Homework 5

Stat 428

Homework submission guideline: For the convenience of grading, you are required to submit the **Rmd** and **PDF** files separately and NOT zipped, as a zip file cannot be previewed in Canvas. You may get a penalty if wrong format is submitted.

Before the homework, please set seed as your UIN for the sake of reproducibility.

```r
set.seed(677978239)
```

1. Cross Validation (40 points)

We're going to look at a data set on 97 men who have prostate cancer (from the book The Elements of Statistical Learning). There are 10 variables measured on these 97 men:

1. `lpsa`: log PSA score
2. `lcavol`: log cancer volume
3. `lweight`: log prostate cancer weight
4. `age`: age of patient
5. `lbph`: log of the amount of benign prostatic hyperplasia
6. `svi`: seminal vesicle invasion
7. `lcp`: log of capsular penetration
8. `gleason`: Gleason score
9. `pgg45`: percent of Gleason scores 4 or 5
10. `train`: if belonging to training data set

To load this prostate cancer data set and store it as a matrix `pros.data`, we can do as following:

```r
pros.data =
  as.matrix(read.table("https://web.stanford.edu/~hastie/ElemStatLearn/datasets/prostate.data"))
```

We only use the first 9 columns

```r
pros.data = pros.data[,1:9]
```

In this question, we are going to predict `lpsa` by a linear model of other variables. In particular, we consider the following three models:

$$X_{lpsa} = \beta_0 + \beta_1 X_{lcavol} + \beta_2 X_{lweight} + \beta_3 X_{age} + \epsilon$$

$$X_{lpsa} = \beta_0 + \beta_1 X_{lcavol} + \beta_2 X_{lweight} + \epsilon$$

$$X_{lpsa} = \beta_0 + \beta_1 X_{lcavol} + \epsilon$$

- **1a.** Randomly split the prostate cancer data frame into $k = 5$ folds of roughly equal size. (Make sure your code is general enough to handle an arbitrary number of folds k; you will be asked to change the number of folds in questions that follow.) Report the number of observations that fall in each fold.

```
summary(pros.data)
```

```
##      lcavol           lweight           age             lbph
##  Min.   :-1.3471   Min.   :2.375   Min.   :41.00   Min.   :-1.3863
##  1st Qu.: 0.5128   1st Qu.:3.376   1st Qu.:60.00   1st Qu.:-1.3863
##  Median : 1.4469   Median :3.623   Median :65.00   Median : 0.3001
##  Mean   : 1.3500   Mean   :3.629   Mean   :63.87   Mean   : 0.1004
##  3rd Qu.: 2.1270   3rd Qu.:3.876   3rd Qu.:68.00   3rd Qu.: 1.5581
##  Max.   : 3.8210   Max.   :4.780   Max.   :79.00   Max.   : 2.3263
##       svi              lcp             gleason          pgg45
##  Min.   :0.0000   Min.   :-1.3863   Min.   :6.000   Min.   :  0.00
##  1st Qu.:0.0000   1st Qu.:-1.3863   1st Qu.:6.000   1st Qu.:  0.00
##  Median :0.0000   Median :-0.7985   Median :7.000   Median : 15.00
##  Mean   :0.2165   Mean   :-0.1794   Mean   :6.753   Mean   : 24.38
##  3rd Qu.:0.0000   3rd Qu.: 1.1787   3rd Qu.:7.000   3rd Qu.: 40.00
##  Max.   :1.0000   Max.   : 2.9042   Max.   :9.000   Max.   :100.00
##       lpsa
##  Min.   :-0.4308
##  1st Qu.: 1.7317
##  Median : 2.5915
##  Mean   : 2.4784
##  3rd Qu.: 3.0564
##  Max.   : 5.5829
```

```
nfolds = 5
folds <- split(sample(nrow(pros.data), nrow(pros.data), replace = FALSE), as.factor(1:nfolds))
datalist <- lapply(folds, function(x) pros.data[x, ])
i = 0
n = nrow(pros.data)


for(data in datalist) {
  print(nrow(data))
}
```

```
## [1] 20
## [1] 20
## [1] 19
## [1] 19
## [1] 19
```

As we can see, the length of the 5 folds are roughly equal.

