Data\_Mining.R

Mon May 29 18:34:42 2017

# Task 1  
  
# General Objective  
# To find the best interesting sequential rules  
# Data sets  
# • diab\_trans.data  
# • http://mlr.cs.umass.edu/ml/datasets/Diabetes does not exist  
# https://archive.ics.uci.edu/ml/datasets/diabetes correct location  
  
#Needed libraries  
library(arules)

## Loading required package: Matrix

##   
## Attaching package: 'arules'

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

library(arulesSequences)  
  
#read in the data  
diab.df <- read.csv("diab\_trans.data", header=TRUE, stringsAsFactors = FALSE)  
  
diab.df <- diab.df[complete.cases(diab.df),]  
  
#define the column names  
colnames(diab.df) <- c("ID", "time", "eventID", "value")  
  
#eventID formatting e.g. id\_65 to 65  
eventID\_to\_int <- function(frame) {  
 apply(frame, 1, function(x) strtoi(unlist(strsplit(x[3], "\_"))[2]))  
}  
diab.df$eventID <- eventID\_to\_int(diab.df)  
  
# clean the data for use, throw out data we don't need  
# events where id < 64 we don't want not transactional data items, 65+ we need. We want to create baskets for the events that do have values  
# so that similar values are grouped together during rule mining.  
num\_baskets <- 3  
out.df <- data.frame()  
for (id in unique(diab.df$eventID)) {  
 cat("processing eventID: ", id, "\n")  
 baskets <- list(dim=(num\_baskets+1))  
 if(id >= 65) {  
 next  
 }  
  
 sorted\_values <- sort(diab.df$value[diab.df$eventID == id])  
 len <- length(sorted\_values)  
 step <- len / num\_baskets  
 for (j in 1:(num\_baskets)) {  
 baskets <- append(baskets, sorted\_values[j \* step])  
 }  
 baskets <- append(baskets, len)  
  
 sub <- subset(diab.df, eventID == id)  
 for (row in 1:nrow(sub)) {  
 s <- sub[row,]  
 for (k in 1:(num\_baskets)) {  
 if ((s[4] >= baskets[k]) && (s[4] <= baskets[k+1])) {  
 mod\_sample <- s  
 mod\_sample[3] = 100 \* s[3] + k # change event ID  
 out.df <- rbind(out.df, mod\_sample)  
 break  
 }  
 }  
 }  
}

## processing eventID: 58   
## processing eventID: 33   
## processing eventID: 34   
## processing eventID: 62   
## processing eventID: 48   
## processing eventID: 65   
## processing eventID: 60   
## processing eventID: 35   
## processing eventID: 56   
## processing eventID: 64   
## processing eventID: 61   
## processing eventID: 67   
## processing eventID: 63   
## processing eventID: 57   
## processing eventID: 72   
## processing eventID: 68   
## processing eventID: 69   
## processing eventID: 59   
## processing eventID: 71   
## processing eventID: 66   
## processing eventID: 70   
## processing eventID: 36

unique(out.df$eventID)

## [1] 5801 5803 5802 3303 3302 3301 3402 3403 3401 6202 6203 6201 4801 4803  
## [15] 4802 6003 6001 6002 3501 3502 3503 5601 5602 5603 6402 6401 6403 6101  
## [29] 6102 6103 6302 6301 6303 5702 5701 5703 5903 5901 5902 3601

data.df <- out.df  
data.df <- rbind(data.df, subset(diab.df, eventID > 64))  
  
head(subset(data.df, eventID==5803))

## ID time eventID value  
## 7 1 96852901 5803 216  
## 13 1 96938701 5803 257  
## 18 1 97026721 5803 239  
## 39 1 97293781 5803 259  
## 84 1 97804081 5803 305  
## 118 1 98235781 5803 251

data.df.sorted <- data.df[order(data.df["ID"], data.df["time"]),]  
  
write.table(data.df.sorted, "diab\_baskets.data", sep = ",", row.names = FALSE, col.names = FALSE)  
write.table(data.df.sorted[,-c(4)], "diab\_baskets\_novalues.data", sep = ",", row.names = FALSE, col.names = FALSE)  
  
diabSeq <- read\_baskets(con = "diab\_baskets\_novalues.data", sep =",", info = c("sequenceID","eventID"))  
  
seqParam = new ("SPparameter",support = 0.5, maxsize = 4, mingap=600, maxgap =172800, maxlen = 3 )  
patSeq= cspade(diabSeq,seqParam, control = list(verbose = TRUE, tidLists = FALSE, summary= TRUE))

