nba\_stats.R

stell

Wed Feb 28 22:14:19 2018

library(data.table)  
library(dplyr)

##   
## Attaching package: 'dplyr'

## The following objects are masked from 'package:data.table':  
##   
## between, first, last

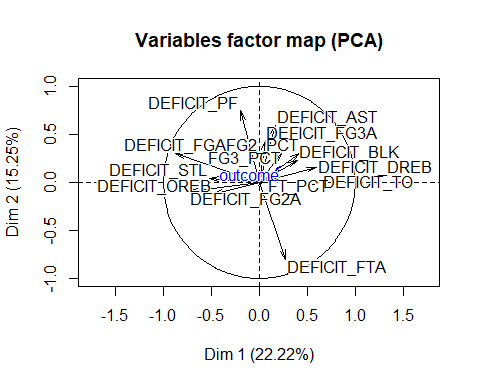
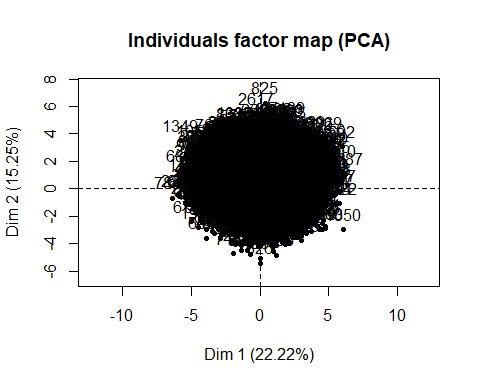
## The following objects are masked from 'package:stats':  
##   
## filter, lag

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

library(ggplot2)  
library(mclust)

## Package 'mclust' version 5.4  
## Type 'citation("mclust")' for citing this R package in publications.

library(FactoMineR)  
  
  
# read all csv files into one data.table  
read.files <- function(){  
 files <- list.files(pattern = '\\.csv')  
 tables <- lapply(files, read.csv, header = TRUE)  
 combined.df <- do.call(rbind , tables)  
 return(data.table(combined.df))  
}  
  
  
match = read.files()  
match = data.table(match[complete.cases(match)])  
match[,DEFICIT\_REB:=NULL]  
# organize data  
# outcome - The outcome of a match  
# deficty table - each row contains data of a team in a certain match.  
outcome=match$OUTCOME  
deficity\_table = match[ ,(grepl('(?:DEFICIT|PCT).\*(?<!M)$', names(match), perl = T)), with=F]  
  
# Plotting data in pairs, trying to find correlation between features.  
# plot(deficity\_table, col=outcome+2  
pca = PCA(cbind(deficity\_table, outcome), quanti.sup = ncol(deficity\_table)+1)



pca\_desc = dimdesc(pca,1)  
print(pca\_desc)

## $Dim.1  
## $Dim.1$quanti  
## correlation p.value  
## DEFICIT\_TO 0.64782496 0.000000e+00  
## DEFICIT\_DREB 0.59689117 0.000000e+00  
## FG2\_PCT 0.42325025 0.000000e+00  
## DEFICIT\_BLK 0.39837306 0.000000e+00  
## DEFICIT\_FTA 0.27627513 1.252301e-163  
## FG3\_PCT 0.23840636 4.008164e-121  
## outcome 0.23476968 2.062951e-117  
## DEFICIT\_AST 0.16766272 5.453914e-60  
## FT\_PCT 0.07536078 2.853709e-13  
## DEFICIT\_FG3A 0.06169043 2.316470e-09  
## DEFICIT\_PF -0.19443711 1.976765e-80  
## DEFICIT\_OREB -0.48243768 0.000000e+00  
## DEFICIT\_STL -0.51519499 0.000000e+00  
## DEFICIT\_FG2A -0.72989726 0.000000e+00  
## DEFICIT\_FGA -0.86995385 0.000000e+00

# scale data.  
scaled\_data = scale(deficity\_table)  
  
# picked what apeers to be the strongest feature:  
# DEFICIT\_DREB - Defensieve rebound.  
# Finding most correlated features with DEFICIT\_DREB  
  
pair.correlation = function(tb, feature, thresh){  
 len = ncol(tb)  
 correlated = vector()  
 correlated[1] = feature  
 rand.index = vector()  
 i = 1  
 for ( m in 1:len ) {  
 if (m == feature ){  
 next()  
 }  
 clust = kmeans(scaled\_data[, c(feature, m)], 2)  
 rand.index[i] = adjustedRandIndex(outcome, clust$cluster)  
 if( rand.index[i] > thresh){  
 correlated[i+1] = m  
 i = i + 1  
 }  
 }  
 return(list(correlated, mean(rand.index)))  
}  
  
