TDDE01: Lab 2

A report by

Adam Nyberg
adany869

Assigment 1

The first assignment was to use cross-validation for feature selection on a linear model. The data used was Swiss which is a dataset that includes data about fertility and other socioeconomic indicators. 47 rows of data were used in this assignment.

The goal was to implement a R function that performs feature selection in linear regression by using k-fold cross-validaton. The R function depends on the following data:

- Y: vector with fertility rates
- X: matrix with the data of all other columns (X is assumed to have 5 columns)
- Nfolds: number of folds used in the cross-validation

The first thing was to create the linear regression function and that was made with Ordinary least squares (OLS) regression with help of the following formulas.

$$\hat{\beta} = (X^T X)^{-1} X^T y \,.$$

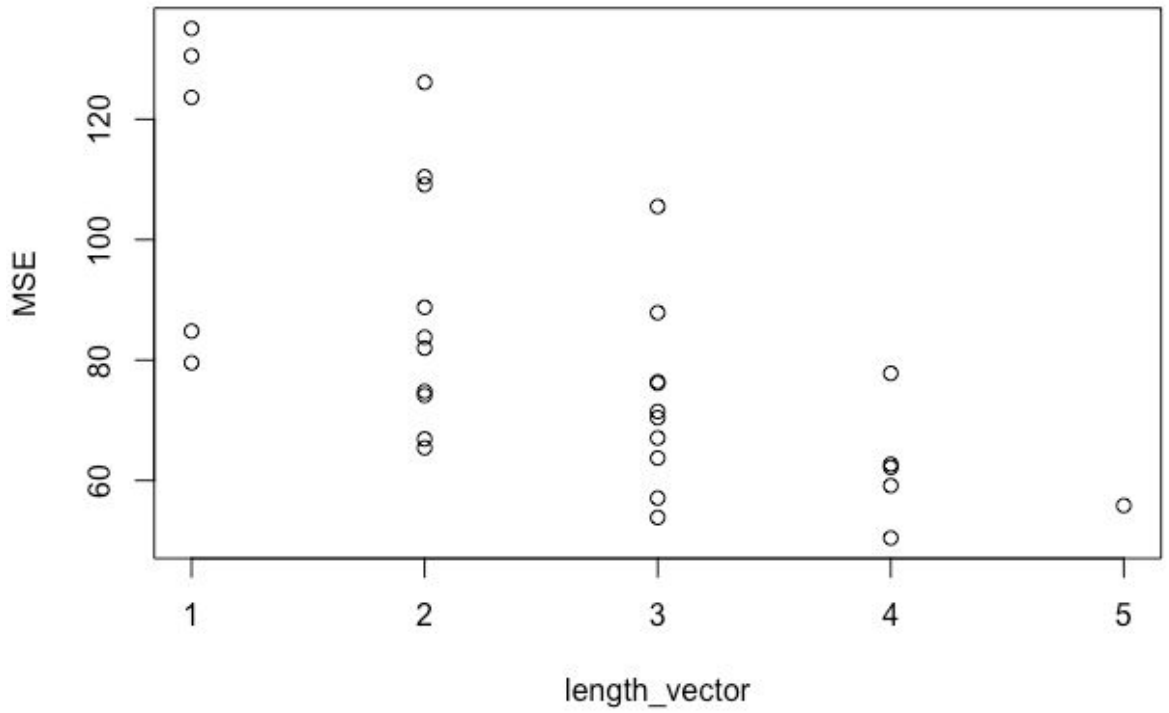
$$\hat{y} = X\hat{\beta} = Py,$$

I now have a linear model to use for prediction. The first thing I do now is to shuffle the data to remove some possible bias later when I do my K-fold divid. Next I create every subset of the features X. That gives me 31 different feature sets.

For everyone of those feature sets I split the data up into K folds. The cross-validation is done by iterating K times and run the linear model every time with a new fold as validation and all other folds as training. In our case K was set to 5. Then I sum (CV score) the square of the error where error is the difference between the real Y and the predicted Y. I then save the CV score together with the information about what features used in a list.

When I now have done this for every subset of the features I can now use BSS (best subset selection) which means to select the feature subset with the lowest CV score.

I also plot CV score against number of features.



