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.

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 "
```

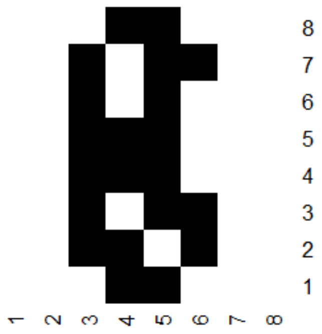
```
acc(test$V65,pred_test)
```

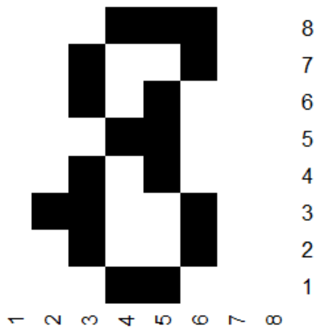
```
## [1] 0.9571578
```

A.1.3

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

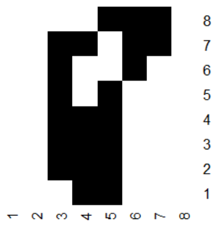
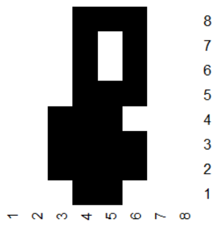
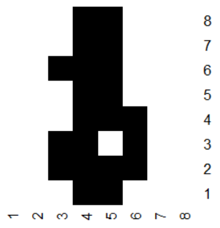
```
library(png)
library(grid)
library(gridExtra)
img1 <-  rasterGrob(as.raster(readPNG("images/best1.png")), interpolate = FALSE)
img2 <-  rasterGrob(as.raster(readPNG("images/best2.png")), interpolate = FALSE)
grid.arrange(img1, img2, ncol = 2)
```





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)
```

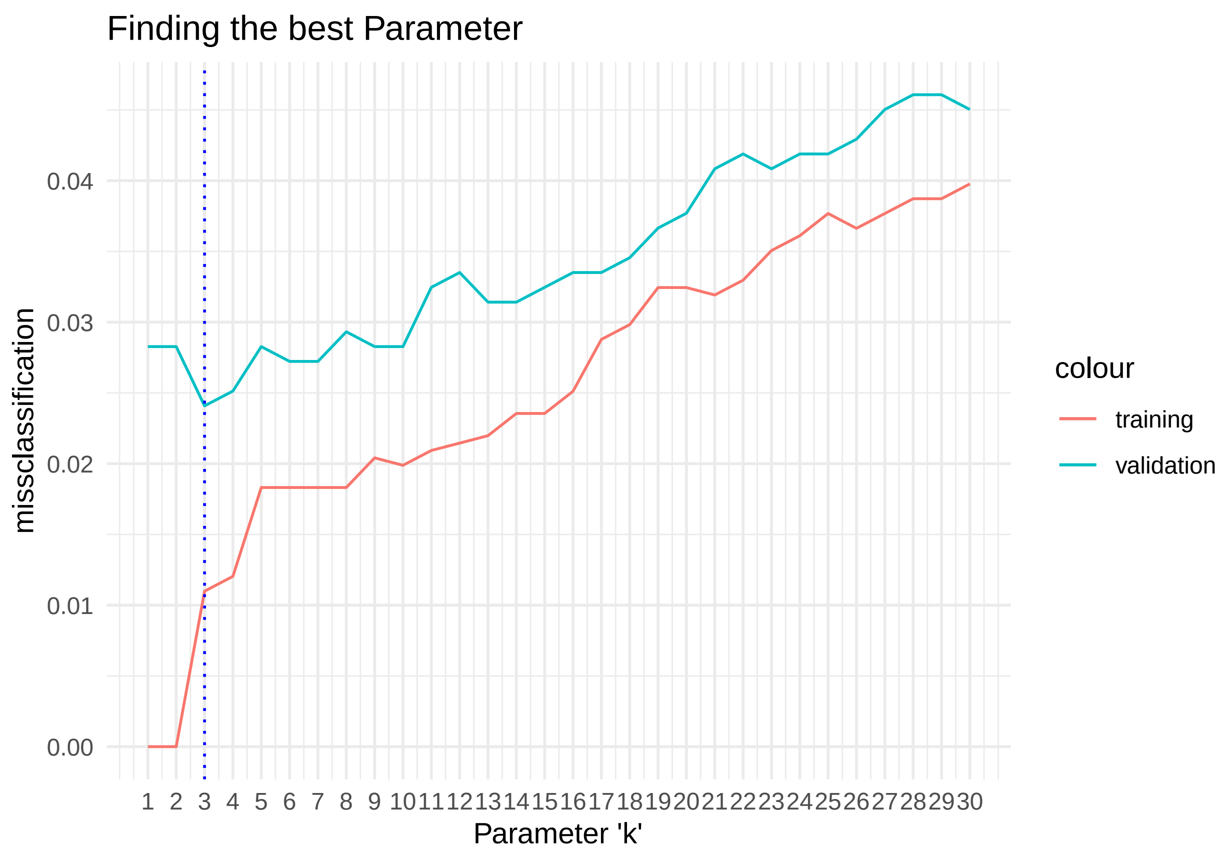
These cases is also hard to be recognize as 8 by human as well (my opinion).