- **1b.** Over the folds you computed in the previous question, compute the cross-validation error of the above three linear models.

```r
e1 <- e2 <- e3 <- numeric(nfolds)
for(i in 1:nfolds){
  A = datalist[[i]]

  B = data.frame()
  for (j in 1:nfolds) {
    if (i != j) {
      B = rbind(B, datalist[[j]])
    }
  }
  A.lpsa <- A[,"lpsa"]
  A.lcavol = A[,"lcavol"]
  A.lweight = A[,"lweight"]
  A.age = A[,"age"]

  A = as.data.frame(A)
  B = as.data.frame(B)

  L1 <- lm(lpsa ~ lcavol+lweight+age, data=B);
  hatAlpsa=L1$coef[1]+L1$coef[2]*A.lcavol + L1$coef[3]*A.lweight + L1$coef[4]*A.age
  e1[i]=sum((hatAlpsa-A.lpsa)^2)

  #2
  L2 <- lm(lpsa ~ lcavol+lweight, data=B);
  hatAlpsa=L2$coef[1]+L2$coef[2]*A.lcavol + L2$coef[3]*A.lweight
  e2[i]=sum((hatAlpsa-A.lpsa)^2)

  #3
  L3 <- lm(lpsa ~ lcavol, data=B);
  hatAlpsa=L3$coef[1]+L3$coef[2]*A.lcavol
  e3[i]=sum((hatAlpsa-A.lpsa)^2)
}
c(sum(e1)/n,sum(e2)/n,sum(e3)/n)
```

```
## [1] 0.5676379 0.5622533 0.6224067
```

- **1c.** Write a function `pros.cv()`, which takes three arguments: `df`, a data frame of prostate cancer measurements, with a default of `pros.df`; `k`, an integer determining the number of cross-validation folds, with a default of 5; and `seed`, an integer to be passed to `set.seed()` before defining the folds, with a default of NULL (meaning no seed shall be set). Your function should split up the given data `df` into `k` folds of roughly equal size, and using these folds, compute the cross-validation error of the above three linear model. Its output should simply be a vector of cross-validation errors. When `k` is equal to the row of the data set, leave-one-out cross validation should be used.

```r
pros.cv <- function(df, k=5, seed=NULL) {
  if (!is.null(seed)) {
    set.seed(seed)
  }

  nfolds = k
  folds <- split(sample(nrow(df), nrow(df), replace = FALSE), as.factor(1:nfolds))
  datalist <- lapply(folds, function(x) df[x, ])
  i = 0
  e1 <- e2 <- e3 <- numeric(nfolds)
  n = nrow(pros.data)
  for(i in 1:nfolds){
    A = data.frame(matrix(nrow=1, ncol = ncol(pros.data)))
    colnames(A) = colnames(pros.data)
    A = rbind(A, datalist[[i]])
    B = data.frame(matrix(nrow=1, ncol = ncol(A)))
    colnames(B) = colnames(A)

    for (j in 1:nfolds) {
      if (i != j) {
        B = rbind(B, datalist[[j]])
      }
    }
    A = na.omit(A)
    B = na.omit(B)


    A.lpsa <- A$lpsa
    A.lcavol = A$lcavol
    A.lweight = A$lweight
    A.age = A$age


    L1 <- lm(lpsa ~ lcavol+lweight+age, data=B);

    hatAlpsa=L1$coef[1]+L1$coef[2]*A.lcavol + L1$coef[3]*A.lweight + L1$coef[4]*A.age
    e1[i]=sum((hatAlpsa-A.lpsa)^2)

    #2
    L2 <- lm(lpsa ~ lcavol+lweight, data=B);

    hatAlpsa=L2$coef[1]+L2$coef[2]*A.lcavol + L2$coef[3]*A.lweight

    e2[i]=sum((hatAlpsa-A.lpsa)^2)

    #3

    L3 <- lm(lpsa ~ lcavol, data=B);

    hatAlpsa=L3$coef[1]+L3$coef[2]*A.lcavol

    e3[i]=sum((hatAlpsa-A.lpsa)^2)

    #Use the test and train data partitions however you desire...
  }
  return(c(sum(e1)/n,sum(e2)/n,sum(e3)/n))
}
pros.cv(pros.data,5)
```