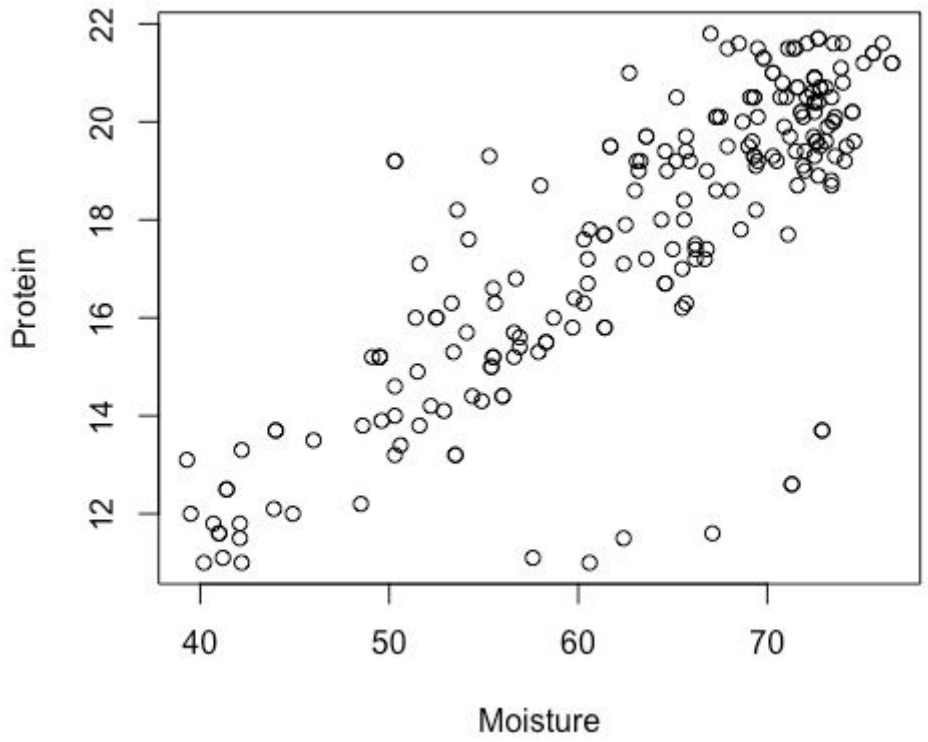
By looking at the plot and examinating the result of my function I can see that the best subset of features contains 4 features. When examining further those features are “Agriculture, Education, Catholic, Inf.Mortality” and excludes “Examination”. To me this makes sense because being in the army doesn’t affect the fertility rate that much.

Assigment 2

In this assignment I were given a dataset with information about foods properties such as Fat, Protein, Moistrue and 100 properties of infrared information (channel 1-100). The assignment was divided into a number of small problems to solve with this data.

2.1

This problem was to decide if there was a correlation between Moistrue and Protein that could be described well with a linerar model.



By looking at the plot I can assume that a linear model could describe this data well.

2.2

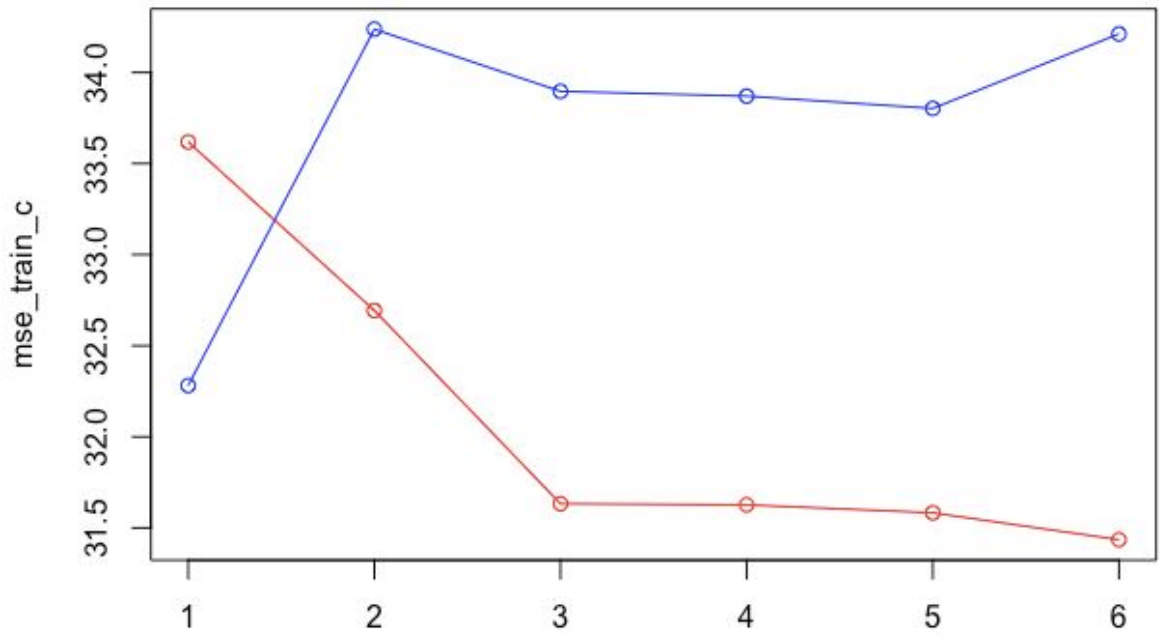
In this problem I was asked to provide a model Mi of the Moisture and expected Moisture is a polynomial function of Protein including the polynomial terms up to power i. So that M1 is a linear model, M2 is a quadratic model and so on. I decided to create the model with the following code.

```
M = lm(Moisture ~ poly(Protein, i), data=train)
```

