Lab 1 Topic 1 Machine Learning

Hugo Knape & Zahra Jalil Pour & Niklas Larsson

11/12/2020

State of contribution

Assignment1: Hugo Knape

Assignment2: Zahra Jalilpour

Assignment3: Niklas Larsson

Assignment 1. Handwritten digit recognition with K-means

1

```
digits <- read.csv("optdigits.csv")
digits$X0.26 <- as.factor(digits$X0.26)
n <-dim(digits)[1]
set.seed(12345)
id <-sample(1:n, floor(n*0.5))
train <-digits[id,]

id1 <-setdiff(1:n, id)
set.seed(12345)
id2 <-sample(id1, floor(n*0.25))
valid <-digits[id2,]

id3 <-setdiff(id1,id2)
test <-digits[id3,]
```

2

Train

```
digits_kknn_train <- kknn(X0.26~., train, train,  k = 30, kernel = "rectangular")
CM_train <- table(train$X0.26, digits_kknn_train$fitted.values)
CM_train
```

```
##
##       0   1   2   3   4   5   6   7   8   9
##   0 177   0   0   0   1   0   0   0   0   0
##   1   0 174   9   0   0   0   1   0   1   3
##   2   0   0 170   0   0   0   0   1   2   0
##   3   0   0   0 197   0   2   0   1   0   0
##   4   0   1   0   0 166   0   2   6   2   2
##   5   0   0   0   0   0 183   1   2   0  11
##   6   0   0   0   0   0   0 200   0   0   0
##   7   0   1   0   1   0   1   0 192   0   0
##   8   0  10   0   1   0   0   2   0 190   2
##   9   0   3   0   4   2   0   0   2   4 181
```

```
accuracy_train <- (sum(diag(CM_train)))/sum(CM_train)
accuracy_train
```

```
## [1] 0.9576138
```

```
miss_train <- 1-accuracy_train
miss_train
```

```
## [1] 0.04238619
```

We can see that we had very easy to classify digit 0 but had some problem with 1. 10 of the observations 1 were classified as instead as a 8. For digit 9, 11 observations were classified as a 5 instead. The overall prediction quality for the train data was good and were around 95.7 percent. The overall missclassiftion errors for the train data was around 4 percent.

Test

```
digits_kknn_test <- kknn(X0.26~., train, test,  k = 30, kernel = "rectangular")
CM_test <- table(test$X0.26, digits_kknn_test$fitted.values)
CM_test
```

```
##
##      0  1  2  3  4  5  6  7  8  9
##   0 97  0  0  0  0  0  1  0  0  0
##   1  0 91  3  0  0  0  0  0  0  3
##   2  0  0 93  1  0  0  0  0  1  0
##   3  0  0  0 95  0  0  0  2  1  0
##   4  1  0  0  0 89  0  1  5  1  3
##   5  0  1  0  1  0 79  1  0  0  5
##   6  0  0  0  0  0  0 94  0  0  0
##   7  0  2  0  0  0  1  0 91  1  0
##   8  0  3  0  1  0  0  1  0 86  0
##   9  0  0  0  4  0  0  0  2  1 94
```

```
accuracy_test <- (sum(diag(CM_test)))/sum(CM_test)
accuracy_test
```

```
## [1] 0.9508368
```

```
miss_test <- 1-accuracy_test
miss_test
```

```
## [1] 0.04916318
```

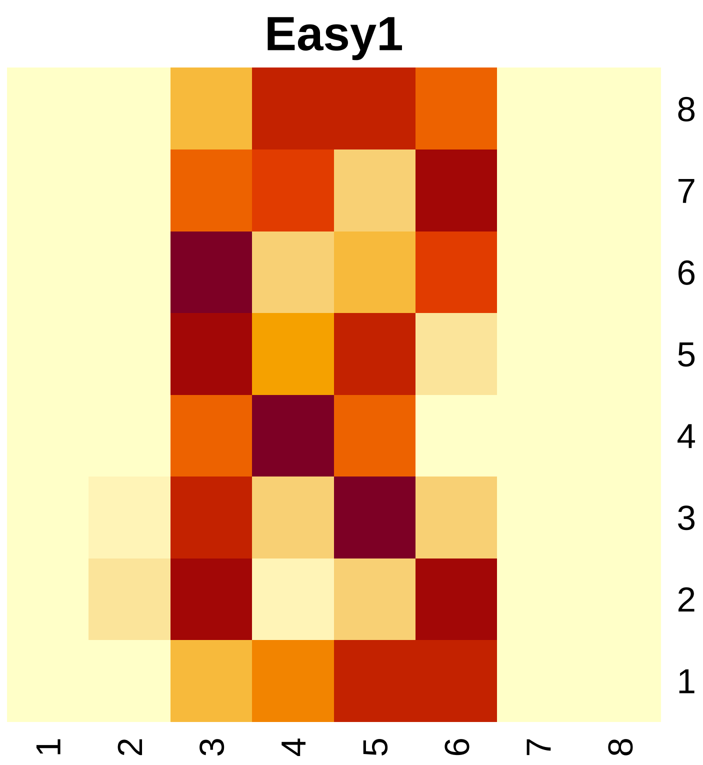
We can see that we had very easy to classify digit 0 for the test data with only one missclassified observation but had som problem with digit 9. Where some observaations where predicted as digit 1,4 and 5. The overall missclassifition errors for the train data was around 5 percent and prediction around 95.

3

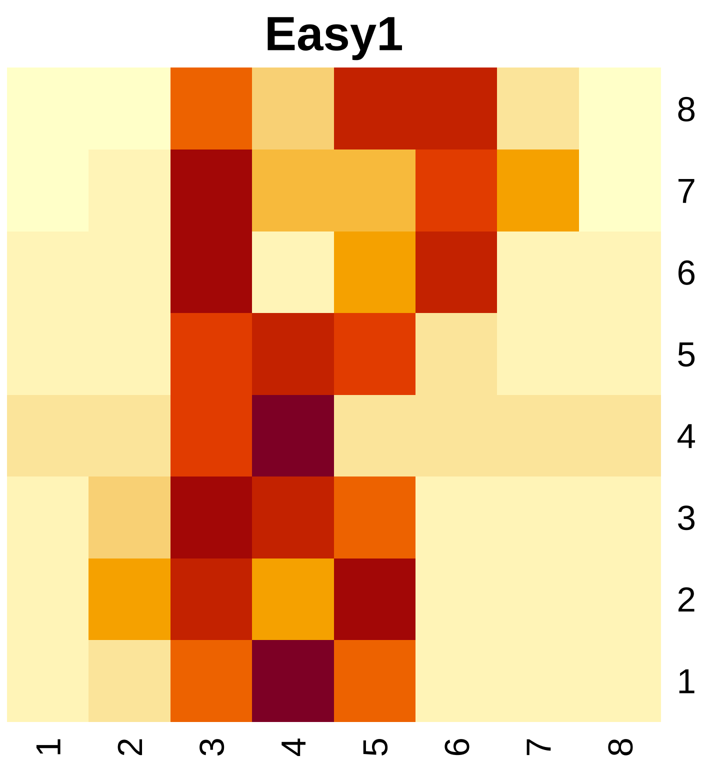
```
probs <- as.data.frame(digits_kknn_train$prob)
probs$digit <- train$X0.26
probs_8 <- probs[probs$digit==8,]
```

Easisest to classify

```
easy_probs_8 <- probs_8[probs_8$`8` == 1,]
easy_rows <- as.numeric(row.names(easy_probs_8))
easy_1 <-  sample(easy_rows, 1)
easy_2 <-  sample(easy_rows, 1)

map_easy_1 <- as.numeric(train[easy_1,-65])
map_m_easy_1 <- matrix((map_easy_1), nrow = 8, ncol = 8, byrow = TRUE)
m_easy1 <- map_m_easy_1[rev(1:8),]

map_easy_2 <- as.numeric(train[easy_2,-65])
map_m_easy_2 <- matrix((map_easy_2), nrow = 8, ncol = 8, byrow = TRUE)
m_easy2 <- map_m_easy_2[rev(1:8),]
heatmap(m_easy1, Colv=NA , Rowv=NA, main = "Easy1")
```



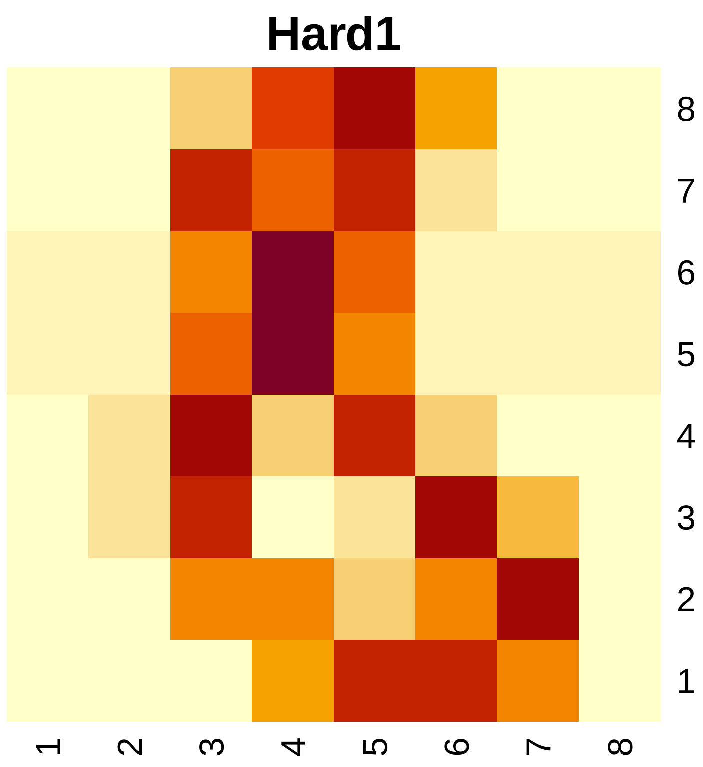
```r
heatmap(m_easy2, Colv=NA , Rowv=NA, main = "Easy1")
```



