Statement of Contribution

Alexander Bois is responsible for assignment 1, Daniel Bissessar for assignment 2 and Filip Brunander for assignment 3. Moreover, several programming sessions was held where we discussed problems and helped each other. However, the help and discussions where limited where the one responsible for the assignment also wrote all the code and answered all questions related to that assignment.

Assignment 1

Responsible: Alexander Bois

Task 1

Here we import the necessary packages for this assignment. Then we import the data from the csv file and divides the data set into train, test and validation sets.

```
library(kknn)

## Warning: package 'kknn' was built under R version 4.0.3

DF=read.csv(file = 'optdigits.csv', header = FALSE)
n=dim(DF)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=DF[id,]
id1=setdiff(1:n, id)
set.seed(12345)
id2=sample(id1, floor(n*0.25))
valid=DF[id2,]
id3=setdiff(id1,id2)
test=DF[id3,]
```

Task 2

First we define a function needed to compute the misclassification rate.

```
missclass = function(y, y_h){
  l=length(y)
  return(1-sum(diag(table(y,y_h)))/l)
}
```

Then we fit our model to the training set and predicts to the training data as well. We also present the confusion matrix for the predictions made. We also present the misclassification rate.

```
knn_fit_train <- kknn(as.factor(V65)~., train, train, k = 30, kernel = "rec
tangular")
train_confusion_matrix <- table(train$V65, knn_fit_train$fitted.values)
miss_train = missclass(train$V65, knn_fit_train$fitted.values)
train_confusion_matrix

##
##       0   1   2   3   4   5   6   7   8   9
##   0 202   0   0   0   0   0   0   0   0   0
##   1   0 179  11   0   0   0   0   1   1   3
##   2   0   1 190   0   0   0   0   1   0   0
##   3   0   0   0 185   0   1   0   1   0   1
##   4   1   3   0   0 159   0   0   7   1   4
##   5   0   0   0   1   0 171   0   1   0   8
##   6   0   2   0   0   0   0 190   0   0   0
##   7   0   3   0   0   0   0   0 178   1   0
##   8   0  10   0   2   0   0   2   0 188   2
##   9   1   3   0   5   2   0   0   3   3 183

miss_train

## [1] 0.04500262
```

With the train set it seems to have problems with doing the right prediction, when presented with the true numbers of 1, 4, and 5. It also seems to be predicting to 1, 7 and 9 more often than others.

We then do the same procedure for the test set, both fitting of and predicting on the test set.

```
knn_fit_test <- kknn(as.factor(V65)~., train, test, k = 30, kernel = "recta
ngular")
test_confusion_matrix <- table(test$V65, knn_fit_test$fitted.values)
miss_test = missclass(test$V65, knn_fit_test$fitted.values)
test_confusion_matrix

##
##       0   1   2   3   4   5   6   7   8   9
##   0  77   0   0   0   1   0   0   0   0   0
##   1   0  81   2   0   0   0   0   0   0   3
##   2   0   0  98   0   0   0   0   0   3   0
##   3   0   0   0 107   0   2   0   0   1   1
##   4   0   0   0   0  94   0   2   6   2   5
##   5   0   1   1   0   0  93   2   1   0   5
##   6   0   0   0   0   0   0  90   0   0   0
##   7   0   0   0   1   0   0   0 111   0   0
##   8   0   7   0   1   0   0   0   0  70   0
##   9   0   1   1   1   0   0   0   1   0  85

miss_test

## [1] 0.05329154
```

With the test set it seems to have problems when reading the true 4, 8 and 9. It also seems to predict to 9, 7 and 1 more often than others.

Over all prediction quality is better for the training set with a lower misclassification rate, which is explained by the fact that the model is trained by and is doing predictions on the same data set. But a misclassification rate of around 5% seems good since it is better than guessing, which would be 10% probability of being right(90% of being wrong).

Task 3

Now we need to identify the probabilities for a prediction to be 8, when it actually is 8. We will choose 2 prediction that had a high probability of being right, and 3 that had a lower probability of being right.

```
prob8s <- knn_fit_train$prob[,9] #probabilities of being 8


easy1 <- which.max(prob8s)
prob8s[easy1] <- NA
easy2 <- which.max(prob8s)
prob8s[easy2] <- NA

t <- 0
vhard <- c(0,0,0)
while (t<3) {
  temp <- which.min(prob8s)
  if(train[temp, 65] == 8){
  vhard[t+1] <- temp
  t <- t+1
  }
  prob8s[temp] <- NA
}

hard1 <- vhard[1]
hard2 <- vhard[2]
hard3 <- vhard[3]
```

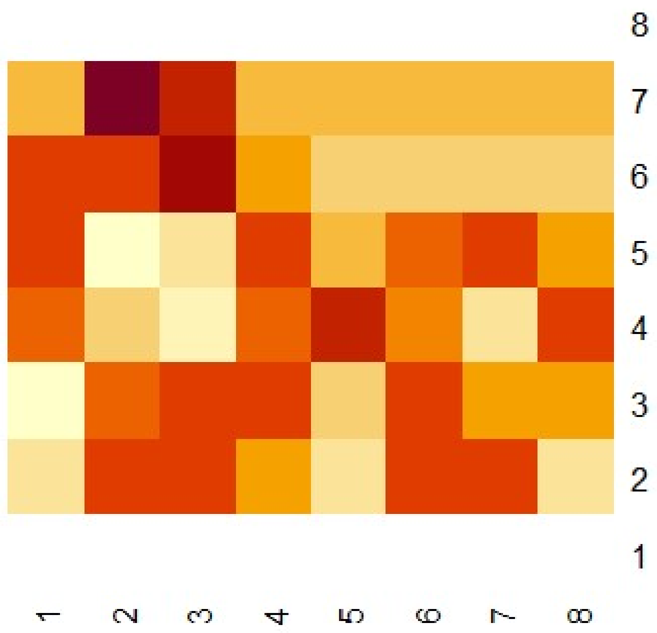
After these rows had been identified as having the lowest and highest probability of being an 8, while also being an 8, we transformed then rows into 8x8-matrices and plotted heat maps for these to determine how hard it was to determine if they were 8's or not.

```
e1matrix <- as.matrix(train[easy1, 1:64])
v <- c(e1matrix[1:64])
e1matrix <- matrix(v, nrow = 8)

e2matrix <- as.matrix(train[easy2, 1:64])
v <- c(e2matrix[1:64])
e2matrix <- matrix(v, nrow = 8)

h1matrix <- as.matrix(train[hard1, 1:64])
v <- c(h1matrix[1:64])
h1matrix <- matrix(v, nrow = 8)

h2matrix <- as.matrix(train[hard2, 1:64])
v <- c(h2matrix[1:64])
h2matrix <- matrix(v, nrow = 8)
```

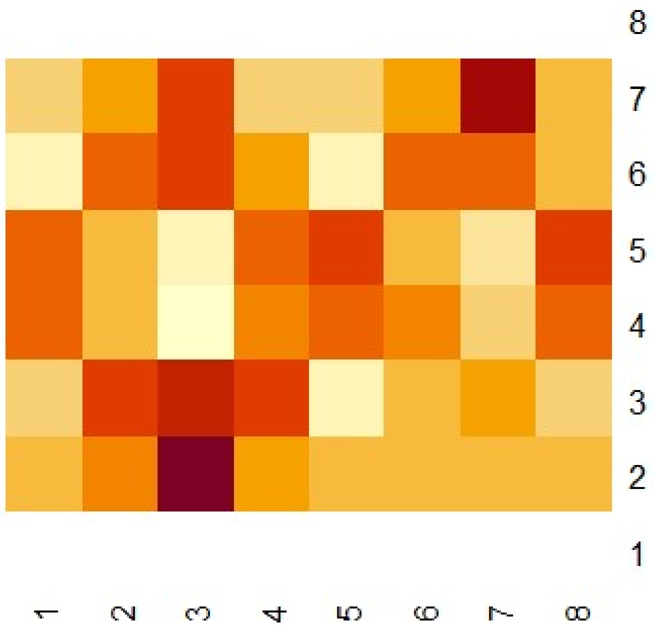
```r
h3matrix <- as.matrix(train[hard3, 1:64])
v <- c(h3matrix[1:64])
h3matrix <- matrix(v, nrow = 8)
```

From the first two heat maps showing the easy to classify 8’s, it is quite easy to recognize the patterns of a 8, albeit rotated. For the following three heat maps depicting the hard to recognize 8’s it is much harder for us to see the pattern of an 8 in the picture.

