Maboudou Independent Study

## Multivariate Change Point Using Hotelling T2

require(plyr)

## Loading required package: plyr

## Warning: package 'plyr' was built under R version 3.3.1

require(MASS)

## Loading required package: MASS

require(methods)  
  
## HotellingT2- return Hotelling T2 value for given Y and W matrix  
  
HotellingT2 <- function(y, w){  
 y = matrix(y, nrow = 1)  
 w = as.matrix(w)  
 y %\*% solve(w) %\*% t(y)  
}  
  
  
## T2ChangePoint - return two values (1) Maximum T2 and (2) the value of k  
  
T2ChangePoint <- function(x, keepobs = NULL){  
 x = as.matrix(x)  
 #print(x)  
 N = nrow(x)  
 ncol = ncol(x)  
 if (ncol > 1) {ccm = getFunction(colMeans)  
 }else {ccm = get0("mean", as.environment("package:base"))}  
   
 if (is.null(keepobs)){  
 keepobs = ceiling(N\*0.1)  
 if (keepobs < ncol) keepobs = ncol + 1  
 }  
   
 aa <- function(z, n, nobs){  
 x = z[1:n,]  
 y = z[(n+1):nobs,]  
 wk = (var(x)\*n + var(y)\*(nobs - n)) / (nobs - 2)  
 yk = sqrt(n \*(nobs - n) / nobs) \* (ccm(x) - ccm(y))  
 T2k = HotellingT2(yk, wk)  
 return(T2k)  
 }  
   
 ntest = c((keepobs+1):(N-keepobs))  
 T2 = sapply(ntest, aa, z = x, nobs = N)  
 maxk = which.max(T2)  
 return(c(T2Max = T2[maxk], Change\_Point = maxk +keepobs, T2 = T2))  
}  
  
## Repeat Max T^2 for N (SimNum) times and return as a Data Frame containing two columns (T2Max and Change\_Point (position))  
  
set.seed(2345)  
alpha = 0.05  
p = 1  
SimNum = 20  
n =8  
mu = rep(0, p)  
sigma = diag(x = 1, nrow = p, ncol = p)  
a = rdply(SimNum, T2ChangePoint (mvrnorm(n = n, mu = mu, Sigma = sigma), keepobs = p+1))  
b = a[,4:ncol(a)]  
print(a)

## .n T2Max Change\_Point T21 T22 T23  
## 1 1 1.8829980 5 1.976256e-01 1.18928439 1.882998043  
## 2 2 2.0967854 6 8.684432e-01 1.61662263 0.372369886  
## 3 3 0.2575637 4 3.487597e-02 0.25756367 0.021267580  
## 4 4 0.1594644 5 7.142481e-02 0.03850383 0.159464395  
## 5 5 14.5026479 3 1.450265e+01 3.46340996 3.202057352  
## 6 6 0.6875516 3 6.875516e-01 0.42903856 0.197372528  
## 7 7 7.8049864 4 1.551667e+00 7.80498636 5.333896699  
## 8 8 2.1744871 6 7.730171e-01 0.01824885 0.896953217  
## 9 9 0.1587509 3 1.587509e-01 0.11077765 0.042786125  
## 10 10 2.1665637 6 1.413523e-01 0.05266637 0.323061132  
## 11 11 4.1975502 3 4.197550e+00 0.88762688 0.001032129  
## 12 12 2.4023162 3 2.402316e+00 1.02088929 0.929425476  
## 13 13 0.4402615 3 4.402615e-01 0.34262963 0.308200510  
## 14 14 2.0852057 3 2.085206e+00 0.25522824 0.050946226  
## 15 15 1.0769353 5 1.082131e-03 0.01762906 1.076935329  
## 16 16 2.1001482 6 1.873204e-02 0.20743859 0.500004178  
## 17 17 1.1985305 5 6.563868e-02 0.00330262 1.198530481  
## 18 18 1.5399035 6 7.576673e-01 0.16630984 0.007474626  
## 19 19 0.9173667 5 1.180747e-01 0.03966890 0.917366748  
## 20 20 0.1913494 5 4.278521e-04 0.13630954 0.191349432  
## T24  
## 1 0.60607440  
## 2 2.09678543  
## 3 0.02298761  
## 4 0.01403302  
## 5 0.74381224  
## 6 0.60546422  
## 7 6.10836265  
## 8 2.17448712  
## 9 0.06863711  
## 10 2.16656367  
## 11 0.03845976  
## 12 0.50956755  
## 13 0.18243332  
## 14 0.22399582  
## 15 0.09089440  
## 16 2.10014819  
## 17 0.19930501  
## 18 1.53990350  
## 19 0.36210154  
## 20 0.04349802

quant = list(c(NULL, NULL))  
for (i in 1:ncol(b)){  
 quant[[i]] = c(as.integer(p+1+i), quantile(b[,i], probs = 0.95, names = FALSE, type = 1))  
 b = b[-which(unlist(b[,i]) >= quant[[i]][2]),]  
}  
bb = t(as.data.frame(quant))  
row.names(bb) = NULL  
print(bb)

## [,1] [,2]  
## [1,] 3 4.197550  
## [2,] 4 7.804986  
## [3,] 5 1.882998  
## [4,] 6 2.174487

## Change Point KMMD File

require(kernlab)

## Loading required package: kernlab

## Warning: package 'kernlab' was built under R version 3.3.1

require(plyr)  
require(MASS)  
x <- matrix(rnorm(100, 10, 5),50)  
y <- matrix(rnorm(200, 20, 5),100)  
z <- matrix(rnorm(100, 15, 10), ncol = ncol(x))  
  
colSums(sweep(x, 2,colMeans(x)))

## [1] 1.687539e-14 -1.065814e-14

############################################################  
  
HotellingT2 <- function(y, w){  
 y = matrix(y, nrow = 1)  
 w = as.matrix(w)  
 y %\*% solve(w) %\*% t(y)  
}  
  