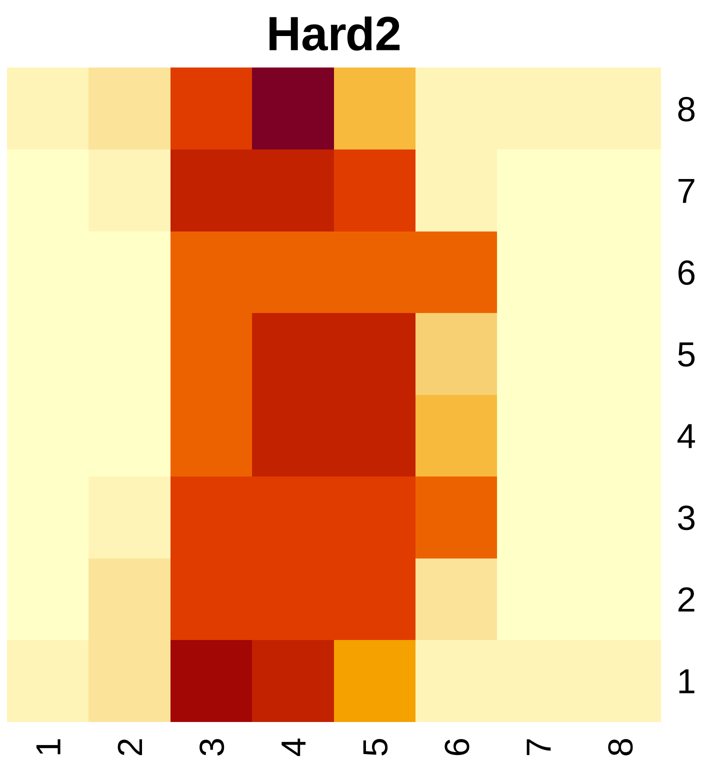
The first “easy” picture to classify seems to be pretty easy to visually classify as an 8 because we can see that is seems to be 2 holes in the digit. The second “easy” picture to classify seems to be pretty easy to visually classify as an 8 because it doesn’t look like an 8.

Hardest to classify

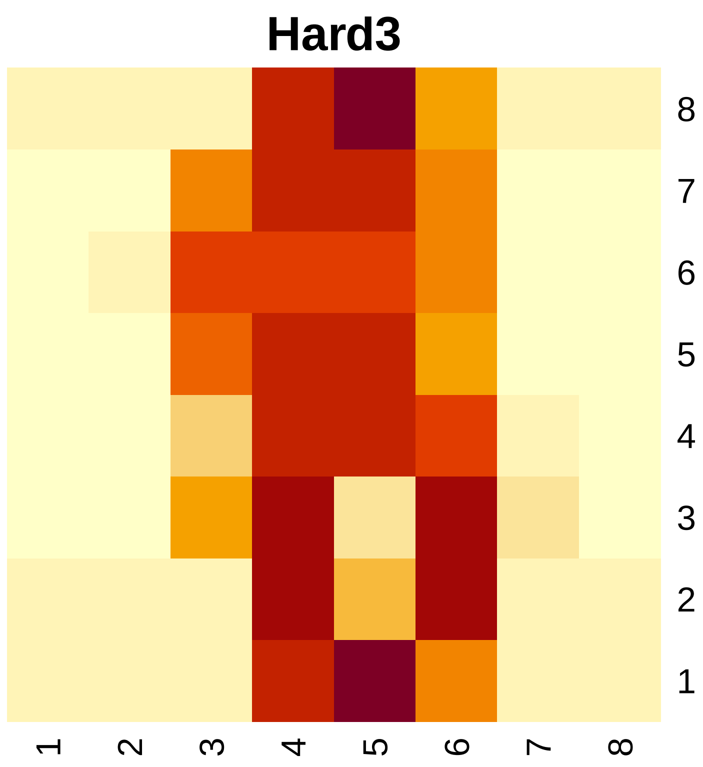
```
hard_probs_8 <- probs_8[order(probs_8$`8`),]
hard_rows <- as.numeric(row.names(hard_probs_8))
hard_1 <- hard_rows[1]
hard_2 <- hard_rows[2]
hard_3 <- hard_rows[3]

map_hard_1 <- as.numeric(train[hard_1,-65])
map_m_hard_1 <- matrix((map_hard_1), nrow = 8, ncol = 8, byrow = TRUE)
m_hard_1 <- map_m_hard_1[rev(1:8),]

map_hard_2 <- as.numeric(train[hard_2,-65])
map_m_hard_2 <- matrix((map_hard_2), nrow = 8, ncol = 8, byrow = TRUE)
m_hard_2 <- map_m_hard_2[rev(1:8),]

map_hard_3 <- as.numeric(train[hard_3,-65])
map_m_hard_3 <- matrix((map_hard_3), nrow = 8, ncol = 8, byrow = TRUE)
m_hard_3 <- map_m_hard_3[rev(1:8),]
heatmap(m_hard_1, Colv=NA , Rowv=NA, main = "Hard1")
```



```r
heatmap(m_hard_2, Colv=NA , Rowv=NA, main = "Hard2")
```



```r
heatmap(m_hard_3, Colv=NA , Rowv=NA, main = "Hard3")
```

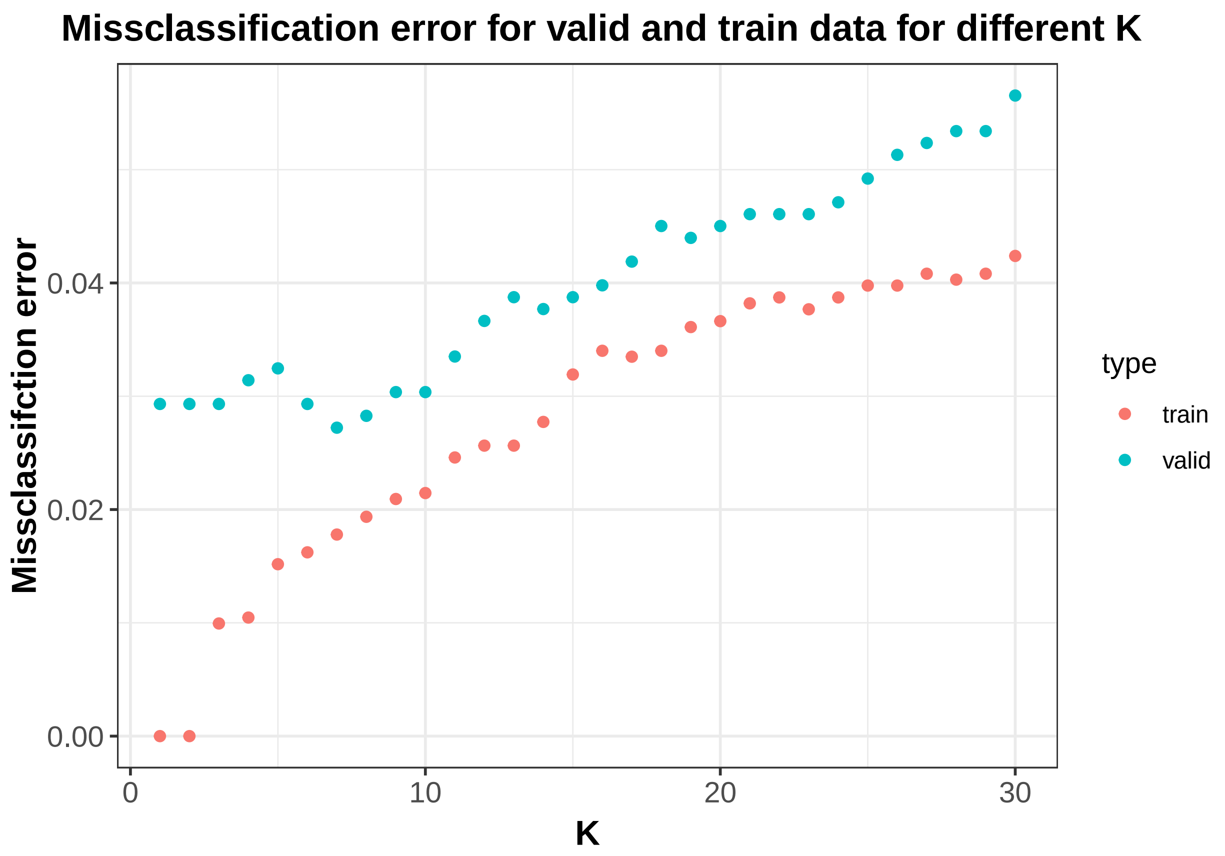


The 3 “hard” pictures to classify are all very hard to visually classify as an 8.