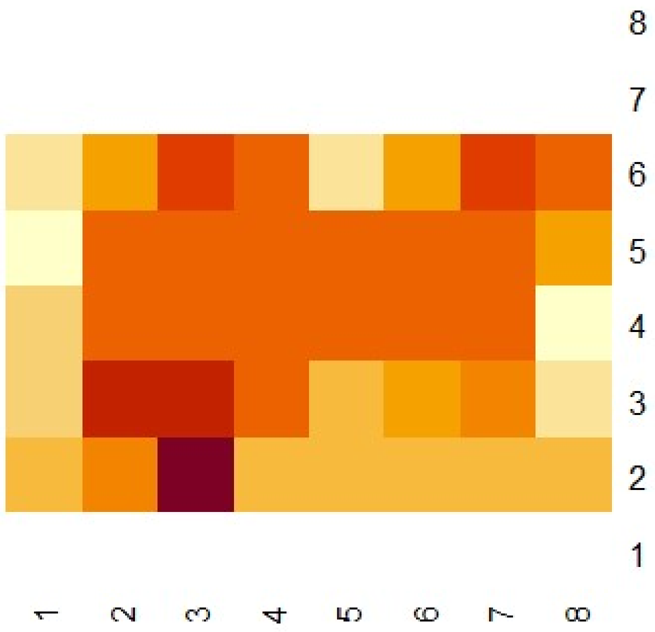
```r
heatmap(e1matrix, Colv = NA, Rowv = NA)
```



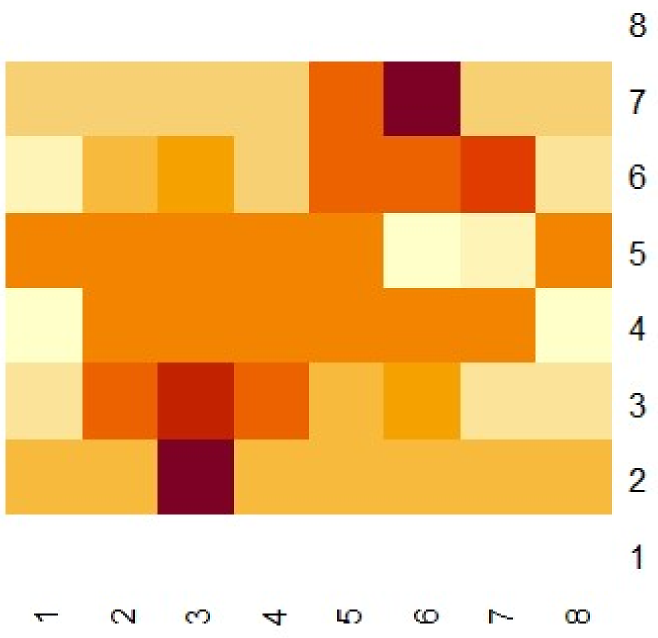
```r
heatmap(e2matrix, Colv = NA, Rowv = NA)
```



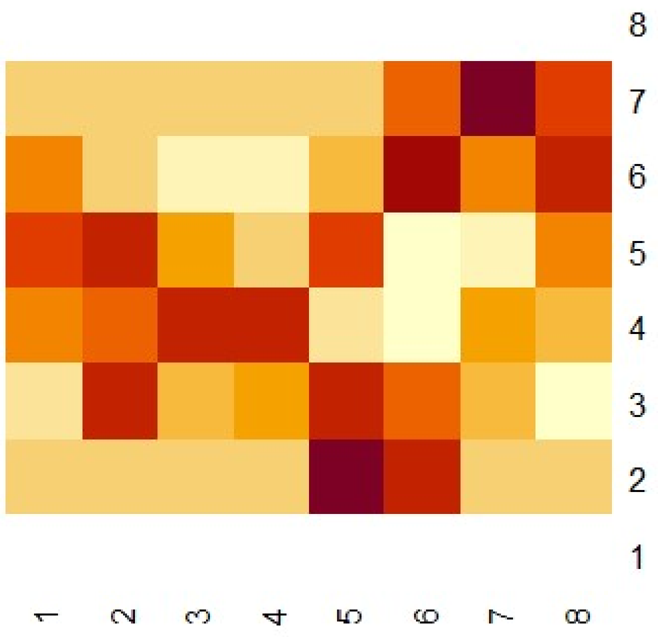
```r
heatmap(h1matrix, Colv = NA, Rowv = NA)
```



```r
heatmap(h2matrix, Colv = NA, Rowv = NA)
```



```
heatmap(h3matrix, Colv = NA, Rowv = NA)
```



Task 4 and 5

First we define a function to calculate the cross-entropy.

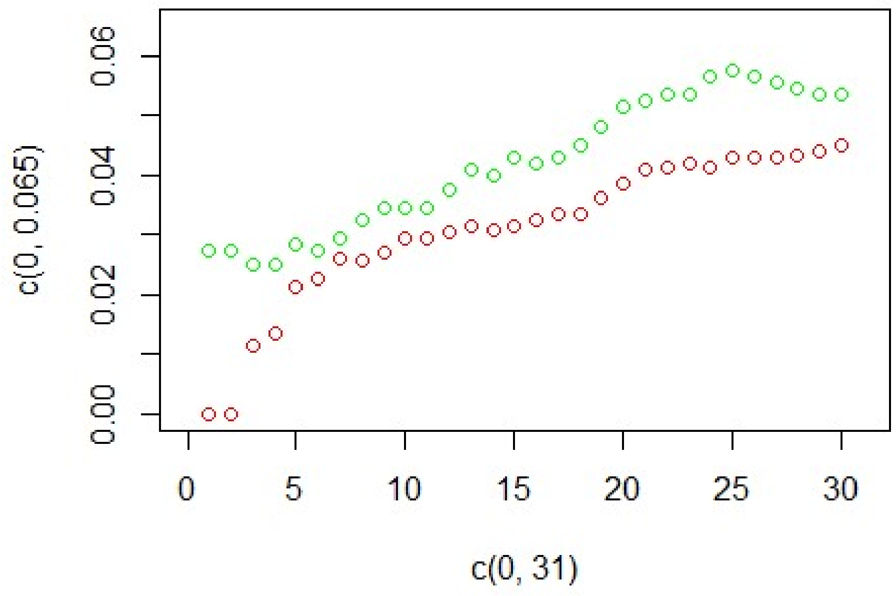
```
cross_entropy = function(x, y){
  res <- c(1:nrow(y))
  for (i in 1:nrow(y)){
    res[i] <-log(x$prob[i, (y$V65[i]+1)]+(1e-15))#x:knn_fit_loop and y:vali
d
  }
  return(res)
}
```

