932A99 - Group K7 - Lab 1

Hoda Fakharzadehjahromy, Otto Moen & Ravinder Reddy Atla

2020-11-18

Statement of contribution: Assignment 1 was contributed mostly by Hoda, assignment 2 mostly by Ravinder and assignment 3 mostly by Otto.

1. Assignment 1 - Handwritten digit recognition with K-means

A.1.1

Import the data into R and divide it into training, validation and test sets (50%/25%/25%)

```
#-----------------------------------------------------------------------
# A.1.1
#-----------------------------------------------------------------------
```

```
library(tidyverse)
Data <- read.csv("optdigits.csv",header = FALSE)
#intended to represent values of a categorical variable
Data$V65 <- as.factor(Data$V65)
n <- dim(Data)[1] # Number of rows(Samples) = 3822
set.seed(1234)
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,] #Validation Set
id3 <- setdiff(id1,id2) #Test Set
test <- Data[id3,]
```

A.1.2

Use training data to fit 30-nearest neighbor classifier with function kknn() and kernel=”rectangular” from package kknn.

```
#-----------------------------------------------------------------------
# A.1.2
#-----------------------------------------------------------------------
```

confusion matrix

```
temp <- table(train$V65, pred_train)
temp2 <- table(test$V65,pred_test )
print("confusion matrix for Train ")
```

```
## [1] "confusion matrix for Train "
```

```
temp
```

```
##    pred_train
##       0   1   2   3   4   5   6   7   8   9
##   0 182   0   0   0   1   0   0   0   0   0
##   1   0 171   9   1   0   0   0   0   0   1
##   2   0   0 175   0   1   0   0   1   2   1
##   3   0   0   1 188   0   2   0   1   0   1
##   4   0   1   0   0 203   0   1   7   0   4
##   5   0   1   0   1   0 171   0   1   1   9
##   6   0   1   0   0   0   0 173   0   0   0
##   7   0   0   0   0   0   0   0 200   0   0
##   8   0   8   0   0   0   0   1   0 199   0
##   9   1   3   0   4   2   0   0   7   1 173
```

```
print("confusion matrix for test ")
```

```
## [1] "confusion matrix for test "
```

```
temp2
```

```
##    pred_test
##       0   1   2   3   4   5   6   7   8   9
##   0 103   0   0   0   0   0   0   0   0   0
##   1   0  86   5   0   0   0   0   1   1   3
##   2   0   0  94   0   0   0   0   0   0   0
##   3   0   0   0  98   0   0   0   1   1   0
##   4   0   0   0   0  76   0   0   0   3   0
##   5   0   0   0   0   0  90   1   1   0   3
##   6   0   1   0   0   0   0  91   0   0   0
##   7   0   1   0   1   1   0   0 102   0   0
##   8   0   7   0   1   0   0   0   0  76   0
##   9   0   1   0   3   1   0   0   2   2 100
```

From the Confusion Matrix for we can see that the model is most successful on predicting digits 4. The quality of the model for fitting the input image to the correct output on training set is as follows:

Comment on the quality of predictions for different digits:

From the Confusion Matrix for we can see that the model is most successful on predicting digits 4. The quality of the model for fitting the input image to the correct output on training set is as follows:

```
t$ix
```

```
##  [1] 0 7 9 3 2 6 5 1 4 8
```

on the other hand,in the test set , the model is more successful on predicting digit 0 . the quality of the model prediction on different digits is respectively :

```
t2$ix
```

```
##  [1] 0 7 9 3 2 6 5 1 4 8
```

missclassification Error

```
missclass=function(X,X1){
  n=length(X)
  return(1-sum(diag(table(X,X1)))/n)
}
print("missclassification Error for Train Data")
```

```
## [1] "missclassification Error for Train Data"
```

```
missclass(train$V65,pred_train)
```

```
## [1] 0.03976975
```

```
print("missclassification Error for Test Data")
```

```
## [1] "missclassification Error for Test Data"
```

```
missclass(test$V65,pred_test)
```

```
## [1] 0.04284222
```

overall performance of the model

```
acc=function(x,x2)
{
  n=length(x)
  ac=sum(diag(table(x,x2)))/n
  return(ac)
}
print("accuracy on training Data ")
```

```
## [1] "accuracy on training Data "
```

```
acc(train$V65,pred_train)
```

```
## [1] 0.9602302
```

```
print("accuracy on test set ")
```

```
## [1] "accuracy on test set "
```

```
acc(test$V65,pred_test)
```

```
## [1] 0.9571578
```

A.1.3

```
#-----------------------------------------------------------------------
# A.1.3
#-----------------------------------------------------------------------
```

```
#any 2 cases of digit 8 in the training data which were
#easiest to classify and 3 cases that were
#hardest to classify

library(dbplyr)
new_train <-train
new_train$prob <-kknn_classifier_train$prob[,"8"]
df <- data.frame(new_train)
hard <- df[order(df$prob),] %>% head(3) #three hardest case
easy <- df[rev(order(df$prob)),] %>% head(2) # 2 easiest case
#the easiest case
heatmap(matrix(as.numeric(easy[1,1:64]),
               byrow = TRUE,nrow = 8,ncol = 8), Colv = NA, Rowv = NA,col=c("white","black"))
```

```
heatmap(matrix(as.numeric(easy[2,1:64]),byrow =
                 TRUE,nrow = 8,ncol = 8), Colv = NA, Rowv = NA,
        col=c("white","black"))
```

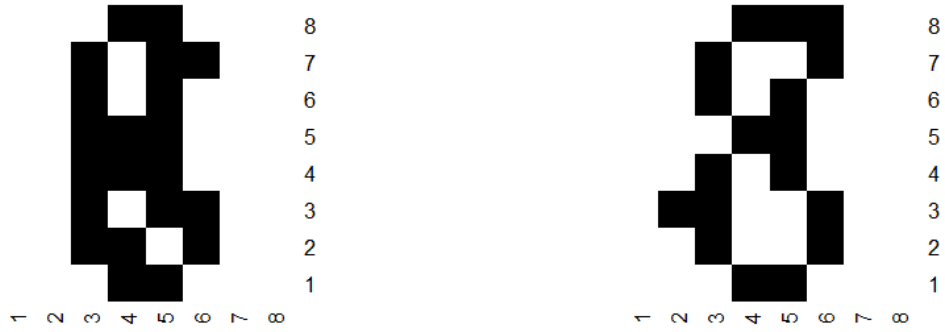
```
#The hardest Case
heatmap(matrix(as.numeric(hard[1,1:64]),
               byrow = TRUE,nrow = 8,ncol = 8), Colv = NA, Rowv = NA,col=c("white","black"))
```

```
heatmap(matrix(as.numeric(hard[2,1:64]),
               byrow = TRUE,nrow = 8,ncol = 8), Colv = NA, Rowv = NA,col=c("white","black"))
```

```
heatmap(matrix(as.numeric(hard[3,1:64]),
               byrow = TRUE,nrow = 8,ncol = 8), Colv = NA, Rowv = NA,col=c("white","black"))
```

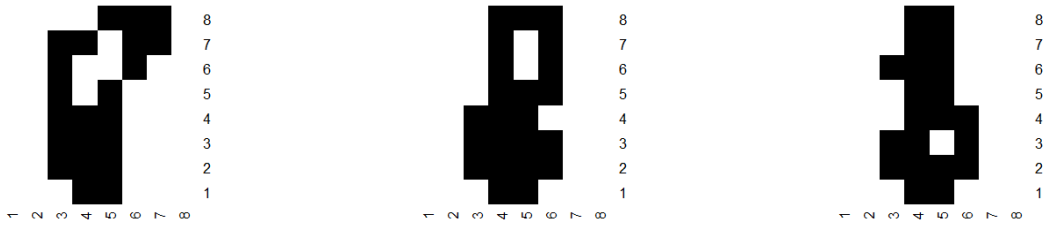
2 easiest cases that were easy to to classify(i.e having the highest probabilities of the correct class):

```
library(png)
library(grid)
library(gridExtra)
img1 <-  rasterGrob(as.raster(readPNG("images/best1.png")), interpolate = FALSE)
img2 <-  rasterGrob(as.raster(readPNG("images/best2.png")), interpolate = FALSE)
grid.arrange(img1, img2, ncol = 2)
```



3 cases that were hardest to classify (i.e. having lowest probabilities of the correct class). worst case:

```
library(png)
library(grid)
library(gridExtra)
img1 <-  rasterGrob(as.raster(readPNG("images/worst1.png")), interpolate = FALSE)
img2 <-  rasterGrob(as.raster(readPNG("images/worst2.png")), interpolate =
                      FALSE)
img3 <- rasterGrob(as.raster(readPNG("images/worst3.png")), interpolate =
                      FALSE)
grid.arrange(img1, img2, img3,ncol = 3)
```



These cases is also hard to be recognize as 8 by human as well (my opinion).