4

```
miss_train <- c()
for (i in 1:30) {
  model <- kknn(X0.26~., train, train,  k = i, kernel = "rectangular")
  CM <- table(train$X0.26, model$fitted.values)
  accuracy <- (sum(diag(CM)))/sum(CM)
  miss <- 1 - accuracy
  miss_train <- c(miss_train, miss)
}

df_train <- data.frame(miss = miss_train, i = 1:30, type = rep("train", 30))

miss_valid <- c()
for (i in 1:30) {
  model <- kknn(X0.26~., train, valid,  k = i, kernel = "rectangular")
  CM <- table(valid$X0.26, model$fitted.values)
  accuracy <- (sum(diag(CM)))/sum(CM)
  miss <- 1 - accuracy
  miss_valid <- c(miss_valid, miss)
}
```

```r
df_valid <- data.frame(miss = miss_valid, i = 1:30,  type = rep("valid", 30))
df <- rbind(df_train,df_valid)

ggplot(df, aes(x=i, y = miss, color = type)) +
  geom_point()  + theme_bw() +
  labs(title = "Missclassification error for valid and train data for different K"
      , x = "K" , y = "Missclassifction error")  +
  theme(axis.title.y = element_text(vjust = 0.5, size = 13 , face = "bold")) +
  theme(axis.title.x = element_text(vjust = 0.5 ,size = 13 , face = "bold")) +
  theme(plot.title = element_text(size = 14, face = "bold" , hjust = 0.4 )) +
  theme(axis.text.y = element_text(size = 11)) +
  theme(axis.text.x = element_text(size = 11))
```



Optimal K is different for train and valid. But we don't want to overfit the data as we do for example with k = 1,2. We want the valid error to be as small as possible and that's why we choose the optimal k as 7. If we have high bias the model take very little consideration to the training data and simplify it to much. If we on the other hand have high variance we take to much consideration to the training data and don't generalize it. The graph shows that when K is smaller then 7 we can see that we have overfitting(bias is low and variance high) because the missclassification error is much lesser compared to valid. When K is higher then 7 we can see that graph as underfitting(the variance is low and the bias is high) because we have a higher missclassification for both train and valid compared to when K = 7.

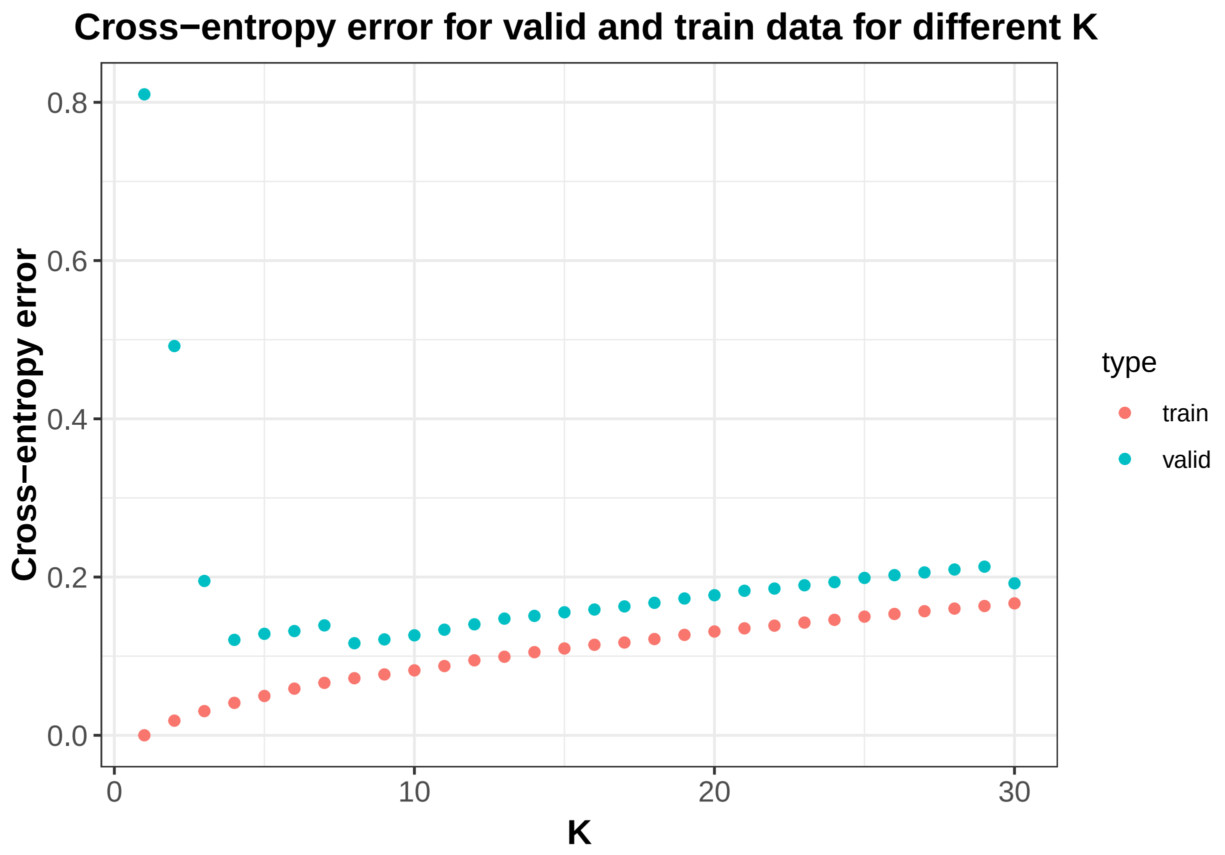
```r
model_test <- kknn(X0.26~., train, test,  k = 7, kernel = "rectangular")
CM <- table(test$X0.26, model_test$fitted.values)
accuracy <- (sum(diag(CM)))/sum(CM)
miss <- 1 - accuracy
```

```
miss_df <- data.frame( "test" = miss, "Valid" = miss_valid[7], "Train" = miss_train[7])
miss_df
```

```
##         test      Valid      Train
## 1 0.03870293 0.02722513 0.01779173
```

If we compare the test error for all different data sets with K = 7 we can se that the lowest test error has the Train data and the highest error has the test data. Even if the test and train error differs almost 2 percent we would say that the model quailty is good when the test error only is around 3.8 percent. ## 5

```
#Function for cross entropy
cross.entropy <- function(p, q){
  entropy <- 0
  row <- 1
  for (label in q) {
    entropy = - log(p[row,label] + 0.000000000001) + entropy
    row <- row + 1
  }
  m_entropy <- entropy / length(q)
  return(m_entropy)
}

#Train

cross_train <- c()
for(i in 1:30){
  model_train <- kknn(X0.26~., train, train, k = i, kernel = "rectangular")
  cross_t <- cross.entropy(model_train$prob, train$X0.26)
  cross_train <- c(cross_train, cross_t)
}
df_1_train <- data.frame(cross = cross_train, i = 1:30, type = rep("train", 30))

#Valid

cross_valid <- c()
for(j in 1:30){
  model_valid <- kknn(X0.26~., train, valid, k = j, kernel = "rectangular")
  cross_v <- cross.entropy(model_valid$prob, valid$X0.26)
  cross_valid <- c(cross_valid,cross_v )
}
df_1_valid <- data.frame(cross = (cross_valid), i = 1:30, type = rep("valid", 30))
df_new <- rbind(df_1_train, df_1_valid)
ggplot(df_new, aes(x=i, y = cross, color = type)) +
  geom_point() + theme_bw() +
  labs(title = "Cross-entropy error for valid and train data for different K"
       , x = "K" , y = "Cross-entropy error") +
  theme(axis.title.y = element_text(vjust = 0.5, size = 13 , face = "bold")) +
  theme(axis.title.x = element_text(vjust = 0.5 ,size = 13 , face = "bold")) +
  theme(plot.title = element_text(size = 14, face = "bold" , hjust = 0.4 )) +
  theme(axis.text.y = element_text(size = 11)) +
  theme(axis.text.x = element_text(size = 11))
```



The figure shows that the optimal K seems to be when valid has the lowest cross-entropy error which is when k = 8. Why cross-entropy can be a more suitable choice of the empirical risk function than the missclassification error is because cross-entropy looks for the probability of every single observations and for every digit instead of the missclassification error that only looks if it classified right or wrong.

Assignment 2. Ridge regression and model selection

1

In Bayes rules:

$$Posterior \sim Prior\ Probability *\ Likelihood$$

or

$$p(w|D) \sim p(w).p(D|w)$$

Where w is the model parameters, D is the observed data, p(w) is the prior distribution on the model parameters, p(D|w)is the likelihood of the data, and p(w|D) is the posterior probability.