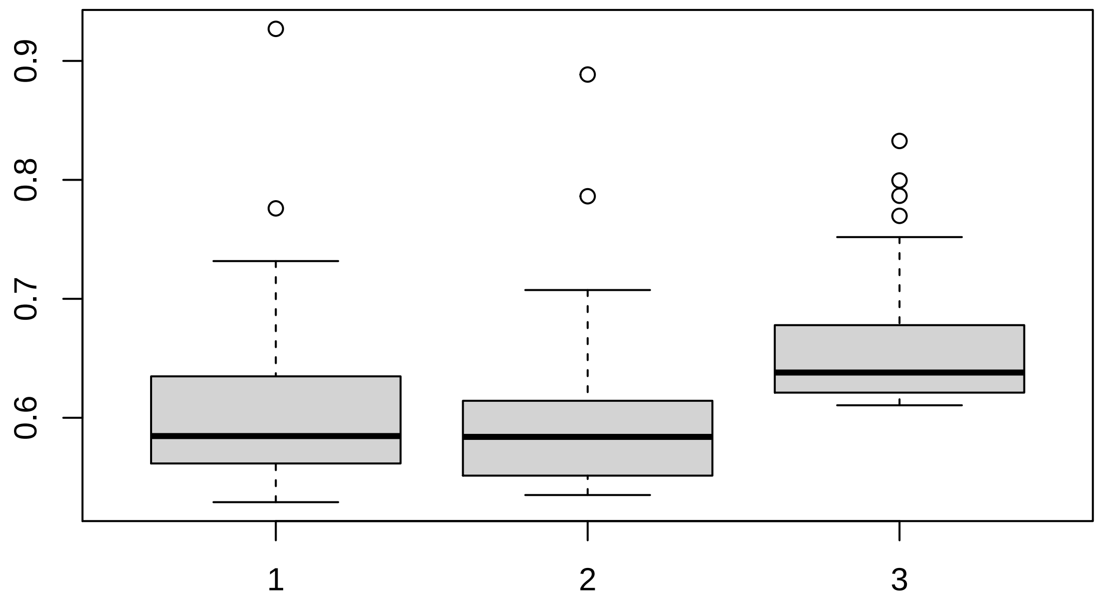
```
## [1] 0.5873896 0.5868482 0.6583445
```