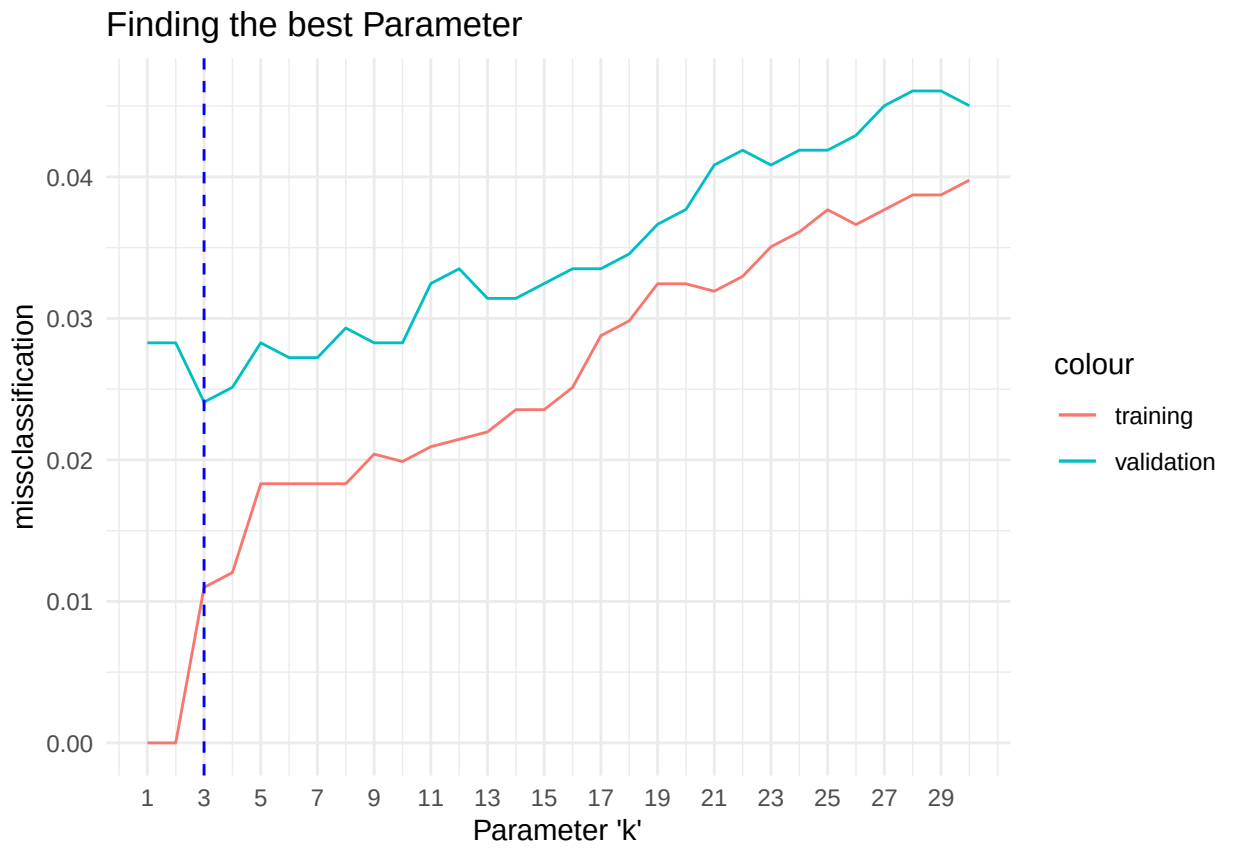
A.1.4

```
#-------------------------------------------------------------------------
# A.1.4
#-------------------------------------------------------------------------
```

we fitted K-nearest neighbor classifiers to the training data for different values of $k = 1, 2, \ldots, 30$ and plot the dependence of the training and validation misclassification.

```
missclass=function(X,X1){
  n=length(X)
  return(1-sum(diag(table(X,X1)))/n)
}
train_missclass <- c()
valid_missclass <- c()
for (k in 1:30) {
   KKNN_model_train <- kknn(formula =V65~., kernel = "rectangular", train = train,
                            test = train, k = k)
   KKNN_model_valid <- kknn(formula =V65~., kernel = "rectangular", train = train,
                            test = valid, k = k)
   train_missclass <- c(train_missclass,missclass(train$V65
                                                  ,KKNN_model_train$fitted.values))
   valid_missclass <- c(valid_missclass,missclass(valid$V65,KKNN_model_valid$fitted.values))
```

```
}
### plotting the result
library(ggplot2)
K <- 1:30
df<-data.frame(train_missclass,K1 =K)
df2<-data.frame(valid_missclass,K2 =K)
df3 <- data.frame(df,df2)
ggplot( ) +
   geom_line(aes(x=df3$K1,y=df3$train_missclass,
                 colour="training")) +
   geom_line(aes(x=df3$K2,y=df3$valid_missclass,
                 colour="validation")) +
   theme_minimal()+ggtitle("Finding the best Parameter") +
   scale_x_continuous(breaks = seq(1,30,2)) +
   xlab("Parameter 'k'") + ylab("missclassification") +
   geom_vline(xintercept = 3
                  , linetype = "dashed",color = "blue")
```



from the plot we can see that $k = 3$ is the optimal parameter for our model as it result in the lowest missclassification error. Furthermore, we test our model with $k = 3$ and we see :

```
KKNN_model_test <- kknn(formula =V65~., kernel = "rectangular", train = train,
                        test = test, k = 3)
cat("for k = 3 :\n")
```

```
## for k = 3 :
```

```
cat("miss classification error on training set =",train_missclass[3],"\n")
```

```
## miss classification error on training set = 0.01098901
```

```
cat("miss classification error on validation set = ",valid_missclass[3],"\n")
```

```
## miss classification error on validation set =  0.02408377
```

```
cat("miss classification error on test set = ",
    missclass(test$V65,KKNN_model_test$fitted.values),"\n")
```

```
## miss classification error on test set =  0.02298851
```

from the plot we can see that $k = 3$ is the optimal parameter for our model as it result in the lowest missclassification error. Furthermore, we test our model with $k = 3$ and we see :

```
KKNN_model_test <- kknn(formula =V65~., kernel = "rectangular", train = train,
                        test = test, k = 3)
cat("for k = 3 :\n")
```

```
## for k = 3 :
```

```
cat("miss classification error on training set =",train_missclass[3],"\n")
```

```
## miss classification error on training set = 0.01098901
```

```
cat("miss classification error on validation set = ",valid_missclass[3],"\n")
```

```
## miss classification error on validation set =  0.02408377
```

```
cat("miss classification error on test set = ",
    missclass(test$V65,KKNN_model_test$fitted.values),"\n")
```

```
## miss classification error on test set =  0.02298851
```

miss classification error rate is smaller in training set as expected. The MSE is almost equal on validation set and test set which indicate the model does not overfit. from the plot we observe that as k increases, the complexity of the model also increases. optimal value of k in this plot is 3.
Usually when the complexity is high we observe a high variance and when the model complexity is we face a model with high bias. Our goal is to minimize these 2 variable, hence there is always a trade-off between variance and bias. for $k = 3$ bias is the smallest because missclassification error is the lowest and the difference between missclassification error on the training set and validation set is also small, and that is why $k = 3$ is the most suitable choice for our model.

A.1.5

```
#------------------------------------------------------------------------
# A.1.5
#------------------------------------------------------------------------
```

Fit K-nearest neighbor classifiers to training data for different values of $k = 1, 2, \ldots, 30$ and compute the empirical risk for the validation data as cross-entropy.

Cross Entropy formula is :

$$CrossEntropy = -\sum_{c=1}^{M} y_{o,c} \log(p_{o,c})$$

- M:number of classes $(0, 1, 2, ...)$

- y: binary indicator (0 or 1) if class label
- c: is the correct classification for observation o
- p: predicted probability observation o is of class c From the plot we can see $k = 15$ is the better parameter for our model.