Ridge regression is a special form of Bayesian linear regression with constant σ^2. A probabilistic model for target variable(motor_UPDRS) is as below:

$$motor - UPDRS = y \sim N(y|w_0 + Xw, \sigma^2.I)$$
$$w \sim N(0, \sigma^2 I/\lambda)$$

The ridge prior corresponds to normal priors centered around 0 on the regression coefficients.

$$Ridge\ prior\ p(w) \sim Normal(\ 0, \frac{\sigma^2}{\lambda}) = \frac{1}{\sqrt{2\pi\frac{\sigma^2}{\lambda}}} e^{-\frac{(w)^2}{2\frac{\sigma^2}{\lambda}}}$$

p(D|w) is a product of independent likelihoods for each observation(x, y)

$$p(D|w) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y-\ w^t X_i)^2}{2\sigma^2}}$$

$$p(w|D) = \frac{1}{\sqrt{2\pi\frac{\sigma^2}{\lambda}}} e^{-\frac{(w)^2}{2\frac{\sigma^2}{\lambda}}} * \frac{1}{(\sqrt{2\pi\sigma^2})^n} e^{-\sum_{i=1}^{n} \frac{(y_i-\ w^t X_i)^2}{2\sigma^2}}$$

2

```r
dt <- read.csv("parkinsons.csv")
dt <- dt[c(-1:-4, -6)]

## scale the data and divide to train and test
scaled.dt <- scale(dt)
m<- nrow(scaled.dt)
set.seed(12345)
id <- sample(1:m, floor(m*0.6))
train <- scaled.dt[id,]
test <- scaled.dt[-id,]
 # for train
x<- train[,2:17]
y<- train[,1]

# for test
x_test <- test[,2:17]
y_test <- test[,1]
```

3

3-a

For log likelihood function:

$$Ln(Posterior) = Ln(\ Prior\ Probability *\ Likelihood)$$

$$Ln(Posterior) = Ln(\ Prior\ Probability) + Ln(\ Likelihood)$$

$$lnL(p(w|D)) = -\frac{n}{2} ln(2\pi\sigma^2) - \sum_{i=1}^{n} \frac{(y_i - w^t X_i)^2}{2\sigma^2}$$

```r
Loglikelihood <- function(w, sigma,y,x){
  n <- nrow(x)
  #y <- train[, 1]
  w<- as.matrix(w)
  log_l<- -0.5*n*(log(2*pi))-(0.5*n*log(sigma^2))-(1/(2*sigma^2))*(sum((y-x %*% w)^2))
  return(log_l)
}
```

3-b

In Ridge Regression , shrink the coefficients to make model less complex.

$$minimize - logLikelihood + \lambda_0||w||_2^2$$

Where λ adds a penalty and reduces the slop of the line. Ridge function $\sim$ log Ridge prior, where $\tau = \frac{\sigma^2}{\lambda}$:

$$log(prior) = -\frac{1}{2}log(2\pi\tau) - \frac{(w)^2}{2\tau}$$

function returns log Ridge penalty added to the minus log-likelihood

```r
Ridge <- function(w,sigma,lambda,y,x){
  y <- as.matrix(y)
  x<- as.matrix(x)

  ridge_fun<- -Loglikelihood(w,sigma,y,x)+lambda*sum(w^2)
  return(ridge_fun)

}
```

3-c

Create RidgeOpt function to return the optimal value of w andσ by using of function in 2-b

```r
RidgeOpt <- function(R,lambda,y,x)
  {
  y <- as.matrix(y)
  x<- as.matrix(x)
    new_ridge<-function(R,lambda,y,x)
      {
        out2<-Ridge(w=R[1:16],sigma=R[17],lambda,y,x)
        return(out2)
        }
    optimize<- optim(par=R, fn=new_ridge,lambda=lambda,y=y, x=x, gr=NULL,method="BFGS")
    return(optimize)

}
```

3-d

DF function return degrees of freedom of regression model in step1 We should find trace of Hat Matrix of:

$$X(X^tX + \lambda I)^{-1}X^t$$

```r
DF <- function(lambda){
  mat <- as.matrix(train[ ,2:17])
  ## Calculate trace of Hat Matrix
  H <- mat %*% (solve((t(mat) %*% mat) + (lambda * diag(16)))%*% t(mat))  ### solve() calculates the in
  traceMat <- sum(diag(H))
  return(traceMat)

}
```

```r
DF(1)
```

```
## [1] 13.86281
```

```r
DF(100)
```

```
## [1] 9.939085
```

```r
DF(1000)
```

```
## [1] 5.643351
```

4

Now we should report training and test MSE values:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(e_i)^2 = \frac{1}{n}e^t.e$$

$$e_i = Y_i - Y_i^{hat}$$

```r
R<-rep(1,17)
out1<- RidgeOpt(R, lambda=1,y=y, x=x)
out2<- RidgeOpt(R, lambda=100,y=y, x=x)
out3<- RidgeOpt(R, lambda=1000,y=y, x=x)

#lambda=1
#x<- as.matrix(x)
pred_y1<-x %*% out1$par[-17]
MSE1<- mean((y-pred_y1)^2)

#lambda=100
pred_y2<-x %*% out2$par[-17]
MSE2<- mean((y-pred_y2)^2)

#lambda=1000
pred_y3<-x %*% out3$par[-17]
MSE3<- mean((y-pred_y3)^2)

# For test data:
#lambda=1
pred_y_test1<-x_test %*% out1$par[-17]
MSE_test1<- mean((y_test - pred_y_test1)^2)

#lambda=100
pred_y_test2<-x_test %*% out2$par[-17]
MSE_test2<- mean((y_test - pred_y_test2)^2)

#lambda=1000
pred_y_test3<-x_test %*% out3$par[-17]
MSE_test3<- mean((y_test - pred_y_test3)^2)
prediction <- data.frame(predY_training = head(c(pred_y1, pred_y2, pred_y3)), predY_testing = head(c(pr
prediction
```

```
##   predY_training predY_testing
## 1     -0.1416234    0.45336095
## 2      0.1939502    0.36701743
## 3      0.1612918    0.30948134
## 4     -0.3929647    0.45600592
## 5     -0.6716758    0.21571619
## 6      0.1939838   -0.07040793
```

```
optimal_val<- matrix(c(MSE1, MSE2, MSE3,MSE_test1, MSE_test2, MSE_test3),ncol=2)
rownames(optimal_val)<- c("lambda=1", "lambda=100","lambda=1000")
colnames(optimal_val)<-c("MSE_train", "MSE_test")
optimal_val
```

```
##             MSE_train  MSE_test
## lambda=1    0.8732770 0.9290357
## lambda=100  0.8790599 0.9263172
## lambda=1000 0.9156267 0.9479166
```

In penalty parameter(Lambda = 1), we can see the value of MSE in training set is lower than other penalty parameters. By minimizing MSE we can maximize the probability.The prior is a distribution we have to choose based on assumptions outside of our data. Minimizing mean squared error maximizes the likelihood of the parameters. In short, we have found the maximum likelihood estimator (MLE). Hence MSE is a more appropriate measure here than other empirical risk functions. ## 5

```
AIC <- function(w, sigma, y, x, lambda)
{
step1 <- -2*Loglikelihood(w, sigma, y, x) + 2*DF(lambda)
return(step1)
}
# Training Data
AIC1 <- AIC(w=out1$par[-17], sigma = out1$par[17], y=y, x=x, lambda = 1)
AIC100 <- AIC(w=out2$par[-17], sigma = out2$par[17], y=y, x=x, lambda = 100)
AIC1000 <- AIC(w=out3$par[-17], sigma = out3$par[17], y=y, x=x, lambda = 1000)

# Test Data
AIC_test1 <- AIC(w=out1$par[-17], sigma = out1$par[17], y=y_test, x=x_test, lambda = 1)
AIC_test100 <- AIC(w=out2$par[-17], sigma = out2$par[17], y=y_test, x=x_test, lambda = 100)
AIC_test1000 <- AIC(w=out3$par[-17], sigma = out3$par[17], y=y_test, x=x_test, lambda = 1000)

AIcscore<- matrix(c(AIC1,AIC100,AIC1000,AIC_test1,AIC_test100,AIC_test1000), ncol=2)
rownames(AIcscore)<- c("lambda=1", "lambda=100","lambda=1000")
colnames(AIcscore)<- c("AIcscore_train", "AIcscore_test")
AIcscore
```

```
##             AIcscore_train AIcscore_test
## lambda=1          9553.596      6528.354
## lambda=100        9569.014      6512.300
## lambda=1000       9704.087      6556.028
```

Akaike information criterion (AIC) is a fined technique based on in-sample fit to estimate the likelihood of a model to predict/estimate the future values.

A good model is the one that has minimum AIC among all the other models. Here AIC by Lambda = 1 is the lowest AIC.

In holdout model selection, we should have test data and train data that are smaller than our original data. So these two random samples may be so noisier, and for plenty of data it is better to choose holdout model selection. But here , selecting model bi minimize AIC is best.

Assignment 3. Linear regression and LASSO

Load data

```
data = read.csv('tecator.csv')
data = data[,2:102] # Pre-processing


# Split data 50/50

n = dim(data)[1]
set.seed(12345)
id = sample(1:n, floor(n*0.5))
train = data[id,]
test = data[-id,]
```

Task 1

Fit with linear regression

The distribution of underlying model

$$y(w^T x, \sigma^2) \rightarrow Fat(w^T x, \sigma^2)$$

where

$$w = \{w_0, w_1 ... w_{100}\}, x = \{1, Channel_1, Channel_2 ... Channel_{100}\}$$

The RMSE scores between train and test data is quite similar which indicates fairly good model as there seems to be neither over or under fitting. The score magnitude could on the other hand been lower.

```
lm_model1 = lm(formula = Fat ~ ., data = as.data.frame(train))
#summary(lm_model1)

print("Choose Channel63 due to being highly significant!", quote = FALSE)
```

```
## [1] Choose Channel63 due to being highly significant!
```

```
fit1 = lm(Fat ~ Channel63, as.data.frame(train));

pred_train = predict(fit1, as.data.frame(train))
pred_test = predict(fit1, as.data.frame(test))

print("RMSE score for test-prediction:", quote = FALSE)
```

```
## [1] RMSE score for test-prediction:
```

```r
(rmse_test= sqrt(mean((pred_test - test$Fat)^2)))
```

```
## [1] 11.45302
```

```r
print("RMSE score for train-prediction:", quote = FALSE)
```

```
## [1] RMSE score for train-prediction:
```

```r
(rmse_train = sqrt(mean((pred_train - train$Fat)^2)))
```

```
## [1] 11.63363
```

Task 2

The objective function which should be minimized is

$$\sum_{i=1}^{n}(Fat_i - w_i * Channel_i)^2 + \lambda\sum_{i=1}^{n}(|w_i|)$$

The λ is the penalty which "decided" which features to keep, depending on how much they contribute.