  
  
cpkmmd <- function(x, keepobs = NULL){  
 x = as.matrix(x)  
 N = nrow(x)  
 if (is.null(keepobs)){  
 keepobs = ceiling(N\*0.1)  
 }  
   
 aa <- function(z, n, nobs){  
 x = z[1:n,]  
 y = z[(n+1):nobs,]  
 wk = (var(x)\*n + var(y)\*(nobs - n)) / (nobs - 2)  
 yk = sqrt(n \*(nobs - n) / nobs) \* (colMeans(x) - colMeans(y))  
 T2k = HotellingT2(yk, wk)  
 return(T2k)  
 }  
   
   
 ntest = c((keepobs+1):(N-keepobs))  
 stat = sapply(ntest, aa, z = x, nobs = N)  
 mk = which.max(stat)  
 return(c(T2Max = stat[mk], change\_Point = mk+keepobs))  
}  
  
a = rdply(20, cpkmmd(mvrnorm(n = 20, mu = c(11,15,32),   
 Sigma = matrix(c(1, .6, .7, .6, 1, .75, .7, .75,1),   
 nrow = 3, byrow = TRUE))))  
  
print(a)

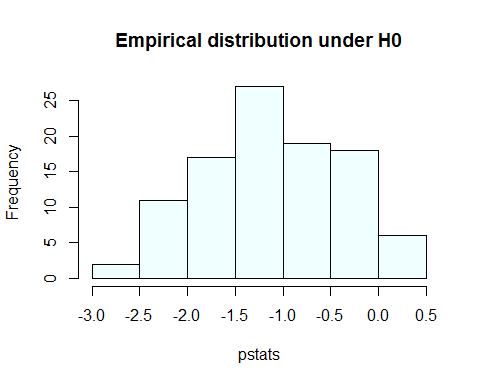
## .n T2Max change\_Point  
## 1 1 12.242666 16  
## 2 2 9.213460 14  
## 3 3 4.582759 18  
## 4 4 4.565112 14  
## 5 5 5.886297 12  
## 6 6 5.975853 4  
## 7 7 3.542667 4  
## 8 8 6.244345 6  
## 9 9 13.586066 16  
## 10 10 11.481422 3  
## 11 11 2.987013 3  
## 12 12 29.403018 3  
## 13 13 16.798784 7  
## 14 14 4.441895 18  
## 15 15 10.897145 17  
## 16 16 8.596549 18  
## 17 17 11.264867 3  
## 18 18 12.146210 17  
## 19 19 6.711628 17  
## 20 20 5.656718 8

# KMMD function   
# only return KMMD value  
  
kmmdfunconly <- function(Kxx,Kyy, Kxy)  
{  
   
 m <- dim(Kxx)[1]  
 n <- dim(Kyy)[1]  
   
 sumKxx <- sum(Kxx)  
   
 sumKyy <- sum(Kyy)  
   
 sumKxy <- sum(Kxy)  
   
 mmd1 <- sqrt(max(0,sumKxx/(m\*m) + sumKyy/(n\*n) - 2/m/n\* sumKxy))  
  
 return(list(mmd=mmd1))  
}  
  
  
############################################################  
  
# KMMD function   
# return KMMD with 1st and 3rd order   
  
kmmdfunc <- function(Kxx,Kyy, Kxy, alpha = 0.05)  
{  
   
 m <- dim(Kxx)[1]  
 n <- dim(Kyy)[1]  
   
 N <- max(m,n)  
 M <- min(m,n)  
   
 sumKxx <- sum(Kxx)  
   
 if(m!=n)  
 sumKxxM <- sum(Kxx[1:M,1:M])  
 else  
 sumKxxM <- sumKxx  
   
 dgxx <- diag(Kxx)  
   
 sumKxxnd <- sumKxx - sum(dgxx)  
 R <- max(dgxx)  
 RM <- max(dgxx[1:M])  
 hu <- colSums(Kxx[1:M,1:M]) - dgxx[1:M]  
   
 sumKyy <- sum(Kyy)  
 if(m!=n)  
 sumKyyM <- sum(Kyy[1:M,1:M])  
 else  
 sumKyyM <- sumKyy  
   
 dgyy <- diag(Kyy)  
   
 sumKyynd <- sum(Kyy) - sum(dgyy)  
 R <- max(R,dgyy)  
 RM <- max(RM,dgyy[1:M]) # RM instead of R in original  
 hu <- hu + colSums(Kyy[1:M,1:M]) - dgyy[1:M]  
   
 sumKxy <- sum(Kxy)  
 if (m!=n)  
 sumKxyM <- sum(Kxy[1:M,1:M])  
 else  
 sumKxyM <- sumKxy  
   
 dg <- diag(Kxy) # up to M only  
 hu <- hu - colSums(Kxy[1:M,1:M]) - colSums(t(Kxy[1:M,1:M])) + 2\*dg # one sided sum  
   
 mmd1 <- sqrt(max(0,sumKxx/(m\*m) + sumKyy/(n\*n) - 2/m/n\* sumKxy))  
 mmd3 <- sum(hu)/M/(M-1)  
 D1 <- 2\*sqrt(RM/M)+sqrt(log(1/alpha)\*4\*RM/M)  
   
 return(list(mmd1=mmd1,mmd3=mmd3,D1=D1))  
}  
  
  
## Find Change point location by calculating maximum KMMD for data  
  
cpkmmd <- function(x, keepobs = NULL){  
 x = as.matrix(x)  
 N = nrow(x)  
 if (is.null(keepobs)){  
 keepobs = ceiling(N\*0.1)  
 }  
 aa <- function(z, n, rbf = rbf){  
 nobs = nrow(z)  
 ncoll = ncol(z)  
 x = z[1:n,1:ncoll]  
 y = z[(n+1):nobs,1:ncoll]  
 stat <- kmmdfunconly(Kxx=kernelMatrix(rbf, x), Kyy =kernelMatrix(rbf, y), Kxy=kernelMatrix(rbf, x,y))  
 return(stat$mmd)  
 }  
   
 sig.opt <- sigest(x, scaled = FALSE)[2]  
 rbf <- rbfdot(sigma = sig.opt)  
   
 ntest = c((keepobs+1):(N-keepobs))  
 stat = sapply(ntest, aa, z = x, rbf = rbf)  
 mk = which.max(stat)  
 return(list(stat = stat, change.Point = mk+keepobs))  
}  
  
bb = cpkmmd(rbind(x,y))  
  
newobscpkmmd <- function(newobs, x, y){  
 ncoll = ncol(x)  
 newobs = matrix(newobs, ncol = ncoll)  
 x = as.matrix(x)  
 y = as.matrix(y)  
 nnew = nrow(newobs)  
 ns = c(1:nnew)  
 sig.opt <- sigest(rbind(x,y), scaled = FALSE)[2]  
 rbf <- rbfdot(sigma = sig.opt)  
   
 aa <- function(i, z, x, y, rbf = rbf){  
 a = z[i,]  
 s1 = kmmdfunconly(Kxx=kernelMatrix(rbf, rbind(x,a)), Kyy =kernelMatrix(rbf, y), Kxy=kernelMatrix(rbf, rbind(x,a),y))  
 s2 = kmmdfunconly(Kxx=kernelMatrix(rbf, x), Kyy =kernelMatrix(rbf, rbind(y,a)), Kxy=kernelMatrix(rbf, x,rbind(y,a)))  
 if (s1$mmd > s2$mmd) c = 1  
 else c = 2  
 return(c)  
 }  
   
 co = sapply(ns, aa, z = newobs, x = x, y = y, rbf = rbf)  
   
 return(co)  
   