Then we run a loop varying the number of neighbours used to fit the model after the training set and doing predictions on the validation set. Using that fit we calculated the misclassification rate for each k=1:30 and the empiracal risk based on the cross-entropy for that fit.

```
vemp <- c(1:30)
vmissV <- c(1:30)
vmissT <- c(1:30)
vk <- c(1:30)
for (i in 1:30){ #this is something weird, maybe should use valid set in kk
nn?
  vk[i] <- i
  knn_fit_loopV <- kknn(as.factor(V65)~., train, valid, k = i, kernel = "re
ctangular")
  knn_fit_loopT <- kknn(as.factor(V65)~., train, train, k = i, kernel = "re
ctangular")
  vmissV[i] <- missclass(valid$V65, knn_fit_loopV$fitted.values)
  vmissT[i] <- missclass(train$V65, knn_fit_loopT$fitted.values)
  vemp[i] <- -1*mean(cross_entropy(knn_fit_loopV,valid))
}
```

We first get the plot for the misclassification rate against k=1:30.

```
plot(c(0,31), c(0,0.065), type = "n")
points(vk, vmissV, col = "green")
points(vk, vmissT, col = "red")
```



```
print(which.min(vmissV))

## [1] 3
```

From this plot we can observe that the misclassification rate increses with number of neighbours as the complexity of the model decreases for both models.The red plot is naturally lower since we have trained and predicted on the train set. The optimal K is 3 since it has the lowest misclassification error in the green plot.

From a Bias Variance trade-off perspective the red plot(prediction on training data) has lower bias in the model since the model is making prediction on the same data as it was trained. The model has an overall higher variance when making predictions on the validation set. Also in respect to complexity, higher complexity reflects lower bias, but it can lead to a high variance in predicting on other data sets, as can be seen in the plot for low K. The green has much higher misclassification rate on K=1-4 than the red since the variance for those K's are high, since the bias are low for those K's.

With optimal k=3 from the green plot we estimate the error with the test set.

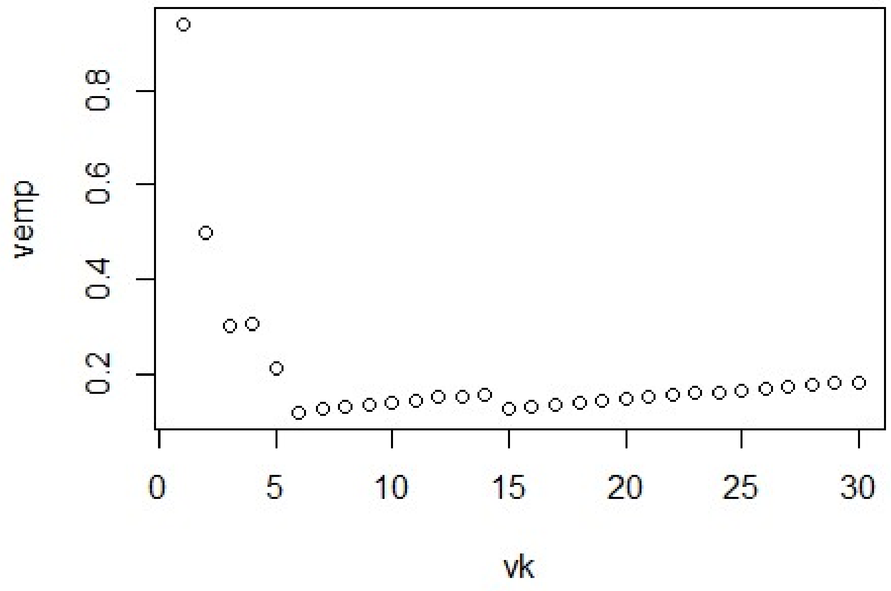
```
knn_fit_test2 <- kknn(as.factor(V65)~., train, test, k = 3, kernel = "recta
ngular")
test_error <- missclass(test$V65, knn_fit_test2$fitted.values)
print(test_error)

## [1] 0.02403344

print(vmissT[3])
```

```
## [1] 0.0115123

print(vmissV[3])

## [1] 0.02513089
```

As can be seen the error for the valid and test set are quite similair and the error for the train set is lower, as expected. The model seems to have a low bias but a higher variance when k=3. The quality of the model is better than before with a error of about 2.5% misclassification on “new” data sets.

Then we plot the empirical risk against K=1:30.

```
plot(vk, vemp)
```



```
print(which.min(vemp))

## [1] 6
```

For the plot with empirical risk gives the lowest empirical risk when K=6. The cross-entropy is better for this problem since it more accurately show the quality of our model since it considers to what probability it bases decisions on, compared to misclassifcation rate which just gives a ratio of wrong to right without taking into consideration of HOW right it’s prediction was.

Assignment 2

Responsible: Daniel Bissessar

Task 1:

From the lecture slides this is the standard model for the probabilistic model:

$$y \sim N\left(y \middle| w_o + Xw, \sigma^2 I\right)$$
$$w \sim N\left(0, \frac{\sigma^2}{\lambda} I\right)$$

Task 2:

The data is scaled, divided into test data and the columns not related to our model are removed.

```
data <- read.csv("C:/Users/Daniel/Desktop/parkinsons.csv")
data <- scale(data)
n=dim(data)[1]
set.seed(12345)
id = sample(1:n, floor(n*0.6))
train = data[id,]
train_motor = train[,5]
train = train[,-(1:6)]
test = data[-id,]
test_motor = test[,5]
test = test[,-(1:6)]
```

Task 3a)

```
loglikelihood = function(w,sigma){
  return(-length(train_motor)/2*log(sigma^2)-1/(2*sigma^2)*(sum((train_moto
r-as.matrix(train)%*%w)^2)))
  }
```

Task 3b)

```
ridge = function(w,sigma){
  return(lambda*(sum(w^2))-loglikelihood(w,sigma))
}
```

Task 3c)

```
helpridge = function(x) {
  w <- c(x[1:16])
  sigma <- c(x[17])
  return(ridge(w,sigma))
}
ridgeOpt = function(){
  return(optim(par=rep(1,17), helpridge, method="BFGS"))
}
```

Task 3d)

```
DF = function(){
  train=as.matrix(train)
  tempT <- train%*%(t(train)%*%train+lambda*diag(ncol(train)))^-1%*%t(train
)
  return(sum(diag(tempT)))

}
```