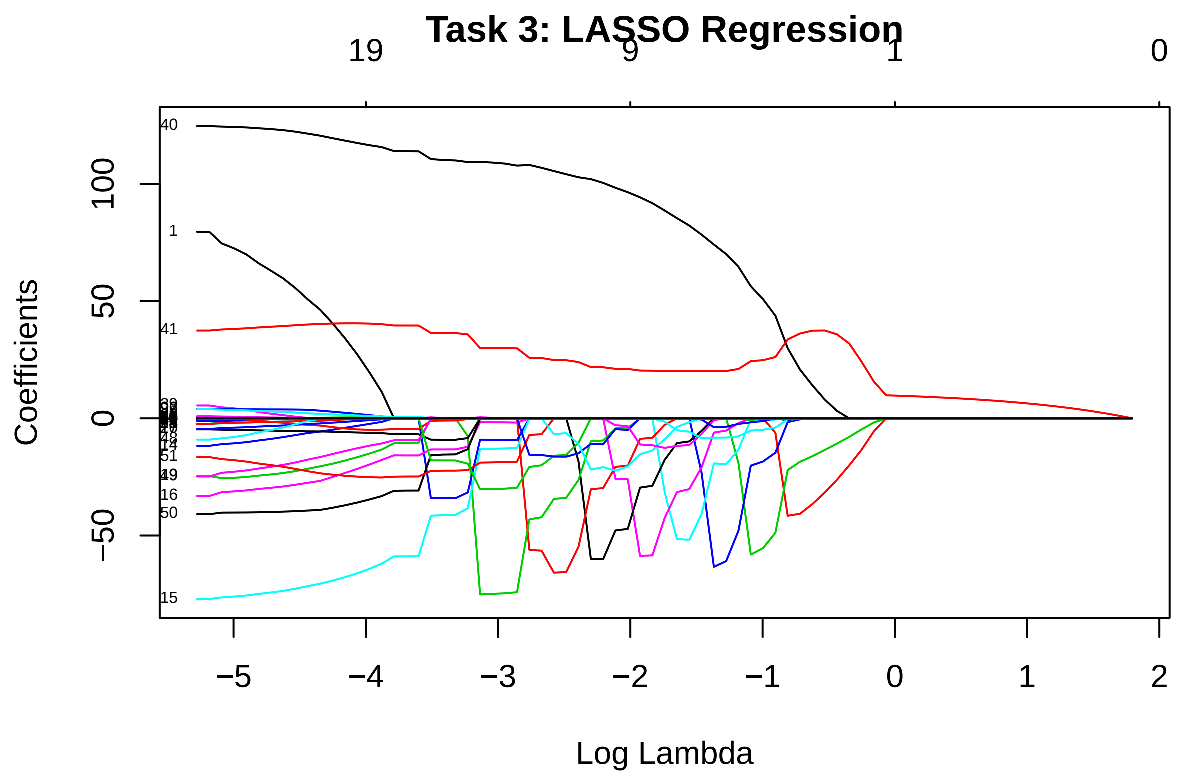
Task 3

As seen in the $Log(\lambda)$-plot where every line corresponds to one coefficient. The number of coefficients decreases as the penalty is increasing. The number of coefficients represents the degree of freedom and by looking at how many degrees of freedom each λ has one will find a suitable penalty cost.

```r
library(glmnet)
```

```
## Loading required package: Matrix

## Loaded glmnet 4.0-2
```

```r
fit_lasso = glmnet(as.matrix(train[,1:100]), train$Fat, alpha = 1.0,
                      family = "gaussian")

plot(fit_lasso, xvar = "lambda", label = TRUE, main = "Task 3: LASSO Regression")
```



```
print("Choosing lambda below gives only three features:", quote = FALSE)
```

```
## [1] Choosing lambda below gives only three features:
```

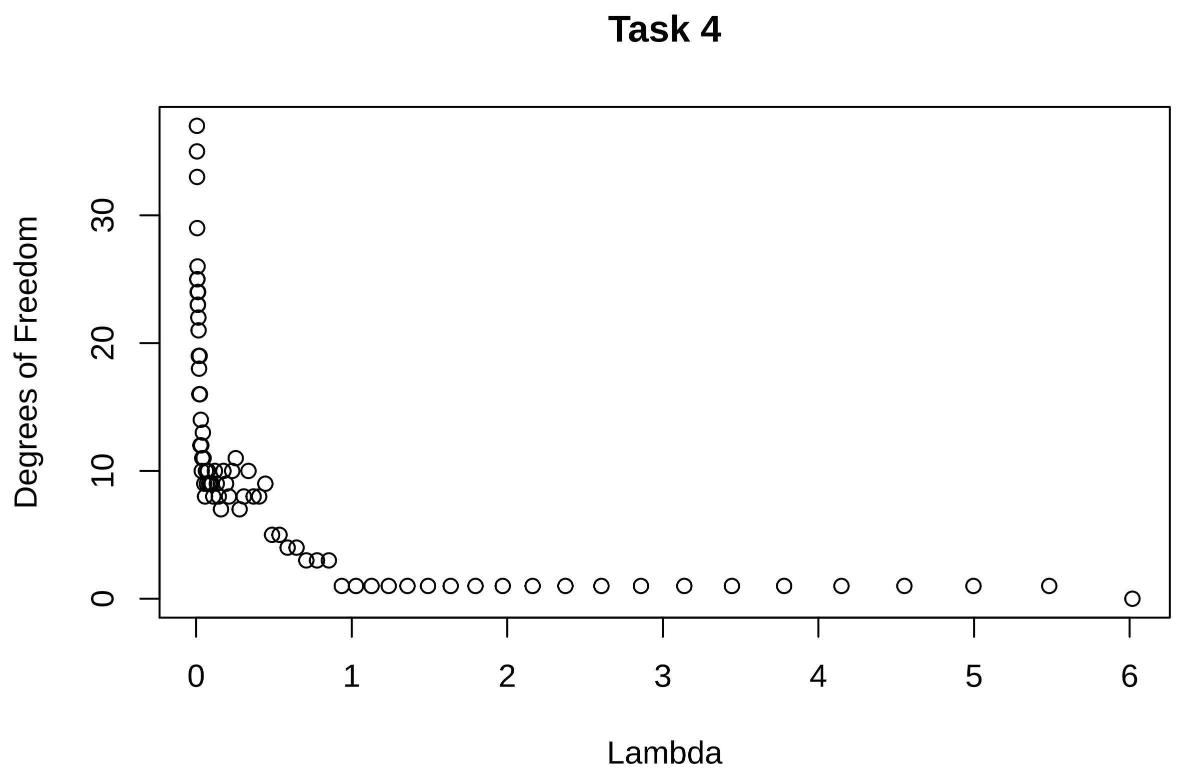
```
(three_features = fit_lasso$lambda[fit_lasso$df == 3][1])
```

```
## [1] 0.8530452
```

Task 4

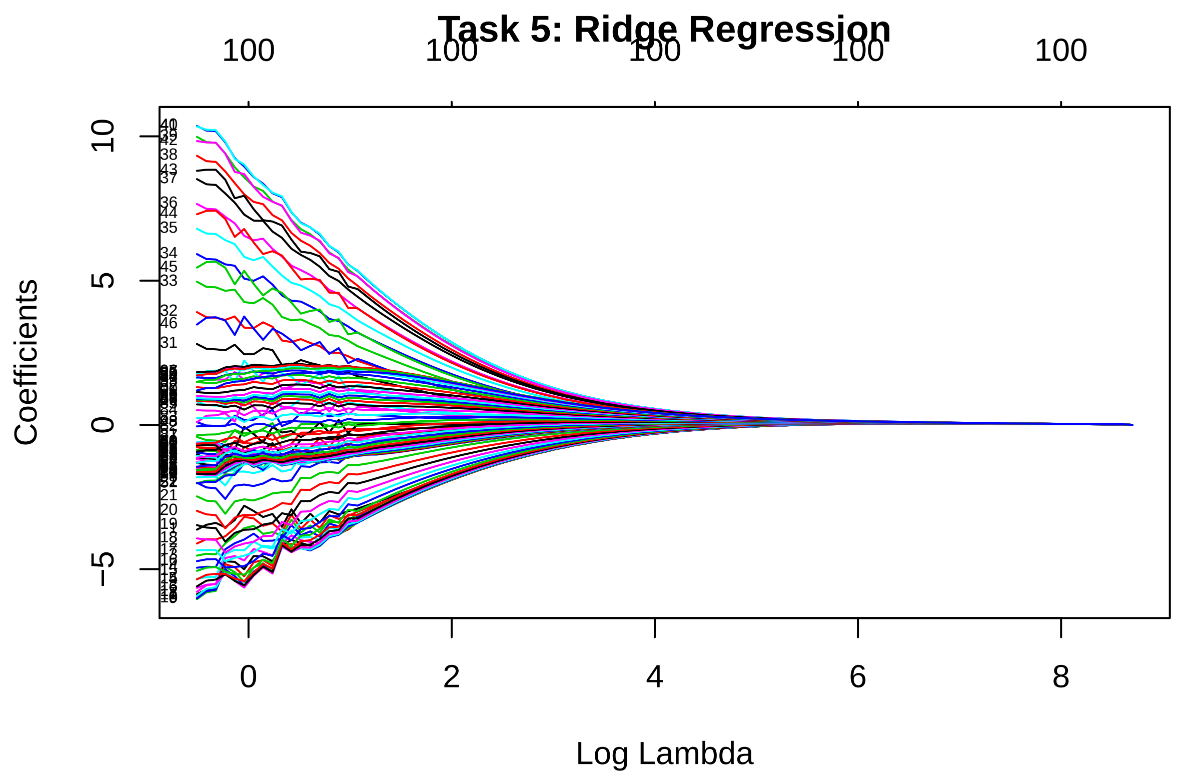
As expected: A lower penalty allows for more coefficients/features to be used and as penalty increases the number of coefficients decreases. Fewer but more significant coefficients/features will be kept as the penalty increases.

```
plot(fit_lasso$lambda, fit_lasso$df, ylab = "Degrees of Freedom", xlab = "Lambda", main = "Task 4")
```



