Convert Record Data to Transactions for Association Rule Mining in R

Bei Yu

April 27, 2017

# Rationale

The HW3 data set bankdata\_csv\_all.csv is in record data format. See below a snippet of the data:

id,age,sex,region,income,married,children,car,save\_act,current\_act,mortgage,pep ID12101,48,FEMALE,INNER\_CITY,17546,NO,1,NO,NO,NO,NO,pep=YES

It can be directly loaded into Weka for AR mining, but the "arules" package in R does not accept record data as input. The record data has to be transformed into transaction data first. An example of transaction data:

citrus fruit,semi-finished bread,margarine,ready soups tropical fruit,yogurt,coffee

Two transformations are needed to convert the record data into transaction data. The first step is to convert all numeric variables to nominal, because AR mining can only analyze nominal data (whether an item occurs in a transaction or not). After that, the bank data might have duplicate items like "NO, NO, NO, NO", which should be converted to "married=NO, car=NO, save\_act=NO, current\_act=NO".

There are actually multiple ways to do the conversion. Amara and Tin found two different ways, which are summarized below.

# Solution 1

Amara's solution is a direct implementation of the aforementioned two-step conversion.

This solution used a package "plyr". Because new version "dplyr" is released, I tweaked the above code to fit dplyr by replacing the "revalue" function with the new "recode" function.

## First, load the libraries

library(plyr)  
library(dplyr)

##   
## Attaching package: 'dplyr'

## The following objects are masked from 'package:plyr':  
##   
## arrange, count, desc, failwith, id, mutate, rename, summarise,  
## summarize

## The following objects are masked from 'package:stats':  
##   
## filter, lag

## The following objects are masked from 'package:base':  
##   
## intersect, setdiff, setequal, union

library(arules)

## Loading required package: Matrix

##   
## Attaching package: 'arules'

## The following object is masked from 'package:dplyr':  
##   
## recode

## The following objects are masked from 'package:base':  
##   
## abbreviate, write

## Then, load the dataset

bd = read.csv("/Users/byu/Desktop/Data/bankdata\_csv\_all.csv")  
str(bd)

## 'data.frame': 600 obs. of 12 variables:  
## $ id : Factor w/ 600 levels "ID12101","ID12102",..: 1 2 3 4 5 6 7 8 9 10 ...  
## $ age : int 48 40 51 23 57 57 22 58 37 54 ...  
## $ sex : Factor w/ 2 levels "FEMALE","MALE": 1 2 1 1 1 1 2 2 1 2 ...  
## $ region : Factor w/ 4 levels "INNER\_CITY","RURAL",..: 1 4 1 4 2 4 2 4 3 4 ...  
## $ income : num 17546 30085 16575 20375 50576 ...  
## $ married : Factor w/ 2 levels "NO","YES": 1 2 2 2 2 2 1 2 2 2 ...  
## $ children : int 1 3 0 3 0 2 0 0 2 2 ...  
## $ car : Factor w/ 2 levels "NO","YES": 1 2 2 1 1 1 1 2 2 2 ...  
## $ save\_act : Factor w/ 2 levels "NO","YES": 1 1 2 1 2 2 1 2 1 2 ...  
## $ current\_act: Factor w/ 2 levels "NO","YES": 1 2 2 2 1 2 2 2 1 2 ...  
## $ mortgage : Factor w/ 2 levels "NO","YES": 1 2 1 1 1 1 1 1 1 1 ...  
## $ pep : Factor w/ 2 levels "NO","YES": 2 1 1 1 1 2 2 1 1 1 ...

## Then the first step of conversion: discretization and numeric-to-nominal transformation.

You can choose your own way of discretization and numeric-to-nominal conversion, like using different numbers of bins or equal-frequency discretization. Below is just one of many ways to transform the variables.

### Discretize age by customized bin

bd$age <- cut(bd$age, breaks = c(0,10,20,30,40,50,60,Inf),labels=c("child","teens","twenties","thirties","fourties","fifties","old"))

### Discretize income by equal-width bin

min\_income <- min(bd$income)  
max\_income <- max(bd$income)  
bins = 3   
width=(max\_income - min\_income)/bins;  
bd$income = cut(bd$income, breaks=seq(min\_income, max\_income, width))

### Convert numeric to nominal for "children"

bd$children=factor(bd$children)

