01a. DQF Playground

2023-02-17

```
dqf.outlier <- function(data = NULL, gram.mat = NULL, g.scale=2, angle=c(30,45,60), kernel="linear", p1=
  # kernelized version of depthity
  #
  # inputs:
  #   data (matrix or data frame) - a data matrix of explanatory variables
  #   kernel - of form "linear", "rbf" or "poly", or a user defined function
  #   g.scale (scalar) - scales the base distribution G
  #   angle (numeric vector of length 3)- angles of cone from midline, must live between 0 and 90
  #   p1 - first parameter for kernel
  #   p2 - second parameter for kernel
  #   n.splits (integer) - the number of split points at which the DQF is computed
  #   subsample (integer)- the number of random pairs for each observation
  #   z.scale (logical) - should the data be z-scaled first
  #   k.w (integer) - the number of points altered in the windsorized standard deviation
  #   adaptive - if TRUE, uses windsorized standard deviation to scale base distribution
  #   G - base distribution:   "norm" or "unif"
  ##
  # Output:
  #   angle  - vector of angles used, same as inputted
  #   dqf1, dqf2, dqf3 - matrices of depth quantile functions, rows are observations
  if (G=="norm") {
    param1 <- 0; param2 <- 1 }
  if (G=="unif") {
    param1 <- -1; param2 <- 1 }
  if (is.null(data) & is.null(gram.mat))
    stop("Either a data set or Gram matrix must be provided")
  if (min(angle) <= 0 | max(angle) >= 90)
    stop("Angles must be between 0 and 90")
  if (is.null(data))
    n.obs <- nrow(gram.mat)
  if (is.null(gram.mat))
    n.obs <- nrow(data)
  scram <- sample(n.obs)
  pairs <- subsamp.dqf(n.obs, subsample)
  if (is.null(gram.mat)) {
    if (z.scale==TRUE)
      data <- apply(data, 2, scale) #z-scale data
    if (is.function(kernel)==TRUE)
      kern <- kernel
    if (kernel == "linear") {
      kern <- function(x,y)
        return(sum(x*y))
    }
    if (kernel == "rbf") {
      kern <- function(x,y)
```

```
      return(exp(-sum((x-y)^2)/p1))
  }
  if (kernel == "poly") {
    kern <- function(x,y)
      return((sum(x*y)+p2)^p1)
  }
  data <- data[scram,]
  gram <- matrix(0,n.obs, n.obs)
  for (i in 1:n.obs) {
    for (j in i:n.obs) {
      gram[i,j] <- kern(data[i,], data[j,])
      gram[j,i] <- gram[i,j]
    }
  }
} else {
  if (diff(dim(gram.mat))!=0)
    stop("Gram matrix must be square")
  if (isSymmetric(gram.mat) == FALSE)
    stop("Gram matrix must be symmetric")
  gram <- gram.mat
  gram <- gram[scram,]
  gram <- gram[,scram]
}
splits <- get(paste("q", G, sep=""))((1:n.splits)/(n.splits+1),param1, param2) * g.scale
depthity1 <- depthity2 <- depthity3 <- rep(0,length(splits))
norm.k2 <- error.k <- k.to.mid <- rep(0, n.obs)
dep1 <- dep2 <- dep3 <- matrix(0, nrow=nrow(pairs), ncol=n.splits)
qfs1 <- qfs2 <- qfs3 <- matrix(0, nrow=nrow(pairs), ncol=100)
for (i.subs in 1:nrow(pairs)) {
  i <- pairs[i.subs,1];  j <- pairs[i.subs,2]
  for (k in 1:n.obs) {
    norm.k2[k] <- gram[k,k] + 1/4*(gram[i,i]+gram[j,j]) + 1/2*gram[i,j]-gram[k,i]-gram[k,j]
    k.to.mid[k] <- (gram[k,i]-gram[k,j]+1/2*(gram[j,j]-gram[i,i]))/sqrt(gram[i,i]+gram[j,j]-2*gram[i,
    error.k[k] <- sqrt(abs(norm.k2[k] - k.to.mid[k]^2))
  }
  for (c in 1:length(splits)) {
    good <- rep(1, n.obs)
    s <- splits[c] * (sd.w(k.to.mid, k.w)*adaptive + (adaptive==FALSE))
    good[k.to.mid/s > 1] <- 0  #points on other side of cone tip removed
    d.to.tip <- abs(k.to.mid - s)
    good1 <- good * (abs(atan(error.k / d.to.tip)) < (angle[1]/360*2*pi))  #points outside of cone re
    good1 <- good1 * (1 - 2*(sign(k.to.mid)==sign(s)))  #which side of midpoint are they on
    depthity1[c] <- min(c(sum(good1==-1), sum(good1==1)))
    good2 <- good * (abs(atan(error.k / d.to.tip)) < (angle[2]/360*2*pi))  #points outside of cone re
    good2 <- good2 * (1 - 2*(sign(k.to.mid)==sign(s)))  #which side of midpoint are they on
    depthity2[c] <- min(c(sum(good2==-1), sum(good2==1)))
    good3 <- good * (abs(atan(error.k / d.to.tip)) < (angle[3]/360*2*pi))  #points outside of cone re
    good3 <- good3 * (1 - 2*(sign(k.to.mid)==sign(s)))  #which side of midpoint are they on
    depthity3[c] <- min(c(sum(good3==-1), sum(good3==1)))
  }
  qfs1[i.subs,] <- quantile(depthity1, seq(0,1,length=100), na.rm=TRUE)
  qfs2[i.subs,] <- quantile(depthity2, seq(0,1,length=100), na.rm=TRUE)
  qfs3[i.subs,] <- quantile(depthity3, seq(0,1,length=100), na.rm=TRUE)
```

```
  }
  dqf1 <- dqf2 <- dqf3 <- matrix(0,n.obs, 100)
  for (i in 1:n.obs) {
    dqf1[i,] <- apply(qfs1[which(pairs[,1]==i | pairs[,2]==i),],2,mean, na.rm=TRUE)
    dqf2[i,] <- apply(qfs2[which(pairs[,1]==i | pairs[,2]==i),],2,mean, na.rm=TRUE)
    dqf3[i,] <- apply(qfs3[which(pairs[,1]==i | pairs[,2]==i),],2,mean, na.rm=TRUE)
  }
  dqf1 <- dqf1[order(scram),]  #map back to original indicies
  dqf2 <- dqf2[order(scram),]
  dqf3 <- dqf3[order(scram),]

  return(list(angle=angle, dqf1=dqf1, dqf2=dqf2, dqf3=dqf3))
}
```