Association Rule Mining in R

Source: https://towardsdatascience.com/association-rule-mining-in-r-ddf2d044ae50

By Avinash Kadimisetty May 12, 2018 7 min read

Association Rule Mining (also called as Association Rule Learning) is a common technique used to find associations between many variables. It is often used by grocery stores, e-commerce websites, and anyone with large transactional databases. A most common example that we encounter in our daily lives — Amazon knows what else you want to buy when you order something on their site. The same idea extends to Spotify too — They know what song you want to listen to next. All of these incorporate, at some level, data mining concepts and association rule mining algorithms.

Market Basket Analysis is similar to ARM. Market Basket Analysis is a modelling technique based upon the theory that if you buy a certain group of items, you are more (or less) likely to buy another group of items. For example, if you are in an English pub and you buy a pint of beer and don't buy a bar meal, you are more likely to buy crisps at the same time than somebody who didn't buy beer.

Based on the concept of strong rules, Rakesh Agrawal, Tomasz Imieliński and Arun Swami introduced association rules for

discovering regularities between products in large-scale transaction data recorded by point-of-sale (POS) systems in supermarkets.

This article explains the concept of Association Rule Mining and how to use this technique in R

To perform Association Rule Mining in R, we use the `arules` and the `arulesViz` packages in R.

Michael Hahsler, et al. has authored and maintains two very useful R packages relating to association rule mining: the arules package and the arulesViz package.

If you don't have these packages installed in your system, please use the following commands to install them.

```
> install.packages("arules")
> install.packages("arulesViz")
```

Data

I'm using the AdultUCI dataset that comes bundled with the `arules` package.

```
> data("Groceries")
```

Lets inspect the Groceries data first.

```
> class(Groceries)
      [1] "transactions"
      attr(,"package")
      [1] "arules"
```

It is a transactional dataset.

```
> inspect(head(Groceries, 2))
          items
      [1] {citrus fruit,semi-finished bread,margarine,ready
soups}
      [2] {tropical fruit,yogurt,coffee}
```

The first two transactions and the items involved in each transaction can be observed from the output above.

Generating Rules

There are three parameters controlling the number of rules to be generated *viz*. **Support and Confidence**. Another parameter **Lift** is generated using Support and Confidence and is one of the major parameters to filter the generated rules.

- ***Support*** is an indication of how frequently the itemset appears in the dataset. Consider only the two transactions from the above output. The support of the item *citrus fruit* is 1/2 as it appears in only 1 out of the two transactions.
- ***Confidence*** is an indication of how often the rule has been found to be true. We will discuss more about confidence after generating the rules.

Lets find out the rules using the *apriori* algorithm.

```
> grocery_rules <- apriori(Groceries, parameter = list(support =
0.01, confidence = 0.5))AprioriParameter specification:
      confidence minval smax arem  aval originalSupport maxtime
support minlen maxlen target   ext
        0.5    0.1    1 none FALSE            TRUE       5
0.01      1     10  rules FALSEAlgorithmic control:
      filter tree heap memopt load sort verbose
      0.1 TRUE TRUE  FALSE TRUE    2    TRUEAbsolute minimum
support count: 98set item appearances ...[0 item(s)] done
```

```
[0.00s].
      set transactions ...[169 item(s), 9835 transaction(s)]
done [0.00s].
      sorting and recoding items ... [88 item(s)] done [0.00s].
      creating transaction tree ... done [0.00s].
      checking subsets of size 1 2 3 4 done [0.00s].
      writing ... [15 rule(s)] done [0.00s].
      creating S4 object  ... done [0.00s].
```

The Apriori algorithm generated 15 rules with the given constraints. Lets dive into the Parameter Specification section of the output.

- ***minval*** is the minimum value of the support an itemset should satisfy to be a part of a rule.
- ***smax*** is the maximum support value for an itemset.
- ***arem*** is an Additional Rule Evaluation Parameter. In the above code we have constrained the number of rules using Support and Confidence. There are several other ways to constrain the rules using the **arem** parameter in the function and we will discuss more about it later in the article.
- ***aval*** is a logical indicating whether to return the additional rule evaluation measure selected with arem.
- ***originalSupport*** The traditional support value only considers both LHS and RHS items for calculating support. If you want to use only the LHS items for the calculation then you need to set this to FALSE.

- ***maxtime*** is the maximum amount of time allowed to check for subsets.
- ***minlen*** is the minimum number of items required in the rule.
- ***maxlen*** is the maximum number of items that can be present in the rule.

```
> inspect(head(sort(rules, by = "confidence"), 3))
      lhs                                  rhs
support    confidence lift     count
      [1] {citrus fruit,root vegetables}   => {other vegetables}
0.01037112 0.5862069  3.029608 102
      [2] {tropical fruit,root vegetables} => {other vegetables}
0.01230300 0.5845411  3.020999 121
      [3] {curd,yogurt}                    => {whole milk}
0.01006609 0.5823529  2.279125  99
```

The top 3 rules sorted by confidence are shown above.

