Assignment 4

Exercise 1

Task 1

Basis of step functionss

```r
age <- ISLR::Wage$age
```

```r
#'
#' @param `x` "vector" a continuous variable
#' @param `K` "constant"
region_matrix <- function(x,K) {

  clevels <- levels(cut(x,K))
  binmat <- cbind(
    a = as.numeric(sub("\\((.+),.*","\\1",clevels)),
    b = as.numeric(sub("[^,]*,([^]]*)\\]","\\1",clevels))
  )
  return(as.data.frame(binmat))
}
```

```r
cut_2 <- function(x,K) {
  n <- length(x)
  K <- K+1

  r <- region_matrix(x,K)
  m <- matrix(0,n,K)

  for(j in 1:K) {
    a <- r$a[j]
    b <- r$b[j]

    m[,j] <- x < b & x >= a
  }

  return(
    list(
      c_matrix = as.matrix(m),
      c_regions = r
    )
  )
}

#'
#' @param `x` vector of obs
#' @param `K` number of steps
```

```r
stepfun <- function(x,K) {

  X <- cut_2(x,K)$c_matrix
  y <- x

  # linear solution
  weights <- solve(t(X)%*%X)%*%t(X)%*%y

  return (
    # matrix of step functions
    structure(
      list(
        weights = t(weights),
        step_matrix = as.matrix(X)
      )
    )
  )
}
```

```r
stepfun(age,5)$weights
```

```
##          [,1]    [,2]     [,3]     [,4]  [,5]     [,6]
## [1,] 24.41795 33.6635 43.43983 53.30525 62.64 73.31429
```

Task 2

Hierarchical clustering

```r
X=mtcars[1:6,]
# plot.new()
# plot(hclust(dist(X)))
```

```r
#' Calculate the pairwise dissimilarity matrix
#' @return `data.frame` object containing pairwise dissimilarity matrix
pd <- function(X)
{
  N <- nrow(X)

  # negative row indices for building merge table
  rownames(X) <- 1:N*-1

  # Distance matrix with
  # upper and lower
  x <- as.data.frame(as.matrix(dist(X)))

  # Set diag to inf
  diag(x) <- Inf
  return(
    x
  )
}

#' Perform hierarchical clustering algorithm
#' @param Matrix X of n obs in p variables
#' @return object class 'hclust', including n-1 list of cluster indices
h_clust <- function(X)
{
  # pairwise dissimilarity matrix
  # Euclidian distance
  x <- pd(X)

  N <- nrow(x)
  N_1 <- N-1

  # Initialise objects
  merge <- matrix(0,N_1,2)
  height <- vector(length=N_1)
  clusters <- lapply(1:N_1,function(x) x)

  # for(m in 1:N_1) {
  for(m in 1:2) { # only want first iteration

    # Indices
    ind <- colnames(x)

    # find pair of clusters of least dissimilar
    cl <- which(x==min(x),arr.ind=TRUE)[1,,drop=FALSE]

    # record the pair of clusters to merge
    merge[m,] <- as.numeric(ind[cl])

    # merge the pair
    l <- apply(x[cl,],2,max)
    x[min(cl),] <- l
    x[,min(cl)] <- l

    #
    x[cl] <- Inf
    x[max(cl),] <- Inf
    x[,max(cl)] <- Inf

    # List of indices in cluster
    cluster <- c(
      cl,
      which(ind %in% ind[cl[1,ind[cl[1]] > 0]])
    )

    # record cluster indices
    colnames(x)[cluster] <- m

    clusters[[m]] <- unique(cluster)
    # clusters[[m]] <- unique(clusters[[m]])

    # height of dendogram at fusion point
    height[m] <- min(x)

  }
  return(
    structure(
      list(
        merge=merge,
        height=height,
        clusters=clusters
      ),
      class='hclust'
    )
  )
}
```

```
# List of cluster indices
h_clust(X)$cluster
```

```
## [[1]]
## [1] 2 1
##
## [[2]]
## [1] 6 4
##
## [[3]]
## [1] 3
##
## [[4]]
## [1] 4
##
## [[5]]
## [1] 5
```

