       => {iron maiden} 0.01353     0.5075  8.563

inspect(sort(subset(model, subset=lift>8), by="confidence"))

##   lhs                     rhs           support confidence   lift
## 1 {the pussycat dolls} => {rihanna}    0.01040     0.5778 13.416
## 2 {t.i.}               => {kanye west} 0.01040     0.5673  8.854
## 3 {sonata arctica}     => {nightwish}  0.01347     0.5101  8.236
## 4 {judas priest}       => {iron maiden} 0.01353     0.5075  8.563
```

2 Understanding the Algorithm: Sample Dataset

We now use a smaller artificial dataset to illustrate the model building algorithm.

To illustrate the concepts of association analysis we will generate a random dataset of some 5 items and 10 baskets. Before doing so we use `set.seed()` to ensure we get the same "random" dataset each time and we set up some constants.

```
set.seed(42)
nb <- 10     # Number of baskets.
ni <- 5      # Number of items.
nc <- 40     # Number of combinations.
```

A simple use of `sample()` combined with `sprintf()` generates the baskets identified with an initial "b" and items with an initial "i". The use of `sort()` ensures a nicely sorted order to the baskets. With `unique()` we remove duplicate items from a basket. We revert the `rownames()` to a complete sequence for convenience.

```
ds <- data.frame(id=sort(sprintf("b%02d", sample(1:nb, nc, replace=TRUE))),
                 item=sprintf("i%1d", sample(1:ni, nc, replace=TRUE)))
ds <- unique(ds)
rownames(ds) <- NULL
```

The different baskets contain differing number of items as we can easily see with a simple dplyr sequence.

```
ds %>% group_by(id) %>% tally()

## Source: local data frame [10 x 2]
##
##     id n
## 1  b01 3
## 2  b02 2
## 3  b03 2
....
```

We can also compactly list the contents of the baskets, again using dplyr.

```
ds %>% group_by(id) %>% summarise(items=paste(sort(item), collapse=", "))

## Source: local data frame [10 x 2]
##
##     id          items
## 1  b01     i1, i2, i3
## 2  b02         i3, i5
## 3  b03         i4, i5
## 4  b04         i2, i4
## 5  b05     i1, i2, i4
## 6  b06         i1, i4
## 7  b07 i2, i3, i4, i5
## 8  b08 i1, i3, i4, i5
## 9  b09     i2, i4, i5
## 10 b10 i1, i2, i3, i4
```

2.1 1-Itemsets and Support

The basic concept of association analysis is that of **items** in baskets. Our aim is usually to identify sets of items that commonly occur together in a basket. Thus we talk about **itemsets**.

The simplest itemset is a set containing a single item. Thus $\{i_1\}$ is an itemset containing a single item, i_1. We say that this itemset has a frequency of 5. That is, it appears in 5 of the 10 baskets:

```
ds %>% group_by(id) %>% summarise(i1="i1" %in% item) %>% filter(i1)

## Source: local data frame [5 x 2]
##
##    id   i1
## 1 b01 TRUE
## 2 b05 TRUE
## 3 b06 TRUE
## 4 b08 TRUE
## 5 b10 TRUE
```

Because this itemset has a single item (i_1) we refer to it as a **1-itemset**.

The frequencies of the other 1-itemsets can be easily calculated:

```
ds %>% group_by(item) %>% tally()

## Source: local data frame [5 x 2]
##
##   item n
## 1   i1 5
## 2   i2 6
## 3   i3 5
## 4   i4 8
## 5   i5 5
```

This leads us to the concept of **support**. We define support as the proportion of all baskets that contain the itemset. There are 10 baskets, so the support for the 1-itemset $\{i_1\}$ is 5/10 or 0.5. Similarly we can calculate the support for each of our 1-itemsets:

```
ds %>% group_by(item) %>% tally() %>% mutate(s=n/nb)

