Machine learning 732A95

Lab 1, block 1

Eric Herwin
erihe068

Contents

All the code will be presented in the appendix of this lab report. If there is code inside the actual report, it is there because it is asked for.

Assignment 1. Spam classification with nearest neighbors

In this task i will use the provided `spambase.xlsx` data where i see spam = 1 and regualr emails as spam = 0.

1.1

Here i will divide the data in a training and validation sets. They will be split 50 percent of the data set each.

With the code provided:

```
spambase <- read_excel("spambase.xlsx")
n <- dim(spambase)[1]
set.seed(12345)
id <- sample(1:n, floor(n*0.5))
train <- spambase[id,]
test <- spambase[-id,]
```

I have split up the data.

1.2

For this assignment i will fit a K-nearest neighbor classifier to the training data. The function, `knearest()`, i have implemented is in the appendix 1.2.

This function computes returns a vector of probabilities and a distance matrix that is based on the cosine similarity. The top of the distance matrix is printed below.

```
dist <- knearest(train, 5, test)$Distancematr

dist[1:10, 1:10]
```

```
##            [,1]      [,2]      [,3]      [,4]      [,5]      [,6]
##  [1,] 1.0000000 1.0000000 1.0000000 1.0000000 0.9296314 1.0000000
##  [2,] 1.0000000 0.9333354 1.0000000 1.0000000 0.9004838 0.7397750
##  [3,] 0.3106880 0.2285334 0.9651570 0.8541353 0.3159146 0.2558663
##  [4,] 0.4190554 0.3529327 0.9706347 0.8770668 0.4281123 0.3850166
##  [5,] 0.7359715 0.6885723 0.8174168 0.5978832 0.5179976 0.6284180
##  [6,] 0.3108562 0.2396126 0.8690269 0.8303518 0.3237340 0.2721695
##  [7,] 0.8527008 0.8500528 0.8635101 0.8824937 0.8549266 0.8431047
##  [8,] 0.8342823 0.8207864 0.8061595 0.9381844 0.8238071 0.8243111
##  [9,] 0.8025920 0.8824462 0.4857129 0.9403825 0.8719007 0.8861205
## [10,] 0.5445958 0.4872289 0.8157174 0.8506925 0.5349043 0.4739639
##            [,7]      [,8]      [,9]     [,10]
##  [1,] 1.0000000 1.0000000 1.0000000 1.0000000
##  [2,] 1.0000000 0.8490343 1.0000000 1.0000000
##  [3,] 0.2528591 0.2309649 0.2361742 0.2443410
##  [4,] 0.3703179 0.3589226 0.3562560 0.3631389
##  [5,] 0.6897024 0.6594163 0.7138509 0.6788963
##  [6,] 0.2397718 0.2466515 0.2425629 0.2301235
##  [7,] 0.8127336 0.8514408 0.8710782 0.7986005
##  [8,] 0.7949854 0.8224454 0.8209465 0.7690826
##  [9,] 0.8315530 0.8835344 0.8626397 0.8176354
## [10,] 0.4872092 0.4777549 0.4972984 0.4689274
```

1.3

Here i will classify the training and the test data by using $K = 5$ and the observation will be classified as spam if the probability is above 0.5.

I will use the function i made to produce the confusion matrix and with that, the missclassification rate.

This is the confusion matrix and missclassification rate for the training data.

```
con_matr_k5_tr
```

```
##
##       0   1
##   0 787 158
##   1 119 306
```

```
miss_class_k5_tr
```

```
## [1] 0.2021898
```

This is the confusion matrix and missclassification rate for the test data.

```
con_matr_k5_tst
```

```
##
##       0   1
##   0 695 242
##   1 193 240
```

```
miss_class_k5_tst
```

```
## [1] 0.3175182
```

1.4

Here is will do the same procedure as step 1.3, but i will use $K = 1$.

This is the confusion matrix and missclassification rate for the train data.

```
conf_matr_train_1
```

```
##
##       0   1
##   0 939   6
##   1   2 423
```

```
miss_class_train_1
```

```
## [1] 0.005839416
```

We can see that the missclassification rate is much lower when we run it for $K = 1$ then for $K = 5$. This is logical due to that when the algorithm looks for the closest value it will most often choose itself and therefore become less missclassifications.

This is the confusion matrix and missclassification rate for the test data.

```
conf_matr_test_1
```

```
##
##       0   1
##   0 639 298
##   1 178 255
```

```
miss_class_test_1
```

```
## [1] 0.3474453
```

When i run it for the test data with $K = 1$ i get a little bit higher missclassification then when we run with $K = 5$. This might be because that the decision boundary becomes much more “smoothened” and might indicate that it exists some clusters that can decide if a mail is a spam or not.

1.5

For this task i will use the function `kknn()` from the package `kknn`. I will compute the confusion matrix and the missclassification rate for the test data and compare this to my `knearest()` function.

For when $K = 5$:

```
#Confusion matrix
conf_matr
```

```
##
##       0   1
##   0 640 297
##   1 177 256
```

```
#Missclassification rate
miss_k5_tst
```

```
## [1] 0.3459854
```

For when $K = 1$.

```
#Confusion matrix
conf_matr_1
```

```
##
##       0   1
##   0 640 297
##   1 177 256
```

```
#Missclassification rate
miss_k1_tst
```

```
## [1] 0.3459854
```

We can see that the missclassification for $K = 1$ and $K = 5$ is exacly the same and by observing the confusion matrix we can see that it has classified an missclassified the same. We can conlude that the value of K is too low for the `kknn()` function.