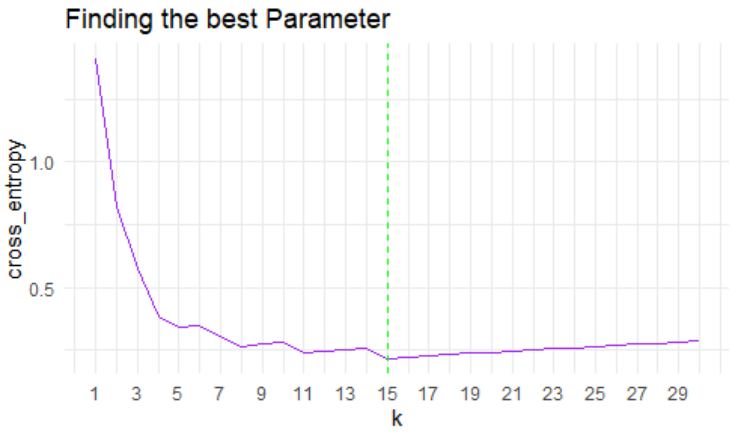


Figure 1: Cross Entropy

$k = 15$ results to the lowest empirical risk error.

from a probabilistic view MSE arises from the Liner Regression problem while Cross Enropy arises from Logistic Regression problems. we want to classify we are dealing with discrete output so, Cross Entropy is the better choice. Furthermore, MSE does not penalizes the model for wrong classification as much as Cross Entropy does.Actually, Cross Entropy penalizes the model for incorrect classification a lot more than MSE , as, its derivative is higher.

```
#------------------------------------------------------------------------
# A.1.5
#------------------------------------------------------------------------
library(kknn)
library(reshape)
library(ggplot2)
crossEntropy <- function(x){
   for (row in 1:nrow(x)){
      x[row, "cross_entropy"] <- -sum(log(x[row, 1:10]+1e-15,
                                          base = 2)*v_prob[[row,"coded"]] )
   }
   return(x)
}

encode_ordinal <- function(j) {
   x <- rep(0,10)
   x[j+1] <- 1
   return(I(x))
   }
ce <- c()
for (k in 1:30){

   KKNN_model_valid <- kknn(train$V65 ~ ., train = train ,test = valid, k =
                               k, kernel = "rectangular")
```

```
   prob_df <- KKNN_model_valid$prob
   digits <- colnames(prob_df)[apply(prob_df,1,which.max)]
   v_prob <- data.frame(prob_df)
   v_prob$y <- valid$V65
   v_prob$y_hat <- KKNN_model_valid$fitted.values
   v_prob$digits <- digits

   v_prob$coded <- (lapply(as.numeric(v_prob$y )-1,  encode_ordinal ))

   v_prob <- crossEntropy(v_prob)

   ce[k] <- mean(v_prob$cross_entropy)

}

metrics <- data.frame(ce, 1:30)
names(metrics) <- c("cross_entropy", "k")

df1 <- melt(metrics, id.var='k')

names(df1)[3] <- "cross_entropy"
ggplot(df1, aes(x=k, y=cross_entropy, col='cross_Entropy')) +
   geom_line(colour="purple") + theme_minimal()+
   ggtitle("Finding the best Parameter") +
   scale_x_continuous(breaks = seq(1,30,2)) +
   geom_vline(xintercept = 15
 , linetype = "dashed",color = "green")
```

```
best_K <- which(ce==min(ce))
cat("The best performant K parameter is k =", as.character(best_K))
```

```
## The best performant K parameter is k = 15
```

2. Assignment 2 - Ridge regression and model selection

```
#------------------------------------------------------------------------
# A.2.1
#------------------------------------------------------------------------
```

Data Description : The data consists of voice characteristic measurements from 43 people suffering from early stage of Parkinson's disease. The voice characteristic data are obtained by monitoring patients for six months using a tele-monitoring device. The aim is to predict a score for the symptoms of the disease.

- The data consists of 16 voice characteristics which constitutes our input(consider it as X).
- Output will be the score which is motor UPDRS variable(consider it as Y).

Since our target variable can be obtained by ridge regression of voice characteristics and that target variable(Y) is normally distributed, our target variable can be written as

Y $\sim N(XW,\sigma^2)$,

W $\sim N(0,\sigma^2\ \lambda^{-1}I)$, where W is weight matrix and lambda is penalty constant.

2.Scale and split data :

```
#-------------------------------------------------------------------------
# A.2.2
#-------------------------------------------------------------------------
```

The data is scaled to supress the magnitudes and is split into train and test data. Train and Test data has an extra column of 1's added for intercept($w0$). This data is divided into input and output for train and test respectively for better readability.

```
parkinson_data <- read.csv('parkinsons.csv')
parkinson_scaled_data <- scale(parkinson_data[,5:22])

n <- nrow(parkinson_scaled_data)
set.seed(1234)

# Obtaining train and test values randomly using sample function
id = sample(1:n, floor(n * 0.6))
train_data <- parkinson_scaled_data[id,]
train_data <- cbind(1,train_data)
test_data <-parkinson_scaled_data[-id,]
test_data <- cbind(1,test_data)

# Train and test values are separated by input and output for better readability
x_train <- train_data[,c(1,4:19)]
y_train <- train_data[,2]

x_test <- test_data[,c(1,4:19)]
y_test <- test_data[,2]
```

```
#-------------------------------------------------------------------------
# A.2.3
#-------------------------------------------------------------------------
```

3.a Log Likelihood :

likelihood, P(D|w,σ) = π $\frac{1}{\sqrt{2\pi\sigma^2}}$ exp[$\frac{-(Y-X^TW)^2}{2\sigma^2}$]

log likelihood = -nlog σ - $\frac{n}{2}$ log 2π - $\frac{1}{2\sigma^2}$ * Σ $\{(ytrain - (xtrain * w))^2\}$

```
loglikelihood <- function(data,w, si){
  x <- data[,c(1,4:19)]
  y <- data[,2]
  n <- ncol(x)
  loss <- (y - (x %*% (as.matrix(w))))^2
  ll <- -((n/2)*log(si^2)) - ((n/2) * log(2 * pi)) - ((1/2*si*si) * sum(loss))
  return(ll)
}
```

3.b Ridge loss function :

penalty $\sim$ $exp(-\lambda W^T W/2\sigma^2)$

ridge loss = $loglikelihood(w, \sigma) + \frac{\lambda}{2\sigma^2} * \Sigma \{w^2\}$

```
ridge <- function(data,par){
    w <- par[1:17]
    si <- par[18]

    ridge_loss <- loglikelihood(data, w, si) + ((1/(2*si*si))*(lambda * sum(w^2)))
```

```
    return(ridge_loss)
  }
```

3.c Ridge Opt function

```
ridge_opt <- function(data,lamda){

  ridge <- function(par){
    w <- par[1:17]
    si <- par[18]
    ridge_loss <- (-loglikelihood(data, w, si)) + ((1/(2*si*si))*(lambda * sum(w^2)))#-((n/2)*log(si^2)
    return(ridge_loss)
  }

  set.seed(1234)
  w <- sample(17)
  si <-  sample(1)
  lambda <- lamda
  out<- optim(c(w, si), ridge, method = 'BFGS')$par
  return(out)
}
```

3.d Degrees of freedom :

degrees of freedom = trace(hat_matrix)

```
df <- function(lambda){
  tobeinv <- (t(x_train) %*% x_train) + (lambda * diag(17))
  hat_matrix <- x_train %*% solve(tobeinv) %*% t(x_train)
  deg_of_freedom <- sum(diag(hat_matrix))
  return(deg_of_freedom)
}
```

```
#-----------------------------------------------------------------------
# A.2.4
#-----------------------------------------------------------------------
```

4. The values obtained below are MSE for λ = 1,100,1000 respectively for both train and test data.

```
## [1] 0.9947448 0.9949366 0.9949551
```

```
## [1] 1.006975 1.007169 1.007189
```

As observed from mean squared error values for train and test data, it is evident that the penalty parameter, $\lambda = 1$ is the most appropriate one as the loss in this case is less comparatively.

5.

```
#-----------------------------------------------------------------------
# A.2.5
#-----------------------------------------------------------------------
```

```
## [1] -174.1566 -227.8409 -191.8753
```

From the AIC(Akaike Information Criteria) values obtained using different λ values, it is observed that the model with $\lambda = 100$ and respective optimal parameters has lower AIC value comparatively which implies that the model is less complex.Hence, concluding the optimal model to be the one with $\lambda = 100$.

AIC measures balance between model fit and model complexity. It is using likelihood and degrees of freedom to obtain the measure. The complexity of the model can be interpreted from this measure This feature cannot

be obtained using MSE(Mean Squared Error) which chooses a random penalty parameter and obtained the model with less error.