   
}  
  
  
  
newobscpkmmd(z, x,y)

## [1] 2 1 2 1 2 1 2 1 1 1 2 2 1 2 1 2 1 1 2 2 2 2 1 1 1 1 2 2 1 2 2 2 2 2 2  
## [36] 1 1 2 2 1 1 2 1 2 2 1 2 1 1 2

## Functions\_kmmd\_sent file

#  
# Compute kernel MMD test  
f.kmmd<-function(k1,k2,k3){  
   
 kmmd2<-0  
 m<-nrow(k1)  
 for(i in 1:m){  
 t<-0  
 for(j in 1:m){  
 if( i != j){  
   
 t <- t + k1[i,j]+k2[i,j]-k3[i,j]-k3[j,i]  
   
 }# end if  
   
 } #end loop j  
   
 kmmd2<-kmmd2+t/(m\*(m-1))  
   
 } # end loop i  
   
 return(kmmd2)  
   
}  
  
  
#Kernel function  
  
kernel11 <- function(x1, y = NULL){  
 if (!is.null(y)){ x1 = x1-y}  
 if(all(x1 == 0)){return(0)}  
 K<- sum(x1 / base::norm(as.matrix(x1),type = 'f'))  
 return(K)  
}  
  
# Permutation test  
PermTest.knl1 <- function(x, y, R=499, testfun=f.kmmd) {  
 z <- rbind(x, y) # pooled sample  
 # Create kernel matrix  
 sig.opt <- sigest(z, scaled = FALSE)[2]  
 #rbf <- rbfdot(sigma = sig.opt)  
 rbf <- getFunction("kernel11")  
   
 myfun <- function(a, b,c) suppressWarnings(unname(testfun(a, b,c)))  
 #set.seed(123)   
 DoIt <- function() {  
 i <- sample(nrow(z), nrow(x))  
 myfun( a=kernelMatrix(rbf, z[i,]), b =kernelMatrix(rbf, z[-i,]), c=kernelMatrix(rbf, z[i,],z[-i,]))  
 }  
 pstats <- replicate(R, DoIt())  
 stat <- myfun(a=kernelMatrix(rbf, x), b =kernelMatrix(rbf, y), c=kernelMatrix(rbf, x,y))  
 hist(pstats, col="azure", main="Empirical distribution under H0")  
 abline(v=stat, col="red", lty=2)  
 p.v <-mean(c(stat, pstats) >= stat)  
 res <- list(p.v=p.v, sig.opt=sig.opt, R=R, stat=stat, pstats=pstats, sum=sum(c(stat, pstats) >= stat))  
 return(res)  
}  
  
  
  
  
x <- matrix(rnorm(100, 10, 5),20)  
y <- matrix(rnorm(100, 20, 5),20)  
  
  
PermTest.knl1(x, y, R=100)



## $p.v  
## [1] 0.00990099  
##   
## $sig.opt  
## 50%   
## 0.001912114   
##   
## $R  
## [1] 100  
##   
## $stat  
## [1] 3.522715  
##   
## $pstats  
## [1] -0.573244605 -1.544318055 -2.858193959 -1.175930731 -1.240145971  
## [6] -1.593102055 -0.467843331 0.051493695 -0.955575516 -0.409234591  
## [11] -2.429147879 -0.313282578 -2.083194558 -1.296442973 -0.767396209  
## [16] -1.735550264 -2.490883554 -1.234734515 -0.991722215 -0.237391620  
## [21] -1.335606365 -1.152808618 -0.370974485 -1.348666213 -1.961935255  
## [26] -0.840291821 0.065800997 -2.046440293 -0.754747608 -2.291439852  
## [31] -1.376248446 -0.499649547 -0.547827900 -1.336667246 -0.684022673  
## [36] -2.206328016 -1.113366184 -1.310434025 -0.587861193 -0.169762582  
## [41] -1.085419829 -0.034644588 -1.052660067 -0.803948860 0.357001366  
## [46] -0.929056438 -0.088465891 -0.510154527 -0.671429605 -2.017312873  
## [51] -0.931204346 -1.123687588 -0.298417606 -0.412319236 -1.446975174  
## [56] -1.534812295 -1.687899543 -1.326247138 -0.721851354 -0.114043001  
## [61] -0.268721828 -0.568919190 -1.094618045 -1.313777926 -0.723052391  
## [66] -1.949709774 -1.844523372 -1.043960010 -1.874126441 0.151099774  
## [71] -1.177840883 -1.043986457 -0.126754266 -1.424391797 -0.435615483  
## [76] -1.403777453 0.006327668 -1.569261905 -0.566617363 -2.094851301  
## [81] -0.361778722 -2.072303798 -1.714773126 -1.481449196 -2.552269372  
## [86] -1.848687159 -1.961517241 -0.465407998 -1.185238776 -2.421271535  
## [91] -1.420836524 -1.518451899 -0.619272939 -1.989355498 -1.507808399  
## [96] -1.945341781 -1.484631733 0.006947217 -2.322984383 -0.489397585  
##   
## $sum  
## [1] 1

## kmmd\_nutrimouse\_sent file

#############################################################################  
# Kernel Maximum Mean Discrepancy   
# "INFERRING DIFFERENTIALLY EXPRESSED PATHWAYS BY USING KERNEL MAXIMUM MEAN DISCREPANCY-BASED TEST  
# Esteban Vegas, Ferran Reverter and Josep Maria Oller  
# Statistical Department, University of Barcelona  
#############################################################################  
  
  
#############################################################################  
# Nutrimouse data set  
#############################################################################  
  
  
if(!require(mixOmics, quietly =T)) install.packages("mixOmics")