## Source: local data frame [5 x 3]
##
##   item n   s
## 1   i1 5 0.5
## 2   i2 6 0.6
## 3   i3 5 0.5
## 4   i4 8 0.8
## 5   i5 5 0.5
```

This might suggest that if a 1-itemset has a support of less than 0.6 then it might not be so interesting (since all items occur with at least that frequency). We use support to tune the number of "interesting" itemsets we extract from our dataset.

2.2 1-Itemsets—Using ARules

The arules package provides support for association analysis in R. Once loaded we can transform our dataset into a *transactions* data structure. To do so we use `split()` to transform our data frame into a list of lists, each list being a basket of items, represented as a factor. The levels of the factors are the basket items.

```
library(arules)
dst <- as(split(ds$item, ds$id), "transactions")
dst

## transactions in sparse format with
##  10 transactions (rows) and
##  5 items (columns)
```

The item frequencies can be calculated simply using `itemFrequency()`:

```
itemFrequency(dst)

##  i1  i2  i3  i4  i5
## 0.5 0.6 0.5 0.8 0.5
```

Notice these are the same frequencies we manually calculated earlier.

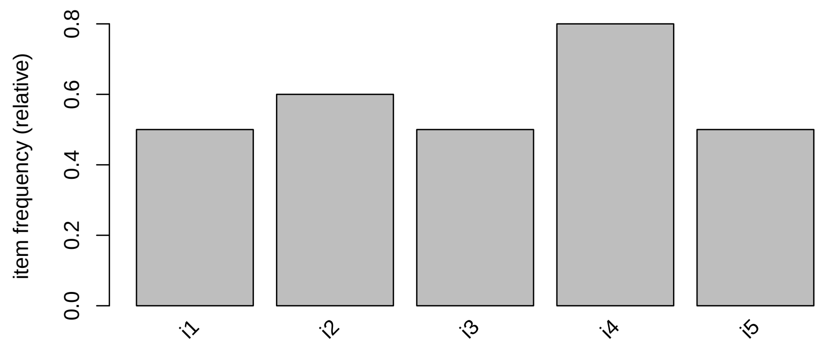
We can obtain the actual frequency count by using `type=` where the default is `"relative"`. Using `"absolute"` we see the actual counts of the items. These again agree with our earlier calculations.

```
itemFrequency(dst, type="absolute")

## i1 i2 i3 i4 i5
##  5  6  5  8  5
```

We can also obtain a frequency plot for the 1-itemsets:

```
itemFrequencyPlot(dst)
```



2.3 k-Itemsets—Using ARules

We extend the concept of a 1-itemset to the general concept of a k-itemset. For example, the 2-itemset i_1, i_2 has a frequency of 3 and so a support of 0.3.

We can calculate the 2-itemsets and their frequencies:

```
merge(ds, ds, by="id") %>%
  subset(as.character(item.x) < as.character(item.y)) %>%
  mutate(itemset=paste(item.x, item.y)) %>%
  group_by(itemset) %>%
  tally()
```

```
## Source: local data frame [10 x 2]
##
##    itemset n
## 1    i1 i2 3
## 2    i1 i3 3
## 3    i1 i4 4
## 4    i1 i5 1
## 5    i2 i3 3
## 6    i2 i4 5
## 7    i2 i5 2
## 8    i3 i4 3
## 9    i3 i5 3
## 10   i4 i5 4
```

Similarly, the 3-itemset $\{i_1, i_2, i_3\}$ has a frequency of 2 and so a support of 0.2.