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

```r
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)
```

```r
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 = 1:30) +
   xlab("Parameter 'k'") + ylab("missclassification") +
   geom_vline(xintercept = 3
                   , linetype = "dotted",color = "blue")
```



# 2. Assignment 2 - Ridge regression and model selection

3. Assignment 3 - Linear regression and LASSO

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:

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

n_tecator <- nrow(tecator)
set.seed(12345)
```

```r
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:

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

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

```r
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

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:

$$\hat{w}^{lasso} = argmin \sum_{i=1}^{N}(y_i - w_0 - w_1x_{1i} - ... - w_px_{pi})^2 + \lambda \sum_{j=1}^{p}|w_j|$$

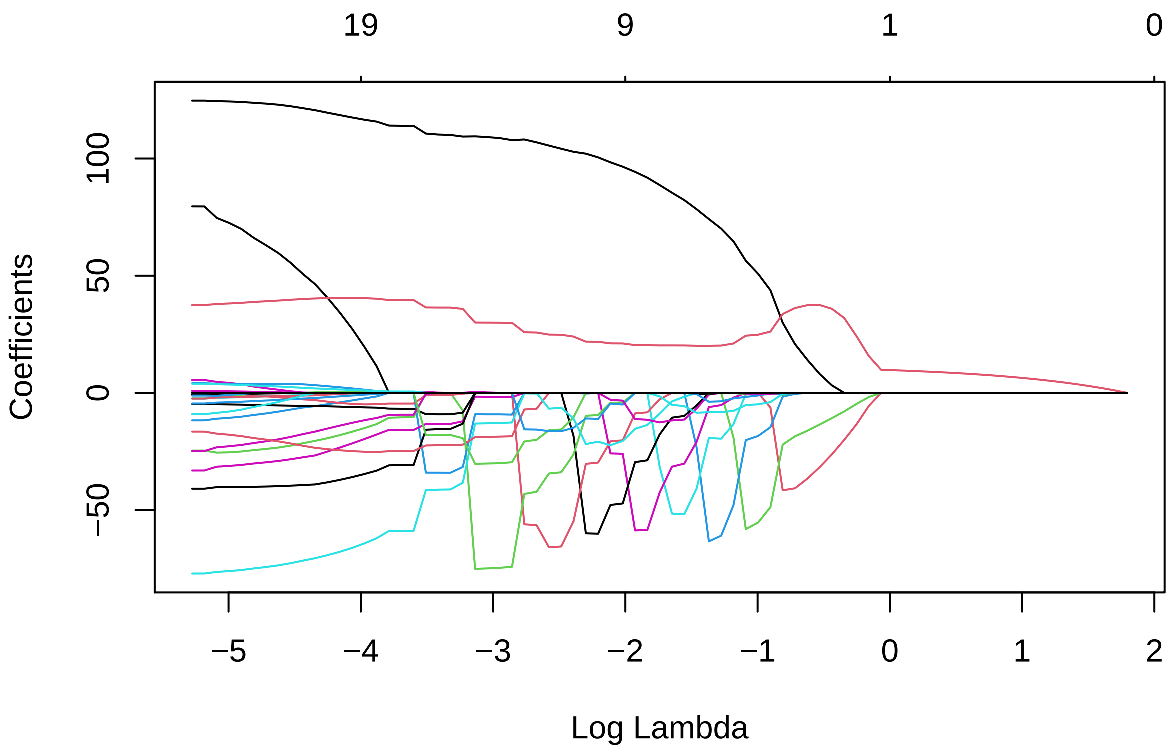
3.3 - LASSO regression model

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.