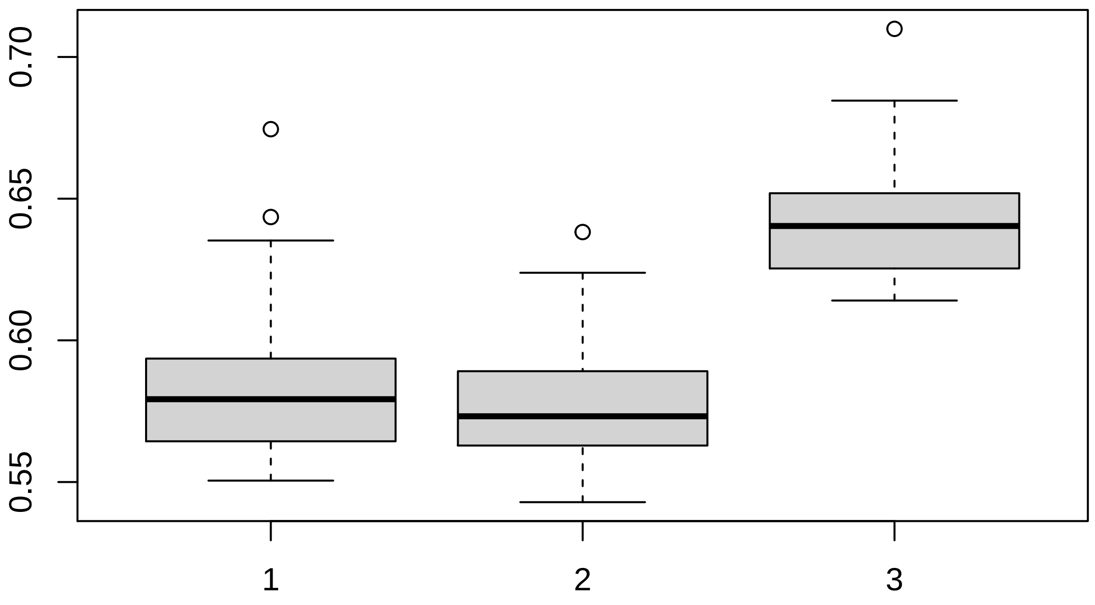
- **1d.** Investigate the result of `pros.cv()` for different values of `k`, specifically, for `k` equal to 2, 4, 8, 16 and 97. For each value, run `pros.cv()` some large number of times (say, 50) and report the average of the cross-validation error estimates, and the standard deviation of these estimates. Then, plot them in an informative way (say, a box plot with `boxplot()`). What do you notice? Is this surprising?

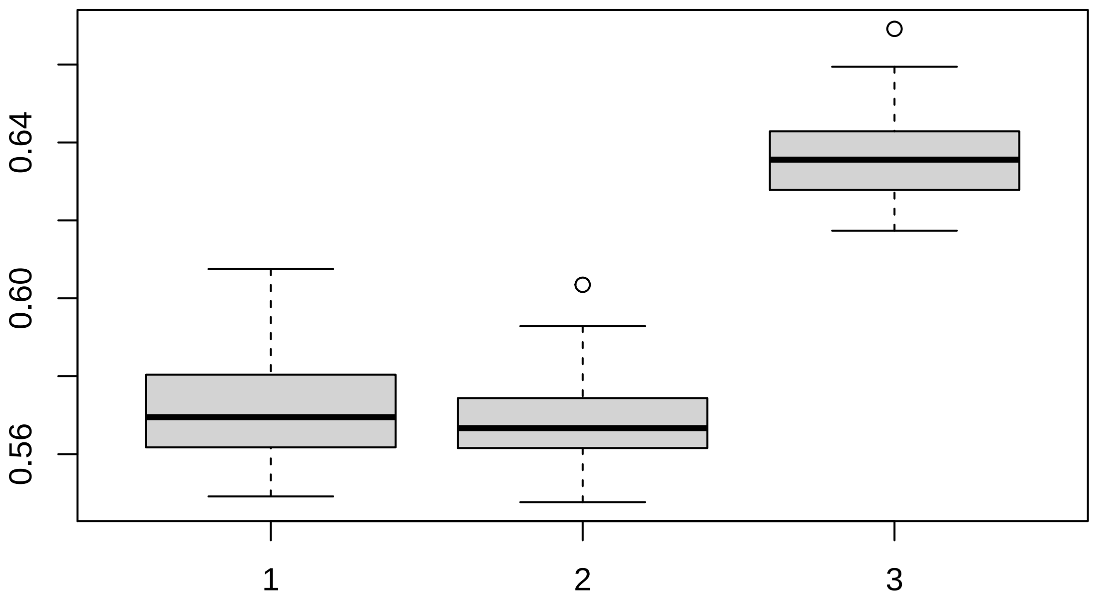
```
lst = c(2,4,8,16,97)

average = list()
std = list()
val = list()


print(nrow(pros.data))
```