3. Assignment 3 - Linear regression and LASSO

```
#-------------------------------------------------------------------------
# A.3.1
#-------------------------------------------------------------------------
```

In this assignment we use LASSO to fit models concerning the fat content in samples of meat. For the assignment the fat content is assumed to be linearly dependent on a 100 channel spectrum of absorbance records. Before fitting any model the data is divided into train, validation and test sets with the following code:

```
tecator <- read.csv("tecator.csv")
names(tecator)[1] <- "Sample"

n_tecator <- nrow(tecator)
set.seed(12345)
id <- sample(1:n_tecator, floor(n_tecator*0.5))
train_data <- tecator[id,]
test_data <- tecator[-id,]

train <- train_data[-c(1,103,104)]
test <- test_data[-c(1,103,104)]
```

3.1 - Linear regression

If the relationship between the fat content and the channels can be assumed to be linear, then a probabilistic regression model can be expressed as follows:

$$p(y|\mathbf{x}, \mathbf{w}) = N(\mathbf{w}^T\mathbf{x}, \sigma^2)$$

where:

$$\mathbf{w} = \{w_0, ..., w_{100}\}$$
$$\mathbf{x} = \{1, Channel_1, ..., Channel_{100}\}$$

This model, with all 100 channels included as features, can be fitted in R with the following code:

```
fat_lm <- lm(formula = Fat~.,
             data = train)
```

The training and test errors of the model are then estimated with the following code:

```
fat_lm_summary <- summary(fat_lm)
train_SSE <- sum(fat_lm_summary$residuals^2)

fat_lm_pred <- predict(fat_lm,
                       newdata = test)
test_resids <- test$Fat - fat_lm_pred
test_SSE <- sum(test_resids^2)

cat("The training error is: ",
```

```
    round(sqrt(train_SSE/(length(fat_lm_summary$residuals)-length(fat_lm$coefficients))),
          digits = 4),
    "\nThe test error is: ",
    sqrt(test_SSE/(length(test_resids)-length(fat_lm$coefficients))))
```

```
## The training error is:  0.3191
## The test error is:  105.5749
```

As can be observed by the above results the training error for this model is very low, but in contrast the test error is a lot higher. This suggests that using all 100 channels to predict the fat content leads to a highly overfitted model which performs very well on the training set but poorly on the test set.

3.2 - Objective function of LASSO regression

```
#------------------------------------------------------------------------
# A.3.2
#------------------------------------------------------------------------
```

If the relationship is still assumed to be linear an alternative to the regression model in 3.1 is the model known as least absolute shrinkage and selection operator, or LASSO. This model minimizes the minus log-likelihood plus a penalty factor λ to force less influential features to zero and thereby exclude them from the model. The function to optimize for this model can be expressed as follows:

$$\hat{w}^{lasso} = argmin \sum_{i=1}^{N} (y_i - w_0 - w_1 x_{1i} - ... - w_p x_{pi})^2 + \lambda \sum_{j=1}^{p} |w_j|$$

3.3 - LASSO regression model

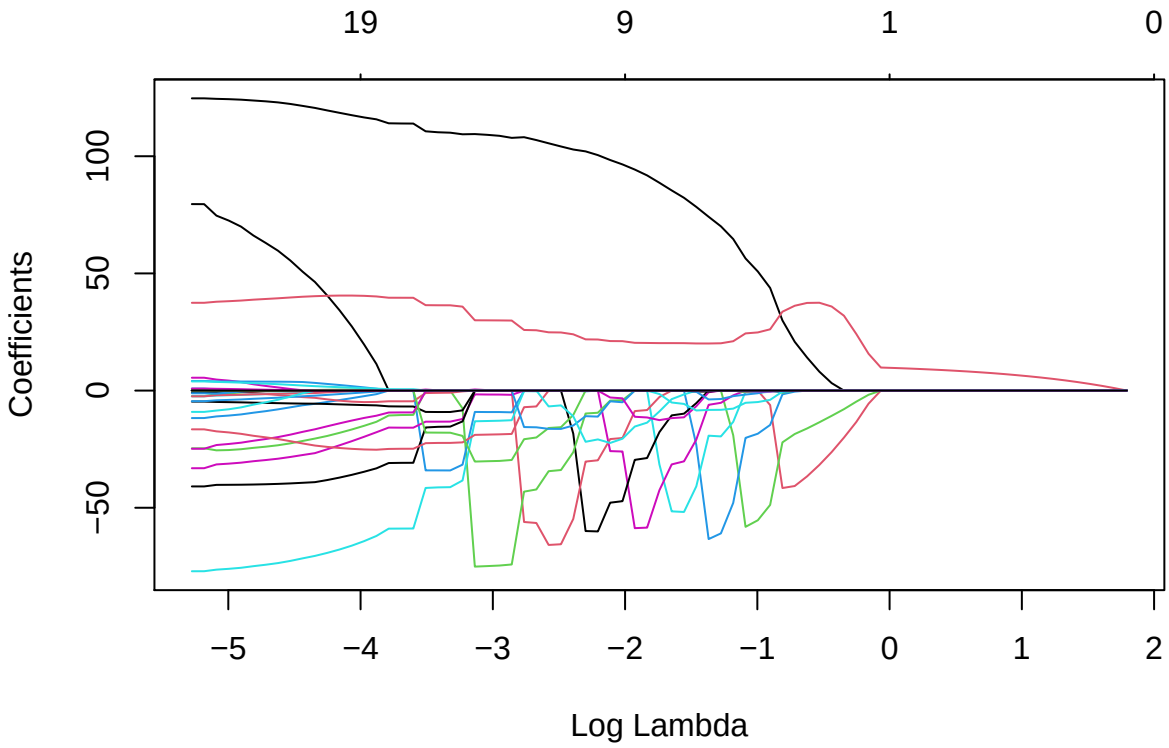
```
#------------------------------------------------------------------------
# A.3.3
#------------------------------------------------------------------------
```

To fit a LASSO model in R the **glmnet()** function is used with parameter $\alpha = 1$ indicating that the model should be trained with the LASSO penalty factor.

```
fat_lasso <- glmnet(as.matrix(train[,1:100]),
                    train[,101],
                    alpha = 1)
```

One way of analyzing the results of this model is to examine how the amount of coefficients included change depending on $log(\lambda)$. This plot can be obtained by plotting the results of the model and setting $xvar = \text{"}lambda\text{"}$:

```
plot(fat_lasso,
     xvar = "lambda")
```



From the plot it can be observed that as $log(\lambda)$ increases the amount of coefficients included in the model decreases. A lot of the coefficients are forced to zero very quickly, which can be seen by observing the individual steps, where already at the first with $\lambda = 0.0051$ the amount of channels have already gone from the initial 100 down to 37. To acquire a model with only three features a λ of about 0.7 has to be used.

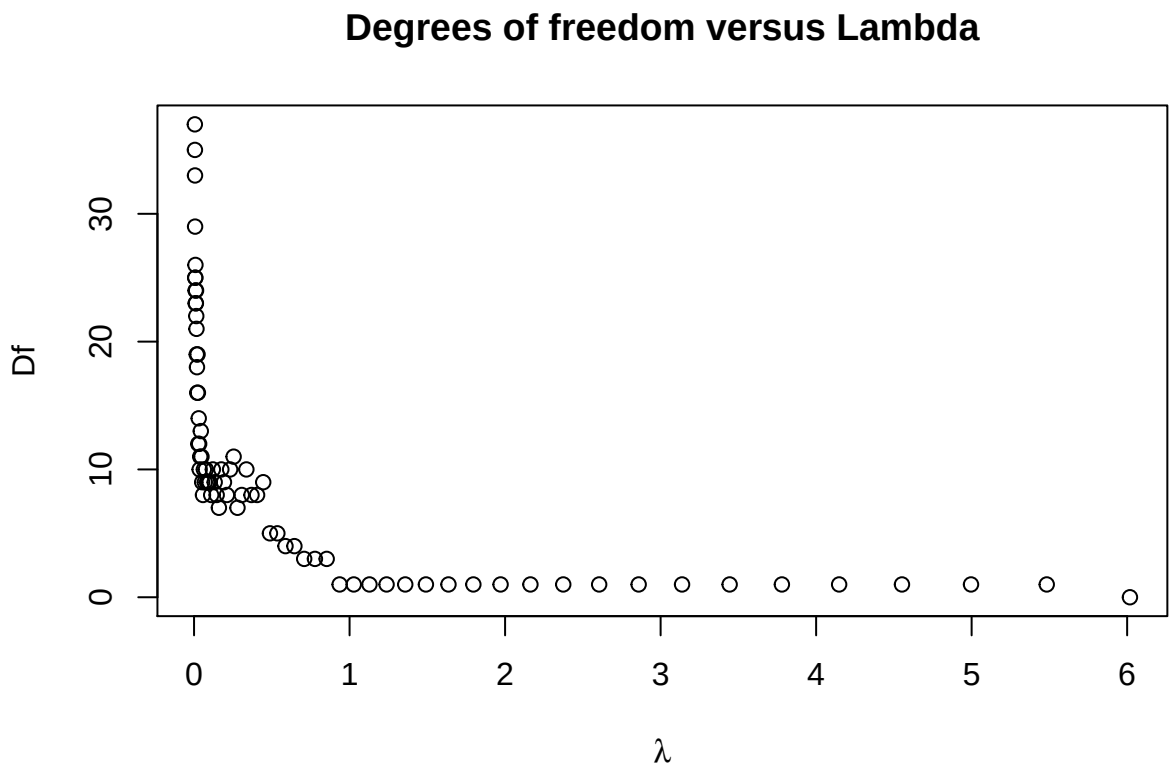
3.4 - Degrees of freedom vs penalty parameter

```
#-----------------------------------------------------------------------
# A.3.4
#-----------------------------------------------------------------------
```

