```
## ----setup,
include=FALSE-------------------------------------------------------------------
knitr::opts_chunk$set(echo = TRUE)


## ---- echo=FALSE, warning=FALSE,
results='hide'------------------------------------------------------
#packages needed for the lab
library(kknn)
library(readxl)


## ---- warning=FALSE, results='hide',
echo=FALSE------------------------------------------------------------

data = read.csv("optdigits.csv", header = F)
n = dim(data)[1]
set.seed(12345)


id=sample(1:n, floor(n*0.5))
train=data[id,]

id1=setdiff(1:n, id)
set.seed(12345)

id2=sample(id1, floor(n*0.25))
valid=data[id2,]

id3=setdiff(id1,id2)
test=data[id3,]


#Exercise 2


k_test <- kknn(as.factor(V65)~., train = train, test = test, k = 30, kernel =
"rectangular")
k_train <- kknn(as.factor(V65)~., train = train, test = train, k = 30, kernel
= "rectangular")
k_valid <- kknn(as.factor(V65)~., train = train, test = valid, k = 30, kernel
= "rectangular")

test_table = table(k_test$fitted.values, as.factor(test$V65))
train_table = table(k_train$fitted.values, train$V65)
valid_table = table(k_valid$fitted.values, valid$V65)

missclass = function(X, X1) {
  n = length(X)
  return (1 - sum(diag(table(X1, X)))/n)
}

missclass_test = missclass(k_test$fitted.values, as.factor(test$V65))
missclass_train = missclass(train$V65, k_train$fitted.values)
```

```
missclass_valid = missclass(k_valid$fitted.values, valid$V65)


## ----
echo=T------------------------------------------------------------------------------------------------
test_table
train_table


## ----
echo=T------------------------------------------------------------------------------------------------
missclass_test
missclass_train

#Comment on the quality of predictions for different digits and on the overall
prediction quality.

#Some numbers have worse predictions than others, for example number 4 was
predicted to be a 7 a couple of times.
#This is most likely due to slopy writing that makes different numbers look
more similar to others.
#The overall prediction quality is good.


## ----
echo=F------------------------------------------------------------------------------------------------

prob_eight <- k_train$prob[, 9]

ordered_high <- order(prob_eight, decreasing = T)
ordered_high <- ordered_high[1:2]

ordered_low <- order(prob_eight, decreasing = F)
ordered_index <- which(prob_eight > 0)

ordered_low_eights <- c()
index = 1
while(length(ordered_low_eights) < 3) {
  value <-train[ordered_low[index], 65]
  if (value == 8) {
    ordered_low_eights <- append(ordered_low_eights, ordered_low[index])
  }
  index <- index + 1
}

print(ordered_low_eights)


  ordered_matrix <- c(ordered_low_eights, ordered_high)

for (i in ordered_matrix){
  my_heatmap <- matrix(as.numeric(train[i,1:64]), nrow=8,ncol=8, byrow = T)
  heatmap(my_heatmap, Colv = NA, Rowv = NA, main =train[i, 65])
}
```

```
#comment on whether these cases seem to be hard or easy to recognize visually.

#The eights that were easy to classify are also very easy to see as eights.
#The eights that were most difficult to spot are very difficult to see without
knowing that they're eights beforehand.



## ----
echo=F------------------------------------------------------------------------------
train_miss_error <- numeric(30)
val_miss_error <- numeric(30)


for (i in 1:30){
  k_valid <- kknn(as.factor(V65)~., train = train, test = valid, k = i, kernel
= "rectangular")
  k_train <- kknn(as.factor(V65)~., train = train, test = train, k = i, kernel
= "rectangular")

  train_miss_error[i] <- missclass(k_train$fitted.values, train$V65)
  val_miss_error[i] <- missclass(k_valid$fitted.values, valid$V65)

}

plot = plot(c(1:30), train_miss_error, ylab = "train_miss_error", xlab = "k",
col="pink")
points(c(1:30), val_miss_error, col="green")

k_test <- kknn(as.factor(V65)~., train = train, test = test, k = 4, kernel =
"rectangular")
k_train <- kknn(as.factor(V65)~., train = train, test = train, k = 4, kernel =
"rectangular")
k_valid <- kknn(as.factor(V65)~., train = train, test = valid, k = 4, kernel =
"rectangular")

missclass_test = missclass(k_test$fitted.values, as.factor(test$V65))
missclass_train = missclass(train$V65, k_train$fitted.values)
missclass_valid = missclass(k_valid$fitted.values, valid$V65)


## ----
echo=T------------------------------------------------------------------------------
missclass_test
missclass_train
missclass_valid

#How does the model complexity change when K increases and how does it affect
the training and validation errors?