## Task 5

```r
fit_ridge = glmnet(as.matrix(train[,1:100]), train$Fat, alpha = 0,
                   family = "gaussian")

plot(fit_ridge, xvar = "lambda", label = TRUE, main = "Task 5: Ridge Regression")
```



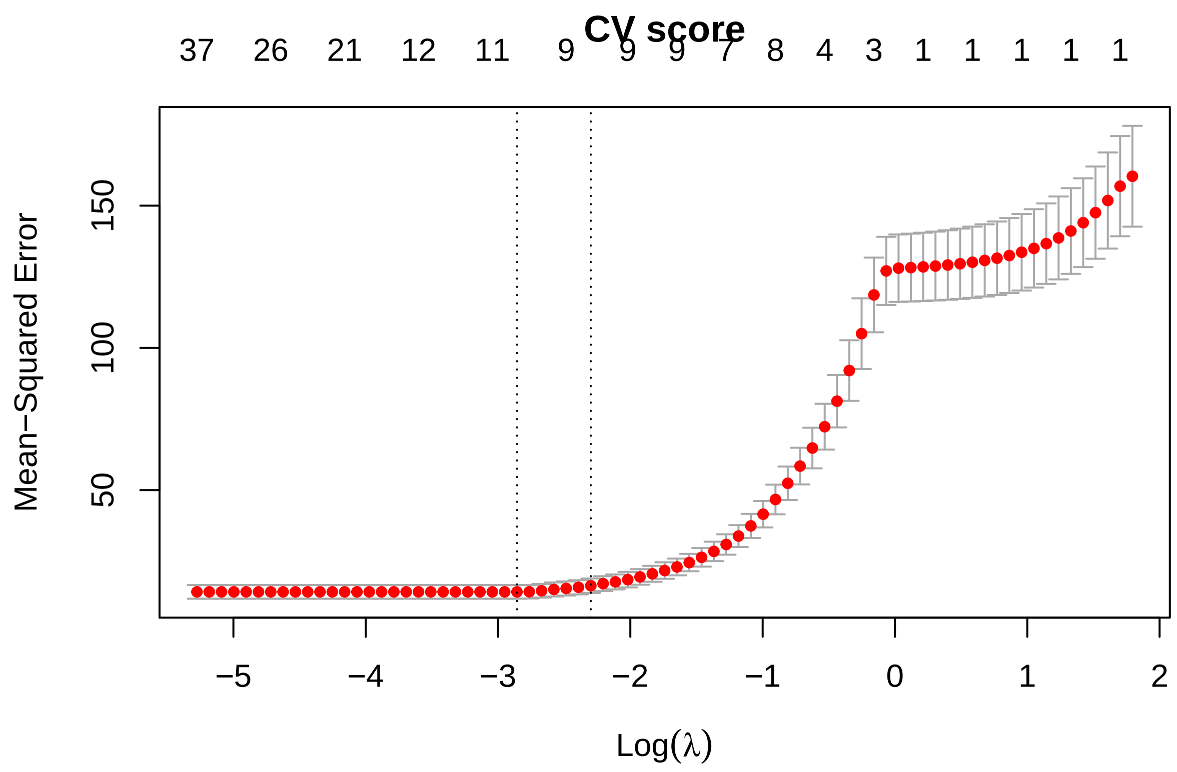
Solution looks uniform as the method does not give a omit features. The number of features are the same for all penalty costs but the magnitude of them are lowered due to high penalty cost, hence the penalty "removes" their impact. Using LASSO method removes the insignificant features which would be preferable in many cases when using wide data to lower computational time and optimize the prediction.

Task 6

As the penalty increases the losses increases and the degrees of freedom / features decreases. The model have a constant error until $Log(\lambda_{min})$. Which gives that the number of features would be 13.

Yes the chosen λ is statistically significantly better than $\lambda = -2$ as the error rate is lower.

```r
fit_cv = cv.glmnet(as.matrix(train[,1:100]), train$Fat, alpha = 1,
                   family = "gaussian")
plot(fit_cv, main = "CV score")
```



```
print('Optimal Lambda:', quote = FALSE)
```

```
## [1] Optimal Lambda:
```

```
(opt_lambda = fit_cv$lambda.min)
```

```
## [1] 0.05744535
```

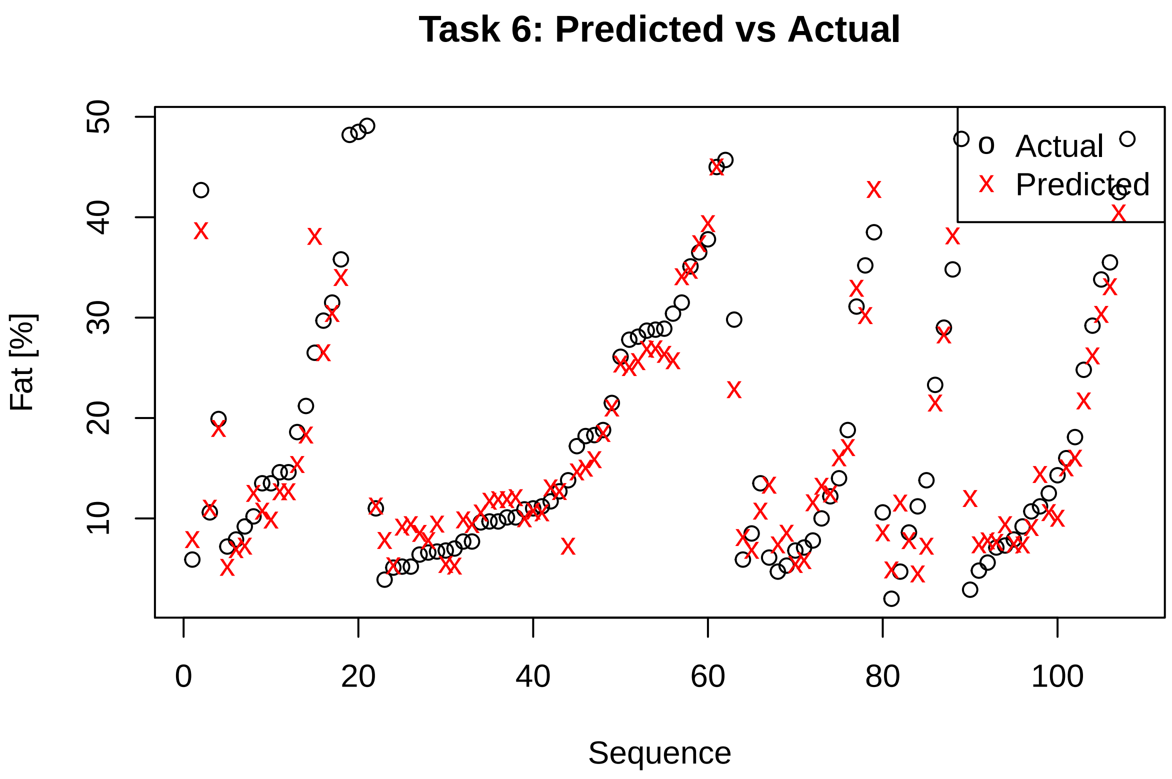
```
print("Number of variables choosen:", quote = FALSE)
```

```
## [1] Number of variables choosen:
```

```
(n_vars = fit_cv$nzero[fit_cv$lambda == opt_lambda])
```

```
## s50
##   8
```