##   
## parameter specification:  
## support : 0.5  
## maxsize : 4  
## maxlen : 3  
## mingap : 600  
## maxgap : 172800  
##   
## algorithmic control:  
## bfstype : FALSE  
## verbose : TRUE  
## summary : TRUE  
## tidLists : FALSE  
##   
## preprocessing ... 1 partition(s), 0.48 MB [0.057s]  
## mining transactions ... 0.03 MB [0.037s]  
## reading sequences ... [0.096s]  
##   
## total elapsed time: 0.19s

#set the confidence at 80% but can be increased to 90%  
seqRules = ruleInduction(patSeq,confidence = 0.8)  
  
length(seqRules)

## [1] 531

#Summary of the sequence rules  
summary(seqRules)

## set of 531 sequencerules with  
##   
## rule size distribution (lhs + rhs)  
## sizes  
## 2 3   
## 99 432   
##   
## rule length distribution (lhs + rhs)  
## lengths  
## 2 3   
## 137 394   
##   
## summary of quality measures:  
## support confidence lift   
## Min. :0.5000 Min. :0.8000 Min. :0.8949   
## 1st Qu.:0.5758 1st Qu.:0.8265 1st Qu.:0.9483   
## Median :0.6364 Median :0.8545 Median :0.9859   
## Mean :0.6368 Mean :0.8623 Mean :0.9941   
## 3rd Qu.:0.6970 3rd Qu.:0.8936 3rd Qu.:1.0342   
## Max. :0.8333 Max. :0.9821 Max. :1.5178   
##   
## mining info:  
## data ntransactions nsequences support confidence  
## diabSeq 22491 66 0.5 0.8

#inspect the firs 100 rules  
#inspect(head(seqRules,100))  
#inspect all rules 531  
#inspect(seqRules,531)  
  
#top 10 rules  
inspect(head(sort(seqRules, by=c("confidence", "support")),10))

## lhs rhs support confidence lift   
## 1 <{5803}> => <{6203}> 0.8333333 0.9821429 1.117611   
## 2 <{5803}> => <{5803}> 0.8333333 0.9821429 1.157526   
## 3 <{5803}> => <{5802}> 0.8333333 0.9821429 1.098668   
## 4 <{5803},   
## {6001}> => <{5802}> 0.6969697 0.9787234 1.094843   
## 5 <{3302},   
## {6001}> => <{5802}> 0.6060606 0.9756098 1.091360   
## 6 <{3303},   
## {3303}> => <{3302}> 0.5151515 0.9714286 1.144898   
## 7 <{5801},   
## {3303}> => <{3302}> 0.5000000 0.9705882 1.143908   
## 8 <{5803},   
## {3303}> => <{3302}> 0.5000000 0.9705882 1.143908   
## 9 <{5803}> => <{5801}> 0.8181818 0.9642857 1.078692   
## 10 <{6001}> => <{5802}> 0.8030303 0.9636364 1.077966   
##

inspect(head(sort(seqRules, by=c("support", "confidence")),10))

## lhs rhs support confidence lift   
## 1 <{5803}> => <{6203}> 0.8333333 0.9821429 1.117611   
## 2 <{5803}> => <{5803}> 0.8333333 0.9821429 1.157526   
## 3 <{5803}> => <{5802}> 0.8333333 0.9821429 1.098668   
## 4 <{6203}> => <{5802}> 0.8333333 0.9482759 1.060783   
## 5 <{6201}> => <{6203}> 0.8333333 0.9322034 1.060783   
## 6 <{5802}> => <{6201}> 0.8333333 0.9322034 1.042804   
## 7 <{6201}> => <{5803}> 0.8333333 0.9322034 1.098668   
## 8 <{5802}> => <{5802}> 0.8333333 0.9322034 1.042804   
## 9 <{5802}> => <{5801}> 0.8333333 0.9322034 1.042804   
## 10 <{6201}> => <{5801}> 0.8333333 0.9322034 1.042804   
##

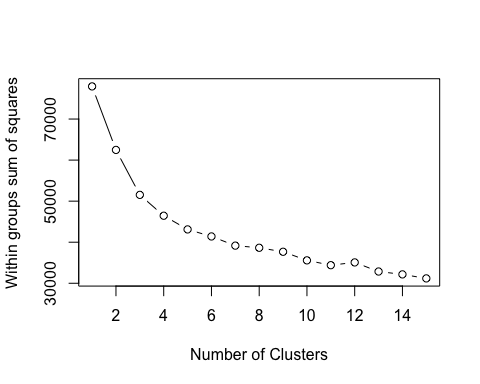
# Task 2  
# To find the best classifier for a selected dataset.  
# Datasets: • Wine Quality (two sets) - http://archive.ics.uci.edu/ml/datasets/Wine+Quality  
  