#The model complexity seems to increase as k increases, given that the
missclass error increases as well.
#The best K values are the ones with the lowest missclass error, in this case
k=3 and k=4 are the best values.
```

```
#We can see that train gives the lowest missclass error because it is the date
we made the model from.
#Both test and validation is slightly higher which is to be expected.
#With the right k value computed with the validation data we get a test error
which is just above 2.5%
#Which results in a pretty high quality model.


## ----
echo=F---------------------------------------------------------------------------
cross.entropy <- function(p, phat){
  x <- 0
  for (i in 1:length(p)){
    x <- x + (p[i] * log(phat[i]))
  }
  return(-x)
}

train_miss_error <- as.vector(matrix(0,ncol = 30))
val_miss_error <- as.vector(matrix(0,ncol = 30))

entropy_error <- as.vector(matrix(0,ncol = 30))

for (i in 1:30){
  k_valid <- kknn(as.factor(V65)~., train = train, test = valid, k = i, kernel
= "rectangular")
  k_train <- kknn(as.factor(V65)~., train = train, test = train, k = i, kernel
= "rectangular")


  for (j in 0:9){
    cross_val <- valid$V65 == j
    cross_train <- train$V65 == j

    bool_val <- (which(cross_val, useNames = T))
    prob_val <- k_valid$prob[cross_val, as.character(j)] + 1e-15

    bool_train <- (which(cross_train, useNames = T))
    prob_train <- k_train$prob[cross_train, as.character(j)] + 1e-15


    val_miss_error[i] <- cross.entropy(bool_val, prob_val)
    train_miss_error[i] <- cross.entropy(bool_train, prob_train)



    entropy_error[i] <- abs(val_miss_error[i]-train_miss_error[i])
  }

}

print(entropy_error)
```

```
plot(c(1:30), entropy_error, ylab = "cross_entropy_error", xlab = "k",
col="blue")
which.min(entropy_error)

#What is the optimal KK value here? Assuming that response has multinomial
distribution, why might the cross-entropy be a more suitable choice of the
error function than the missclassification error for this problem?

#This model might be more suitable because of lower complexity levels than
missclass error.
#The optimal K value here is K = 5.


## ----
echo=F---------------------------------------------------------------------------------
parkin = read.csv("parkinsons.csv", header = T)
parkin_corr <- data.frame(parkin[c(5,7:22)]) #Remove unused voice
characteristics
parkin_scaled <- as.data.frame(scale(parkin_corr))

n = dim(parkin_corr)[1]
set.seed(12345)

id=sample(1:n, floor(n*0.6))
train=parkin_scaled[id,]
test=parkin_scaled[-id,]


## ----
echo=T---------------------------------------------------------------------------------
fit = lm(motor_UPDRS ~ ., data = train)
sum = summary(fit)
mean(sum$residuals^2)
print(sum)

# The variables p-values that are higher than 0.05 does not contribute at all.
#The values that contribute significantly are the ones with high p-values and
high estimates. Jitter.DDP is one example of a value that will contribute
significantly to the model.



## ----
echo=F---------------------------------------------------------------------------------
log_likelihood <- function(train, Y, theta, sigma){

  n <- dim(train)[1]
  train_theta = train%*%theta

  sum1 <- n*log(sigma^2)/2
  sum2 <- n*log(2*pi)/2
  sum3 <- sum((train_theta-Y)^2)
  sum4 <- sum3/(2*sigma^2)
  return (-sum1-sum2-sum4)
}


ridge <- function(train, theta, lambda, Y){
  n<-dim(train)[2]
  sigma <- theta[n+1]
  theta <-as.matrix(theta[1:n])
  log_like <- log_likelihood(theta=theta,Y=Y,sigma=sigma,train=train)
  ridge <- -log_like + lambda*sum(theta^2)
  return(ridge)

}

ridgeOpt <- function(lambda, train, Y){

  train <- as.matrix(train)
  N = dim(train)[2]
  init_theta = integer(N)
  init_sigma = 1

  opt <- optim(par = c(init_theta,init_sigma), fn = ridge, lambda = lambda,
train = train, Y = Y, method = "BFGS")
  return(opt)
}

dF <- function(X, lambda){
  #From the course formula
  X <- as.matrix(X)
  Xt <- t(X)
  n <- dim(X)[2]
  I <- diag(n)
  P <- X%*%solve((Xt%*%X + (lambda*I)))%*%Xt
  return(sum(diag(P)))
}


## ----
echo=F-------------------------------------------------------------------------------------------------
  AIC = function(train,Y,theta, sigma, lambda){
    log_like = log_likelihood(train = train, Y=Y, theta = theta, sigma =
sigma)
    N = dim(train)[1] # No of data points
    df = dF(train,lambda)
    aic = (-2*log_like/N) + (2*df/N) #(-2*Log-likelihood/N) + 2*(df/N)
    return(aic)
  }


## ----
echo=T-------------------------------------------------------------------------------------------------
xtrain<-as.matrix(train[2:17])
ytrain<-as.matrix(train[1])
```

```
xtest=as.matrix(test[2:17])
ytest<-as.matrix(test[1])


for (lambda in c(1, 100, 1000)){
  opt = ridgeOpt(lambda, xtrain, ytrain)
  theta <- as.matrix(opt$par[1:16])
  sigma<- opt$par[17]
  MSE_train = mean((xtrain%*%theta - ytrain)^2)
  MSE_test = mean((xtest%*%theta - ytest)^2)
  aic = AIC(train=xtrain,Y= ytrain,theta = theta,sigma= sigma, lambda= lambda)
  print(paste("Lambda:",lambda))
  print(paste("TrainMSE:",MSE_train))
  print(paste("TestMSE:",MSE_train))
  print(paste("AIC:", aic))


}

#Which penalty parameter is most appropriate among the selected ones? Compute
and compare the degrees of freedom of these models and make appropriate
conclusions.