```r
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 $xvar =$ *"lambda"*:

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



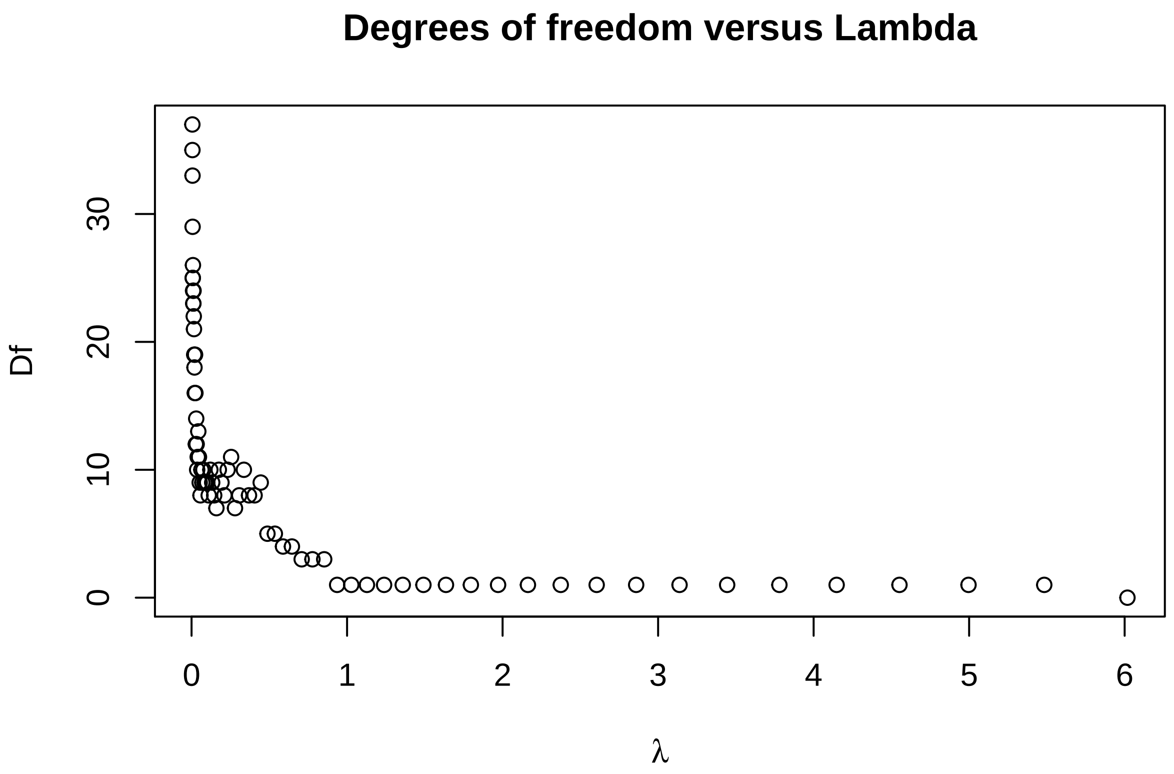
From the plot it can be observed that as $log(\lambda)$ increases the amount of coefficients included in the model decreases. A lot of the coefficients are forced to zero very quickly, which can be seen by observing the individual steps, where already at the first with $\lambda = 0.0051$ the amount of 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

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

```r
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")
```