# Algorithm used k-means  
  
wine <- read.csv('winequality-white.csv', sep=';')  
wine <- rbind(wine, read.csv('winequality-red.csv', sep=';'))  
summary(wine)

## fixed.acidity volatile.acidity citric.acid residual.sugar   
## Min. : 3.800 Min. :0.0800 Min. :0.0000 Min. : 0.600   
## 1st Qu.: 6.400 1st Qu.:0.2300 1st Qu.:0.2500 1st Qu.: 1.800   
## Median : 7.000 Median :0.2900 Median :0.3100 Median : 3.000   
## Mean : 7.215 Mean :0.3397 Mean :0.3186 Mean : 5.443   
## 3rd Qu.: 7.700 3rd Qu.:0.4000 3rd Qu.:0.3900 3rd Qu.: 8.100   
## Max. :15.900 Max. :1.5800 Max. :1.6600 Max. :65.800   
## chlorides free.sulfur.dioxide total.sulfur.dioxide  
## Min. :0.00900 Min. : 1.00 Min. : 6.0   
## 1st Qu.:0.03800 1st Qu.: 17.00 1st Qu.: 77.0   
## Median :0.04700 Median : 29.00 Median :118.0   
## Mean :0.05603 Mean : 30.53 Mean :115.7   
## 3rd Qu.:0.06500 3rd Qu.: 41.00 3rd Qu.:156.0   
## Max. :0.61100 Max. :289.00 Max. :440.0   
## density pH sulphates alcohol   
## Min. :0.9871 Min. :2.720 Min. :0.2200 Min. : 8.00   
## 1st Qu.:0.9923 1st Qu.:3.110 1st Qu.:0.4300 1st Qu.: 9.50   
## Median :0.9949 Median :3.210 Median :0.5100 Median :10.30   
## Mean :0.9947 Mean :3.219 Mean :0.5313 Mean :10.49   
## 3rd Qu.:0.9970 3rd Qu.:3.320 3rd Qu.:0.6000 3rd Qu.:11.30   
## Max. :1.0390 Max. :4.010 Max. :2.0000 Max. :14.90   
## quality   
## Min. :3.000   
## 1st Qu.:5.000   
## Median :6.000   
## Mean :5.818   
## 3rd Qu.:6.000   
## Max. :9.000

wine<-scale(wine)  
wss<-(nrow(wine)-1)\*sum(apply(wine,2,var))  
for(i in 1:15) wss[i]<-sum(kmeans(wine,centers=i)$withinss)

## Warning: did not converge in 10 iterations

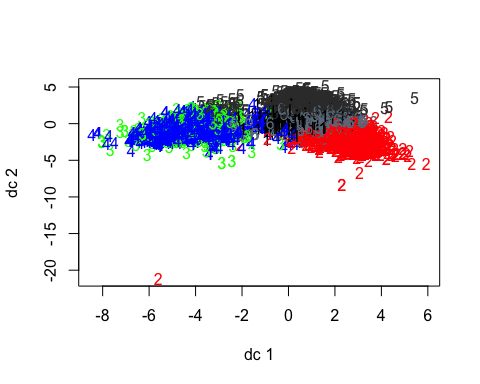
plot(1:15,wss,type='b',xlab="Number of Clusters",ylab='Within groups sum of squares')



fit1 <- kmeans(wine,6)  
fit2 <- kmeans(wine,8)  
  
table(fit1$cluster)

##   
## 1 2 3 4 5 6   
## 1083 1500 600 975 1366 973

library(fpc)  
plotcluster(wine, fit1$cluster)



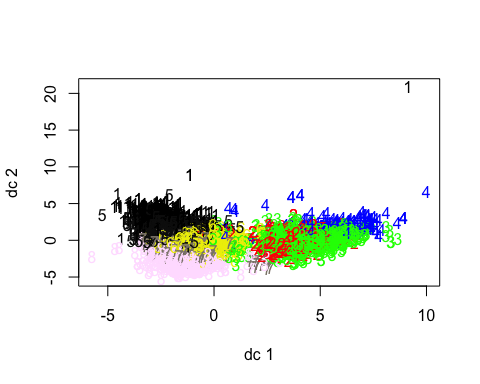
aggregate(wine,by=list(fit1$cluster),FUN=mean)