# Checking that truly DREB is the most corrlated feature.   
max.corr.feat = 0  
max.rand.index= -Inf  
for (i in c(1:ncol(scaled\_data))){  
 pairs = pair.correlation(scaled\_data, i, 0.20)  
 if ( max.rand.index < unlist(pairs[2]) ){  
 max.rand.index = pairs[2]  
 max.corr.feat = i  
 feat.group = unlist(pairs[1])  
 }  
}

## Warning: Quick-TRANSfer stage steps exceeded maximum (= 468100)  
  
## Warning: Quick-TRANSfer stage steps exceeded maximum (= 468100)

print(paste0("Correlated feature: ", max.corr.feat))

## [1] "Correlated feature: 8"

print(feat.group)

## [1] 8 1 7 9

# Find the best combinations from picked features, using kmeans clustering and rand index.  
  
brute.force.F.S <- function(dt, true.lables, chosen\_features)  
{  
 if ( length(chosen\_features) == 1 ){  
 print("Only one feature. No work here.")  
 return(chosen\_features)  
 }  
 selected.features = vector()  
 max.rand.index = -Inf  
 for (m in c(1:length(chosen\_features))) {  
 combs = combn(chosen\_features, m) # for each Combination.  
 for (i in c(1:ncol(combs))) {  
 features = combs[, i]  
 clust = kmeans(dt[, features] , 2)  
 current.rand.index = adjustedRandIndex(true.lables, clust$cluster)  
 if (current.rand.index > max.rand.index) {  
 max.rand.index = current.rand.index  
 selected.features = features  
 }  
 }  
 }  
 print('for')  
 print(selected.features)  
 print(paste0("rand index: ", max.rand.index))  
 print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')  
 return(selected.features)   
}  
  
  
selected\_features = brute.force.F.S(scaled\_data, outcome, feat.group)

## [1] "for"  
## [1] 8 1 9  
## [1] "rand index: 0.296916134517693"  
## [1] "\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"

# find out how many teams whom have greater features actually won the game for each combination  
# of features.  
outcome.to.featurs = function(dt, true.lables, chosen\_features ){  
 num\_feat = length(chosen\_features)  
 dt = dt[,chosen\_features, with = F]  
 dt[,OUTCOME:=true.lables]  
 diff\_dt=dt[seq(2,nrow(dt),2),]-dt[seq(1,nrow(dt)-1,2),]  
 diff\_dt=sign(diff\_dt)  
   
   
 diff\_dt[,SUM\_ALL:=rowSums(diff\_dt)][,SUM\_FEATURES:=rowSums(diff\_dt[,1:num\_feat, with = F])]  
 diff\_dt=diff\_dt[abs(SUM\_FEATURES)==(num\_feat)]  
 won\_features = nrow(diff\_dt)  
 diff\_dt=diff\_dt[abs(SUM\_ALL)==(num\_feat+1)]  
 won\_games = nrow(diff\_dt)  
 print("present data:")  
 print(chosen\_features)  
 print(paste0("won features: ", won\_features, " won games: ", won\_games))  
 print(paste0("score: ", won\_games / won\_features))  
 print('---------------------------------------')  
}  
  
for (m in c(1:length(selected\_features))) {  
 combs = combn(selected\_features, m) # for each Combination.  
 for (i in c(1:ncol(combs))) {  
 features = combs[, i]  
 outcome.to.featurs(as.data.table(scaled\_data), outcome, features)  
 }  
}

## [1] "present data:"  
## [1] 8  
## [1] "won features: 4423 won games: 3284"  
## [1] "score: 0.742482477956138"  
## [1] "---------------------------------------"  
## [1] "present data:"  
## [1] 1  
## [1] "won features: 4615 won games: 3213"  
## [1] "score: 0.696208017334778"  
## [1] "---------------------------------------"  
## [1] "present data:"  
## [1] 9  
## [1] "won features: 4384 won games: 3165"  
## [1] "score: 0.721943430656934"  
## [1] "---------------------------------------"  
## [1] "present data:"  
## [1] 8 1  
## [1] "won features: 2697 won games: 2313"  
## [1] "score: 0.85761957730812"  
## [1] "---------------------------------------"  
## [1] "present data:"  
## [1] 8 9  
## [1] "won features: 2618 won games: 2269"  
## [1] "score: 0.866692131398014"  
## [1] "---------------------------------------"  
## [1] "present data:"  
## [1] 1 9  
## [1] "won features: 2825 won games: 2331"  
## [1] "score: 0.825132743362832"  
## [1] "---------------------------------------"  
## [1] "present data:"  
## [1] 8 1 9  
## [1] "won features: 1859 won games: 1732"  
## [1] "score: 0.93168370091447"  
## [1] "---------------------------------------"