Another way to visualize the result is to show how the degrees of freedom of the model depend on the penalty parameter.

```
fat_lasso_print <- print(fat_lasso)
```

```
plot(fat_lasso_print$Lambda,
     fat_lasso_print$Df,
     xlab = expression(lambda),
     ylab = "Df",
     main = "Degrees of freedom versus Lambda")
```



The plot shows that as λ increases the degrees of freedom go down, which is expected since the degrees of freedom is the amount of features in the model and as λ increases more of these features are forced to 0 and excluded from the model.

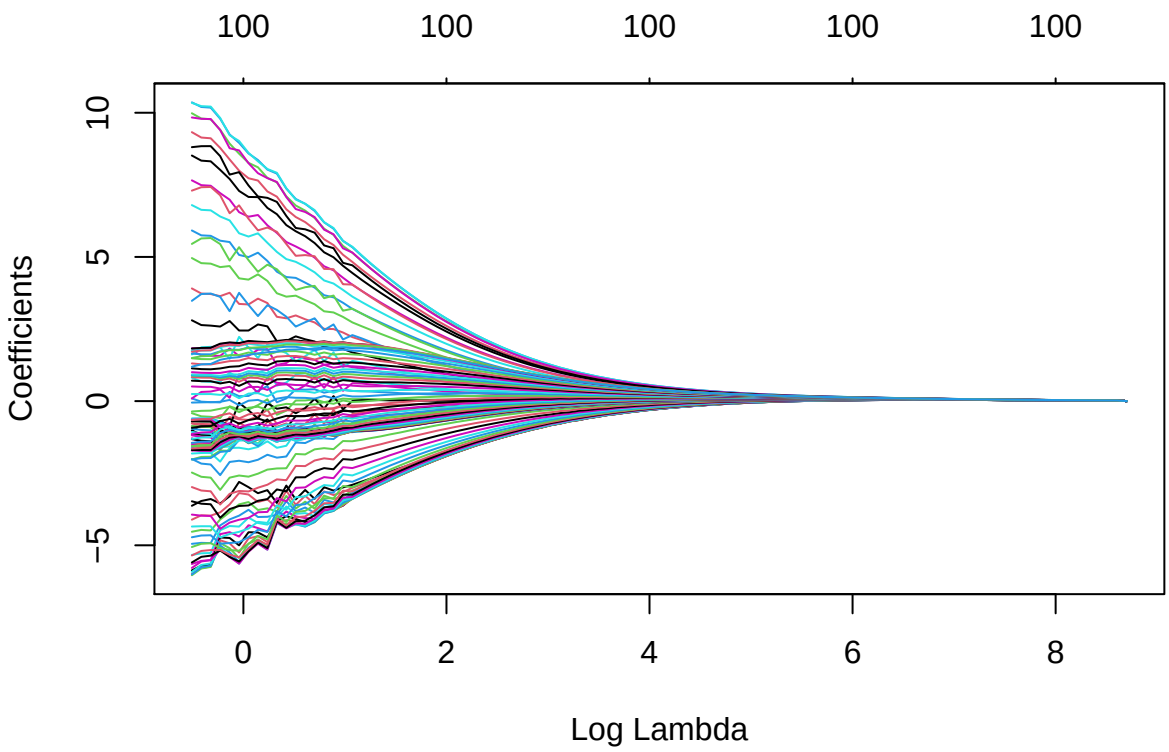
3.5 - Ridge regression model

```
#-----------------------------------------------------------------------
# A.3.5
#-----------------------------------------------------------------------
```

To compare how the LASSO model treats the model features the model from 3.3 is fitted, this time using Ridge regression instead. This is done with the same function, but with the parameter $\alpha = 0$:

```
fat_ridge <- glmnet(as.matrix(train[,1:100]),
                    train[,101],
                    alpha = 0)

plot(fat_ridge,
     xvar = "lambda")
```



The graph shows that unlike for the LASSO model, where a lot of coefficients were very quickly forced to zero, the Ridge regression does this at a slower and more gradual rate. It is also the case that for the Ridge model no feature is completely removed from the model, but are instead kept with the very low coefficient. This can be observed by the axis ticks at the top of the graph, which always say 100, compared to for the LASSO model where this number goes down as features are removed.

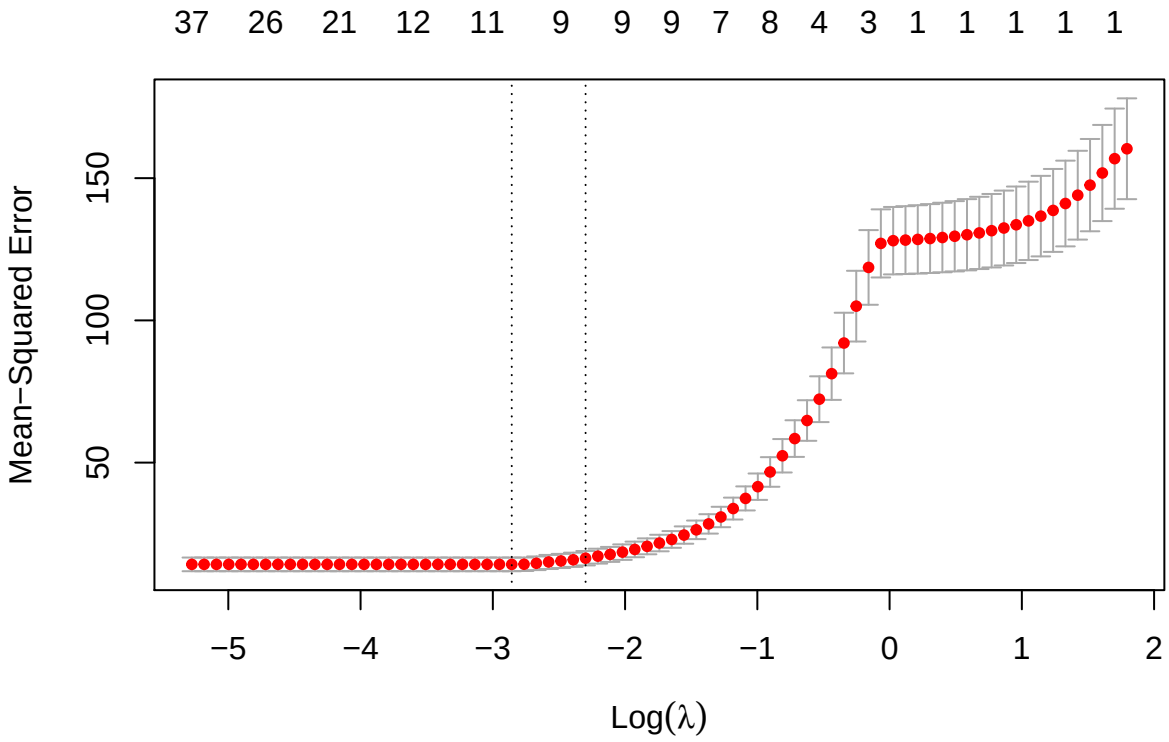
3.6 - LASSO model with cross-validation

```
#-----------------------------------------------------------------------
# A.3.6
#-----------------------------------------------------------------------
```