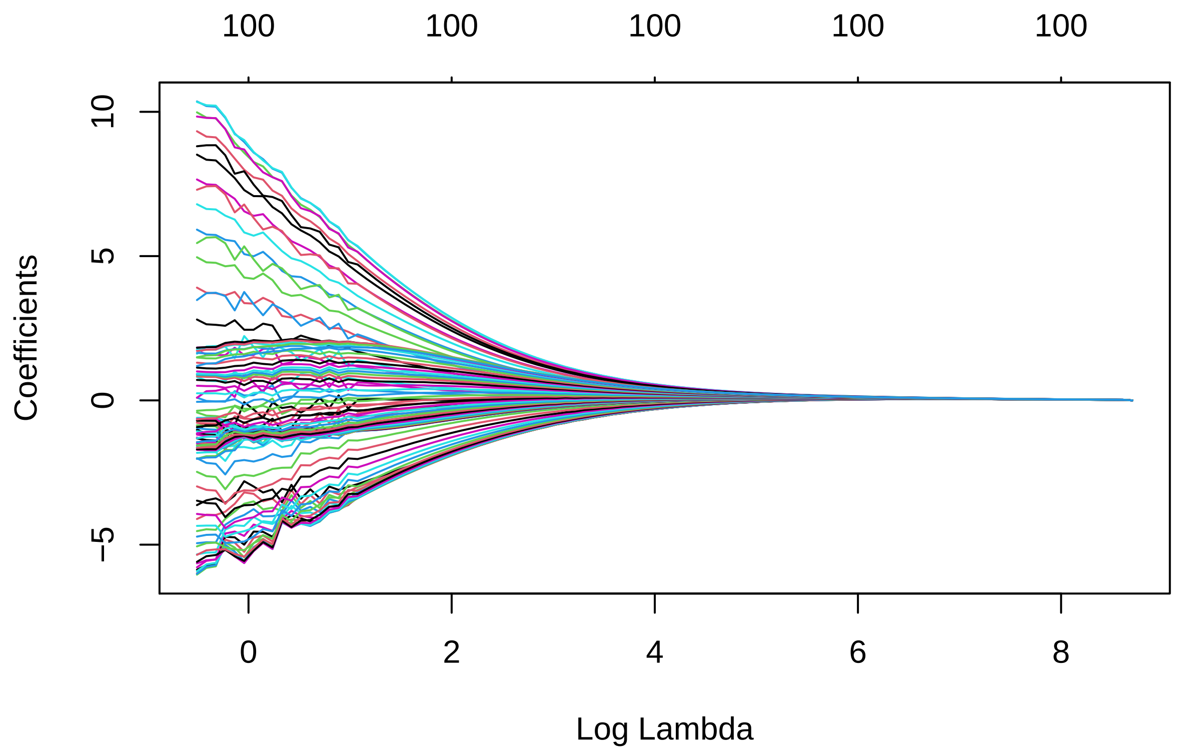
The plot shows that as λ increases the degrees of freedom go down, which is expected since the degrees of freedom is the amount of features in the model and as λ increases more of these features are forced to 0 and excluded from the model.