## Warning: package 'mixOmics' was built under R version 3.3.3

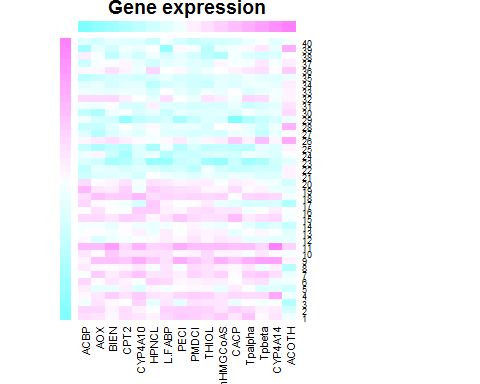
## Warning: package 'ggplot2' was built under R version 3.3.1

##   
## Attaching package: 'ggplot2'

## The following object is masked from 'package:kernlab':  
##   
## alpha

##   
## Loaded mixOmics 6.1.2  
##   
## Visit http://www.mixOmics.org for more details about our methods.  
## Any bug reports or comments? Notify us at mixomics at math.univ-toulouse.fr or https://bitbucket.org/klecao/package-mixomics/issues  
##   
## Thank you for using mixOmics!

require(mixOmics, quietly =T)  
  
if(!require(methods, quietly =T)) install.packages("mixOmics")  
require(mixOmics, quietly =T)  
  
  
data(nutrimouse)  
#help(nutrimouse)  
#str(nutrimouse)  
  
mydata.gene <- nutrimouse$gene # gene expressions data set  
mydata.FA <- nutrimouse$lipid # fatty acids data set  
  
gene.names <- names(mydata.gene)  
FA.names <- names(mydata.FA)  
  
ind.diet <- nutrimouse$diet  
ind.genotype <- nutrimouse$genotype  
  
# There are 2 factors: genotype with 2-levels factor and diet 5-levels factor.  
  
# Change from genotype factor to genotype list   
ind.gen <- list()  
for (i in levels(ind.genotype)) {  
 ind.gen[[i]] <- which(ind.genotype==i)  
}  
  
# Change from diet factor to diet list   
ind.diet1 <- list()  
for (i in levels(ind.diet)) {  
 ind.diet1[[i]] <- which(ind.diet==i)  
}  
  
#############################################################################  
# Subset of genes and fatty acids involved in fatty acids catabolism pathway  
#############################################################################  
  
  
# 1) Selection of genes involved in fatty acids(FA) catabolism pathway  
  
gene.names.sel.1 <- c("ACBP","AOX","BIEN","CPT2","CYP4A10",  
 "HPNCL","L.FABP","PECI","PMDCI", "THIOL",  
 "mHMGCoAS", "CACP","Tpalpha", "Tpbeta", "CYP4A14","ACOTH")  
  
pos.genes <- match(x=c(gene.names.sel.1), table=gene.names)  
  
  
# 2) Selection of fatty acids involved in fatty acids catabolism pathway  
pos.FA <- match(x=FA.names[19:21], table=FA.names)   
  
  
  
# 3) Subset of genes or fatty acids involved in fatty acids catabolism pathway  
mydata.gene.sel <- mydata.gene[,pos.genes]  
mydata.FA.sel <-mydata.FA[,pos.FA]  
  
  
  
#############################################################################  
# Assign the x variable to the sample values of the first condition.  
# Assign the y variable to the sample values of the second condition.  
# Only Gene expresions data set  
#############################################################################  
  
x <- as.matrix(mydata.gene.sel[ind.gen$wt,])  
y <- as.matrix(mydata.gene.sel[ind.gen$ppar,])  
  
  
  
#############################################################################  
# Heatmaps  
#   
#############################################################################  
  
# my heatmap function  
  
myheatmap <-function(z,my.xlab=NULL, my.ylab=NULL, my.main){  
 rc <- cm.colors(nrow(z))  
 cc <- cm.colors(ncol(z))  
 hv <- heatmap(z, col = cm.colors(256), scale = "column",  
 Rowv=NA, Colv=NA,  
 RowSideColors = rc,  
 ColSideColors = cc,   
 margins = c(5,10),  
 xlab = my.xlab, ylab = my.ylab,  
 main = my.main)  
}  
  
  
myheatmap(rbind(x,y), my.main="Gene expression")



#############################################################################  
# Hotelling test  
#   
#############################################################################  
  
if(!require("Hotelling", quietly =T)) install.packages("Hotelling")

## Warning: package 'Hotelling' was built under R version 3.3.3

## Warning: package 'corpcor' was built under R version 3.3.3

require("Hotelling", quietly =T)  
  
(hotelling.test(x, y))

## $stats  
## $stats$statistic  
## [1] 1210.364  
##   
## $stats$m  
## [1] 0.03782895  
##   
## $stats$df  
## [1] 16 23  
##   
## $stats$nx  
## [1] 20  
##   
## $stats$ny  
## [1] 20  
##   
## $stats$p  
## [1] 16  
##   
##   
## $pval  
## [1] 1.1946e-13  
##   
## attr(,"class")  
## [1] "hotelling.test"

source("http://bioconductor.org/biocLite.R")

## Bioconductor version 3.4 (BiocInstaller 1.24.0), ?biocLite for help

## A new version of Bioconductor is available after installing the most  
## recent version of R; see http://bioconductor.org/install

#biocLite("GSAR", suppressUpdates=TRUE)  
library("GSAR")

## Warning: package 'GSAR' was built under R version 3.3.1

## Loading required package: igraph

##   
## Attaching package: 'igraph'

## The following objects are masked from 'package:stats':  
##   
## decompose, spectrum

## The following object is masked from 'package:base':  
##   
## union

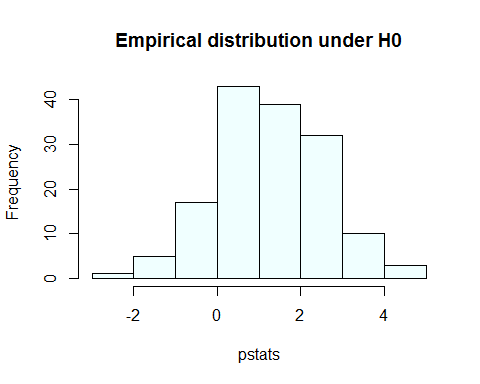
result <- KStest(object=cbind(t(x),t(y)), group=c(rep(1,20),rep(2,20)))   
result