So comparing this `kknn()` function to my `knearest()` for when $K = 5$, we can conlude that the `knearest()` function missclassify less. Comparing the functions when $K = 1$ we can conclude that the functions classify/missclassify about the same.

1.6

For this task i will use the `knearest()` and `kknn()` function by using a vecor of $\pi = 0.05, 0.1, 0.15, ..., 0.95$ for classification limits. I will also compute the specificity and sensitivity and plot ROC curves.

The specificity:

```
#For knearest():
1-FPR_kn
```

```
##  [1] 0.3351121 0.3351121 0.3351121 0.5250800 0.5250800 0.5250800 0.5250800
##  [8] 0.7417289 0.7417289 0.7417289 0.7417289 0.8986126 0.8986126 0.8986126
## [15] 0.8986126 0.9871932 0.9871932 0.9871932 0.9871932
```

```
#For kknn():
1-FPR_kk
```

```
##  [1] 0.3447172 0.4119530 0.4439701 0.4717182 0.5378869 0.5570971 0.6061900
##  [8] 0.6254002 0.6456777 0.6830309 0.7235859 0.7417289 0.7566702 0.7918890
## [15] 0.8121665 0.8762006 0.8964781 0.9146211 0.9519744
```

The sensitivity:

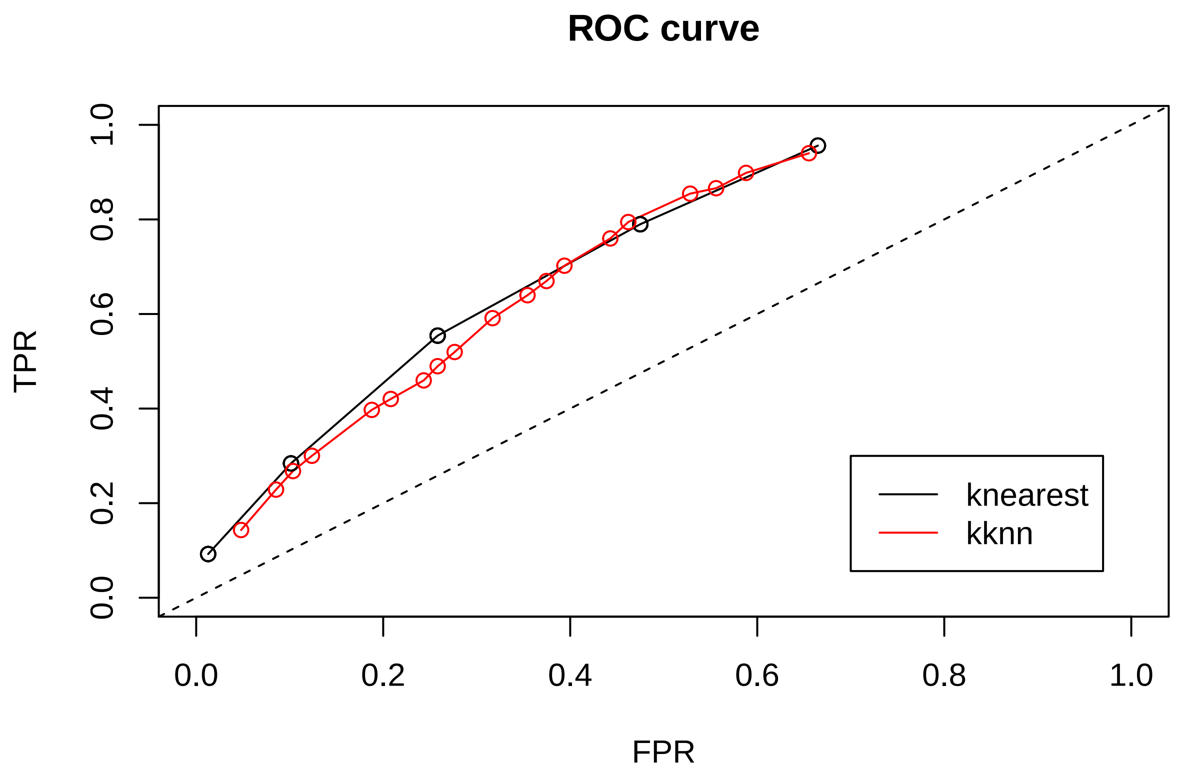
```
#For knearest():
TPR_kn
```

```
##  [1] 0.95612009 0.95612009 0.95612009 0.78983834 0.78983834 0.78983834
##  [7] 0.78983834 0.55427252 0.55427252 0.55427252 0.55427252 0.28406467
## [13] 0.28406467 0.28406467 0.28406467 0.09237875 0.09237875 0.09237875
## [19] 0.09237875
```

```
#For kknn():
TPR_kk
```

```
##  [1] 0.9399538 0.8983834 0.8660508 0.8545035 0.7944573 0.7598152 0.7020785
##  [8] 0.6697460 0.6397229 0.5912240 0.5196305 0.4896074 0.4595843 0.4203233
## [15] 0.3972286 0.3002309 0.2678984 0.2286374 0.1431871
```

The ROC curve:



In the plot the black is `knearest()` and red is `kknn()`. We can see that the both curves are following one another quite closely, but from the plot i will decide that the black line has the greatest area under the curve. Therefore my `knearest()` function are the best, but only for looking at this plot.

We can see that when using my k-nearest funciton that the predictions are grouping together for different π:s. This is mainly due to that my functions uses a way of "naive" calculation of the probability. The `kknn()` function uses a different way of calculating the probabilities and gets more precise probabilities. The functions might be using a more "optimal" solution for calculating the probabilities.