Task 4 & 5

```
AIC = function(w,sigma){
  return(2*DF()-2*loglikelihood(w,sigma))
}

lambda = 1
OptLambda1 = ridgeOpt()
Opt1Var = OptLambda1[[1]]
w1 = Opt1Var[1:16]
sigma1 = Opt1Var[17]
predTrain1 = as.matrix(train)%*%w1
predTest1 = as.matrix(test)%*%w1
MSETrain1 = sum((predTrain1-train_motor)^2)/length(predTrain1)
MSETest1 = sum((predTest1-test_motor)^2)/length(predTest1)
AIC1 <- AIC(w1,sigma1)

lambda <- 100
OptLambda100 = ridgeOpt()
Opt100Var = OptLambda100[[1]]
w100 = Opt100Var[1:16]
sigma100 = Opt100Var[17]
predTrain100 = as.matrix(train)%*%w100
predTest100 = as.matrix(test)%*%w100
MSETrain100 = sum((predTrain100-train_motor)^2)/length(predTrain100)
MSETest100 = sum((predTest100-test_motor)^2)/length(predTest100)
AIC100 <- AIC(w100,sigma100)

lambda <- 1000
OptLambda1000 = ridgeOpt()
Opt1000Var = OptLambda1000[[1]]
w1000 = Opt1000Var[1:16]
sigma1000 = Opt1000Var[17]
predTrain1000 = as.matrix(train)%*%w1000
predTest1000 = as.matrix(test)%*%w1000
MSETrain1000 = sum((predTrain1000-train_motor)^2)/length(predTrain1000)
MSETest1000 = sum((predTest1000-test_motor)^2)/length(predTest1000)
AIC1000 <- AIC(w1000,sigma1000)
```

For question 4 the best lambda value is 1 and MSE is a more appropriate measure since it takes into account both variance and bias which is good since ridge regression optimizes between variying bias and variance

For question 5 the best lambda value is also 1 AIC does not need divided data while holdout methods do

Assignment 3

Responsible: Filip Brunander

Setup of data

Data is read from an excel file and divided into train and test data (50/50).

```
csvData <- read.csv("tecator.csv", header=TRUE)

set.seed(12345)
n <- dim(csvData)[1]
id <- sample(1:n, floor(n*0.5))
trainData <- csvData[id,]
testData <- csvData[-id,]
```

Task 1

Probabilistic model:

$$p(y|x,w) = N(w_0 + w_1 x, \sigma^2)$$

```
fitLm <- lm(formula= Fat~., data=trainData)
summary(fitLm)

## Residual standard error: 5.429e-11 on 4 degrees of freedom
## Multiple R-squared:      1,  Adjusted R-squared:      1
## F-statistic: 5.593e+22 on 102 and 4 DF,  p-value: < 2.2e-16

predTrain <- predict(fitLm, newdata=trainData)

predTest <- predict(fitLm, newdata=testData)

errorTrain <- sum((trainData$Fat-predTrain)^2)/length(predTrain)
errorTest <- sum((testData$Fat-predTest)^2)/length(predTest)
print(errorTrain)

## [1] 3.455219e-18

print(errorTest)

## [1] 1546.499
```

Comment on the quality of fit and prediction and therefore the quality of the model: The two errors are displayed above, the first and lowest is the MSE for the training data and the second is the MSE for the testing data. The low MSE for the training data is no surprise, since that was the data that fitted the model from the beginning. If we compare that to the MSE of the testing error, we can see that the test that is much larger. This could perhaps be signs of overfitting, the low MSE for the training data but very high MSE for the testing data. The quality of the model is therefore rather low. ## Task 2 **We get the objective function:**

$$\hat{\beta}^{lasso} = \underset{\beta}{\operatorname{argmin}} \sum_{i=1}^{N} (y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\,\beta_j)^2$$

Subject to:

$$\sum_{j=1}^{p} |\,\beta_j| \le t$$

Here β_j is the coefficients. The x_{ij} the feature (Channels). The y_i is the Fat.

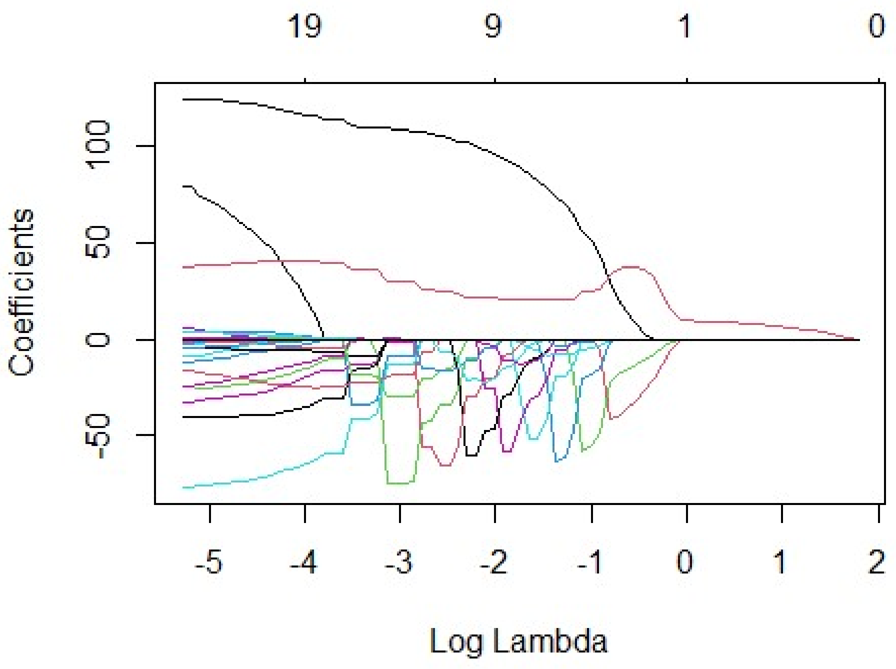
Alternatively we can display the function as:

$$\hat{\beta}^{lasso} = \underset{\beta}{\operatorname{argmin}} \{\frac{1}{2} \sum_{i=1}^{N} (y_i - \beta_0 - \sum_{j=1}^{p} x_{ij}\,\beta_j)^2 + \lambda \sum_{j=1}^{p} |\,\beta_j|\}$$

Where the penalty factor λ is displayed as well.

Task 3

```
covariates <- trainData[,2:101]
response <- trainData$Fat
fitLasso <- cv.glmnet(as.matrix(covariates), response, alpha=1, family="gau
ssian")
plot(fitLasso$glmnet.fit, xvar="lambda")
```



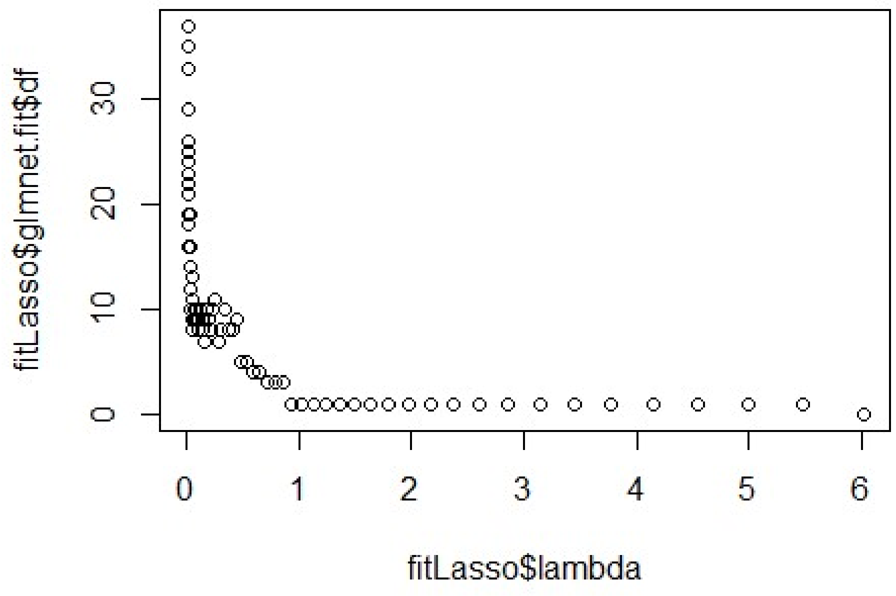
Interpret the plot: This plot displays our coefficients and how they depend on lambda. The numbers at the "Top" show how many coefficients are active at the same time. We can see how as we increase the penalty parameter lambda, the amount of "active" coefficients decreases and only the most important predictors has coefficients that is not zero. We can see the red line showing an important predictor throughout all lambdas. Further, the black line standing out also seems to be an important predictor, but only lasts until log lambda = -0.35.