Task 3

Linear aggregation of M classifiers

Bagging?

```
M <- 1000
w <- runif(M)
response <- rep(0,M)
response[w > .7] <- 1

input <- w
```

credit score and default

leading economic indicator and interest rate change

barometer and rainfall event

Task 4

```
Smarket <- ISLR::Smarket
x <- Smarket$Today
Y <- Smarket$Direction
```

```r
#' Prior probabilites
#' @param Y vector in qualitvative response variable
#' @return length K vector of prior probabilities
.pi_k <- function(Y)
{
  # calc priors
  priors <- as.numeric(prop.table(table(Y)))
  names(priors) <- levels(Y)

  return(
    structure(
    .Data=priors
    )
  )
}

#' sample averages
#' @return length K vector of sample averages
.mk <- function(x,Y)
{
  mu <- as.numeric(tapply(x,Y,mean))
  names(mu) <- levels(Y)
  return(
    structure(
      .Data=mu
    )
  )
}

#'
#' @return A n*K covariance matrix
.Sk <- function(x,Y)
{
  S <- as.numeric(tapply(x,Y,var))
  names(S) <- levels(Y)
  return(
    structure(
      .Data=S
    )
  )
}

#' Object containing the QDA k parameters
#' @return list of parameters `mu_k`, `Sigma_k`, and prior probabilities
pars <- function(x,Y)
{
  structure (
    list(
      prior=.pi_k(Y),
      mean=.mk(x,Y),
      variance=.Sk(x,Y)
    )
  )
}

#' Quadratice discriminant function
#' @param x input var length n
#' @param pars list of length K vectors containing estimated prior probabilities, sample averages, and
#' @return object containing `nK` matrix of posterior probabilities, and length `n` vector of predicted
quadratic_df <- function(x,pars)
{
  V <- t(as.data.frame(pars))
  K <- dim(V)[2]
  n <- length(x)
  d <- as.data.frame(matrix(0,n,K))

  p <- V[1,] #prior
  m <- V[2,] #mean
  S <- V[3,] #variance

  # the magic
  pk <- function(k)
  {
    t1 <- -(x^2/(2*S[k]))
    t2 <- (x*(m[k]/S[k]))
    t3 <- -(m[k]^2/(2*S[k]))
    t4 <- log(p[k])
    t5 <- -log(S[k])

    return(exp(t1+t2+t3+t4+t5))
  }

  d <- sapply(1:K,pk)

  class <- apply(d,1,function(x)
    {
      colnames(V)[which.max(x)]
    })

  colnames(d) <- colnames(V)

  return(
    structure(
      .Data = list(
        posteriors=d,
        class=rownames(d)
      )
    )
  )
}
```

```r
posteriors <- quadratic_df(x,pars(x,Y))
# qda_pred <- MASS::qda(Y~x,CV=TRUE)$class
# table(rownames(posteriors),qda_pred)

# predict <- posteriors[,apply(posteriors,1,which.max)]
```

Exercise 2

The code depicts a k-fold cross validation algorithm. The algorithm takes arguments `d` and `k`, where `d` is a $n \times 2$ design matrix of normal random variables with input and response vectors, and `k` is a paramter controlling the number of k-fold subsamples.

```
#' k-fold cross validation
#' @param d data matrix n * 2
#' @param k number of CV intervals
#' @return
#' Average MSE of k-fold cross-validation
c_k = function(d,k=10) {

  # Get size of n -- num rows from d
  n = dim(d)[1]

  # Input var X
  X = d[,1]
  # Response var Y
  Y = d[,2]

  # k-folds
  kf = k

  # int vector ck length kf
  # initialised with zeros
  ck = rep(0,kf)

  # For each ith fold
  # estimate k-fold train MSE
  for (i in 1:kf) {

    # lower bound fold
    ii = ceiling(1+n*(i-1)/kf)

    # upper bound fold
    ii2 = ceiling(n*i/kf)

    # train subset
    tt = ii:ii2

    # index of train sample
    # out of sample
    tr = setdiff(1:n,tt)

    # beta hat estimate
    # simple OLS on train sample
    bh = sum(X[tr]*Y[tr]/sum(X[tr]^2))

    # estimate response over
    # using test sample
    yh = X[tt]*bh

    # ith C_k=kf
    # test mean square error
    ck[i] = mean((Y[tt] - yh)^2)
  }
  return(mean(ck))
}
```

The algorithm assumes that input and response variables are characterised by simple ordinary least squares relationship, and computes the average TSME from k-fold cross validation of the regression.

```
#  random normal input vector
X <- rnorm(20,0,1)

# response vector
Y <- 1+2*X+rnorm(20,0,2)
m <- matrix(c(X,Y),nrow=20)
c_k(m,10)
```

```
## [1] 5.738396
```