## [1] 0.000999001

result <- WWtest(object=cbind(t(x),t(y)), group=c(rep(1,20),rep(2,20)))   
result

## [1] 0.000999001

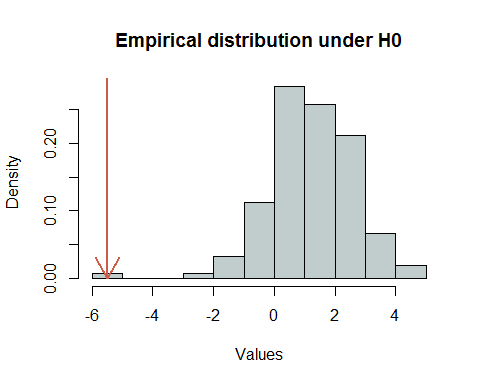
#############################################################################  
# Kernel Maximum Mean Discrepancy: asymptotic distribution  
#   
#############################################################################  
  
if(!require("kernlab", quietly =T)) install.packages("kernlab")  
require("kernlab", quietly =T)  
  
source("D:/UCF/STA 6908 - Edgard Maboudou/STA 6908 - Edgard Maboudou/R code/Functions\_kmmd\_sent.R")  
  
time1 <- proc.time()  
#results.kmmd.1 <- by (Tr\_C\_Y.filt\_1[sel\_row,],ffactor, kmmd.my1)  
results.nl1 <- PermTest.knl1(x,y,R=150)



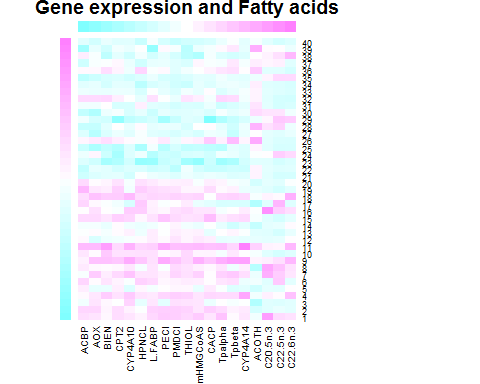
proc.time() - time1

## user system elapsed   
## 9.22 0.01 9.50

# Empirical distribution of kernel MDD of fatty acids catabolism pathway under H0  
  
hist(c(results.nl1[["pstats"]],results.nl1[["stat"]]), col="azure3",freq=FALSE,   
 main= "Empirical distribution under H0", xlab="Values")  
  
x0=results.nl1[["stat"]]  
  
arrows(x0=x0, y0=1.5, x1=x0, y1= 0, col="coral3", lty=1, lwd=2)  
text(x=results.nl1[["stat"]], y=1.5, pos=3, labels="stat")



#############################################################################  
# Assign the x variable to the sample values of the first condition.  
# Assign the y variable to the sample values of the second condition.  
# Gene expresions data set and Fatty acid data set  
#############################################################################  
  
  
# Gene expresions data set (1)  
x1 <- as.matrix(mydata.gene.sel[ind.gen$wt,])  
y1 <- as.matrix(mydata.gene.sel[ind.gen$ppar,])  
  
# Fatty acids data set (2)  
x2 <- as.matrix(mydata.FA.sel[ind.gen$wt,])  
y2 <- as.matrix(mydata.FA.sel[ind.gen$ppar,])  
  
# data set joined by condition  
x <- cbind(x1,x2)  
y <- cbind(y1,y2)  
  
#############################################################################  
# New results  
#   
#############################################################################  
  
  
# heatmap  
myheatmap(rbind(x,y), my.main="Gene expression and Fatty acids")



# Hotelling test  
  
(hotelling.test(x, y))

## $stats  
## $stats$statistic  
## [1] 1231.028  
##   
## $stats$m  
## [1] 0.02770083  
##   
## $stats$df  
## [1] 19 20  
##   
## $stats$nx  
## [1] 20  
##   
## $stats$ny  
## [1] 20  
##   
## $stats$p  
## [1] 19  
##   
##   
## $pval  
## [1] 2.936407e-11  
##   
## attr(,"class")  
## [1] "hotelling.test"

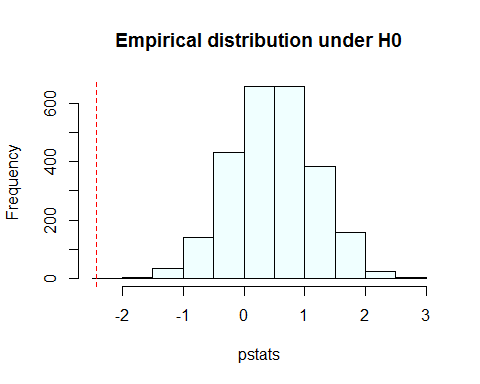
result <- KStest(object=cbind(t(x),t(y)), group=c(rep(1,20),rep(2,20)))   
result

## [1] 0.001998002

result <- WWtest(object=cbind(t(x),t(y)), group=c(rep(1,20),rep(2,20)))   
result

## [1] 0.000999001

# Kernel Maximum Mean Discrepancy: asymptotic distribution  
  
time1 <- proc.time()  
results.nl1 <- PermTest.knl1(x,y,R=2499)



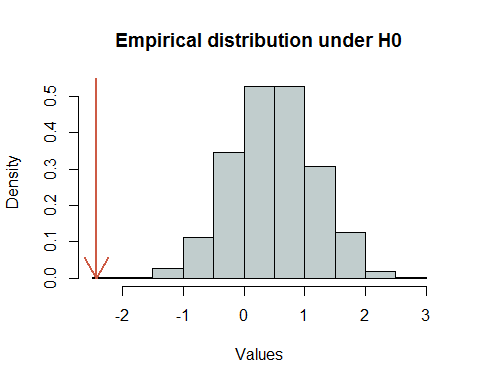
proc.time() - time1

## user system elapsed   
## 152.57 0.35 156.84

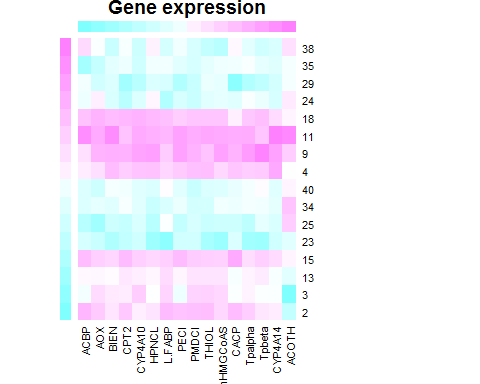
results.nl1$p.v #p-value

## [1] 1

# Empirical distribution of kernel MDD of fatty acids catabolism pathway under H0  
  
hist(c(results.nl1[["pstats"]],results.nl1[["stat"]]), col="azure3",freq=FALSE,   
 main= "Empirical distribution under H0", xlab="Values")  
  
x0=results.nl1[["stat"]]  
  
arrows(x0=x0, y0=1.5, x1=x0, y1= 0, col="coral3", lty=1, lwd=2)  
text(x=results.nl1[["stat"]], y=1.5, pos=3, labels="stat")