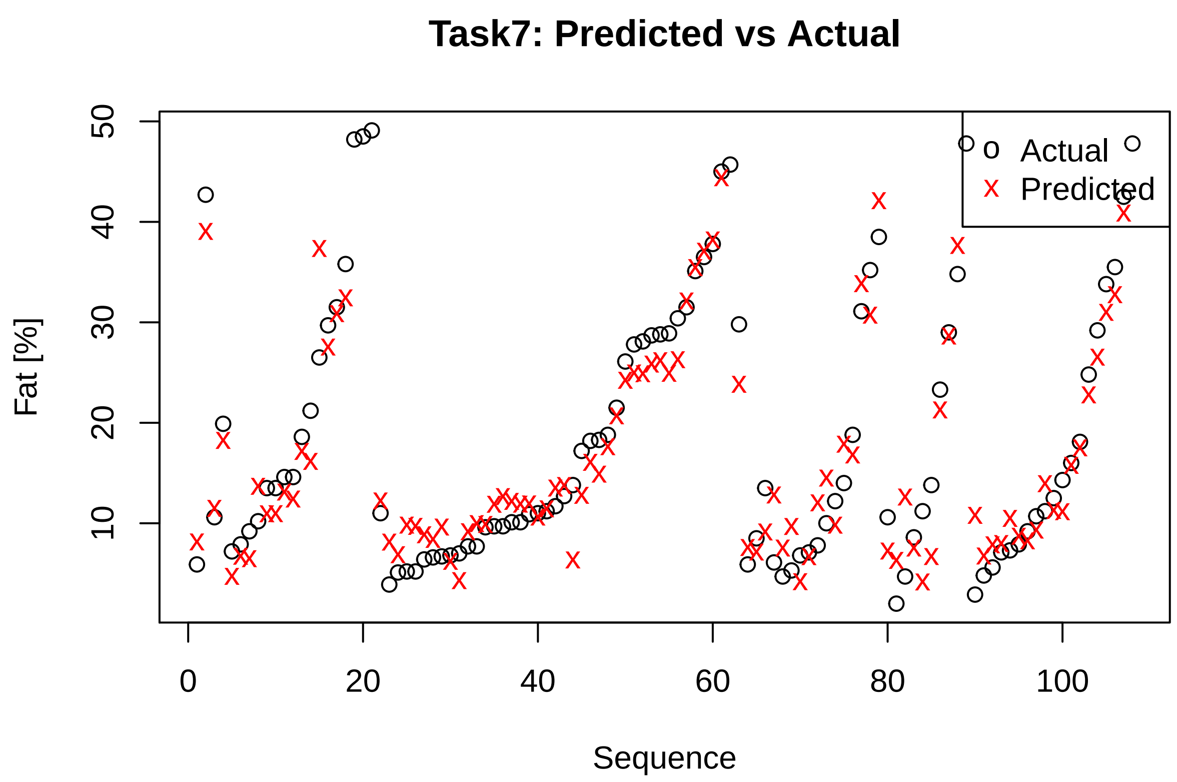
```
pred_cv = predict(fit_cv, newx = as.matrix(test[,1:100]), s = opt_lambda)

plot(test$Fat, xlab = "Sequence", ylab = "Fat [%]", main = "Task 6: Predicted vs Actual")
points(pred_cv, col = "red", pch = "x")
legend("topright", legend = c("Actual", "Predicted"), pch = c("o","x"), col= c(1,2))
```



Task 7

Comparing figure below (Task 7) with the one above (Task 6) one can see that the quality of the predictions are kept more or less. Also the RMSE-score are very similar where the newly generated data does just perform some what worse.

```
new_pred = predict(fit_cv, newx = as.matrix(test[,1:100]), s = opt_lambda)
new_pred = new_pred + rnorm(new_pred)

plot(test$Fat, xlab = "Sequence", ylab = "Fat [%]", main = "Task7: Predicted vs Actual")
points(new_pred, col = "red", pch = "x")
legend("topright", legend = c("Actual", "Predicted"), pch = c("o","x"), col= c(1,2))
```



```r
print("RMSE for Task 6 prediciton:", quote = FALSE)
```

```
## [1] RMSE for Task 6 prediciton:
```

```r
(rmse_test_task6 = sqrt(mean((pred_cv - test$Fat)^2)))
```

```
## [1] 3.697754
```

```r
print("RMSE for new generated data:", quote = FALSE)
```

```
## [1] RMSE for new generated data:
```

```r
(rmse_test_task7 = sqrt(mean((new_pred - test$Fat)^2)))
```

```
## [1] 3.874528
```

Appendix

~~~
```r
knitr::opts_chunk$set(echo = TRUE)
~~~

```
library(kknn)
library(ggplot2)
digits <- read.csv("optdigits.csv")
digits$X0.26 <- as.factor(digits$X0.26)
n <-dim(digits)[1]
set.seed(12345)
id <-sample(1:n, floor(n*0.5))
train <-digits[id,]

id1 <-setdiff(1:n, id)
set.seed(12345)
id2 <-sample(id1, floor(n*0.25))
valid <-digits[id2,]

id3 <-setdiff(id1,id2)
test <-digits[id3,]
digits_kknn_train <- kknn(X0.26~., train, train,  k = 30, kernel = "rectangular")
CM_train <- table(train$X0.26, digits_kknn_train$fitted.values)
CM_train
accuracy_train <- (sum(diag(CM_train)))/sum(CM_train)
accuracy_train
miss_train <- 1-accuracy_train
miss_train
digits_kknn_test <- kknn(X0.26~., train, test,  k = 30, kernel = "rectangular")
CM_test <- table(test$X0.26, digits_kknn_test$fitted.values)
CM_test
accuracy_test <- (sum(diag(CM_test)))/sum(CM_test)
accuracy_test
miss_test <- 1-accuracy_test
miss_test
probs <- as.data.frame(digits_kknn_train$prob)
probs$digit <- train$X0.26
probs_8 <- probs[probs$digit==8,]
easy_probs_8 <- probs_8[probs_8$`8` == 1,]
easy_rows <- as.numeric(row.names(easy_probs_8))
easy_1 <-  sample(easy_rows, 1)
easy_2 <-  sample(easy_rows, 1)

map_easy_1 <- as.numeric(train[easy_1,-65])
map_m_easy_1 <- matrix((map_easy_1), nrow = 8, ncol = 8, byrow = TRUE)
m_easy1 <- map_m_easy_1[rev(1:8),]

map_easy_2 <- as.numeric(train[easy_2,-65])
map_m_easy_2 <- matrix((map_easy_2), nrow = 8, ncol = 8, byrow = TRUE)
m_easy2 <- map_m_easy_2[rev(1:8),]
heatmap(m_easy1, Colv=NA , Rowv=NA, main = "Easy1")
heatmap(m_easy2, Colv=NA , Rowv=NA, main = "Easy1")
hard_probs_8 <- probs_8[order(probs_8$`8`),]
hard_rows <- as.numeric(row.names(hard_probs_8))
hard_1 <- hard_rows[1]
hard_2 <- hard_rows[2]
hard_3 <- hard_rows[3]

map_hard_1 <- as.numeric(train[hard_1,-65])
```

```
map_m_hard_1 <- matrix((map_hard_1), nrow = 8, ncol = 8, byrow = TRUE)
m_hard_1 <- map_m_hard_1[rev(1:8),]

map_hard_2 <- as.numeric(train[hard_2,-65])
map_m_hard_2 <- matrix((map_hard_2), nrow = 8, ncol = 8, byrow = TRUE)
m_hard_2 <- map_m_hard_2[rev(1:8),]

map_hard_3 <- as.numeric(train[hard_3,-65])
map_m_hard_3 <- matrix((map_hard_3), nrow = 8, ncol = 8, byrow = TRUE)
m_hard_3 <- map_m_hard_3[rev(1:8),]
heatmap(m_hard_1, Colv=NA , Rowv=NA, main = "Hard1")
heatmap(m_hard_2, Colv=NA , Rowv=NA, main = "Hard2")
heatmap(m_hard_3, Colv=NA , Rowv=NA, main = "Hard3")
miss_train <- c()
for (i in 1:30) {
  model <- kknn(X0.26~., train, train,  k = i, kernel = "rectangular")
  CM <- table(train$X0.26, model$fitted.values)
  accuracy <- (sum(diag(CM)))/sum(CM)
  miss <- 1 - accuracy
  miss_train <- c(miss_train, miss)
}

df_train <- data.frame(miss = miss_train, i = 1:30, type = rep("train", 30))

miss_valid <- c()
for (i in 1:30) {
  model <- kknn(X0.26~., train, valid,  k = i, kernel = "rectangular")
  CM <- table(valid$X0.26, model$fitted.values)
  accuracy <- (sum(diag(CM)))/sum(CM)
  miss <- 1 - accuracy
  miss_valid <- c(miss_valid, miss)
}

df_valid <- data.frame(miss = miss_valid, i = 1:30,  type = rep("valid", 30))
df <- rbind(df_train,df_valid)

ggplot(df, aes(x=i, y = miss, color = type)) +
  geom_point()  + theme_bw() +
  labs(title = "Missclassification error for valid and train data for different K"
       , x = "K" , y = "Missclassifction error")  +
  theme(axis.title.y = element_text(vjust = 0.5, size = 13 , face = "bold")) +
  theme(axis.title.x = element_text(vjust = 0.5 ,size = 13 , face = "bold")) +
  theme(plot.title = element_text(size = 14, face = "bold" , hjust = 0.4 )) +
  theme(axis.text.y = element_text(size = 11)) +
  theme(axis.text.x = element_text(size = 11))

model_test <- kknn(X0.26~., train, test,  k = 7, kernel = "rectangular")
CM <- table(test$X0.26, model_test$fitted.values)
accuracy <- (sum(diag(CM)))/sum(CM)
miss <- 1 - accuracy
miss_df <- data.frame( "test" = miss, "Valid" = miss_valid[7], "Train" = miss_train[7])
miss_df
#Function for cross entropy
cross.entropy <- function(p, q){
  entropy <- 0
  row <- 1
  for (label in q) {
    entropy = - log(p[row,label] + 0.000000000001) + entropy
    row <- row + 1
  }
  m_entropy <- entropy / length(q)
  return(m_entropy)
}