Exercise 3

The following code provides an algorithm to find the binary split that minimises SSE of the sample.

```
#' Binary split that minimise sse
#' @param x vector or matrix of inputs
#' @param y vector of responses
#' @return
#' sse - minimized sse score
#' split
obj <- function(x,y)
{
   regions <- sort(unique(x))
   SSE <- c()
   for(i in seq_along(regions))
   {
     si <- regions[i]
     SSE[i] <- sum((y[x < si] - mean(y[x < si]))^2) +
               sum((y[x >= si] - mean(y[x >= si]))^2)
   }

   split <- regions[which.min(SSE)]

   return(c(sse=min(SSE),split=split)
   )
}
```

```
X <- runif(10)
Z <- runif(10)
Y <- X-Z
d <- data.frame(matrix(cbind(Y,X,Z),ncol = 3))
colnames(d) <- c("Y","X","Z")

formula <- terms.formula(Y~X+Z-1)
X <- model.matrix(formula,d)
```

```
y <- d[,as.character(formula)[2]]
```

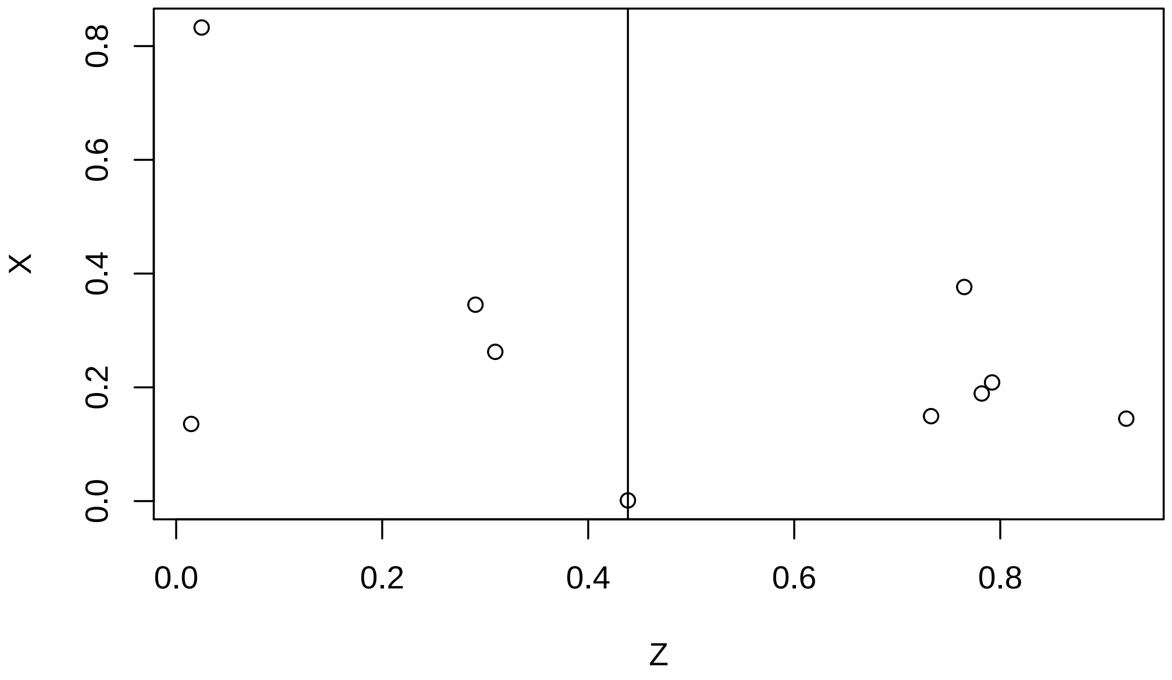
```
split <- apply(X,2,obj,y)
j <- which.min(split[1,])
l <- which.max(split[1,])

root_stump <- c(paste(names(j), ">=",
                      round(split[2,j],5)),
                paste(names(j), "<",
                      round(split[2,j],5))
)

root_stump
```

```
## [1] "Z >= 0.43853" "Z < 0.43853"
```

```
{
  xlabs <- names(j)
  ylabs <- names(l)
  plot(x=X[,j], y=X[,l], xlab=xlabs, ylab=ylabs)
  abline(v=split[2,j])
}
```


X
0.0
0.2
0.4
0.6
0.8
Z