To find which value of λ leads to the most optimal value cross-validation is used. The object of this process is plotted to observe how different values of λ affects the result.

```
fat_lasso_cv <- cv.glmnet(as.matrix(train[,1:100]),
                          train[,101],
                          alpha = 1)

plot(fat_lasso_cv)
```



It can be observed that as $log(\lambda)$ increases so does the Mean-Squared error, at seemingly an exponential rate up until $log(\lambda) = 0$ where the increase halts. After this point the MSE once again starts growing, slowly at first, at what again could be seen as being an exponential rate. To find the optimal value for λ the **print()** function is used:

```
print(fat_lasso_cv)
```

```
##
## Call:  cv.glmnet(x = as.matrix(train[, 1:100]), y = train[, 101], alpha = 1)
##
## Measure: Mean-Squared Error
##
##      Lambda Measure    SE Nonzero
## min 0.05745   14.21 2.403       8
## 1se 0.10039   16.39 2.531       9
```

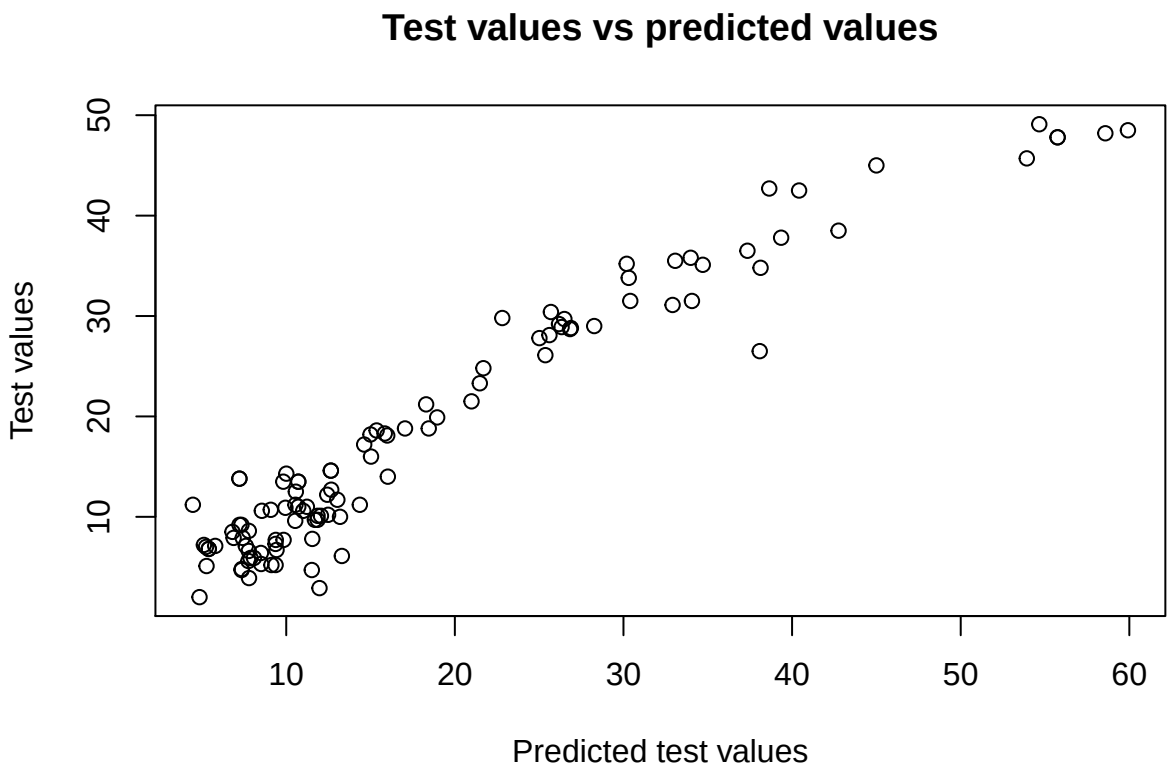
Optimal λ is 0.05745 and the amount of chosen variables is 8. The print function also shows that the largest λ where MSE is still within 1 standard error of the minimum error is 0.10039. Compared to this λ value, $log(\lambda) = -2 <=> lambda = 0.1353$, and as such would suggest that the error for $log(\lambda) = -2$ is more than 1 standard deviation away from the minimum value.

Finally a scatter plot is created to show how the predicted values for the test set correspond to the true values for the model based on the optimal value of λ:

```
fat_lasso_cv_preds <- predict(fat_lasso_cv,
                              newx = as.matrix(test[,1:100]),
                              s = "lambda.min")

plot(x = fat_lasso_cv_preds,
```

```
     y = test$Fat,
     xlab = "Predicted test values",
     ylab = "Test values",
     main = "Test values vs predicted values")
```



Overall it seems the model looks to be making fairly solid predictions, at least for lower levels of Fat. The correlation between the predicted values and the true values is 0.96. It can be seen that as Fat levels increase there is a slight curve to the slope of the points, as the model predicts too high levels of fat. As an example of this there are 5 points predicted to have a fat level between 50 and 60 while in reality these observations all had levels below 50.

3.7 - Generate new data

```
#-----------------------------------------------------------------------
# A.3.7
#-----------------------------------------------------------------------
```

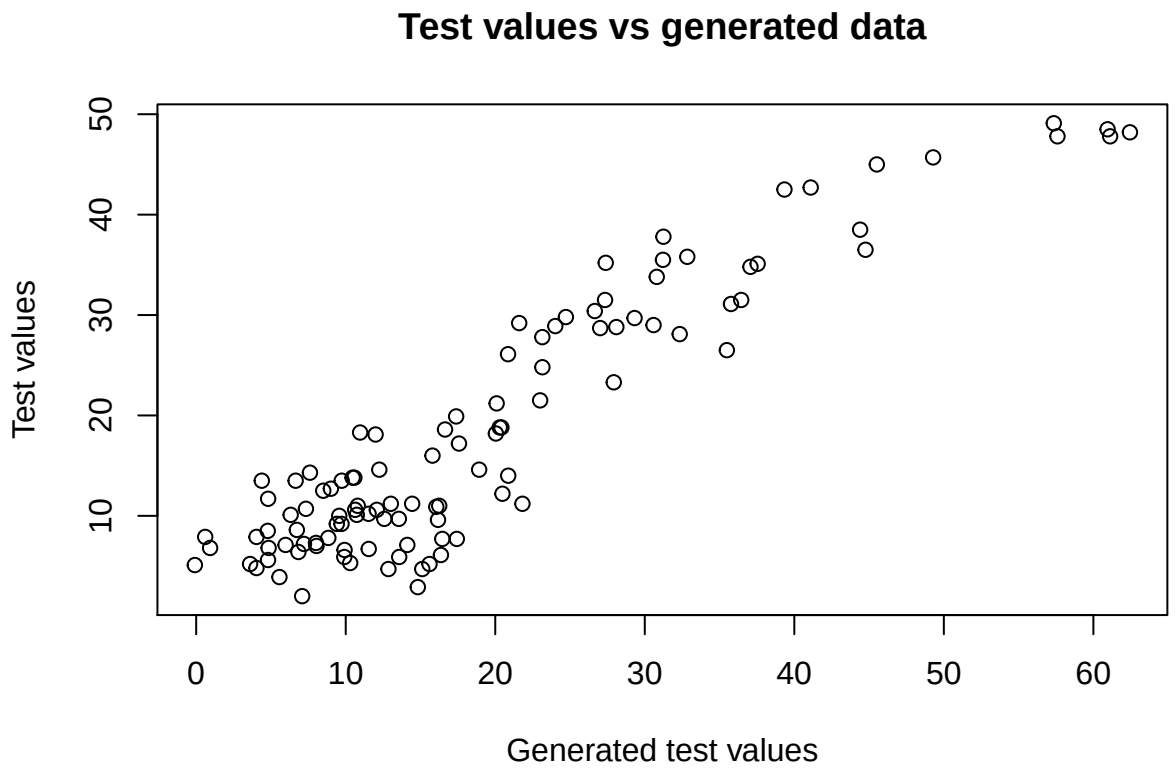
The last step is to use the feature values from the test data and use the model based on the optimal λ to generate new values for fat content. The values are generated from a normal distribution with a mean based on the coefficients of the chosen model and a standard deviation set as the standard deviation of the residuals from the predictions on the train set. The generated values are then plotted against the true values from the test set.

```
train_fits <- predict(fat_lasso_cv,
                      newx = as.matrix(train[,1:100]),
                      s = "lambda.min")
```

```r
train_resid <- train$Fat - train_fits
train_sigma <- sd(train_resid)

new_data <- 0
set.seed(12345)
for (obs in 1:108) {
  intercept <- coef(fat_lasso_cv, s ="lambda.min")[1]
  wx <- sum(coef(fat_lasso_cv, s ="lambda.min")[-1] * test[obs,1:100])
  new_data[obs] <- rnorm(n = 1,
                         mean = (intercept + wx),
                         sd = train_sigma)
}

plot(x = new_data,
     y = test$Fat,
     xlab = "Generated test values",
     ylab = "Test values",
     main = "Test values vs generated data")
```



In general the conclusions that can be drawn from this plot are similar to those from the predicted values in that the generated data appears to follow the true data fairly well, in particular for lower levels of fat. The correlation between the true data and the generated data is 0.93, which is slightly lower than for the predicted values. This makes sense as it can be observed that for the generated data there are now points that go above 60, whereas for the predicted value the upper boundary was around 60.