## Now the second step of conversion, changing "YES" to "[variable\_name]=YES".

bd$married=dplyr::recode(bd$married, YES="married=YES", NO="married=NO")  
bd$car=dplyr::recode(bd$car, YES="car=YES", NO="car=NO")  
bd$save\_act=dplyr::recode(bd$save\_act, YES="save\_act=YES", NO="save\_act=NO")  
bd$current\_act=dplyr::recode(bd$current\_act, YES="current\_act=YES", NO="current\_act=NO")  
bd$mortgage=dplyr::recode(bd$mortgage, YES="mortgage=YES", NO="mortgage=NO")  
bd$pep=dplyr::recode(bd$pep, YES="pep=YES", NO="pep=NO")

## Now load the transformed data into the apriori algorithm

myRules = apriori(bd, parameter = list(supp = 0.001, conf = 0.9, maxlen = 3))

## Apriori  
##   
## Parameter specification:  
## confidence minval smax arem aval originalSupport maxtime support minlen  
## 0.9 0.1 1 none FALSE TRUE 5 0.001 1  
## maxlen target ext  
## 3 rules FALSE  
##   
## Algorithmic control:  
## filter tree heap memopt load sort verbose  
## 0.1 TRUE TRUE FALSE TRUE 2 TRUE  
##   
## Absolute minimum support count: 0   
##   
## set item appearances ...[0 item(s)] done [0.00s].  
## set transactions ...[631 item(s), 600 transaction(s)] done [0.00s].  
## sorting and recoding items ... [631 item(s)] done [0.00s].  
## creating transaction tree ... done [0.00s].  
## checking subsets of size 1 2 3

## Warning in apriori(bd, parameter = list(supp = 0.001, conf = 0.9, maxlen  
## = 3)): Mining stopped (maxlen reached). Only patterns up to a length of 3  
## returned!

## done [0.01s].  
## writing ... [72691 rule(s)] done [0.01s].  
## creating S4 object ... done [0.01s].

# Solution 2

Tin's solution used two packages "readr" and "dplyr", which provided some powerful data manipulation "verbs". See below his code:

library(readr)  
library(dplyr)  
library(arules)

bankdata <- read\_csv("/Users/byu/Desktop/Data/bankdata\_csv\_all.csv")

## Parsed with column specification:  
## cols(  
## id = col\_character(),  
## age = col\_integer(),  
## sex = col\_character(),  
## region = col\_character(),  
## income = col\_double(),  
## married = col\_character(),  
## children = col\_integer(),  
## car = col\_character(),  
## save\_act = col\_character(),  
## current\_act = col\_character(),  
## mortgage = col\_character(),  
## pep = col\_character()  
## )

## Preprocess the data

# drop the id variable  
# convert categorical variables to factor  
# discretize numeric variables  
bankdata <- bankdata %>%   
 select(-id) %>%   
 mutate\_if(is.character, funs(as.factor)) %>%   
 mutate\_if(is.numeric, funs(discretize))

## Generate rules and explore

# generate rules  
rules <- apriori(bankdata, parameter = list(supp = 0.001, conf = 0.9, maxlen = 4))

## Apriori  
##   
## Parameter specification:  
## confidence minval smax arem aval originalSupport maxtime support minlen  
## 0.9 0.1 1 none FALSE TRUE 5 0.001 1  
## maxlen target ext  
## 4 rules FALSE  
##   
## Algorithmic control:  
## filter tree heap memopt load sort verbose  
## 0.1 TRUE TRUE FALSE TRUE 2 TRUE  
##   
## Absolute minimum support count: 0   
##   
## set item appearances ...[0 item(s)] done [0.00s].  
## set transactions ...[27 item(s), 600 transaction(s)] done [0.00s].  
## sorting and recoding items ... [27 item(s)] done [0.00s].  
## creating transaction tree ... done [0.00s].  
## checking subsets of size 1 2 3 4

## Warning in apriori(bankdata, parameter = list(supp = 0.001, conf =  
## 0.9, : Mining stopped (maxlen reached). Only patterns up to a length of 4  
## returned!

## done [0.00s].  
## writing ... [953 rule(s)] done [0.00s].  
## creating S4 object ... done [0.00s].

# Solution 3

Use a Weka wrapper "RWeka" in R. See more usage details in the official tutorial: <https://cran.r-project.org/web/packages/RWeka/RWeka.pdf>

library(RWeka)  
bankdata = read.csv("/Users/byu/Desktop/Data/bankdata\_csv\_all.csv")  
rules <- Apriori(bankdata, control = Weka\_control(C = 0.8))