The reason MSE is appropriate criterion when fitting this model is because a few outliers have a bigger effect and in turn makes them more important. Without squaring the error the outliers would essentially get ignored.

2.3

This problem was to create Mi where i goes from 1 to 6 and examine which model fitted the data best. The data was divided by half into a training and a testing set and then the MSE of all models was computed and shown in the plot below.



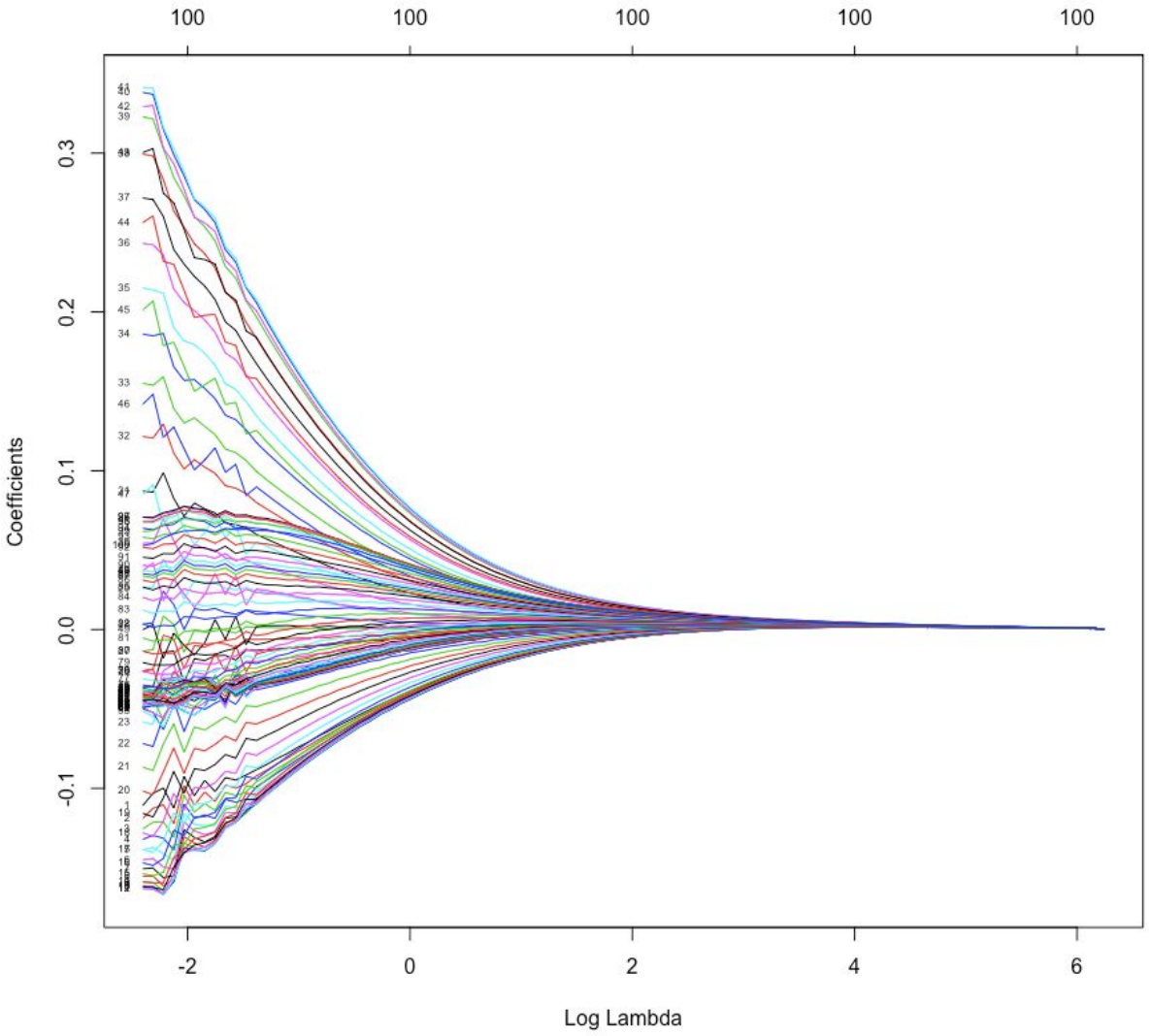
The red points are the training set and from this I can see that using a model with higher polynomial gives a lower MSE. When I look at the blue points (testing set) I see the opposite and here the linear model (M1) performs the best which seems logical according to the data studied in 2.1. This is a clear example of bias-variance tradeoff where using a too high polynomial will create overfitting.