Appendix: Assignment Code

```
knitr::opts_chunk$set(echo = TRUE)
library(glmnet)
library(kknn)
#---------------------------------------------------------------------
# A.1.1
#---------------------------------------------------------------------


library(tidyverse)
Data <- read.csv("optdigits.csv",header = FALSE)
#intended to represent values of a categorical variable
Data$V65 <- as.factor(Data$V65)
n <- dim(Data)[1] # Number of rows(Samples) = 3822
set.seed(1234)
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,] #Validation Set
id3 <- setdiff(id1,id2) #Test Set
test <- Data[id3,]
#---------------------------------------------------------------------
# A.1.2
#---------------------------------------------------------------------
library(kknn)

kknn_classifier_train <- kknn(V65~.,train = train, test = train,
                              kernel = "rectangular", k = 30)

kknn_classifier_test <- kknn(V65~.,train = train, test = test,
                              kernel = "rectangular", k = 30)

pred_train <- kknn_classifier_train$fitted.values
pred_test <- kknn_classifier_test$fitted.values
temp <- table(train$V65, pred_train)
temp2 <- table(test$V65,pred_test )
print("confusion matrix for Train ")
temp
print("confusion matrix for test ")
temp2
t<-sort(diag(temp), index.return=TRUE,decreasing = TRUE)
t2 <- t<-sort(diag(temp2), index.return=TRUE,decreasing = TRUE)
t <- lapply(t, x<- function(x){return(x-1)})
t2 <- lapply(t2, x<- function(x){return(x-1)})
t$ix
t2$ix
missclass=function(X,X1){
  n=length(X)
  return(1-sum(diag(table(X,X1)))/n)
}
```

```
print("missclassification Error for Train Data")
missclass(train$V65,pred_train)

print("missclassification Error for Test Data")

missclass(test$V65,pred_test)
acc=function(x,x2)
{
  n=length(x)
  ac=sum(diag(table(x,x2)))/n
  return(ac)
}
print("accuracy on training Data ")
acc(train$V65,pred_train)

print("accuracy on test set ")
acc(test$V65,pred_test)
#------------------------------------------------------------------------
# A.1.3
#------------------------------------------------------------------------

#any 2 cases of digit 8 in the training data which were
#easiest to classify and 3 cases that were
#hardest to classify

library(dbplyr)
new_train <-train
new_train$prob <-kknn_classifier_train$prob[,"8"]
df <- data.frame(new_train)
hard <- df[order(df$prob),] %>% head(3) #three hardest case
easy <- df[rev(order(df$prob)),] %>% head(2) # 2 easiest case
#the easiest case
heatmap(matrix(as.numeric(easy[1,1:64]),
               byrow = TRUE,nrow = 8,ncol = 8), Colv = NA, Rowv = NA,col=c("white","black"))
heatmap(matrix(as.numeric(easy[2,1:64]),byrow =
                 TRUE,nrow = 8,ncol = 8), Colv = NA, Rowv = NA,
        col=c("white","black"))
#The hardest Case
heatmap(matrix(as.numeric(hard[1,1:64]),
               byrow = TRUE,nrow = 8,ncol = 8), Colv = NA, Rowv = NA,col=c("white","black"))
heatmap(matrix(as.numeric(hard[2,1:64]),
               byrow = TRUE,nrow = 8,ncol = 8), Colv = NA, Rowv = NA,col=c("white","black"))
heatmap(matrix(as.numeric(hard[3,1:64]),
               byrow = TRUE,nrow = 8,ncol = 8), Colv = NA, Rowv = NA,col=c("white","black"))

library(png)
library(grid)
library(gridExtra)
img1 <-  rasterGrob(as.raster(readPNG("images/best1.png")), interpolate = FALSE)
img2 <-  rasterGrob(as.raster(readPNG("images/best2.png")), interpolate = FALSE)
grid.arrange(img1, img2, ncol = 2)
library(png)
library(grid)
```

```
library(gridExtra)
img1 <-  rasterGrob(as.raster(readPNG("images/worst1.png")), interpolate = FALSE)
img2 <-  rasterGrob(as.raster(readPNG("images/worst2.png")), interpolate =
                      FALSE)
img3 <- rasterGrob(as.raster(readPNG("images/worst3.png")), interpolate =
                      FALSE)
grid.arrange(img1, img2, img3,ncol = 3)

#------------------------------------------------------------------------
# A.1.4
#------------------------------------------------------------------------
missclass=function(X,X1){
  n=length(X)
  return(1-sum(diag(table(X,X1)))/n)
}
train_missclass <- c()
valid_missclass <- c()
for (k in 1:30) {
   KKNN_model_train <- kknn(formula =V65~., kernel = "rectangular", train = train,
                            test = train, k = k)
   KKNN_model_valid <- kknn(formula =V65~., kernel = "rectangular", train = train,
                            test = valid, k = k)
   train_missclass <- c(train_missclass,missclass(train$V65
                                                  ,KKNN_model_train$fitted.values))
   valid_missclass <- c(valid_missclass,missclass(valid$V65,KKNN_model_valid$fitted.values))

}
### plotting the result
library(ggplot2)
K <- 1:30
df<-data.frame(train_missclass,K1 =K)
df2<-data.frame(valid_missclass,K2 =K)
df3 <- data.frame(df,df2)
ggplot( ) +
   geom_line(aes(x=df3$K1,y=df3$train_missclass,
                 colour="training")) +
   geom_line(aes(x=df3$K2,y=df3$valid_missclass,
                 colour="validation")) +
   theme_minimal()+ggtitle("Finding the best Parameter") +
   scale_x_continuous(breaks = seq(1,30,2)) +
   xlab("Parameter 'k'") + ylab("missclassification") +
   geom_vline(xintercept = 3
                    , linetype = "dashed",color = "blue")
KKNN_model_test <- kknn(formula =V65~., kernel = "rectangular", train = train,
                            test = test, k = 3)
cat("for k = 3 :\n")
cat("miss classification error on training set =",train_missclass[3],"\n")
cat("miss classification error on validation set = ",valid_missclass[3],"\n")
cat("miss classification error on test set = ",
    missclass(test$V65,KKNN_model_test$fitted.values),"\n")
KKNN_model_test <- kknn(formula =V65~., kernel = "rectangular", train = train,
                            test = test, k = 3)
cat("for k = 3 :\n")
```

```r
cat("miss classification error on training set =",train_missclass[3],"\n")
cat("miss classification error on validation set = ",valid_missclass[3],"\n")
cat("miss classification error on test set = ",
    missclass(test$V65,KKNN_model_test$fitted.values),"\n")

#---------------------------------------------------------------------
# A.1.5
#---------------------------------------------------------------------
#---------------------------------------------------------------------
# A.1.5
#---------------------------------------------------------------------
library(kknn)
library(reshape)
library(ggplot2)
crossEntropy <- function(x){
   for (row in 1:nrow(x)){
      x[row, "cross_entropy"] <- -sum(log(x[row, 1:10]+1e-15,
                                          base = 2)*v_prob[[row,"coded"]] )
   }
   return(x)
}

encode_ordinal <- function(j) {
   x <- rep(0,10)
   x[j+1] <- 1
   return(I(x))
   }
ce <- c()
for (k in 1:30){

   KKNN_model_valid <- kknn(train$V65 ~ ., train = train ,test = valid, k =
                               k, kernel = "rectangular")


   prob_df <- KKNN_model_valid$prob
   digits <- colnames(prob_df)[apply(prob_df,1,which.max)]
   v_prob <- data.frame(prob_df)
   v_prob$y <- valid$V65
   v_prob$y_hat <- KKNN_model_valid$fitted.values
   v_prob$digits <- digits

   v_prob$coded <- (lapply(as.numeric(v_prob$y )-1,  encode_ordinal ))

   v_prob <- crossEntropy(v_prob)

   ce[k] <- mean(v_prob$cross_entropy)

}

metrics <- data.frame(ce, 1:30)
names(metrics) <- c("cross_entropy", "k")

df1 <- melt(metrics, id.var='k')
```

```
names(df1)[3] <- "cross_entropy"