```
merge(ds, ds, by="id") %>%
  merge(ds, by="id") %>%
  subset(as.character(item.x) < as.character(item.y) &
         as.character(item.y) < as.character(item)) %>%
  mutate(itemset=paste(item.x, item.y, item)) %>%
  group_by(itemset) %>%
  tally()
```

```
## Source: local data frame [9 x 2]
##
##    itemset n
## 1 i1 i2 i3 2
## 2 i1 i2 i4 2
## 3 i1 i3 i4 2
## 4 i1 i3 i5 1
## 5 i1 i4 i5 1
## 6 i2 i3 i4 2
## 7 i2 i3 i5 1
## 8 i2 i4 i5 2
## 9 i3 i4 i5 2
```

3 APriori Principle—ARules

We noted above that we are interested in finding the frequent itemsets—that is, the itemsets that have support that is at least some user specified threshold. A basic observation is that if a 2-itemset is frequent, then both of the items, by themselves as 1-itemsets, must also be frequent. Similarly, for a 3-itemset to be frequent, each of the possible 2-itemsets that only contain two items from the 3-itemset, must also be frequent. And so on.

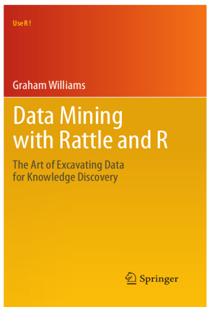
This observation is the basis for an algorithm to identify all frequent itemsets. We begin with identifying the 1-frequent itemsets and then only consider the 2-itemsets that can be constructed from the items in the frequent 1-itemsets. And so on.

The arules package implements the apriori algorithm for association rules.

4 Confidence

Once we have generated candidate (i.e., frequent enough) itemsets, we next generate the candidate association rules and retain those that have enough confidence. The confidence is a measure of the strength of an association rule. It is the frequency of occurrence of the right-hand items in the rule from among those baskets that contain the items on the left-hand side of the rule.

5 Further Reading


Use R!
Graham Williams
Data Mining with Rattle and R
The Art of Excavating Data for Knowledge Discovery
Springer

The Rattle Book, published by Springer, provides a comprehensive introduction to data mining and analytics using Rattle and R. It is available from Amazon. Other documentation on a broader selection of R topics of relevance to the data scientist is freely available from `http://datamining.togaware.com`, including the Datamining Desktop Survival Guide.

This chapter is one of many chapters available from `http://HandsOnDataScience.com`. In particular follow the links on the website with a * which indicates the generally more developed chapters.

Other resources include:

- Christian Borgelt is owed a much gratitude for the early work he did on implementing association rules in C.

6 References

Bache SM, Wickham H (2014). *magrittr: magrittr - a forward-pipe operator for R*. R package version 1.0.1, URL http://CRAN.R-project.org/package=magrittr.

Hahsler M, Buchta C, Gruen B, Hornik K (2014). *arules: Mining Association Rules and Frequent Itemsets*. R package version 1.1-3, URL http://CRAN.R-project.org/package=arules.

R Core Team (2014). *R: A Language and Environment for Statistical Computing*. R Foundation for Statistical Computing, Vienna, Austria. URL http://www.R-project.org/.

Wickham H, Francois R (2014). *dplyr: dplyr: a grammar of data manipulation*. R package version 0.2, URL http://CRAN.R-project.org/package=dplyr.

Williams GJ (2009). "Rattle: A Data Mining GUI for R." *The R Journal*, **1**(2), 45–55. URL http://journal.r-project.org/archive/2009-2/RJournal_2009-2_Williams.pdf.

Williams GJ (2011). *Data Mining with Rattle and R: The art of excavating data for knowledge discovery*. Use R! Springer, New York. URL http://www.amazon.com/gp/product/1441998896/ref=as_li_qf_sp_asin_tl?ie=UTF8&tag=togaware-20&linkCode=as2&camp=217145&creative=399373&creativeASIN=1441998896.

This document, sourced from ARulesO.Rnw revision 470, was processed by KnitR version 1.6 of 2014-05-24 and took 5.4 seconds to process. It was generated by gjw on nyx running Ubuntu 14.04.1 LTS with Intel(R) Xeon(R) CPU W3520 @ 2.67GHz having 4 cores and 12.3GB of RAM. It completed the processing 2014-08-03 17:34:03.