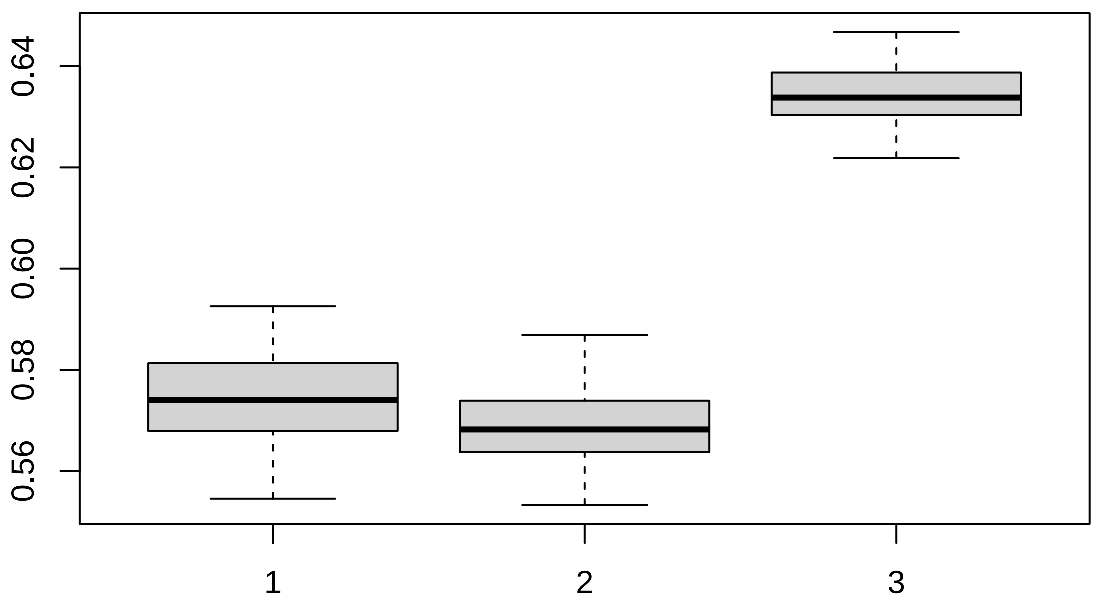
```
## [1] 97
```

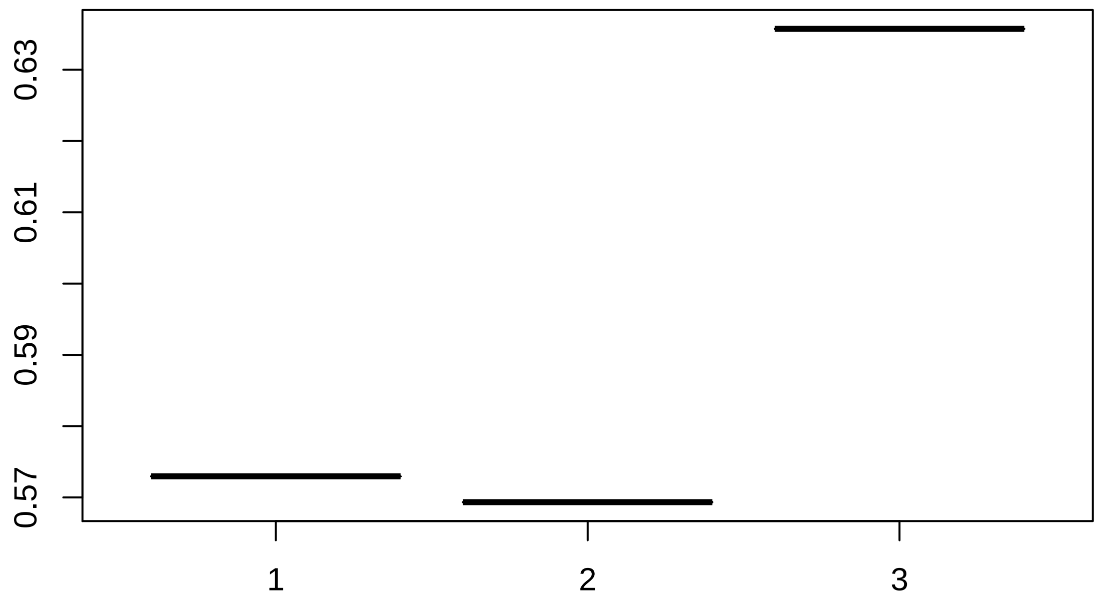
```
index = 1
for(k in lst) {
  first = rep(0, 50)
  second = rep(0, 50)
  third = rep(0, 50)

  for(i in 1:50) {
    temp = pros.cv(pros.data, k)
    first[i] = temp[1]
    second[i] = temp[2]
    third[i] = temp[3]
  }
  boxplot(first,second, third)
  val[index] <- c((first), (second), (third))
  average[index] <- c(mean(first), mean(second), mean(third))
  std[index] <- c(sd(first), sd(second), sd(third))
  index = index + 1
}
```