What value of the penalty factor can be chosen if we want a model with three features: From the graph (we scaled it so it was easier to see breaking points), we can see that three coefficients differs from zero when log(lambda) is around -0.07 to -0.35. This is respectively lambda 0,9324 and 0,7047, which in turn is the range for our penalty factor with three features.

Task 4

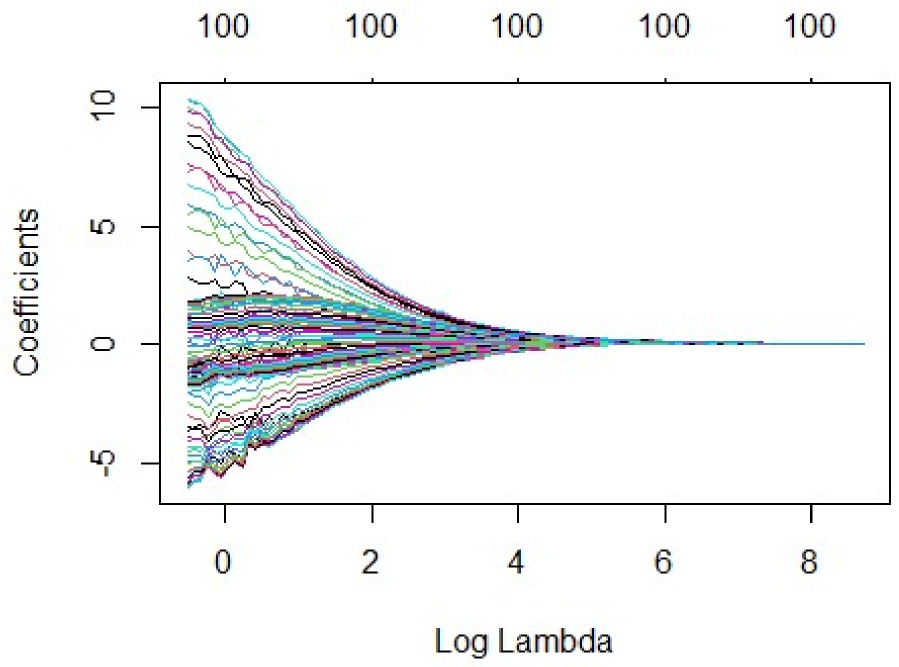
Is the observed trend expected: A decrease in degrees of freedom is expected when lambda increases, which is also displayed by the plot. Further, a lot of coefficients are “active” when log(lambda) is smaller, say around -6, where 53 coefficients are active. In this plot with the degrees of freedom, we can see how it rapidly increases when lambda comes close to zero, which is expected. Additionally, our calculations that lambda would be in the range from 0.7 to 0.93 with 3 active features does fit in well in this graph, where the degrees of freedom is free in a range that looks like 0.7 to 0.93.

```
plot(fitLasso$lambda, fitLasso$glmnet.fit$df)
```



Task 5

```
fitRidge <- cv.glmnet(as.matrix(covariates), response, alpha=0, family="gau
ssian")
plot(fitRidge$glmnet.fit, xvar="lambda")
```



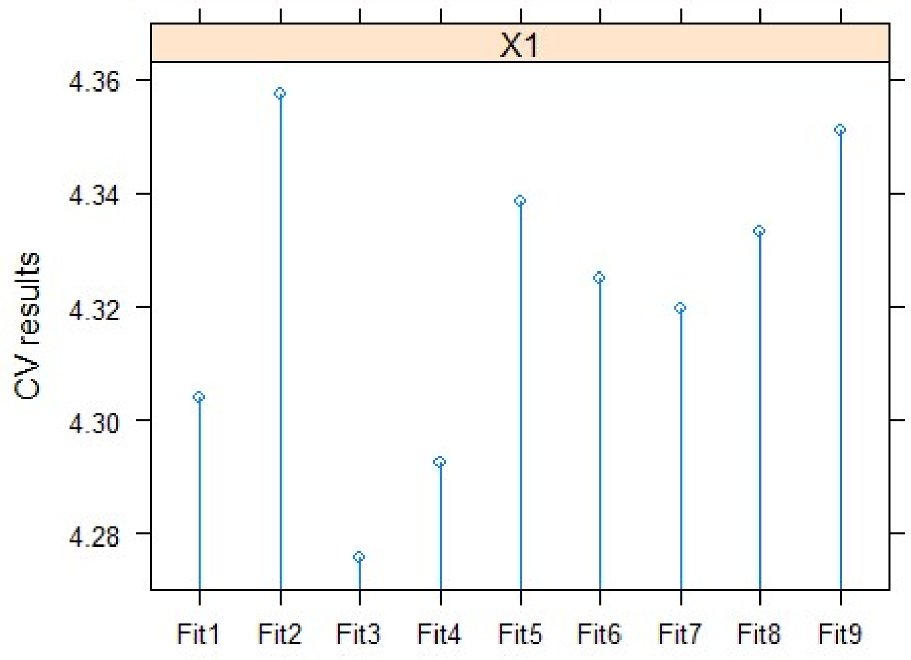
Comparison of the plot in task 3 and task 5: Here, all coefficients are active at all times (not zero), even though they come close to zero they never reach it. We can also in the ridge plot that the solutions are more compact compared to the more sparse solutions in the lasso plot.

Task 6

How K was chosen: K = 10 is a usual K that is selected. A larger K usually brings longer computation time, however, that is not really relevant in our case due to our comparable small dataset, so we could have chosen a larger K and it would also have been fine. However, with too large of a dataset, the iteration for each validation is very limited, which might not be preferrable. Low K → less variance and more bias, high K → more variance and lower bias. Further, 10 is usually in between these two cases providing rather low bias and modest variance. Lastly, 10 is a devisor of our dataset and by these points raised, K=10 was chosen.