Assignment 3. Feature selection by cross-validation in a linear model.

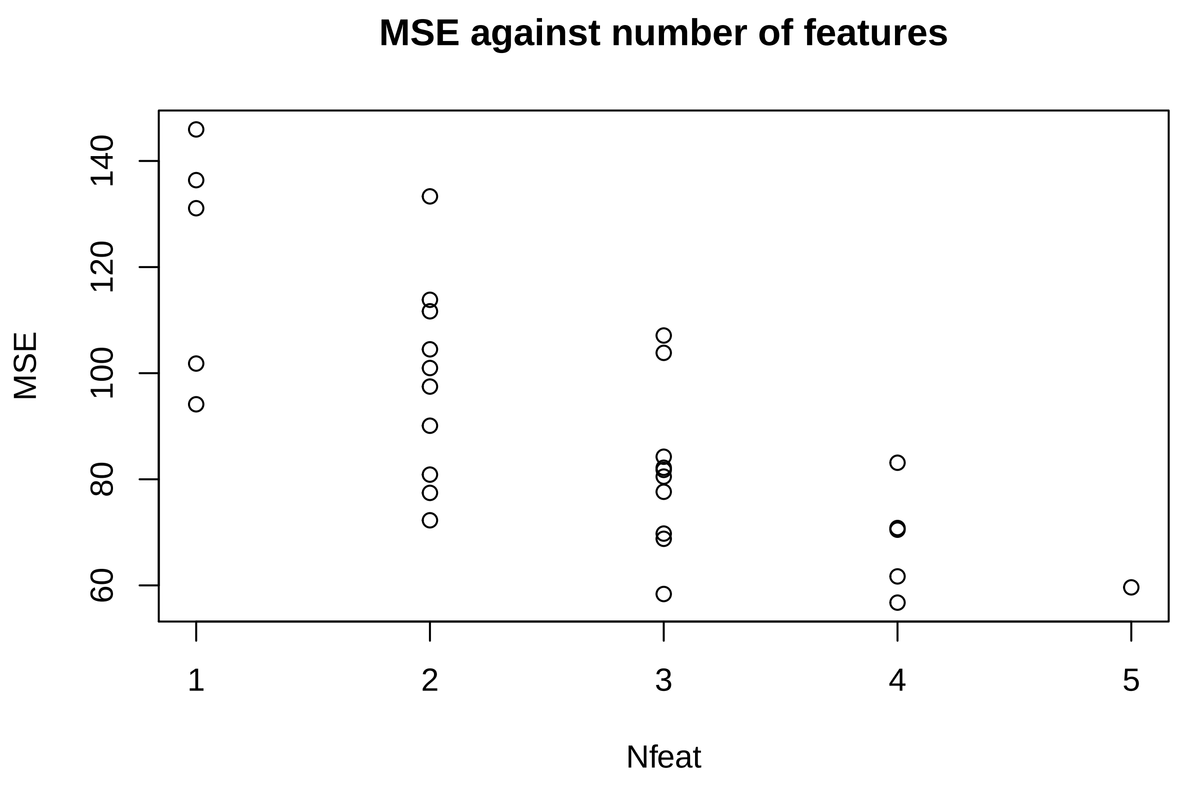
3.1

For this task i have implemented a function that performes a N fold cross validation with taking in to acount of 5 features. The code for this funtion is in the appendix 3.1.

3.2

We will now test the function `myCV` on the `swiss` data set where Y is Fertility, the rest of the variables will be X. This will be done with `Nfolds = 5`. So we will get:

```
myCV(as.matrix(swiss[,2:6]), swiss[[1]], 5)
```



```
## $CV
## [1] 56.75421
##
## $Features
## [1] "Agriculture"      "Education"        "Catholic"
## [4] "Infant.Mortality"
```