## Group.1 fixed.acidity volatile.acidity citric.acid residual.sugar  
## 1 1 0.02658223 -0.3652429 0.13013296 -0.3790194  
## 2 2 -0.17199007 -0.3486631 0.31269464 1.4390319  
## 3 3 2.06915799 0.4596025 1.02109573 -0.5718819  
## 4 4 0.11129198 1.6896941 -1.23949244 -0.6257476  
## 5 5 -0.43000575 -0.3331523 0.04900162 -0.4071079  
## 6 6 -0.54822206 -0.5648248 -0.08331397 -0.2453519  
## chlorides free.sulfur.dioxide total.sulfur.dioxide density  
## 1 -0.1704648 -0.43034870 -0.002423892 -0.4376988  
## 2 -0.1381111 0.94330029 0.999213951 0.9063185  
## 3 1.2857981 -0.91178591 -1.309928258 0.9832681  
## 4 0.6936047 -0.79186131 -1.141286886 0.5137972  
## 5 -0.5714709 -0.03578313 -0.106633997 -1.2392309  
## 6 -0.2829745 0.43076391 0.563389370 -0.2914426  
## pH sulphates alcohol quality  
## 1 -0.64735986 -0.48234740 -0.17844888 -0.6061981  
## 2 -0.50145479 -0.27679202 -0.87350969 -0.2493097  
## 3 -0.07173568 1.45983298 0.08242585 0.1068290  
## 4 0.94012888 0.41246945 -0.25513001 -0.5272559  
## 5 -0.11784150 -0.27777239 1.32495996 0.9599531  
## 6 0.76121292 0.04003067 -0.11004562 0.1738524

mydata <- data.frame(wine, fit1$cluster)  
  
  
table(fit2$cluster)

##   
## 1 2 3 4 5 6 7 8   
## 864 458 910 260 894 1093 852 1166

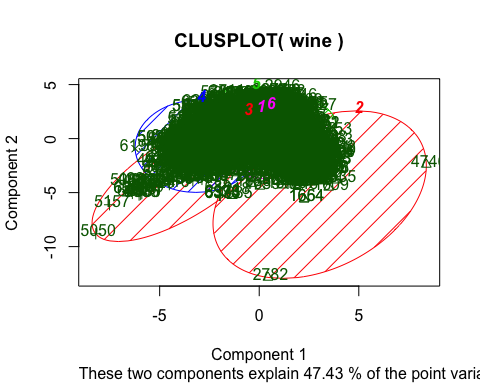
library(fpc)  
plotcluster(wine, fit2$cluster)



aggregate(wine,by=list(fit2$cluster),FUN=mean)

## Group.1 fixed.acidity volatile.acidity citric.acid residual.sugar  
## 1 1 -0.04948163 -0.3999160 0.42802802 1.8795740  
## 2 2 1.15632859 0.2313947 0.60770241 -0.5840333  
## 3 3 0.12178003 1.7601146 -1.30102233 -0.6287055  
## 4 4 2.82200233 0.7851060 1.39628563 -0.5028914  
## 5 5 -0.34476312 -0.3179427 0.09869520 0.5987685  
## 6 6 -0.01230106 -0.3744696 0.08363466 -0.3872400  
## 7 7 -0.66197453 -0.6082574 -0.10698176 -0.4028683  
## 8 8 -0.38226520 -0.3040409 0.07225936 -0.3622580  
## chlorides free.sulfur.dioxide total.sulfur.dioxide density  
## 1 -0.16393840 0.56585273 0.82153751 1.2108108  
## 2 0.56442951 -0.87875489 -1.31351654 0.6045845  
## 3 0.68838734 -0.79300256 -1.14462314 0.5194984  
## 4 2.22766311 -0.87469546 -1.17058801 1.3642640  
## 5 -0.07322061 1.33450424 1.19607574 0.3341502  
## 6 -0.21263108 -0.45577996 0.01446590 -0.4320395  
## 7 -0.38842154 0.23371999 0.28394066 -0.5646911  
## 8 -0.59493197 -0.02691476 -0.07656799 -1.2829217  
## pH sulphates alcohol quality  
## 1 -0.6438572 -0.2085724 -0.9687819 -0.08095288  
## 2 0.3997338 1.2909045 0.3656766 0.32049685  
## 3 0.9295901 0.3627107 -0.2896918 -0.56844740  
## 4 -0.5741038 1.6653149 -0.2734607 -0.24566878  
## 5 -0.2004244 -0.3009329 -0.6484230 -0.42735219  
## 6 -0.5678343 -0.4840992 -0.2031437 -0.59560596  
## 7 0.9564684 0.1323480 0.2167924 0.48889193  
## 8 -0.2903382 -0.4191101 1.3904683 0.96126159

mydata <- data.frame(wine, fit2$cluster)  
  
library(cluster)  
clusplot(wine, fit1$cluster, color=TRUE, shade=TRUE,labels=2, lines=0)



cluster.stats(fit1$cluster, fit2$cluster)