#############################################################################  
# Fatty acids catabolism pathway: sun vs fish diet  
#  
#############################################################################  
  
  
#############################################################################  
# Assign the x variable to the sample values of the first condition.  
# Assign the y variable to the sample values of the second condition.  
# Only Gene expresions data set  
#############################################################################  
  
x <- as.matrix(mydata.gene.sel[ind.diet1$sun,])   
y <- as.matrix(mydata.gene.sel[ind.diet1$fish,])  
  
  
  
#############################################################################  
# New results  
#   
#############################################################################  
  
  
# heatmap  
myheatmap(rbind(x,y), my.main="Gene expression")



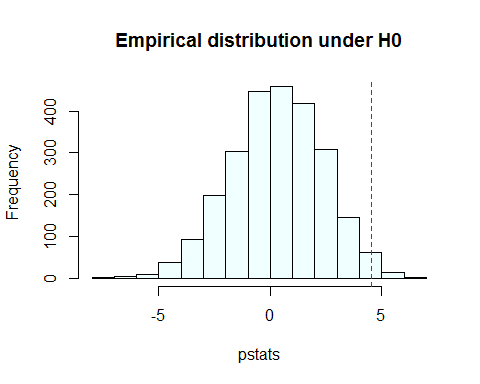
# Hotelling test  
  
#(hotelling.test(x, y)) # error number of samples < number of variables  
  
  
  
result <- KStest(object=cbind(t(x),t(y)), group=c(rep(1,8),rep(2,8)))   
result

## [1] 0.08591409

result <- WWtest(object=cbind(t(x),t(y)), group=c(rep(1,8),rep(2,8)))   
result

## [1] 0.3906094

# Kernel Maximum Mean Discrepancy: asymptotic distribution  
  
time1 <- proc.time()  
results.nl1 <- PermTest.knl1(x,y,R=2499)



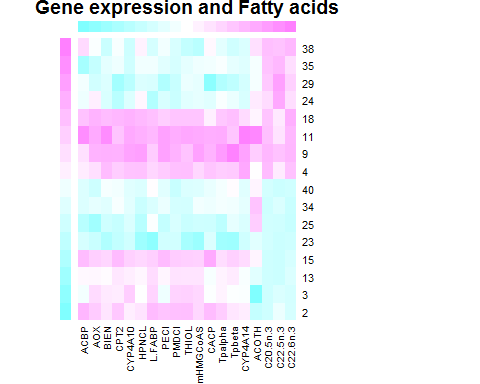
proc.time() - time1

## user system elapsed   
## 43.59 0.05 44.81

results.nl1$p.v #p-value

## [1] 0.0124

#############################################################################  
# Assign the x variable to the sample values of the first condition.  
# Assign the y variable to the sample values of the second condition.  
# Gene expresions data set and Fatty acid data set  
#############################################################################  
  
# Gene expresions data set (1)  
x1 <- as.matrix(mydata.gene.sel[ind.diet1$sun,])   
y1 <- as.matrix(mydata.gene.sel[ind.diet1$fish,])  
  
# Fatty acids data set (2)  
x2 <- as.matrix(mydata.FA.sel[ind.diet1$sun,])   
y2 <- as.matrix(mydata.FA.sel[ind.diet1$fish,])  
  
# data set joined by condition  
x <- cbind(x1,x2)  
y <- cbind(y1,y2)  
  
  
  
# heatmap  
myheatmap(rbind(x,y), my.main="Gene expression and Fatty acids")



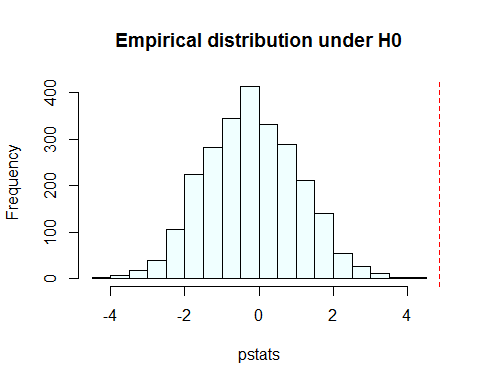
# Hotelling test  
  
#(hotelling.test(x, y)) # error number of samples < number of variables  
  
  
result <- KStest(object=cbind(t(x),t(y)), group=c(rep(1,8),rep(2,8)))   
result

## [1] 0.000999001

result <- WWtest(object=cbind(t(x),t(y)), group=c(rep(1,8),rep(2,8)))   
result

## [1] 0.000999001

# Kernel Maximum Mean Discrepancy: asymptotic distribution  
  
time1 <- proc.time()  
results.nl1 <- PermTest.knl1(x,y,R=2499)



proc.time() - time1

## user system elapsed   
## 43.91 0.11 45.00

results.nl1$p.v #p-value

## [1] 4e-04

## KMMD File

library(kernlab)  
datapath = 'D:/UCF/STA 6908 - Edgard Maboudou/STA 6908 - Edgard Maboudou/Data/'  
df = read.table(paste(datapath, 'pb2.txt', sep = ''))  
x = df[df$V1 == 1, 2:5]  
y = df[df$V1 == 2, 2:5]  
  
  
kernel11 <- function(x1, y = NULL){  
 if (!is.null(y)){ x1 = x1-y}  
 if(all(x1 == 0)){return(0)}  
 K<- sum(x1 / base::norm(as.matrix(x1),type = 'f'))  
 return(K)  
}  
  
kcalculator <- function(X, Y = NULL){  
 if (is.null(Y)){  
 N<-dim(X)[1]  
 K<-matrix(0,N,N)  
 for(i in 1:N){  
 K[i,]<-apply(sweep(X, 2, unlist(X[i,])), 1, kernel11)  
 }  
 return(K)  
 }else{  
 N1<-dim(X)[1]  
 N2 = dim(Y)[1]  
 K<-matrix(0,N1,N2)  
 for(i in 1:N2){  
 K[,i]<-apply(sweep(X, 2, unlist(Y[i,])), 1, kernel11)  
 }  
 return(K)  
   