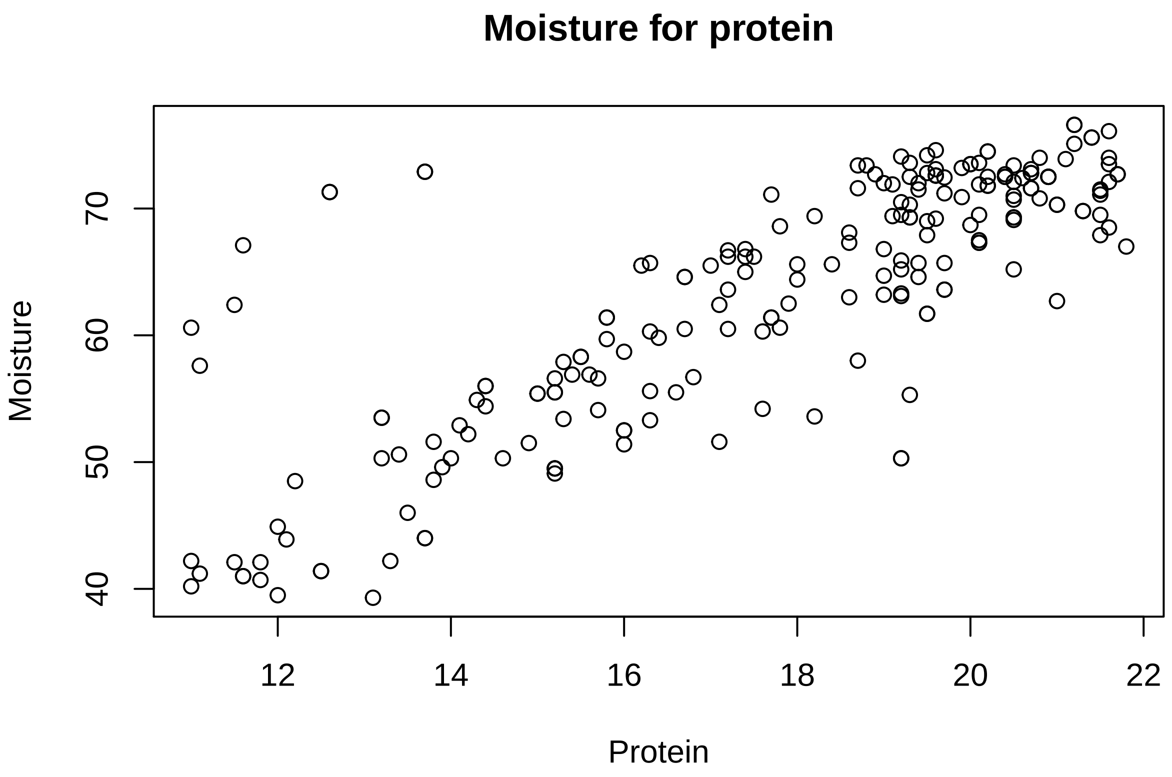
The most opimal model choosen with my `myCV` function is dispayed in the output. The Y variable is Fertility and it seems logical that it would depend on the education, religiosity, the amount of births that live less then 1 year, but the precentage of males involved in agriculture might be strange or reasonable. I am not the expert.

Assignment 4. Linear regression and regularization

4.1

```
tecator <- read_excel("tecator.xlsx")

plot(y =as.numeric(tecator$Moisture),  x= as.numeric(tecator$Protein),
     ylab = "Moisture", xlab = "Protein", main = "Moisture for protein")
```



4.2

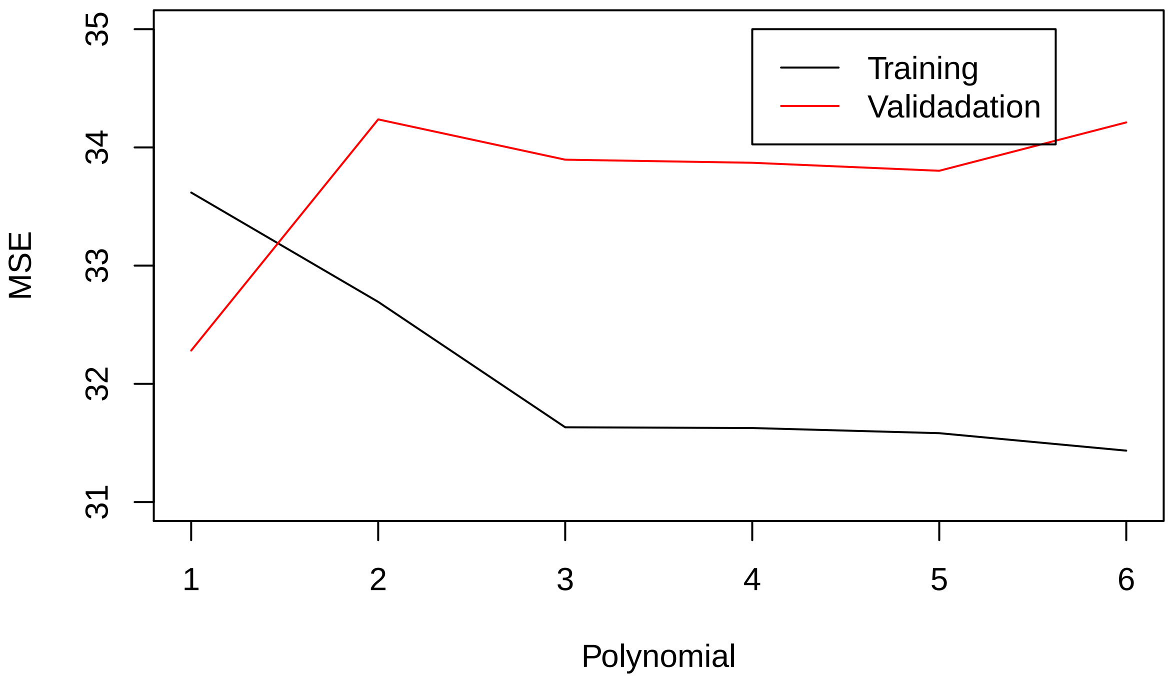
For this task i will choose a probablistic model M_i that can describe the data from the plot in the previous task well. From the plot we can assume that a function with the power of 1 is the most probibalisic model. The model will have the structure:

$$y_i = \beta_0 + \beta_1 x_{i1} + \epsilon_i$$

The MSE measurement is appropriate because it gives a good indication of how the model is performing when fitting to the data, and when minimized it also minimizes the variance. A good MSE value means a better fit, but might be because a model is overfitted and you force the model to the mean of a normal distubution. In my case the linear model will not be over-fitted, so the MSE measurement will be an appropriate indicator if the model is good or not.

4.3

For this task i will plot the training and validation MSE to each other and how it depends on i (eg.the ploynomial). Most of this code will be provided in appendix 4.3.