#Train

cross_train <- c()
for(i in 1:30){
  model_train <- kknn(X0.26~., train, train, k = i, kernel = "rectangular")
  cross_t <- cross.entropy(model_train$prob, train$X0.26)
  cross_train <- c(cross_train, cross_t)
}
df_1_train <- data.frame(cross = cross_train, i = 1:30, type = rep("train", 30))

#Valid

cross_valid <- c()
for(j in 1:30){
  model_valid <- kknn(X0.26~., train, valid, k = j, kernel = "rectangular")
  cross_v <- cross.entropy(model_valid$prob, valid$X0.26)
  cross_valid <- c(cross_valid,cross_v )
}
df_1_valid <- data.frame(cross = (cross_valid), i = 1:30, type = rep("valid", 30))
df_new <- rbind(df_1_train, df_1_valid)
ggplot(df_new, aes(x=i, y = cross, color = type)) +
  geom_point() + theme_bw() +
  labs(title = "Cross-entropy error for valid and train data for different K"
       , x = "K" , y = "Cross-entropy error") +
  theme(axis.title.y = element_text(vjust = 0.5, size = 13 , face = "bold")) +
  theme(axis.title.x = element_text(vjust = 0.5 ,size = 13 , face = "bold")) +
  theme(plot.title = element_text(size = 14, face = "bold" , hjust = 0.4 )) +
  theme(axis.text.y = element_text(size = 11)) +
  theme(axis.text.x = element_text(size = 11))


dt <- read.csv("parkinsons.csv")
dt <- dt[c(-1:-4, -6)]

## scale the data and divide to train and test
scaled.dt <- scale(dt)
m<- nrow(scaled.dt)
set.seed(12345)
id <- sample(1:m, floor(m*0.6))
train <- scaled.dt[id,]
test <- scaled.dt[-id,]
 # for train
x<- train[,2:17]
y<- train[,1]
```

```
# for test
x_test <- test[,2:17]
y_test <- test[,1]

Loglikelihood <- function(w, sigma,y,x){
  n <- nrow(x)
  #y <- train[, 1]
  w<- as.matrix(w)
  log_l<- -0.5*n*(log(2*pi))-(0.5*n*log(sigma^2))-(1/(2*sigma^2))*(sum((y-x %*% w)^2))
  return(log_l)
}
Ridge <- function(w,sigma,lambda,y,x){
  y <- as.matrix(y)
  x<- as.matrix(x)

  ridge_fun<- -Loglikelihood(w,sigma,y,x)+lambda*sum(w^2)
  return(ridge_fun)

}
RidgeOpt <- function(R,lambda,y,x)
  {
  y <- as.matrix(y)
  x<- as.matrix(x)
    new_ridge<-function(R,lambda,y,x)
      {
        out2<-Ridge(w=R[1:16],sigma=R[17],lambda,y,x)
        return(out2)
        }
    optimize<- optim(par=R, fn=new_ridge,lambda=lambda,y=y, x=x, gr=NULL,method="BFGS")
    return(optimize)

}
DF <- function(lambda){
  mat <- as.matrix(train[ ,2:17])
  ## Calculate trace of Hat Matrix
  H <- mat %*% (solve((t(mat) %*% mat) + (lambda * diag(16)))%*% t(mat))  ### solve() calculates the in
  traceMat <- sum(diag(H))
  return(traceMat)

}

DF(1)
DF(100)
DF(1000)
R<-rep(1,17)
out1<- RidgeOpt(R, lambda=1,y=y, x=x)
out2<- RidgeOpt(R, lambda=100,y=y, x=x)
out3<- RidgeOpt(R, lambda=1000,y=y, x=x)

#lambda=1
#x<- as.matrix(x)
pred_y1<-x %*% out1$par[-17]
MSE1<- mean((y-pred_y1)^2)
```

```
#lambda=100
pred_y2<-x %*% out2$par[-17]
MSE2<- mean((y-pred_y2)^2)

#lambda=1000
pred_y3<-x %*% out3$par[-17]
MSE3<- mean((y-pred_y3)^2)

# For test data:
#lambda=1
pred_y_test1<-x_test %*% out1$par[-17]
MSE_test1<- mean((y_test - pred_y_test1)^2)

#lambda=100
pred_y_test2<-x_test %*% out2$par[-17]
MSE_test2<- mean((y_test - pred_y_test2)^2)

#lambda=1000
pred_y_test3<-x_test %*% out3$par[-17]
MSE_test3<- mean((y_test - pred_y_test3)^2)
prediction <- data.frame(predY_training = head(c(pred_y1, pred_y2, pred_y3)), predY_testing = head(c(pr
prediction

optimal_val<- matrix(c(MSE1, MSE2, MSE3,MSE_test1, MSE_test2, MSE_test3),ncol=2)
rownames(optimal_val)<- c("lambda=1", "lambda=100","lambda=1000")
colnames(optimal_val)<-c("MSE_train", "MSE_test")
optimal_val
AIC <- function(w, sigma, y, x, lambda)
{
step1 <- -2*Loglikelihood(w, sigma, y, x) + 2*DF(lambda)
return(step1)
}
# Training Data
AIC1 <- AIC(w=out1$par[-17], sigma = out1$par[17], y=y, x=x, lambda = 1)
AIC100 <- AIC(w=out2$par[-17], sigma = out2$par[17], y=y, x=x, lambda = 100)
AIC1000 <- AIC(w=out3$par[-17], sigma = out3$par[17], y=y, x=x, lambda = 1000)

# Test Data
AIC_test1 <- AIC(w=out1$par[-17], sigma = out1$par[17], y=y_test, x=x_test, lambda = 1)
AIC_test100 <- AIC(w=out2$par[-17], sigma = out2$par[17], y=y_test, x=x_test, lambda = 100)
AIC_test1000 <- AIC(w=out3$par[-17], sigma = out3$par[17], y=y_test, x=x_test, lambda = 1000)

AICscore<- matrix(c(AIC1,AIC100,AIC1000,AIC_test1,AIC_test100,AIC_test1000), ncol=2)
rownames(AICscore)<- c("lambda=1", "lambda=100","lambda=1000")
colnames(AICscore)<- c("AICscore_train", "AICscore_test")
AICscore
data = read.csv('tecator.csv')
data = data[,2:102] # Pre-processing


# Split data 50/50

n = dim(data)[1]
```

```
set.seed(12345)
id = sample(1:n, floor(n*0.5))
train = data[id,]
test = data[-id,]
lm_model1 = lm(formula = Fat ~ ., data = as.data.frame(train))
#summary(lm_model1)

print("Choose Channel63 due to being highly significant!", quote = FALSE)

fit1 = lm(Fat ~ Channel63, as.data.frame(train));

pred_train = predict(fit1, as.data.frame(train))
pred_test = predict(fit1, as.data.frame(test))

print("RMSE score for test-prediction:", quote = FALSE)
(rmse_test= sqrt(mean((pred_test - test$Fat)^2)))
print("RMSE score for train-prediction:", quote = FALSE)
(rmse_train = sqrt(mean((pred_train - train$Fat)^2)))
library(glmnet)
fit_lasso = glmnet(as.matrix(train[,1:100]), train$Fat, alpha = 1.0,
                      family = "gaussian")

plot(fit_lasso, xvar = "lambda", label = TRUE, main = "Task 3: LASSO Regression")

print("Choosing lambda below gives only three features:", quote = FALSE)
(three_features = fit_lasso$lambda[fit_lasso$df == 3][1])
plot(fit_lasso$lambda, fit_lasso$df, ylab = "Degrees of Freedom", xlab = "Lambda", main = "Task 4")
fit_ridge = glmnet(as.matrix(train[,1:100]), train$Fat, alpha = 0,
                   family = "gaussian")

plot(fit_ridge, xvar = "lambda", label = TRUE, main = "Task 5: Ridge Regression")
fit_cv = cv.glmnet(as.matrix(train[,1:100]), train$Fat, alpha = 1,
                   family = "gaussian")
plot(fit_cv, main = "CV score")

print('Optimal Lambda:', quote = FALSE)
(opt_lambda = fit_cv$lambda.min)
print("Number of variables choosen:", quote = FALSE)
(n_vars = fit_cv$nzero[fit_cv$lambda == opt_lambda])

pred_cv = predict(fit_cv, newx = as.matrix(test[,1:100]), s = opt_lambda)

plot(test$Fat, xlab = "Sequence", ylab = "Fat [%]", main = "Task 6: Predicted vs Actual")
points(pred_cv, col = "red", pch = "x")
legend("topright", legend = c("Actual", "Predicted"), pch = c("o","x"), col= c(1,2))
new_pred = predict(fit_cv, newx = as.matrix(test[,1:100]), s = opt_lambda)
new_pred = new_pred + rnorm(new_pred)

plot(test$Fat, xlab = "Sequence", ylab = "Fat [%]", main = "Task7: Predicted vs Actual")
points(new_pred, col = "red", pch = "x")
legend("topright", legend = c("Actual", "Predicted"), pch = c("o","x"), col= c(1,2))

print("RMSE for Task 6 prediciton:", quote = FALSE)
(rmse_test_task6 = sqrt(mean((pred_cv - test$Fat)^2)))
```

```
print("RMSE for new generated data:", quote = FALSE)
(rmse_test_task7 = sqrt(mean((new_pred - test$Fat)^2)))
```