#We have read that to find the best aic value among mutiple aic values is the
lowest. In our example it seems like lambda = 1 and lambda = 100 gives similar
aic values which is hard to draw conclusions from. Lambda = 1 gives the lowest
AIC value.



## ----
echo=F--------------------------------------------------------------------------------
prime = read.csv("pima-indians-diabetes.csv", header = F)

set.seed(12345)



## ----
echo=F--------------------------------------------------------------------------------
coloor <- function(x){
  if (x==1){
    c = "red"
  } else{
    c="green"
  }
  return(c)
}

coloors = sapply(prime$V9, coloor)
plot( prime$V2, prime$V8, xlab = "Plasma", ylab = "Age", main = "doabets", col
= coloors)
```

```
## ----
echo=F--------------------------------------------------------------------------------
glm.fits = glm(V9~ V2 + V8, prime, family = "binomial" )
prob=predict(glm.fits, type="response")
pred=ifelse(prob>0.5, 'red','green')

table = table(pred, prime$V9)

miss <- missclass(pred, prime$V9)

plot(prime$V2, prime$V8,col=pred, xlab = "Plasma glucose levels", ylab =
"Age", main = paste("Missclass error", toString(miss), sep=" = ") )



## ----
echo=F--------------------------------------------------------------------------------

r=.5
glm.fits = glm(V9~ V2 + V8, prime, family = "binomial" )
cf = glm.fits$coefficients
prob=predict(glm.fits, type="response")
pred=ifelse(prob>0.5, 'red','green')

w9 = cf[1]
w2 = cf[2]
w8 = cf[3]

x8 = c(seq(0,100,0.1))
x2 = (log(-r/(r-1)) - w9 - w8*x8)/w2

plot(prime$V2, prime$V8,col=pred, ylab = "Age", xlab= "Plasma", main =
paste("Missclass Error", toString(miss), sep=" = "))
lines(x2,x8,col="blue")


## ----
echo=F--------------------------------------------------------------------------------
for(r in c(.2,.5, .8)) {
  pred=ifelse(prob>r, 'red','green')
  table(pred, prime$V9)

  miss <- missclass(pred, prime$V9)

  x2 = (log(-r/(r-1)) - w9 - w8*x8)/w2

  plot(prime$V2, prime$V8,col=pred, ylab = "Age", xlab= "Plasma", main =
paste("Missclass_Error = ", toString(miss), "\n r = ", r, sep= ""))
  lines(x2,x8,col="blue")
}


## ----
echo=F--------------------------------------------------------------------------------
```

```
expanded <- prime

expanded$z1 <- expanded$V2 ** 4
expanded$z2 <- expanded$V2 ** 3 * expanded$V8
expanded$z3 <- expanded$V2 ** 2 * expanded$V8 ** 2
expanded$z4 <- expanded$V2 * expanded$V8 ** 3
expanded$z5 <- expanded$V8 ** 4

glm.fits = glm(V9~ V2 + V8 + z1 + z2 + z3 + z4 + z5, expanded, family =
"binomial" )


for(r in c(.2,.5, .8)) {
  prob=predict(glm.fits, type="response")
  pred=ifelse(prob>r, 'red','green')
  table(pred, expanded$V9)

  miss <- missclass(pred, prime$V9)

  x2 = (log(-r/(r-1)) - w9 - w8*x8)/w2

  plot(expanded$V2, expanded$V8,col=pred, ylab = "Age", xlab= "Plasma", main =
paste("Missclass_Error = ", toString(miss), "\n r = ", r, sep= ""))
}
```