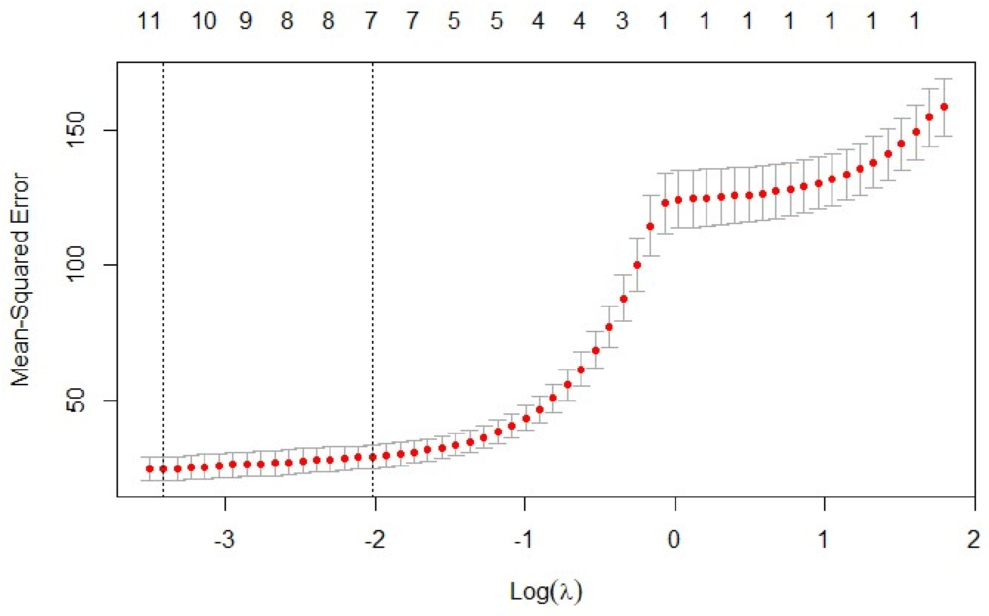
```
cv1 <- trainData[,2:21]
cv2 <- trainData[,2:31]
cv3 <- trainData[,2:41]
cv4 <- trainData[,2:51]
cv5 <- trainData[,2:61]
cv6 <- trainData[,2:71]
cv7 <- trainData[,2:81]
cv8 <- trainData[,2:91]
cv9 <- trainData[,2:101]

fit1 <- cv.glmnet(as.matrix(cv1), trainData$Fat, alpha=1, family="gaussian"
)
```

```
fit2 <- cv.glmnet(as.matrix(cv2), trainData$Fat, alpha=1, family="gaussian"
)

fit3 <- cv.glmnet(as.matrix(cv3), trainData$Fat, alpha=1, family="gaussian"
)
fit4 <- cv.glmnet(as.matrix(cv4), trainData$Fat, alpha=1, family="gaussian"
)
fit5 <- cv.glmnet(as.matrix(cv5), trainData$Fat, alpha=1, family="gaussian"
)
fit6 <- cv.glmnet(as.matrix(cv6), trainData$Fat, alpha=1, family="gaussian"
)
fit7 <- cv.glmnet(as.matrix(cv7), trainData$Fat, alpha=1, family="gaussian"
)
fit8 <- cv.glmnet(as.matrix(cv8), trainData$Fat, alpha=1, family="gaussian"
)
fit9 <- cv.glmnet(as.matrix(cv9), trainData$Fat, alpha=1, family="gaussian"
)

f1 <- cvFit(fit1, x=as.matrix(testData[,2:101]), y=testData$Fat, K=10, fold
Type="consecutive")
f2 <- cvFit(fit2, x=as.matrix(testData[,2:101]), y=testData$Fat, K=10, fold
Type="consecutive")
f3 <- cvFit(fit3, x=as.matrix(testData[,2:101]), y=testData$Fat, K=10, fold
Type="consecutive")
f4 <- cvFit(fit4, x=as.matrix(testData[,2:101]), y=testData$Fat, K=10, fold
Type="consecutive")
f5 <- cvFit(fit5, x=as.matrix(testData[,2:101]), y=testData$Fat, K=10, fold
Type="consecutive")
f6 <- cvFit(fit6, x=as.matrix(testData[,2:101]), y=testData$Fat, K=10, fold
Type="consecutive")
f7 <- cvFit(fit7, x=as.matrix(testData[,2:101]), y=testData$Fat, K=10, fold
Type="consecutive")
f8 <- cvFit(fit8, x=as.matrix(testData[,2:101]), y=testData$Fat, K=10, fold
Type="consecutive")
f9 <- cvFit(fit9, x=as.matrix(testData[,2:101]), y=testData$Fat, K=10, fold
Type="consecutive")

res=cvSelect(f1,f2,f3,f4,f5,f6,f7,f8,f9)
plot(res)
```



Fit 3 with the lowest CV score was selected and future graphs will be based on that model.

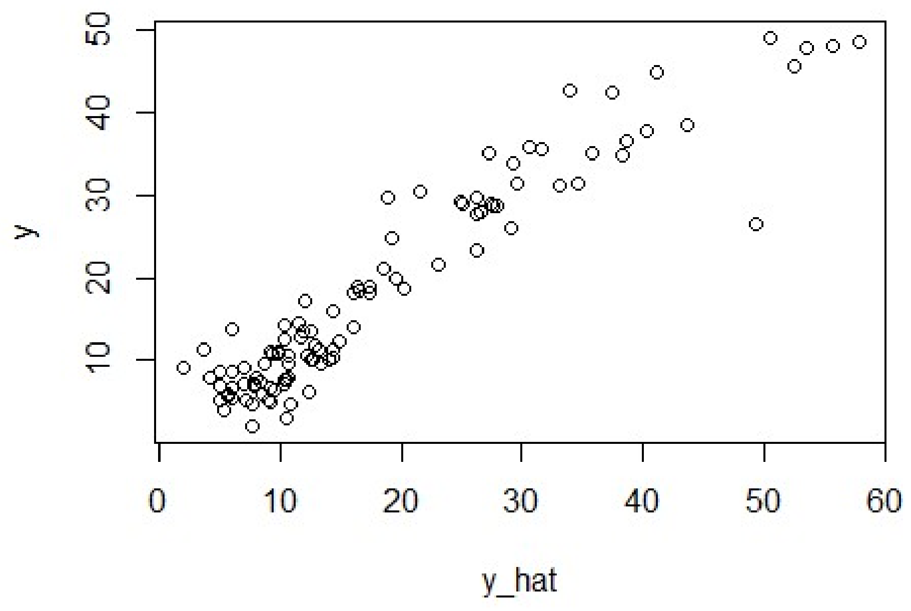
```
plot(fit3)
```



Comment on how the CV score changes with log lambda: As log lambda larger than lambda.min increases, the MSE increases. MSE could also be presented as the CV score, and thereby we see an increase in CV score with an increase in log lambda.

Comment if selected lambda is statistically significantly better than log lambda = -2: The difference between lambda.min and log lambda = -2 in MSE is roughly 5. In a dataset of over 100 observations, and the Fat ranging from values 2 to 50, the difference between MSE=25 and MSE=30 is not too significant. Therefore, log lambda min is probably statistically significantly better than log lambda -2. However, if larger errors bring much larger consequences, a difference in 5 MSE could perhaps be significant.

```
y <- testData$Fat
y_hat <- predict(fit3, newx = as.matrix(testData[,2:41], type="response"),
s = "lambda.min")
plot(y_hat, y)
```



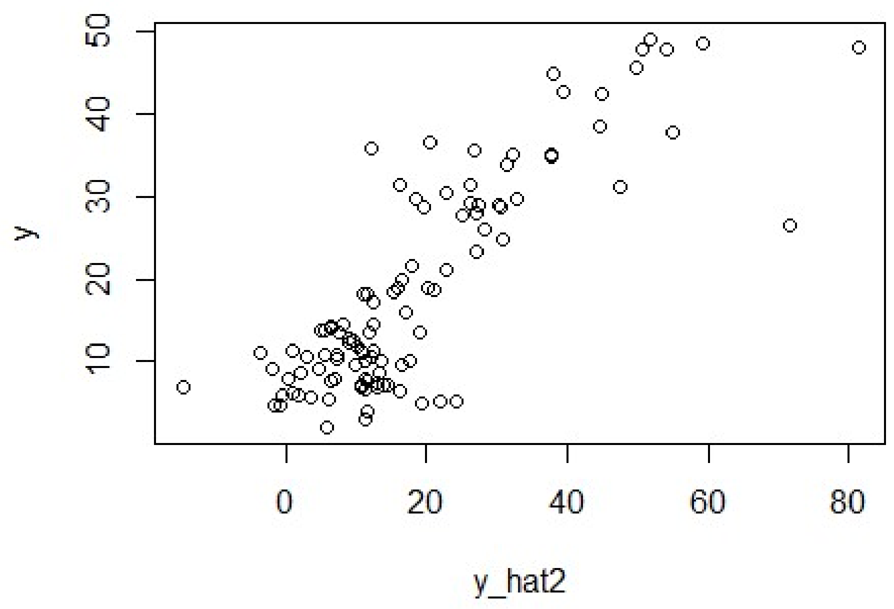
```
coef(fit3, "lambda.min")

## Channel12   -56.380247
## Channel13   -38.651796
## Channel14   -21.444809
## Channel15   -14.606595
## Channel16    -7.862910
## Channel17    -3.881859
## Channel36     5.458304
## Channel37    32.156246
## Channel38    71.210692
## Channel39    20.641761

print(fit3$lambda.min)

## [1] 0.03281783
```