0.6
0.7
0.8
0.9
1
2
3


0.55
0.60
0.65
0.70
1
2
3


0.56
0.60
0.64
1
2
3


0.64
0.62
0.60
0.58
0.56
1
2
3



```r
print("Average:")
```

```
## [1] "Average:"
```

```r
print(unlist(average))
```

```
## [1] 0.6053231 0.5841892 0.5719804 0.5740478 0.5729621
```

```r
print("STD")
```

```
## [1] "STD"
```

```r
print(unlist(std))
```

```
## [1] 7.116361e-02 2.637741e-02 1.327123e-02 8.620154e-03 1.154649e-16
```

As the k gets bigger, the std get smaller while the average have no significant changes. It does not suprise to me

2. Permutation Tests (60 points)

- **2a.** The nearest neighbor test can also be applied to one dimensional data. Can you design a simulation experiment to compare Type I and Type II error of Kolmogorov-Smirnov test and nearest neighbor test on a one-dimensional data set?

Here is the ks.perm function that I got from the lecture.

```r
ks.perm <- function(X,Y,alpha,B) {
   D=ks.test(X,Y)$statistic
   Z=c(X,Y); N=1:length(Z)
   DPerm=rep(0,B)
   for (b in 1:B) {
      selectedindex <- sample(N, size = length(X), replace = FALSE)
      X1 <- Z[selectedindex]
      Y1 <- Z[-selectedindex]
      DPerm[b]=ks.test(X1,Y1)$statistic
   }
   P <- mean(c(D, DPerm) >= D)
   return(P<alpha)
}
```

Check the Type I error:

```r
n = 100; m=100
times = 100
decision <- numeric(times)
for (t in 1:times) {
   X=rnorm(n)
   Y=rnorm(m)
   decision[t]=ks.perm(X,Y,alpha=0.05,B=200)
}
mean(decision)
```

```
## [1] 0.07
```

Check the Type II error:

```r
n = 100; m=100
times = 100
decision <- numeric(times)
for (t in 1:times) {
   X=rnorm(n)
   Y=rnorm(m, mean = 0.2)
   decision[t]=ks.perm(X,Y,alpha=0.05,B=200)
}
mean(1-decision)
```

```
## [1] 0.79
```

Here is the nearest neighbor test function that I got from the lecture.

```r
library(RANN)
NNT <- function(z, ix, sizes, R) {
  n1 <- sizes[1]
  n2 <- sizes[2]
  n <- n1 + n2
  z <- z[ix, ]
  NN <- nn2(z, z, k=R+1)
  block1 <- NN$nn.idx[1:n1, -1]
  block2 <- NN$nn.idx[(n1+1):n, -1]
  i1 <- sum(block1 < n1 + .5)
  i2 <- sum(block2 > n1 + .5)
  return((i1 + i2) / (R * n))
}

NNT.perm <- function(z, sizes, R,alpha,B) {
  T0=NNT(z,1:nrow(z),sizes, R)
  TPerm=rep(0,B)
  for (b in 1:B) {
  permindex=sample(1:nrow(z))
  TPerm[b]=NNT(z,permindex,sizes, R)
  }
  P <- mean(c(T0, TPerm) >= T0)
  return(P<alpha)
}
```