Limiting the number of rules generated

In many cases, you would like to limit the number of rules generated. For example, you can use association rules as predictors in Regression/Classification. You can generate rules with the Right Hand Side of the rule as your response and use the rules generated as the modelling features. In this case, you would not want to use all the rules generated as the predictors because many rules are actually subsets of bigger rules and hence you would want to eliminate them. The following code snippet shows how to generate rules whose RHS is pre-defined.

```
wholemilk_rules <- apriori(data=Groceries, parameter=list
(supp=0.001,conf = 0.08), appearance = list (rhs="whole milk"))#
The above code shows what products are bought before buying
"whole milk" and will generate rules that lead to buying "whole
milk".
```

You can limit the number of rules by tweaking a few parameters. Although the parameter tweaking depends on the kind of data you are dealing with, the most common ways include changing support, confidence and other parameters like minlen, maxlen etc.

```
> grocery_rules_increased_support <- apriori(Groceries,
parameter = list(support = 0.02, confidence = 0.5))# This
generates only one rule in the output.
```

If you want to get **stronger** rules, you have to increase the confidence. If you want **lengthier** rules increase the maxlen parameter. If you want to eliminate **shorter** rules, decrease the minlen parameter.

Sometimes you might be interested in finding the rules involving maximum number of items and remove the shorter rules that are subsets of the longer rules. The below code removes such redundant rules.

```
> subsets <- which(colSums(is.subset(grocery_rules,
groery_rules)) > 1)
> grocery_rules <- grocery_rules[-subsets]
```

Lets look at the **arem** parameter that was described earlier. The rules, after geneartion, are further evaluated based on the value of the *arem* parameter. The *arem* parameter takes the following values — *none*, *diff*, *quot*, *aimp*, *info*, *chi2*.

```
# This gives more than 1,500,000 rules
> rules <- apriori(Groceries, parameter = list(supp = 0.0001,
conf = 0.5))# This gives 982,000 rules.
> rules_chi2 <- apriori(Groceries, parameter = list(supp =
0.0001, conf = 0.5, arem = "chi2"))
```

Converting Data Frame into Transactional data

The `AdultUCI` dataset bundled with `arules` package is used.

```
> data("AdultUCI")
> class(AdultUCI)
      "data.frame"
```

When you look at the structure of the AdultUCI dataframe, you observe that a few columns are numeric. Each transaction of a transactional dataset contains the list of items involved in that transaction. When we convert the dataframe into a transactional dataset, each row of this dataframe will become a transaction. Each column will become an item. But if the value of a column is numeric, it cannot be used as the column can take infinite values. So before converting the dataframe into a transactional dataset, we must ensure that we convert each column into a factor or a logical to ensure that the column takes values only from a fixed set.

```
> str(AdultUCI)
'data.frame':48842 obs. of  15 variables:
 $ age           : int  39 50 38 53 28 37 49 52 31 42 ...
 $ workclass     : Factor w/ 8 levels "Federal-gov",..: 7 6 4 4
4 4
 $ fnlwgt        : int  77516 83311 215646 234721 338409 284582
 $ education     : Ord.factor w/ 16 levels "Preschool"<"1st-
4th"<..:
 $ education-num : int  13 13 9 7 13 14 5 9 14 13 ...
 $ marital-status: Factor w/ 7 levels "Divorced","Married-AF-
Spouse"
 $ occupation    : Factor w/ 14 levels "Adm-clerical",..: 1 4 6
6 10
 $ relationship  : Factor w/ 6 levels "Husband","Not-in-
family",..:
 $ race          : Factor w/ 5 levels "Amer-Indian-Eskimo",..: 5
5 5
 $ sex           : Factor w/ 2 levels "Female","Male": 2 2 2 2 1
1 1
 $ capital-gain  : int  2174 0 0 0 0 0 0 0 14084 5178 ...
```

```
 $ capital-loss  : int  0 0 0 0 0 0 0 0 0 0 ...
 $ hours-per-week: int  40 13 40 40 40 40 16 45 50 40 ...
 $ native-country: Factor w/ 41 levels "Cambodia","Canada",..:
39 39
 $ income        : Ord.factor w/ 2 levels "small"<"large": 1 1 1
1 1
```

In `AdultUCI` dataframe, columns 1, 3, 5, 11, 12, 13 are integers. So convert each of the column into factor.

```
> AdultUCI <- lapply(AdultUCI, function(x){as.factor(x)})>
str(AdultUCI)
List of 15
 $ age           : Factor w/ 74 levels "17","18","19",..: 23 34
22
 $ workclass     : Factor w/ 8 levels "Federal-gov",..: 7 6 4 4
4 4
 $ fnlwgt        : Factor w/ 28523 levels "12285","13492",..:
3462
 $ education     : Ord.factor w/ 16 levels "Preschool"<"1st-
4th"<..:
 $ education-num : Factor w/ 16 levels "1","2","3","4",..: 13 13
9 7
 $ marital-status: Factor w/ 7 levels "Divorced","Married-AF-
 $ occupation    : Factor w/ 14 levels "Adm-clerical",..: 1 4 6
6 10
 $ relationship  : Factor w/ 6 levels "Husband","Not-in-
family",..:
 $ race          : Factor w/ 5 levels "Amer-Indian-Eskimo",..: 5
5 5
 $ sex           : Factor w/ 2 levels "Female","Male": 2 2 2 2 1
1
 $ capital-gain  : Factor w/ 123 levels "0","114","401",..: 28 1
1 1
 $ capital-loss  : Factor w/ 99 levels "0","155","213",..: 1 1 1
1 1
 $ hours-per-week: Factor w/ 96 levels "1","2","3","4",..: 40 13
40
 $ native-country: Factor w/ 41 levels "Cambodia","Canada",..:
39 39
 $ income        : Ord.factor w/ 2 levels "small"<"large": 1 1 1
1 1
```

Now `AdultUCI` dataframe can be converted into a transactional dataset using the code snippet below.

```
> transactional_data <- as(AdultUCI, "transactions")
```