 }  
}  
  
x <- matrix(runif(300),100)  
y <- matrix(runif(300)+10,100)  
a = kmmd(x, y)

## Using automatic sigma estimation (sigest) for RBF or laplace kernel

aa = kernelMatrix(kernel11, as.matrix(x))  
  
  
k = kcalculator(x, y)  
  
library(kernlab)  
kmmd(x, y)

## Using automatic sigma estimation (sigest) for RBF or laplace kernel

## Kernel Maximum Mean Discrepancy object of class "kmmd"   
##   
## Gaussian Radial Basis kernel function.   
## Hyperparameter : sigma = 1.03664381559699   
##   
##   
## H0 Hypothesis rejected : TRUE  
## Rademacher bound : 0.546163676520489  
##   
## 1st and 3rd order MMD Statistics : 1.12666890628934 1.2620028529285

## KMMD File

## calculates the kernel maximum mean discrepancy for samples from two distributions  
## author: alexandros karatzoglou  
  
setGeneric("kmmd",function(x,...) standardGeneric("kmmd"))

## [1] "kmmd"

setMethod("kmmd", signature(x = "matrix"),  
 function(x, y, kernel="rbfdot",kpar="automatic", alpha = 0.05, asymptotic = FALSE, replace = TRUE, ntimes = 150, frac = 1, ...)  
 {  
 x <- as.matrix(x)  
 y <- as.matrix(y)  
   
 res <- new("kmmd")  
   
   
 if(is.character(kernel)){  
 kernel <- match.arg(kernel,c("rbfdot","polydot","tanhdot","vanilladot","laplacedot","besseldot","anovadot","splinedot","matrix"))  
   
 if(kernel == "matrix")  
 if(dim(x)[1]==dim(x)[2])  
 return(kmmd(x= as.kernelMatrix(x), y = y, Kxy = as.kernelMatrix(x)%\*%y, alpha = 0.05, asymptotic = FALSE, replace = TRUE, ntimes = 100, frac = 1, ...))  
 else  
 stop(" kernel matrix not square!")  
   
 if(is.character(kpar))  
 if((kernel == "tanhdot" || kernel == "vanilladot" || kernel == "polydot"|| kernel == "besseldot" || kernel== "anovadot"|| kernel=="splinedot") && kpar=="automatic" )  
 {  
 cat (" Setting default kernel parameters ","\n")  
 kpar <- list()  
 }  
 }  
   
 if (!is.function(kernel))  
 if (!is.list(kpar)&&is.character(kpar)&&(kernel == "laplacedot"|| kernel=="rbfdot")){  
 kp <- match.arg(kpar,"automatic")  
 if(kp=="automatic")  
 kpar <- list(sigma=sigest(rbind(x,y),scaled=FALSE)[2])  
 cat("Using automatic sigma estimation (sigest) for RBF or laplace kernel","\n")  
   
 }  
 if(!is(kernel,"kernel"))  
 {  
 if(is(kernel,"function")) kernel <- deparse(substitute(kernel))  
 kernel <- do.call(kernel, kpar)  
 }  
   
 if(!is(kernel,"kernel")) stop("kernel must inherit from class `kernel'")  
   
 m <- dim(x)[1]  
 n <- dim(y)[1]  
   
 N <- max(m,n)  
 M <- min(m,n)  
   
 Kxx <- kernelMatrix(kernel,x)  
 Kyy <- kernelMatrix(kernel,y)  
 Kxy <- kernelMatrix(kernel,x,y)  
   
 resmmd <- .submmd(Kxx, Kyy, Kxy, alpha)   
   
 H0(res) <- (resmmd$mmd1 > resmmd$D1)   
 Radbound(res) <- resmmd$D1  
 Asymbound(res) <- 0  
 mmdstats(res)[1] <- resmmd$mmd1  
 mmdstats(res)[2] <- resmmd$mmd3  
   
 if(asymptotic){  
 boundA <- .submmd3bound(Kxx, Kyy, Kxy, alpha, frac, ntimes, replace)  
   
 AsympH0(res) <- (resmmd$mmd3 > boundA)   
 Asymbound(res) <- boundA  
 }  
   
 kernelf(res) <- kernel  
 return(res)  
 })

## [1] "kmmd"

setMethod("kmmd",signature(x="list"),  
 function(x, y, kernel="stringdot",kpar=list(type="spectrum",length=4), alpha = 0.05, asymptotic = FALSE, replace = TRUE, ntimes = 150, frac = 1, ...)  
 {  
   
 if(!is(kernel,"kernel"))  
 {  
 if(is(kernel,"function")) kernel <- deparse(substitute(kernel))  
 kernel <- do.call(kernel, kpar)  
 }  
 if(!is(kernel,"kernel")) stop("kernel must inherit from class `kernel'")  
   
 Kxx <- kernelMatrix(kernel,x)  
 Kyy <- kernelMatrix(kernel,y)  
 Kxy <- kernelMatrix(kernel,x,y)  
   
 ret <- kmmd(x=Kxx,y = Kyy,Kxy=Kxy, alpha=alpha, asymptotic= asymptotic, replace = replace, ntimes = ntimes, frac= frac)  
   
 kernelf(ret) <- kernel  
   
 return(ret)  
   
 })

## [1] "kmmd"

setMethod("kmmd",signature(x="kernelMatrix"), function (x, y, Kxy, alpha = 0.05, asymptotic = FALSE, replace = TRUE, ntimes = 100, frac = 1, ...)  
{  
 res <- new("kmmd")  
 resmmd <- .submmd(x, y, Kxy, alpha)   
 H0(res) <- (resmmd$mmd1 > resmmd$D1)   
 Radbound(res) <- resmmd$D1  
 Asymbound(res) <- 0  
 mmdstats(res)[1] <- resmmd$mmd1  
 mmdstats(res)[2] <- resmmd$mmd3  
   
 if(asymptotic){  
 boundA <- .submmd3bound(x, y, Kxy, alpha, frac, ntimes, replace)  
   