The plot suggests that the most optimal model would just include one parameter (eg. a simple linear regression). We can see that the the MSE for the training data just decreases with the complexity, which is logical due to that the model will be overfitted to the data. The MSE for the validation data will just be bigger the higher complexity the model would have, due to that the validation data will have a much more hard time giving a good fit for an overfitted model. The model choosen according to the MSE will have a high bias because of the low complexity, but will have a low variance. This is the bias-variance tradeoff.

4.4

In this task we will perform a variable selection with a setpwise AIC where the variable Fat is the response and Channel1 - Channel100 is the covariates.

```
data_s <- tecator[,2:102]
model <- lm(Fat ~ ., data = data_s)
final <- stepAIC(model, trace = FALSE)
length(final$coefficients)-1
```

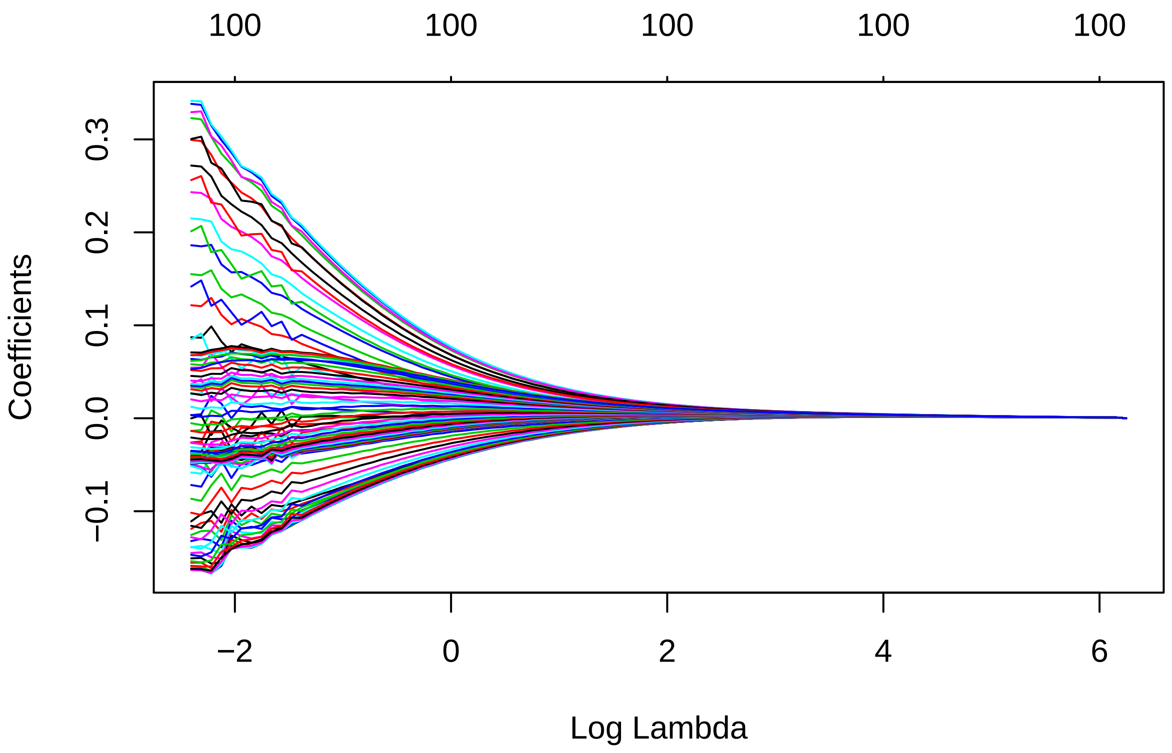
```
## [1] 63
```

We can see that out of 100 variables in the first model the stewise AIC choose 63 variables as the optimal model.

4.5

For this step I will fit a Ridge regression model with the same predictor and response variables as the previous step. I will also plot how the coefficients depend on the penalty factor λ.

```
pred <- scale(as.matrix(data_s[,101]))
xes <- scale(as.matrix(data_s[,1:100]))
ridge <- glmnet(x = xes, y = pred, alpha = 0, family = "gaussian")
plot(ridge,xvar = "lambda")
```



We can see that the coefficients are converging to 0 which means that the more penalized the model are with λ, the more parameters will be unsignificant as they approach 0. The penalazition factor is there to decrease over-fitting of a model and therefore increasing this factor will lead to an increased variance in the model due to the error function is explained as:

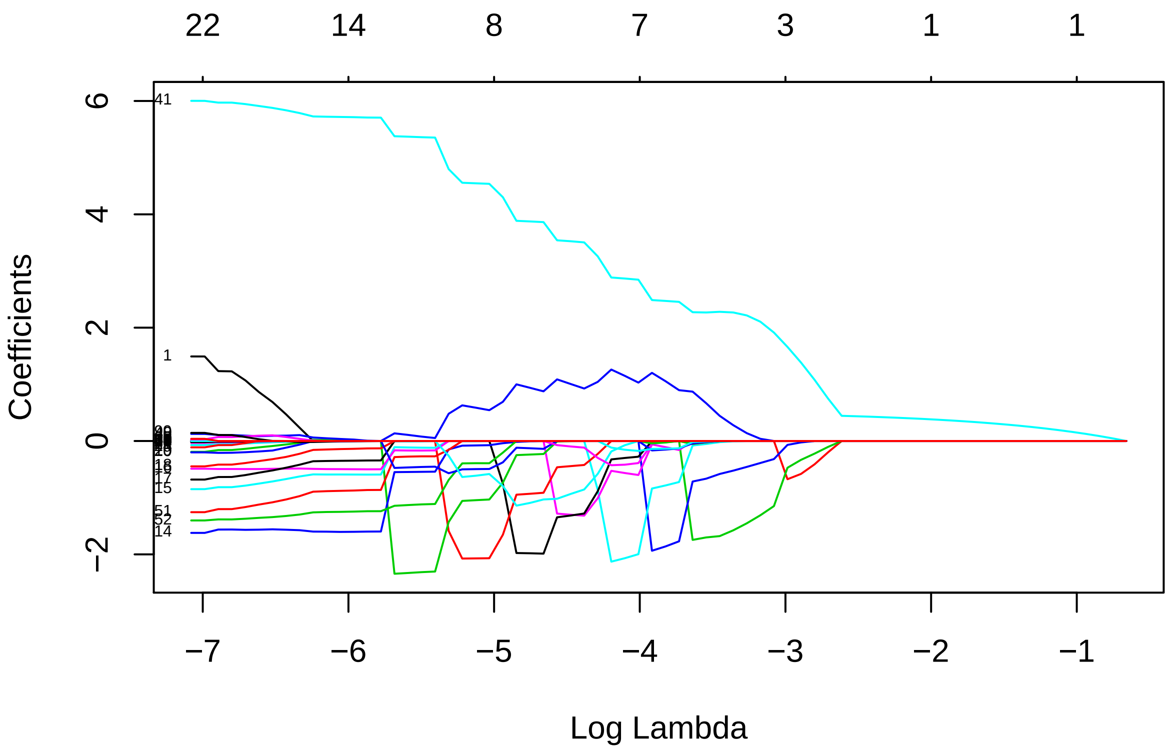
$$1/2\sum_{n=1}^{N}[t_n - \mathbf{w}^T\phi(\mathbf{x}_n)]^2 + \lambda/2\sum_{i=0}^{M}|w_i|^q$$

Where $\mathbf{w}$ is the vector of the parameters w_i and $q = 2$. That is why it is logical for the coefficients to become more un-significant as they are not needed, eg. over-fitting decreases and the model becomes more flexible.

4.6

For this step I will do the same as the previous step, but I will fit a LASSO regression and compare it with the Ridge.

```r
LASSO <- glmnet(x = xes, y = pred, alpha = 1, family = "gaussian")
plot(LASSO, xvar = "lambda", label = TRUE)
```

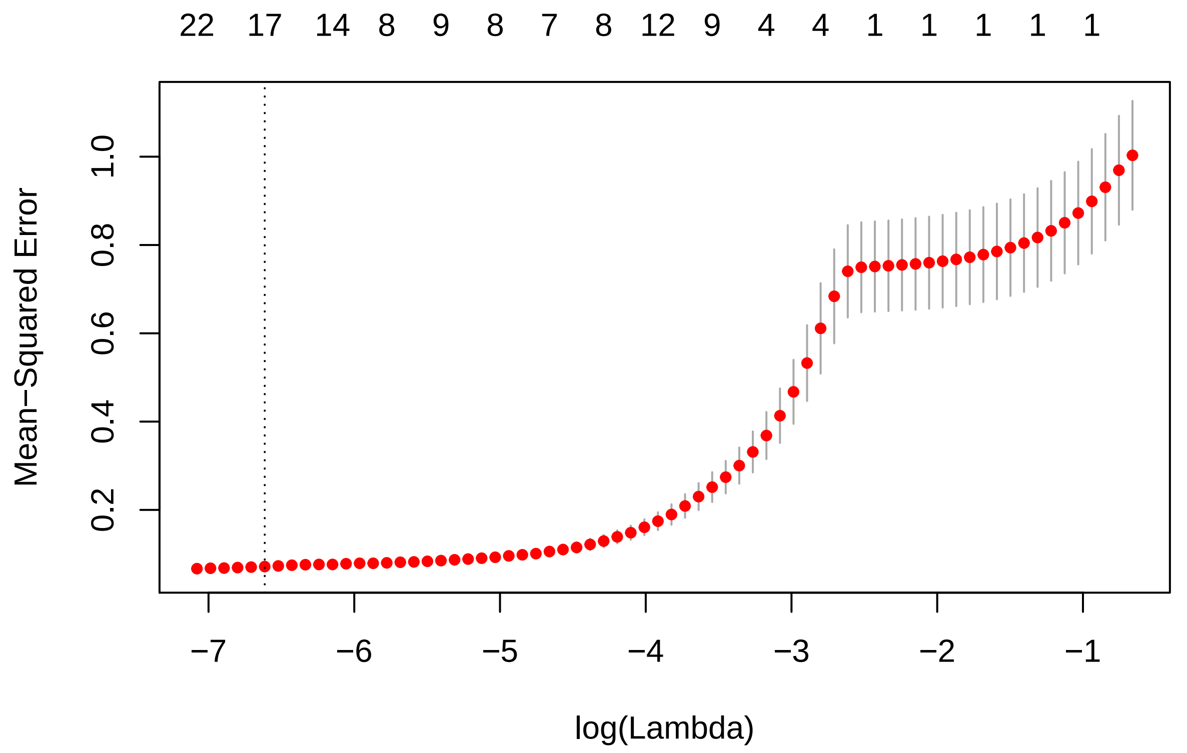


Comparing this plot with the ridge plot we can see that the coefficients are not converging to 0 as smooth as the Ridge regression does. So if we recall the formula of the error function from the previous step, it is the same function but $q = 1$ we will have a more rectangular regularization term. So because of terms becomes the absolute of the parameters the coefficients will become "discrete" and that is why some coeficients die down quickly.