Check the Type I error:

```
n = 100; m=100; d=1
times = 100
decision <- numeric(times)
for (t in 1:times) {
  x <- matrix(rnorm(n*d), n, d)
  y <- matrix(rnorm(m*d), m, d)
  z <- rbind(x, y)
  decision[t]=NNT.perm(z, c(n,m), R=3,alpha=0.05,B=500)
}
mean(decision)
```

```
## [1] 0.09
```

Check the Type II error:

```
n = 100; m=100; d=1
times = 100
decision <- numeric(times)
for (t in 1:times) {
  x <- matrix(rnorm(n*d), n, d)
  y <- matrix(rnorm(m*d, mean = 0.4), m, d)
  z <- rbind(x, y)
  decision[t]=NNT.perm(z, c(n,m), R=3,alpha=0.05,B=200)
}
mean(1-decision)
```

```
## [1] 0.83
```

- **2b.** The choice of distance is important in distance correlation. Besides Euclidean distance, we can also consider L_p distance

$$\|X - Y\|_{\ell_p} = \left(\sum_{i=1}^{d} |X_i - Y_i|^p\right)^{1/p}.$$

Can you design a simulation experiment to assess the Type I and Type II error of distance correlation test for $p = 1, 2, 4, 8$.

```
plst = c(1,2,4,8)
```

Define the needed functions:

```
library(MASS)
Akl <- function(x, pow) {
  d <- as.matrix(dist(x,method="minkowski",p=pow))
  m <- rowMeans(d)
  M <- mean(d)
  a <- sweep(d, 1, m)
  b <- sweep(a, 2, m)
  return(b + M)
}
DCOR <- function(x, y, pow) {
  x <- as.matrix(x)
  y <- as.matrix(y)
  n <- nrow(x)
  m <- nrow(y)
  if (n != m || n < 2) stop("Sample sizes must agree")
  if (! (all(is.finite(c(x, y)))))
  stop("Data contains missing or infinite values")
  A <- Akl(x, pow)
  B <- Akl(y, pow)
  dCov <- sqrt(mean(A * B))
  dVarX <- sqrt(mean(A * A))
  dVarY <- sqrt(mean(B * B))
  dCor <- sqrt(dCov / sqrt(dVarX * dVarY))
  list(dCov=dCov, dCor=dCor, dVarX=dVarX, dVarY=dVarY)
}
DCOR.perm <- function(x,y,alpha,B, pow) {
  distx <- as.matrix(dist(x,method="minkowski",p=pow))
  disty <- as.matrix(dist(y,method="minkowski",p=pow))
  T0=DCOR(distx, disty, pow)$dCor
  TPerm=rep(0,B)
  for (b in 1:B) {
    permindex=sample(1:nrow(x))
    permdisty=disty[permindex,permindex]
    TPerm[b]=DCOR(distx, permdisty, pow)$dCor
  }
  P <- mean(c(T0, TPerm) >= T0)
  return(P<alpha)
}
```