 AsympH0(res) <- (resmmd$mmd1 > boundA)   
 Asymbound(res) <- boundA  
 }  
 kernelf(res) <- " Kernel matrix used as input."  
 return(res)  
   
})

## [1] "kmmd"

.submmd <- function(Kxx,Kyy, Kxy, alpha)  
{  
   
 m <- dim(Kxx)[1]  
 n <- dim(Kyy)[1]  
   
 N <- max(m,n)  
 M <- min(m,n)  
   
 sumKxx <- sum(Kxx)  
   
 if(m!=n)  
 sumKxxM <- sum(Kxx[1:M,1:M])  
 else  
 sumKxxM <- sumKxx  
   
 dgxx <- diag(Kxx)  
   
 sumKxxnd <- sumKxx - sum(dgxx)  
 R <- max(dgxx)  
 RM <- max(dgxx[1:M])  
 hu <- colSums(Kxx[1:M,1:M]) - dgxx[1:M]  
   
 sumKyy <- sum(Kyy)  
 if(m!=n)  
 sumKyyM <- sum(Kyy[1:M,1:M])  
 else  
 sumKyyM <- sumKyy  
   
 dgyy <- diag(Kyy)  
   
 sumKyynd <- sum(Kyy) - sum(dgyy)  
 R <- max(R,dgyy)  
 RM <- max(RM,dgyy[1:M]) # RM instead of R in original  
 hu <- hu + colSums(Kyy[1:M,1:M]) - dgyy[1:M]  
   
 sumKxy <- sum(Kxy)  
 if (m!=n)  
 sumKxyM <- sum(Kxy[1:M,1:M])  
 else  
 sumKxyM <- sumKxy  
   
 dg <- diag(Kxy) # up to M only  
 hu <- hu - colSums(Kxy[1:M,1:M]) - colSums(t(Kxy[1:M,1:M])) + 2\*dg # one sided sum  
   
 mmd1 <- sqrt(max(0,sumKxx/(m\*m) + sumKyy/(n\*n) - 2/m/n\* sumKxy))  
 mmd3 <- sum(hu)/M/(M-1)  
 D1 <- 2\*sqrt(RM/M)+sqrt(log(1/alpha)\*4\*RM/M)  
   
 return(list(mmd1=mmd1,mmd3=mmd3,D1=D1))  
}  
  
  
.submmd3bound <- function(Kxx,Kyy, Kxy, alpha, frac, ntimes, replace)  
{  
 ## implements the bootstrapping approach to the MMD3 bound by shuffling  
 ## the kernel matrix  
 ## frac : fraction of data used for bootstrap  
 ## ntimes : how many times MMD is to be evaluated  
   
 m <- dim(Kxx)[1]  
 n <- dim(Kyy)[1]  
   
 M <- min(m,n)  
 N <- max(m,n)  
   
 poslabels <- 1:m  
 neglabels <- (m+1):(m+n)  
   
 ## bootstrap  
 bootmmd3 <- rep(0,ntimes)  
   
 for (i in 1:ntimes)  
 {  
 nsamples <- ceiling(frac\*min(m,n))  
 xinds <- sample(1:m,nsamples,replace=replace)  
 yinds <- sample(1:n,nsamples,replace=replace)  
 newlab <- c(poslabels[xinds],neglabels[yinds])  
 samplenew <- sample(newlab, length(newlab), replace=FALSE)  
 xinds <- samplenew[1:nsamples]  
 yinds <- samplenew[(nsamples+1):length(samplenew)]  
   
 newm <- length(xinds)  
 newn <- length(yinds)  
 newM <- min(newm,newn)  
   
 ##get new kernel matrices (without concat to big matrix to save memory)  
 xind1 <- xinds[xinds<=m]  
 xind2 <- xinds[xinds>m]- m  
 yind1 <- yinds[yinds<=m]  
 yind2 <- yinds[yinds>m]-m  
   
 ##Kxx (this should be implemented with kernelMult for memory efficiency)  
 nKxx <- rbind(cbind(Kxx[xind1,xind1],Kxy[xind1,xind2]), cbind(t(Kxy[xind1,xind2]),Kyy[xind2,xind2]))  
 dgxx <- diag(nKxx)  
 hu <- colSums(nKxx[1:newM,1:newM]) - dgxx[1:newM] # one sided sum  
 rm(nKxx)  
   
 #Kyy  
 nKyy <- rbind(cbind(Kxx[yind1,yind1],Kxy[yind1,yind2]), cbind(t(Kxy[yind1,yind2]), Kyy[yind2,yind2]))  
 dgyy <- diag(nKyy)  
 hu <- hu + colSums(nKyy[1:newM,1:newM]) - dgyy[1:newM]  
 rm(nKyy)  
   
 ## Kxy  
 nKxy <- rbind(cbind(Kxx[yind1,xind1],Kxy[yind1,xind2]), cbind(t(Kxy[xind1,yind2]),Kyy[yind2,xind2]))  
 dg <- diag(nKxy)  
 hu <- hu - colSums(nKxy[1:newM,1:newM]) - colSums(t(nKxy[1:newM,1:newM])) + 2\*dg  
 rm(nKxy)  
   
 ## now calculate mmd3  
 bootmmd3[i] <- sum(hu)/newM/(newM-1)  
 }  
   
   
 bootmmd3 <- sort(bootmmd3, decreasing=TRUE);  
 aind <- floor(alpha\*ntimes) ## better less than too much (-> floor);  
   
 ## take threshold in between aind and the next smaller value:  
 bound <- sum(bootmmd3[c(aind,aind+1)])/2;  
 return(bound)  
   
}  
  
  
setMethod("show","kmmd",  
 function(object){  
   
 cat("Kernel Maximum Mean Discrepancy object of class \"kmmd\"","\n","\n")  
   
 show(kernelf(object))  
   
 if(is.logical(object@H0)){  
 cat("\n")  
 cat("\n","H0 Hypothesis rejected : ", paste(H0(object)))  
 cat("\n","Rademacher bound : ", paste(Radbound(object)))  
 }  
   
 cat("\n")   
   
 if(Asymbound(object)!=0){  
 cat("\n","H0 Hypothesis rejected (based on Asymptotic bound): ", paste(AsympH0(object)))  
 cat("\n","Asymptotic bound : ", paste(Asymbound(object)))  
 }  
   
 cat("\n","1st and 3rd order MMD Statistics : ", paste( mmdstats(object)))  
 cat("\n")  
 })

## [1] "show"