4.7

For this step I will use a cross-validation to find the optimal LASSO model. I will also include when $\lambda = 0$.

```
set.seed(12345)
CV_LASS <- cv.glmnet(x = xes, y = pred,
                     alpha = 1, lambda  = c(LASSO$lambda, 0))
plot(CV_LASS)
```



```
CV_LASS$cvm[which.min(CV_LASS$cvm)]
```

```
## [1] 0.06375671
```

By just looking at the graph we can conclude that the larger the lambda, the larger the MSE. The plot suggest that the best model would be using as low lambda as possible (eg. 0), but plotting a $log(0)$ would lead to $-Inf$ so it would not be showed in the plot. The cross-validation does not seem to make it better so we will just consider the optimal model would be a linear regression (lambda = 0) where the $MSE = 0.06375671$. If we want to utilize the LASSO regularization, we could try to take a low lambda as possible (eg. lambda = 0.0000001) and we could get some reduction of parameters and not affect the MSE too much.

4.8

In this task I will compare the model choosen by the step AIC and the LASSO regression with the optimal lambda.

We can see from the conclusion from the previous step with LASSO that the most optimal model would be with lambda = 0 and the amount of variables in the model would be 100. The optimal amount of variables for the stepwise AIC is 63. We can conclude that the stepwise AIC would be the best option with this data due to less over-fitting.

Appendix

1.1

```r
spambase <- read_excel("spambase.xlsx")
n <- dim(spambase)[1]
set.seed(12345)
id <- sample(1:n, floor(n*0.5))
train <- spambase[id,]
test <- spambase[-id,]
```

1.2

```r
knearest <- function(data,k,newdata) {

  n1=dim(data)[1]
  n2=dim(newdata)[1]
  p=dim(data)[2]
  Prob=numeric(n2)
  X=as.matrix(data[,-p])
  Xn=as.matrix(newdata[-p])
  X=X/matrix(sqrt(rowSums(X^2)), nrow=n1, ncol=p-1)

  Xn=Xn/matrix(sqrt(rowSums(Xn^2)), nrow=n2, ncol=p-1)

  matr_mass <- X %*% t(Xn)

  #Distance matrix
  D <- 1-matr_mass

  dat <- lapply(1:n2, function(i){

    arr <- arrange(data.frame(Dist = D[,i], S_pam = data$Spam), Dist)
    mean(arr[1:k,2])

  })

  Prob <- unlist(dat)

  svar <- data.frame(Pred = 1, Prob)
  svar$Pred[Prob <= 0.5] <- 0


  conf_matr <- table(newdata$Spam, svar$Pred)

  ret_list <- list(Confusionmatr = conf_matr, Distancematr = D[1:10, 1:10], probabilities = Prob)

  return(ret_list)
}
```

1.3

```
# test
con_matr_k5_tst<- knearest(train, 5, test)$Confusionmatr
miss_class_k5_tst <- (con_matr_k5_tst[upper.tri(as.matrix(con_matr_k5_tst))] +
                        con_matr_k5_tst[lower.tri(as.matrix(con_matr_k5_tst))])/sum(con_matr_k5_tst)

#training
con_matr_k5_tr <- knearest(train, 5, train)$Confusionmatr
miss_class_k5_tr <- (con_matr_k5_tr[upper.tri(as.matrix(con_matr_k5_tr))] +
                       con_matr_k5_tr[lower.tri(as.matrix(con_matr_k5_tr))])/sum(con_matr_k5_tr)
```

1.4

```
conf_matr_test_1<- knearest(test, 1, test)$Confusionmatr

miss_class_test_1 <- (conf_matr_test_1[upper.tri(as.matrix(conf_matr_test_1))] +
                        conf_matr_test_1[lower.tri(as.matrix(conf_matr_test_1))])/sum(conf_matr_test_1)

conf_matr_train_1<- knearest(train, 1, train)$Confusionmatr

miss_class_train_1 <- (conf_matr_train_1[upper.tri(as.matrix(conf_matr_train_1))] +
                         conf_matr_train_1[lower.tri(as.matrix(conf_matr_train_1))])/sum(conf_matr_trai
```

1.5

```
#k=5
knn_form <- kknn(Spam ~ ., train, test, k=5)
Prob <- knn_form$fitted.values

svar <- data.frame(Pred = 1, Prob)
svar$Pred[Prob <= 0.5] <- 0

conf_matr <- table(test$Spam, svar$Pred)

miss_k5_tst <- (conf_matr[upper.tri(as.matrix(conf_matr))] +
                  conf_matr[lower.tri(as.matrix(conf_matr))])/sum(conf_matr)

#k=1
knn_form <- kknn(Spam ~ ., train, test, k=1)
Prob <- knn_form$fitted.values

svar <- data.frame(Pred = 1, Prob)
svar$Pred[Prob <= 0.5] <- 0

conf_matr_1 <- table(test$Spam, svar$Pred)
miss_k1_tst <- (conf_matr_1[upper.tri(as.matrix(conf_matr_1))] +
                  conf_matr_1[lower.tri(as.matrix(conf_matr_1))])/sum(conf_matr_1)
```