## Warning in as.dist.default(d): non-square matrix

## Warning in as.matrix.dist(d): number of items to replace is not a multiple  
## of replacement length

## $n  
## [1] 6497  
##   
## $cluster.number  
## [1] 8  
##   
## $cluster.size  
## [1] 864 458 910 260 894 1093 852 1166  
##   
## $min.cluster.size  
## [1] 260  
##   
## $noisen  
## [1] 0  
##   
## $diameter  
## [1] 6 6 6 6 6 6 6 6  
##   
## $average.distance  
## [1] 3.453417 3.450976 3.457856 3.453519 3.447685 3.454600 3.461912 3.457732  
##   
## $median.distance  
## [1] 4 4 4 4 4 4 4 4  
##   
## $separation  
## [1] 1 1 1 1 1 1 1 1  
##   
## $average.toother  
## [1] 3.454517 3.459380 3.455563 3.454241 3.454866 3.454361 3.456065 3.458052  
##   
## $separation.matrix  
## [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8]  
## [1,] 0 1 1 1 1 1 1 1  
## [2,] 1 0 1 1 1 1 1 1  
## [3,] 1 1 0 1 1 1 1 1  
## [4,] 1 1 1 0 1 1 1 1  
## [5,] 1 1 1 1 0 1 1 1  
## [6,] 1 1 1 1 1 0 1 1  
## [7,] 1 1 1 1 1 1 0 1  
## [8,] 1 1 1 1 1 1 1 0  
##   
## $ave.between.matrix  
## [,1] [,2] [,3] [,4] [,5] [,6] [,7]  
## [1,] 0.000000 3.454227 3.446801 3.449110 3.452589 3.452704 3.461662  
## [2,] 3.454227 0.000000 3.452802 3.447338 3.465156 3.460235 3.462741  
## [3,] 3.446801 3.452802 0.000000 3.455063 3.459433 3.455385 3.457953  
## [4,] 3.449110 3.447338 3.455063 0.000000 3.449802 3.452572 3.457530  
## [5,] 3.452589 3.465156 3.459433 3.449802 0.000000 3.448501 3.452772  
## [6,] 3.452704 3.460235 3.455385 3.452572 3.448501 0.000000 3.453788  
## [7,] 3.461662 3.462741 3.457953 3.457530 3.452772 3.453788 0.000000  
## [8,] 3.459813 3.463330 3.458706 3.462680 3.457574 3.457791 3.452156  
## [,8]  
## [1,] 3.459813  
## [2,] 3.463330  
## [3,] 3.458706  
## [4,] 3.462680  
## [5,] 3.457574  
## [6,] 3.457791  
## [7,] 3.452156  
## [8,] 0.000000  
##   
## $average.between  
## [1] 3.455856  
##   
## $average.within  
## [1] 3.455446  
##   
## $n.between  
## [1] 18139852  
##   
## $n.within  
## [1] 2962404  
##   
## $max.diameter  
## [1] 6  
##   
## $min.separation  
## [1] 1  
##   
## $within.cluster.ss  
## [1] 48569.99  
##   
## $clus.avg.silwidths  
## 1 2 3 4 5 6   
## -0.04714926 -0.02918535 -0.03286101 -0.03120517 -0.04366999 -0.04586399   
## 7 8   
## -0.04677577 -0.04479729   
##   
## $avg.silwidth  
## [1] -0.04207752  
##   
## $g2  
## NULL  
##   
## $g3  
## NULL  
##   
## $pearsongamma  
## [1] 8.170904e-05  
##   
## $dunn  
## [1] 0.1666667  
##   
## $dunn2  
## [1] 0.9956352  
##   
## $entropy  
## [1] 2.006846  
##   
## $wb.ratio  
## [1] 0.9998815  
##   
## $ch  
## [1] -926.7143  
##   
## $cwidegap  
## [1] 1 1 1 1 1 1 1 1  
##   
## $widestgap  
## [1] 1  
##   
## $sindex  
## [1] 1  
##   
## $corrected.rand  
## NULL  
##   
## $vi  
## NULL