ggplot(df1, aes(x=k, y=cross_entropy, col='cross_Entropy')) +
   geom_line(colour="purple") + theme_minimal()+
   ggtitle("Finding the best Parameter") +
   scale_x_continuous(breaks = seq(1,30,2)) +
   geom_vline(xintercept = 15
 , linetype = "dashed",color = "green")

best_K <- which(ce==min(ce))
cat("The best performant K parameter is k =", as.character(best_K))
#-------------------------------------------------------------------------
# A.2.1
#-------------------------------------------------------------------------
#-------------------------------------------------------------------------
# A.2.2
#-------------------------------------------------------------------------
parkinson_data <- read.csv('parkinsons.csv')
parkinson_scaled_data <- scale(parkinson_data[,5:22])

n <- nrow(parkinson_scaled_data)
set.seed(1234)

# Obtaining train and test values randomly using sample function
id = sample(1:n, floor(n * 0.6))
train_data <- parkinson_scaled_data[id,]
train_data <- cbind(1,train_data)
test_data <-parkinson_scaled_data[-id,]
test_data <- cbind(1,test_data)

# Train and test values are separated by input and output for better readability
x_train <- train_data[,c(1,4:19)]
y_train <- train_data[,2]

x_test <- test_data[,c(1,4:19)]
y_test <- test_data[,2]
#-------------------------------------------------------------------------
# A.2.3
#-------------------------------------------------------------------------
loglikelihood <- function(data,w, si){
  x <- data[,c(1,4:19)]
  y <- data[,2]
  n <- ncol(x)
  loss <- (y - (x %*% (as.matrix(w))))^2
  ll <- -((n/2)*log(si^2)) - ((n/2) * log(2 * pi)) - ((1/2*si*si) * sum(loss))
  return(ll)
}
ridge <- function(data,par){
    w <- par[1:17]
    si <- par[18]

    ridge_loss <- loglikelihood(data, w, si) + ((1/(2*si*si))*(lambda * sum(w^2)))
    return(ridge_loss)
  }
```

```r
ridge_opt <- function(data,lamda){

  ridge <- function(par){
    w <- par[1:17]
    si <- par[18]
    ridge_loss <- (-loglikelihood(data, w, si)) + ((1/(2*si*si))*(lambda * sum(w^2)))#-((n/2)*log(si^2)
    return(ridge_loss)
  }

  set.seed(1234)
  w <- sample(17)
  si <-  sample(1)
  lambda <- lamda
  out<- optim(c(w, si), ridge, method = 'BFGS')$par
  return(out)
}
df <- function(lambda){
  tobeinv <- (t(x_train) %*% x_train) + (lambda * diag(17))
  hat_matrix <- x_train %*% solve(tobeinv) %*% t(x_train)
  deg_of_freedom <- sum(diag(hat_matrix))
  return(deg_of_freedom)
}
#-----------------------------------------------------------------------
# A.2.4
#-----------------------------------------------------------------------

lambda_vec <- c(1,100,1000)

mean_squared_error_train <- c()
mean_squared_error_test = c()
l <- c()

for(val in lambda_vec){
  # Obtaining optimal 'w' values using ridge_opt()
  param <- ridge_opt(train_data,val)
  w <- param[1:17]

  # MSE for train data
  pred_motor_UPDRS <- x_train %*% w
  n <- length(y_train)
  mse <- (1/n)* sum((pred_motor_UPDRS - y_train)^2)
  mean_squared_error_train<- append(mean_squared_error_train,mse)

  # MSE for test data
  pred_motor_UPDRS_test <- x_test %*% w
  n <- length(y_test)
  mse_test <- (1/n)* sum((pred_motor_UPDRS_test - y_test)^2)
  mean_squared_error_test <- append(mean_squared_error_test,mse_test)
}

print(mean_squared_error_train)
print(mean_squared_error_test)
```

```r
#-------------------------------------------------------------------------
# A.2.5
#-------------------------------------------------------------------------

aic <- function(lambda){
  param <- ridge_opt(train_data,lambda)
  w <- param[1:17]
  si <- param[18]
  l <- loglikelihood(train_data,w,si)
  d <- df(lambda)
  info_criteria <- 2*(d-l)
  return(info_criteria)
}

lambda_vec <- c(1,100,1000)
aic_vec <- c()
for(a_val in lambda_vec){
  aic_vec <- append(aic_vec,aic(a_val))
}

aic_vec
#-------------------------------------------------------------------------
# A.3.1
#-------------------------------------------------------------------------
tecator <- read.csv("tecator.csv")
names(tecator)[1] <- "Sample"

n_tecator <- nrow(tecator)
set.seed(12345)
id <- sample(1:n_tecator, floor(n_tecator*0.5))
train_data <- tecator[id,]
test_data <- tecator[-id,]

train <- train_data[-c(1,103,104)]
test <- test_data[-c(1,103,104)]
fat_lm <- lm(formula = Fat~.,
             data = train)
fat_lm_summary <- summary(fat_lm)
train_SSE <- sum(fat_lm_summary$residuals^2)

fat_lm_pred <- predict(fat_lm,
                       newdata = test)
test_resids <- test$Fat - fat_lm_pred
test_SSE <- sum(test_resids^2)

cat("The training error is: ",
    round(sqrt(train_SSE/(length(fat_lm_summary$residuals)-length(fat_lm$coefficients))),
          digits = 4),
    "\nThe test error is: ",
    sqrt(test_SSE/(length(test_resids)-length(fat_lm$coefficients))))
#-------------------------------------------------------------------------
# A.3.2
#-------------------------------------------------------------------------
```

```
#-------------------------------------------------------------------------
# A.3.3
#-------------------------------------------------------------------------
fat_lasso <- glmnet(as.matrix(train[,1:100]),
                    train[,101],
                    alpha = 1)
plot(fat_lasso,
     xvar = "lambda")
#-------------------------------------------------------------------------
# A.3.4
#-------------------------------------------------------------------------
fat_lasso_print <- print(fat_lasso)
plot(fat_lasso_print$Lambda,
     fat_lasso_print$Df,
     xlab = expression(lambda),
     ylab = "Df",
     main = "Degrees of freedom versus Lambda")
#-------------------------------------------------------------------------
# A.3.5
#-------------------------------------------------------------------------
fat_ridge <- glmnet(as.matrix(train[,1:100]),
                    train[,101],
                    alpha = 0)

plot(fat_ridge,
     xvar = "lambda")
#-------------------------------------------------------------------------
# A.3.6
#-------------------------------------------------------------------------
fat_lasso_cv <- cv.glmnet(as.matrix(train[,1:100]),
                          train[,101],
                          alpha = 1)

plot(fat_lasso_cv)
print(fat_lasso_cv)
fat_lasso_cv_preds <- predict(fat_lasso_cv,
                              newx = as.matrix(test[,1:100]),
                              s = "lambda.min")

plot(x = fat_lasso_cv_preds,
     y = test$Fat,
     xlab = "Predicted test values",
     ylab = "Test values",
     main = "Test values vs predicted values")
#-------------------------------------------------------------------------
# A.3.7
#-------------------------------------------------------------------------
train_fits <- predict(fat_lasso_cv,
                      newx = as.matrix(train[,1:100]),
                      s = "lambda.min")

train_resid <- train$Fat - train_fits
train_sigma <- sd(train_resid)
```

```r
new_data <- 0
set.seed(12345)
for (obs in 1:108) {
  intercept <- coef(fat_lasso_cv, s ="lambda.min")[1]
  wx <- sum(coef(fat_lasso_cv, s ="lambda.min")[-1] * test[obs,1:100])
  new_data[obs] <- rnorm(n = 1,
                         mean = (intercept + wx),
                         sd = train_sigma)
}

plot(x = new_data,
     y = test$Fat,
     xlab = "Generated test values",
     ylab = "Test values",
     main = "Test values vs generated data")
```