Report the optimal Lambda and how many variables were chose in this model: The optimal lambda is: 0.03281783. And at that lambda, we have 10 channels active (12-17, 36-39).

Are the model predictions good?: Most of the points are along the line y=y_hat, which indicates that our model usually predicts quite good values. The points are gathered rather close to each other in almost a linear fashion, which indicates that the model is good. However, it is hard to say if it is “good” depending on how specific the the predictions has to be in order to be considered “good”, and some points are differing quite a bit, which could be an issue depending on the case. Therefore, no decisive conclusion was made, but the model seems to at least decently predict the level of fat.

Task 7

```
y_hat2 <- rnorm(mean = y_hat, n = length(y), sd = fit3$cvsd)
plot(y_hat2, y)
```



Comment on the quality of the data generation: The points could be gathered in a linear fashion, but it is hard to say. The points are quite sparse, which makes it hard to see any form of pattern, and in turn makes the relationship between y and y_hat2 hard to identify. Due to the points being sparse, the quality of the data generation is not too high, but could still be relevant since the distribution seems to be around y=y_hat2.

Appendix

Code to Assignment 1

```
#lab 1

#exercise 1

library(kknn)

#task 1

DF=read.csv(file = 'optdigits.csv', header = FALSE)


n=dim(DF)[1]

set.seed(12345)

id=sample(1:n, floor(n*0.5))

train=DF[id,]

id1=setdiff(1:n, id)

set.seed(12345)

id2=sample(id1, floor(n*0.25))

valid=DF[id2,]

id3=setdiff(id1,id2)

test=DF[id3,]


#task2

knn_fit_train <- kknn(as.factor(V65)~., train, train, k = 30, kernel = "rectangular")

train_confusion_matrix <- table(train$V65, knn_fit_train$fitted.values)

#problem false positive when reading 1(T) as 2(11) and 9(3), also 4(T) as 7(7) and 9(4),

#also 5(T) as 9(8)

#false negatives when reading 8(T) as 1(10), 3(2) or 6(2), also 9(T) as 3(5), 1,7,8:(3), 4(T) as 1(3) and
7(T) as 1(3)

#generally predicting 1 a lot when it shouldn't especially 8, and misinterpret 4 and 1 alot



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

test_confusion_matrix <- table(test$V65, knn_fit_test$fitted.values)
```

```
#there seems to be a problem with a false positive when reading 4(T) as 7(5), 9(3) and 6(2),
#and also a false negative when reading 8(T) as 1 and 9(T) as 3
#generally having trouble reading 4, predicting 1 and 3 more than it should

missclass = function(y, y_h){
  l=length(y)
  return(1-sum(diag(table(y,y_h)))/l)
}

miss_train = missclass(train$V65, knn_fit_train$fitted.values)
miss_test = missclass(test$V65, knn_fit_test$fitted.values)
#with a smaller missclassification rate in the train set, it seems better,
#but it also had more data do fit

#task 3

#find 8s, 2 easy, 3 hard:
#all probabilities for 8
prob8s <- knn_fit_train$prob[,9] #probabilities on being a 8



easy1 <- which.max(prob8s)
prob8s[easy1] <- NA
easy2 <- which.max(prob8s)
prob8s[easy2] <- NA

t <- 0
vhard <- c(0,0,0)
while (t<3) {
  temp <- which.min(prob8s)
  if(train[temp, 65] == 8){#check in slides
  vhard[t+1] <- temp
  t <- t+1
  }
  prob8s[temp] <- NA
}

hard1 <- vhard[1]
hard2 <- vhard[2]
hard3 <- vhard[3]

e1matrix <- as.matrix(train[easy1, 1:64])
v <- c(e1matrix[1:64])
e1matrix <- matrix(v, nrow = 8)

e2matrix <- as.matrix(train[easy2, 1:64])
v <- c(e2matrix[1:64])
e2matrix <- matrix(v, nrow = 8)

h1matrix <- as.matrix(train[hard1, 1:64])
v <- c(h1matrix[1:64])
h1matrix <- matrix(v, nrow = 8)

h2matrix <- as.matrix(train[hard2, 1:64])
v <- c(h2matrix[1:64])
h2matrix <- matrix(v, nrow = 8)

h3matrix <- as.matrix(train[hard3, 1:64])
v <- c(h3matrix[1:64])
h3matrix <- matrix(v, nrow = 8)

heatmap(e1matrix, Colv = NA, Rowv = NA) #it seems like a 8
```

```r
heatmap(e2matrix, Colv = NA, Rowv = NA) #it could be an 8 or a 9?

heatmap(h1matrix, Colv = NA, Rowv = NA) #These were hard to see if they were 8 or others

heatmap(h2matrix, Colv = NA, Rowv = NA)

heatmap(h3matrix, Colv = NA, Rowv = NA)


cross_entropy = function(x, y){
  res <- c(1:nrow(y))
  for (i in 1:nrow(y)){
    res[i] <- log(x$prob[i, (y$V65[i]+1)]+(1e-15))#x:knn_fit_loop and y:valid
  }
  return(res)
}
#task 4 & 5
vemp <- c(1:30)
vmissV <- c(1:30)
vmissT <- c(1:30)
vk <- c(1:30)
for (i in 1:30){ #this is something weird, maybe should use valid set in kknn?
  vk[i] <- i
  knn_fit_loopV <- kknn(as.factor(V65)~., train, valid, k = i, kernel = "rectangular")
  knn_fit_loopT <- kknn(as.factor(V65)~., train, train, k = i, kernel = "rectangular")
  vmissV[i] <- missclass(valid$V65, knn_fit_loopV$fitted.values)
  vmissT[i] <- missclass(train$V65, knn_fit_loopT$fitted.values)
  vemp[i] <- -1*mean(cross_entropy(knn_fit_loopV,valid))
}


plot(c(0,31), c(0,0.065), type = "n")
points(vk, vmissV, col = "green")
points(vk, vmissT, col = "red")


plot(vk, vemp)
```