1.6

```r
probs <- knearest(train, 5, test)
probs <- probs$probabilities

PIs <- seq(0.05, 0.95, 0.05)

lasta <- list()
for(i in 1:length(PIs)){

  data_f <- data.frame( proba = 0, probs)
  data_f$proba[probs > PIs[i]] <- 1

  lasta[[i]] <- table(test$Spam, data_f$proba)
}
names(lasta) <- paste(PIs)


#Using kknn
probs_kk <- kknn(Spam ~ ., train, test, k=5)
probs_kk <- probs_kk$fitted.values

lasta_kk <- list()
for(i in 1:length(PIs)){

  data_f <- data.frame( proba = 1, probs_kk)
  data_f$proba[probs_kk <= PIs[i]] <- 0

  lasta_kk[[i]] <- table(test$Spam, data_f$proba)
}
names(lasta_kk) <- paste(PIs)

###ROC curve

#my own func
TPR_kn <- c()
FPR_kn <- c()
for(i in 1:length(lasta)){
  TPR_kn[i] <- lasta[[i]][2,2]/ sum(lasta[[i]][2,])
  FPR_kn[i] <- lasta[[i]][1,2]/sum(lasta[[i]][1,])
}

#kknn func
TPR_kk <- c()
FPR_kk <- c()
for(i in 1:length(lasta_kk)){
  TPR_kk[i] <- lasta_kk[[i]][2,2]/ sum(lasta_kk[[i]][2,])
  FPR_kk[i] <- lasta_kk[[i]][1,2]/sum(lasta_kk[[i]][1,])
}
```

3.1

```r
mylin=function(X,Y, Xpred){

  Xpred1 <- cbind(1,Xpred)
  X_ind <- cbind(1,X)

  Xpred1 <- as.matrix(Xpred1)
  X_ind <- as.matrix(X_ind)

  #Xpred1 <- model.matrix(~ ., data = Xpred)
  #X_ind <- model.matrix(~ ., data = X)

  beta <- solve(t(X_ind)%*%X_ind)%*%t(X_ind)%*%Y

  Ypred <- as.matrix(Xpred1)%*%beta
  return(Ypred)
}


myCV=function(X,Y,Nfolds){
  n=length(Y)
  p=ncol(X)
  set.seed(12345)
  ind=sample(n,n)
  sF=floor(n/Nfolds)
  MSE=numeric(2^p-1)
  Nfeat=numeric(2^p-1)
  Features=list()
  curr=0

  #we assume 5 features.
  for (f1 in 0:1)
    for (f2 in 0:1)
      for(f3 in 0:1)
        for(f4 in 0:1)
          for(f5 in 0:1){
            model= c(f1,f2,f3,f4,f5)
            if (sum(model)==0) next()
            SSE=0

            vad <- which(model == 1)

            first <- seq(1, n, sF)
            sec <- seq(0, n, sF)
            for (k in 1:Nfolds){

              if((n %% Nfolds != 0) & k == Nfolds ){
                i <- first[k]
                i <- n
              } else {
                i <- first[k]
                j <- sec[k+1]
              }


              #tar intervallet av de slumpade värdena
              indX <- ind[i:j]

              X_use <- X[-indX, vad]
              Y_use <- Y[-indX]
              X_val <- X[indX, vad]

              #featuresen är kolumnerna i modellen
              Ypred <- mylin(X = X_use, Y=Y_use, Xpred = X_val)
              Yp <- Y[indX]

              SSE=SSE+sum((Ypred-Yp)^2)
            }
            curr=curr+1
            MSE[curr]=SSE/n
            Nfeat[curr]=sum(model)
            nam <- colnames(X[,vad])
            Features[[curr]]=nam

          }
  plot(x = Nfeat, y = MSE,main = "MSE against number of features")
  i=which.min(MSE)
  return(list(CV=MSE[i], Features=Features[[i]]))
}
```

4.3

```
n <- dim(tecator)[1]
set.seed(12345)
id <- sample(1:n, floor(n*0.5))
train <- tecator[id,]
validation <- tecator[-id,]

#TRAINING
Moisture_tr <- as.numeric(train$Moisture)
Protein_tr <- as.numeric(train$Protein)

data <- data.frame(Protein_tr)
poly <- 6
for(i in 1:poly){
  data <- cbind(data,Protein_tr^i)
}
data <- data[,2:ncol(data)]

#simple linear model
M_1 <- lm(Moisture_tr ~ data[,1])

#Multiple linear models
M_list <- list(M_1)
```

```r
for(i in 2:6){
  M_list[[i]] <- lm(Moisture_tr ~ ., data[,1:i])
}

plot(Protein_tr, Moisture_tr)
points(Protein_tr, M_list[[6]]$fitted.values, col = "red")


#MSE
MSE_v <- c()
for(i in 1:6){
  MSE_v[i] <- sum(M_list[[i]]$residuals^2)/nrow(train)
}
names(MSE_v) <- c("M1", "M2", "M3", "M4", "M5", "M6")


#VALIDATION
Moisture_val <- as.numeric(validation$Moisture)
Protein_val <- as.numeric(validation$Protein)

data_val <- data.frame(Protein_val)
poly <- 6
for(i in 1:poly){
  data_val <- cbind(data_val,Protein_val^i)
}
data_val <- data.frame(data_val[,2:ncol(data_val)])

#for simple regression
X_M1 <- model.matrix(Moisture_val~ Protein_val.i , data= data_val)
fits <-  X_M1 %*% M_list[[1]]$coefficients
resid <- Moisture_val - fits


MSE_val <- sum(resid^2)/nrow(validation)
#for the rest
for(i in 2:6){
  X_rest <- model.matrix(Moisture_val ~ ., data= data_val[,1:i])
  fits <-  X_rest %*% M_list[[i]]$coefficients
  resid <- Moisture_val - fits
  MSE_val[i] <-  sum(resid^2)/nrow(validation)

}
```