We can do the Distance correlation test by using the above function, and print the result.

```
z <- as.matrix(iris[1:5, 1:4])
x <- z[ , 1:2]
y <- z[ , 3:4]
for(p in plst) {
  print(c("when p is", p))
  # Get Type I
  n = 100; mu=rep(0,6); sigma=diag(6)
  times = 20
  decision <- numeric(times)
  for (t in 1:times) {
    data <- mvrnorm(n, mu, sigma)
    x <- data[,1:3]
    y <- data[,4:6]
    decision[t]=DCOR.perm(x, y,alpha=0.05,B=100, p)
  }

  print(c("Type I errror is:", mean(decision)))


  # Get Type II
  rho=0.1
  n = 100; mu=rep(0,6); sigma=diag(6)*(1-rho)+rho
  times = 20
  decision <- numeric(times)
  for (t in 1:times) {
  data <- mvrnorm(n, mu, sigma)
  x <- data[,1:3]
  y <- data[,4:6]
  decision[t]=DCOR.perm(x, y,alpha=0.05,B=100, p)
  }
  print(c("Type II errror is:", mean(1-decision)))
  print("-------------")
}
```

```
## [1] "when p is" "1"
## [1] "Type I errror is:" "0"
## [1] "Type II errror is:" "0.75"
## [1] "-------------"
## [1] "when p is" "2"
## [1] "Type I errror is:" "0.05"
## [1] "Type II errror is:" "0.5"
## [1] "-------------"
## [1] "when p is" "4"
## [1] "Type I errror is:" "0.05"
## [1] "Type II errror is:" "0.6"
## [1] "-------------"
## [1] "when p is" "8"
## [1] "Type I errror is:" "0.05"
## [1] "Type II errror is:" "0.75"
## [1] "-------------"
```

- **2c.** The Kendall tau's correlation can also be used to test the independence between variables when they are both one-dimensional variable. Kendall tau's correlation is defined as

$$\frac{1}{n(n-1)}\sum_{i\neq j}\mathrm{sgn}(x_i-x_j)\,\mathrm{sgn}(y_i-y_j).$$

  You can use the implementation in `cor` by setting `method="kendall"`. Can you implement an permutation-based independence test for Kendall tau's correlation? Specifically, write a function kendalltau.perm(x,y,alpha,B), where x and y are input data, alpha is significance level and B is number of replicate in permutation test. kendalltau.perm(x,y,alpha,B) returns the decision on whether the null hypothesis is rejected.

```r
kendalltau.perm <- function(x,y,alpha,B) {
  T0=cor.test(x, y,method="kendall")$statistic
  TPerm=rep(0,B)
  for (b in 1:B) {
    permx=sample(1:length(y))
    permy=y[permx]
    TPerm[b]=cor.test(x, permy,method="kendall")$statistic
  }
  P <- mean(c(T0, TPerm) >= T0)
  return(P<alpha)
}
x = rnorm(100)
y = rnorm(100)
kendalltau.perm(x, y, 0.05, 100)
```

```
## [1] FALSE
```

- **2d.** Both distance correlation test and Kendall tau's correlation test can be used to test the independence between one-dimensional variables. Can you design a simulation experiment to compare Type I and Type II error of distance correlation test and Kendall tau's correlation test? Yes, I can.

```r
n = 100; m=100
times = 60
decision.DCOR <- numeric(times)
decision.KEN <- numeric(times)
for (t in 1:times) {
   X=as.matrix(rnorm(n))
   Y=as.matrix(rnorm(m))
   decision.DCOR[t]=DCOR.perm(X,Y,alpha=0.05,B=100, 2)
   decision.KEN[t]=kendalltau.perm(X, Y,alpha=0.05,B=100)
}
print(c("Type I errror of DCOR is:", mean(decision.DCOR)))
```

```
## [1] "Type I errror of DCOR is:" "0.0833333333333333"
```

```r
print(c("Type I errror of kendalltau is:", mean(decision.KEN)))
```

```
## [1] "Type I errror of kendalltau is:" "0.0833333333333333"
```

```r
# Get Type II

n = 100; m=100
times = 100
decision.DCOR2 <- numeric(times)
decision.KEN2 <- numeric(times)
for (t in 1:times) {
   X=as.matrix(rnorm(n))
   Y=as.matrix(rnorm(m))
   decision.DCOR2[t]=DCOR.perm(X,Y,alpha=0.05,B=100, 2)
   decision.KEN2[t]=kendalltau.perm(X,Y,alpha=0.05,B=100)

}
print(c("Type II errror of DCOR is:", mean(1-decision.DCOR2)))
```

```
## [1] "Type II errror of DCOR is:" "0.96"
```

```
print(c("Type II errror of kendalltau is:", mean(1-decision.KEN2)))
```

```
## [1] "Type II errror of kendalltau is:" "0.97"
```