Code to Assignment 2

```
data <- read.csv("C:/Users/Daniel/Desktop/parkinsons.csv")

data <- scale(data)

n=dim(data)[1]

set.seed(12345)

id = sample(1:n, floor(n*0.6))

train = data[id,]

train_motor = train[,5]

train = train[,-(1:6)]

test = data[-id,]

test_motor = test[,5]

test = test[,-(1:6)]


loglikelihood = function(w,sigma){

  return(-length(train_motor)/2*log(sigma^2)-1/(2*sigma^2)*(sum((train_motor-
as.matrix(train)%*%w)^2)))

  }


ridge = function(w,sigma){

  return(lambda*(sum(w^2))-loglikelihood(w,sigma))

}

helpridge = function(x) {

  w <- c(x[1:16])

  sigma <- c(x[17])

  return(ridge(w,sigma))

}

ridgeOpt = function(){

  return(optim(par=rep(1,17), helpridge, method="BFGS"))

}


DF = function(){

  train=as.matrix(train)
  tempT <- train%*%(t(train)%*%train+lambda*diag(ncol(train)))^-1%*%t(train)
  return(sum(diag(tempT)))

}

AIC = function(w,sigma){
  return(2*DF()-2*loglikelihood(w,sigma))
}

lambda = 1
OptLambda1 = ridgeOpt()
Opt1Var = OptLambda1[[1]]
w1 = Opt1Var[1:16]
sigma1 = Opt1Var[17]
predTrain1 = as.matrix(train)%*%w1
predTest1 = as.matrix(test)%*%w1
MSETrain1 = sum((predTrain1-train_motor)^2)/length(predTrain1)
MSETest1 = sum((predTest1-test_motor)^2)/length(predTest1)
AIC1 <- AIC(w1,sigma1)

lambda <- 100
OptLambda100 = ridgeOpt()
Opt100Var = OptLambda100[[1]]
w100 = Opt100Var[1:16]
sigma100 = Opt100Var[17]
predTrain100 = as.matrix(train)%*%w100
predTest100 = as.matrix(test)%*%w100
MSETrain100 = sum((predTrain100-train_motor)^2)/length(predTrain100)
MSETest100 = sum((predTest100-test_motor)^2)/length(predTest100)
AIC100 <- AIC(w100,sigma100)
```

```
lambda <- 1000

OptLambda1000 = ridgeOpt()

Opt1000Var = OptLambda1000[[1]]

w1000 = Opt1000Var[1:16]

sigma1000 = Opt1000Var[17]

predTrain1000 = as.matrix(train)%*%w1000

predTest1000 = as.matrix(test)%*%w1000

MSETrain1000 = sum((predTrain1000-train_motor)^2)/length(predTrain1000)

MSETest1000 = sum((predTest1000-test_motor)^2)/length(predTest1000)

AIC1000 <- AIC(w1000,sigma1000)


#For question 4 the best lambda value is 1 and MSE is a more appropriate measure

#since it takes into account both variance and bias which is good

#since ridge regression optimizes between variying bias and variance


#For question 4 the best lambda value is also 1

#AIC strictly selects a model that overfits which is theoretically better than MSE
```

Code to Assignment 3

```r
setwd("C:/Users/filip/OneDrive/Skrivbord/TDDE01")



library(glmnet)

library(cvTools)


csvData <- read.csv("tecator.csv", header=TRUE)


#Divide into test/training

set.seed(12345)

n <- dim(csvData)[1]

id <- sample(1:n, floor(n*0.5))

trainData <- csvData[id,]

testData <- csvData[-id,]



#Task 1

fitLm <- lm(formula= Fat~., data=trainData)

summary(fitLm)

predTrain <- predict(fitLm, newdata=trainData)

predTest <- predict(fitLm, newdata=testData)


errorTrain <- sum((trainData$Fat-predTrain)^2)/length(predTrain)

errorTest <- sum((testData$Fat-predTest)^2)/length(predTest)

print(errorTrain)

print(errorTest)


#Task 2 - See one-note


#Task 3
```

```
covariates <- trainData[,2:101]

response <- trainData$Fat

fitLasso <- cv.glmnet(as.matrix(covariates), response, alpha=1, family="gaussian")

plot(fitLasso$glmnet.fit, xvar="lambda")

plot(fitLasso$glmnet.fit, xvar="lambda", xlim=c(-0.4,-0.4))

plot(fitLasso$glmnet.fit, xvar="lambda", xlim=c(-0.1,0), ylim=c(-1,1))


#Task 4

plot(fitLasso$lambda, fitLasso$glmnet.fit$df)


#Task 5

fitRidge <- cv.glmnet(as.matrix(covariates), response, alpha=0, family="gaussian")

plot(fitRidge$glmnet.fit, xvar="lambda")


#Task 6

cv1 <- trainData[,2:21]

cv2 <- trainData[,2:31]

cv3 <- trainData[,2:41]

cv4 <- trainData[,2:51]

cv5 <- trainData[,2:61]

cv6 <- trainData[,2:71]

cv7 <- trainData[,2:81]

cv8 <- trainData[,2:91]

cv9 <- trainData[,2:101]


fit1 <- cv.glmnet(as.matrix(cv1), trainData$Fat, alpha=1, family="gaussian")

fit2 <- cv.glmnet(as.matrix(cv2), trainData$Fat, alpha=1, family="gaussian")

fit3 <- cv.glmnet(as.matrix(cv3), trainData$Fat, alpha=1, family="gaussian")

fit4 <- cv.glmnet(as.matrix(cv4), trainData$Fat, alpha=1, family="gaussian")

fit5 <- cv.glmnet(as.matrix(cv5), trainData$Fat, alpha=1, family="gaussian")

fit6 <- cv.glmnet(as.matrix(cv6), trainData$Fat, alpha=1, family="gaussian")
```

```
fit7 <- cv.glmnet(as.matrix(cv7), trainData$Fat, alpha=1, family="gaussian")

fit8 <- cv.glmnet(as.matrix(cv8), trainData$Fat, alpha=1, family="gaussian")

fit9 <- cv.glmnet(as.matrix(cv9), trainData$Fat, alpha=1, family="gaussian")


f1 <- cvFit(fit1, x=as.matrix(testData[,2:101]), y=testData$Fat, K=10, foldType="consecutive")

f2 <- cvFit(fit2, x=as.matrix(testData[,2:101]), y=testData$Fat, K=10, foldType="consecutive")

f3 <- cvFit(fit3, x=as.matrix(testData[,2:101]), y=testData$Fat, K=10, foldType="consecutive")

f4 <- cvFit(fit4, x=as.matrix(testData[,2:101]), y=testData$Fat, K=10, foldType="consecutive")

f5 <- cvFit(fit5, x=as.matrix(testData[,2:101]), y=testData$Fat, K=10, foldType="consecutive")

f6 <- cvFit(fit6, x=as.matrix(testData[,2:101]), y=testData$Fat, K=10, foldType="consecutive")

f7 <- cvFit(fit7, x=as.matrix(testData[,2:101]), y=testData$Fat, K=10, foldType="consecutive")

f8 <- cvFit(fit8, x=as.matrix(testData[,2:101]), y=testData$Fat, K=10, foldType="consecutive")

f9 <- cvFit(fit9, x=as.matrix(testData[,2:101]), y=testData$Fat, K=10, foldType="consecutive")


res=cvSelect(f1,f2,f3,f4,f5,f6,f7,f8,f9)

plot(res)

plot(fit3)

plot(fit3, xlim=c(-4.1,-2),ylim=c(0,50))


y <- testData$Fat

y_hat <- predict(fit3, newx = as.matrix(testData[,2:41], type="response"), s = "lambda.min")

plot(y_hat, y)

coef(fit3, "lambda.min")

print(fit3$lambda.min)


#Task 7

y_hat2 <- rnorm(mean = y_hat, n = length(y), sd = fit3$cvsd)

plot(y_hat2, y)
```