3.5 - Ridge regression model

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")
```



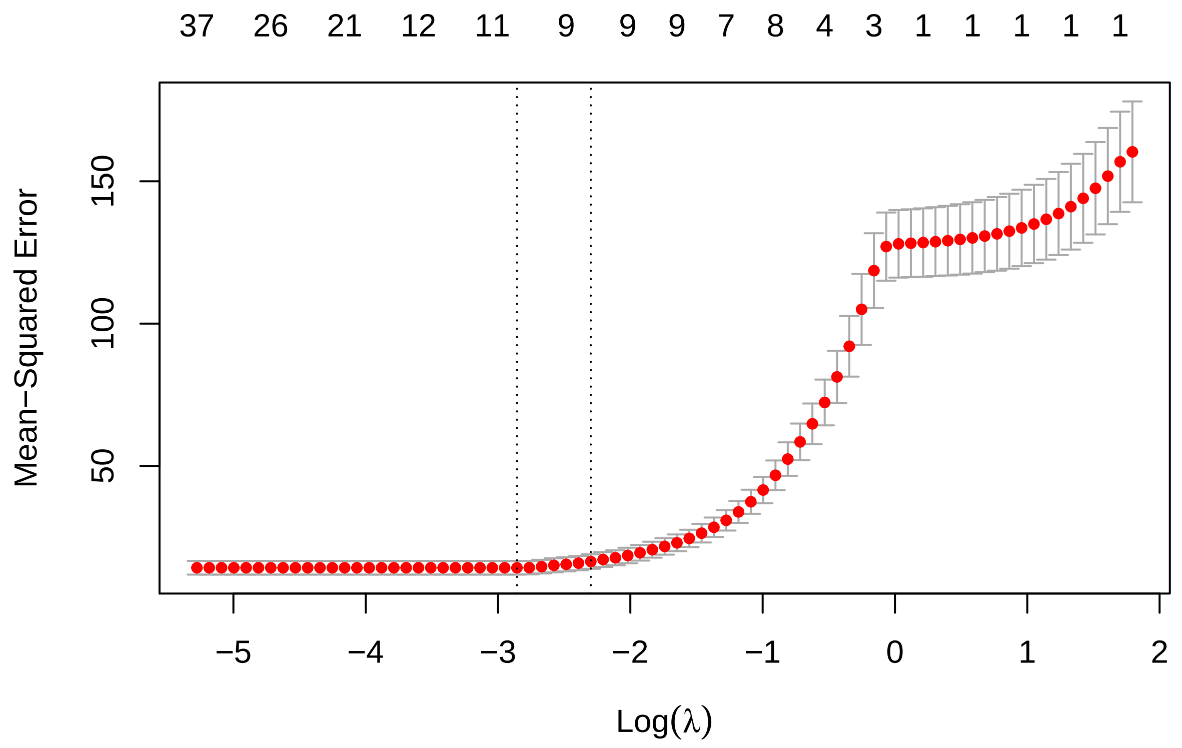
The graph shows that unlike for the LASSO model, where a lot of coefficients were very quickly forced to zero, the Ridge regression does this at a slower and more gradual rate. It is also the case that for the Ridge model no feature is completely removed from the model, but are instead kept with the 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

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:

```r
print(fat_lasso_cv)
```

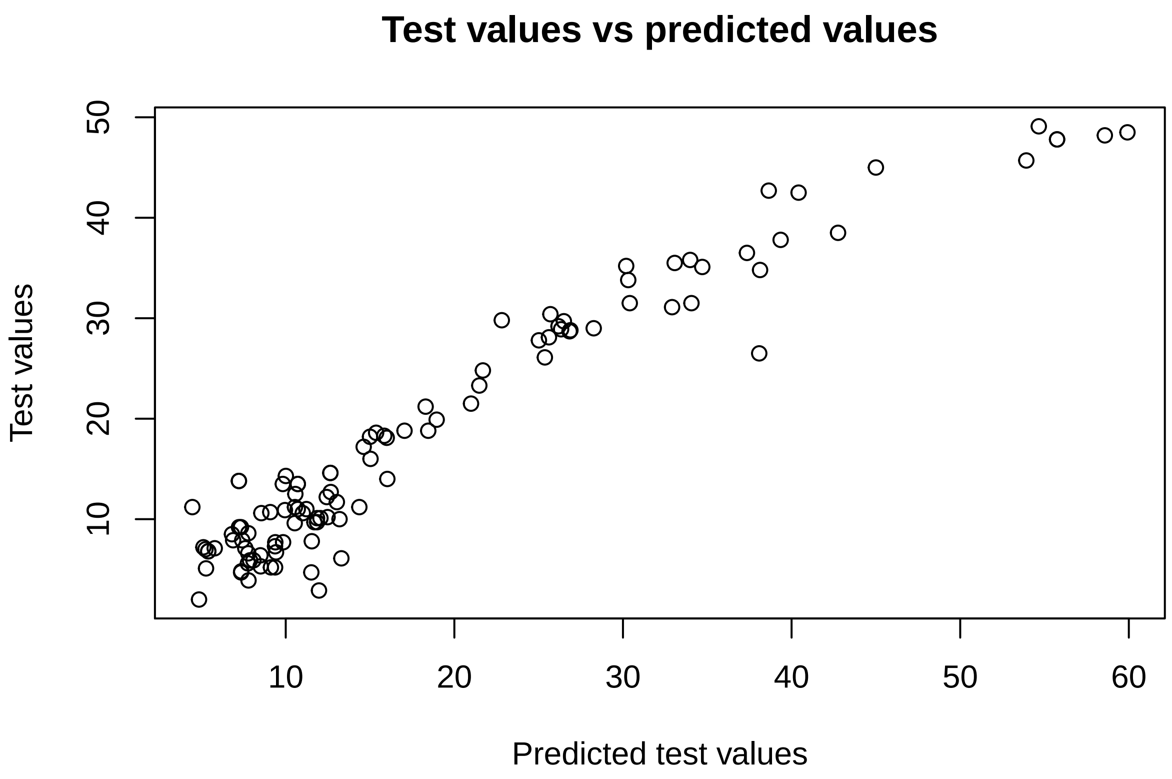
```
##
## Call:  cv.glmnet(x = as.matrix(train[, 1:100]), y = train[, 101], alpha = 1)
##
## Measure: Mean-Squared Error
##
##      Lambda Measure    SE Nonzero
## min 0.05745   14.21 2.403       8
## 1se 0.10039   16.39 2.531       9
```

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:

```r
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

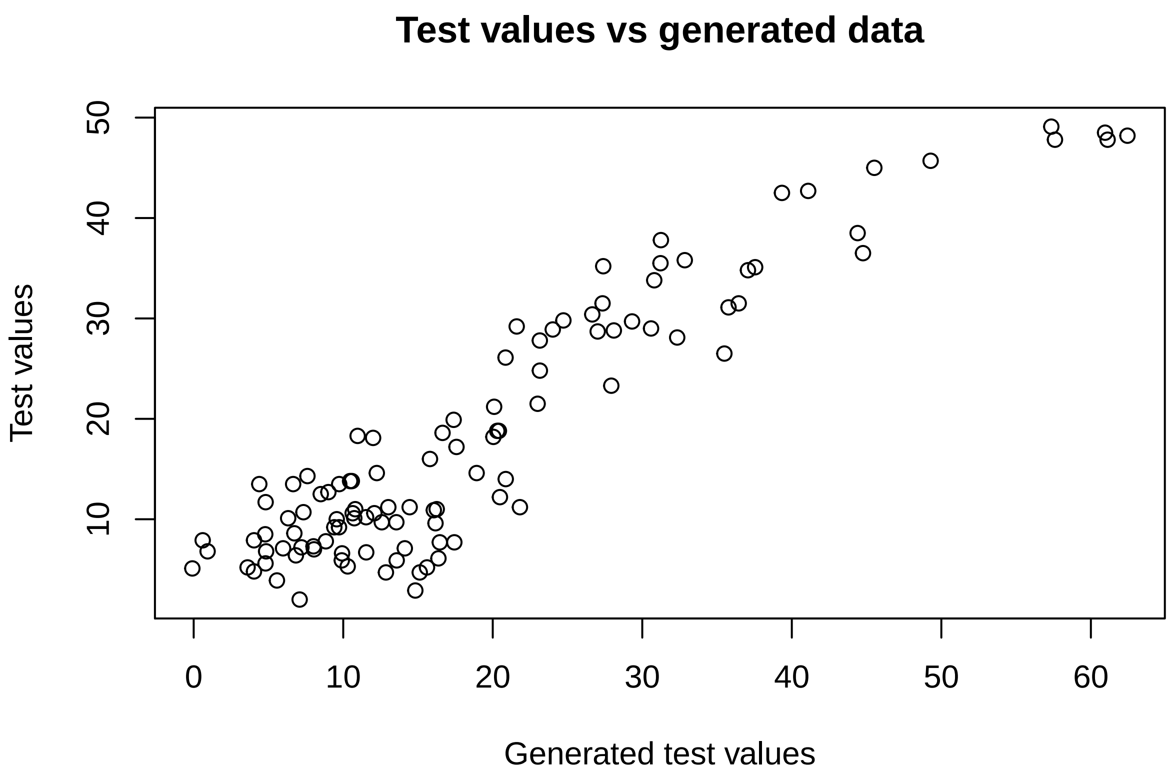
The last step is to use the feature values from the test data and use the model based on the optimal λ to generate new values for fat content. The values are generated from a normal distribution with a mean based on the coefficients of the chosen model and a standard deviation set as the standard deviation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

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")
```



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 3