2.4

I was now asked to perform feature selection of a linear model where Fat is response and channel 1-100 is features. The function stepAIC was used and returned 63 features.

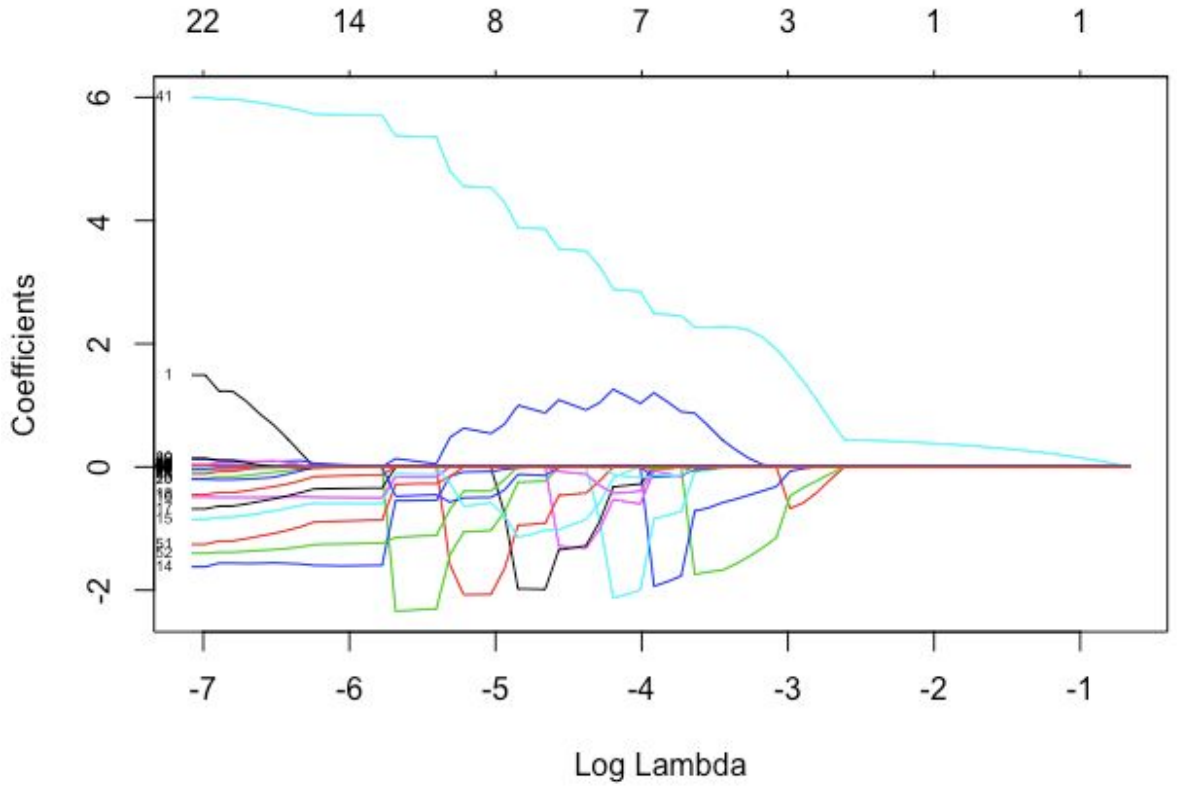
2.5

In this problem I was to performe a ridge regression and plot how the penalty factor lambda affected the coefficients. When looking at the plot below we can see that all the coefficients goes to 0 as lambda increases. As we can see this will surpress all channels equally and not be able to promote some features. This is a characteristic of ridge regression that is good to know.



2.6

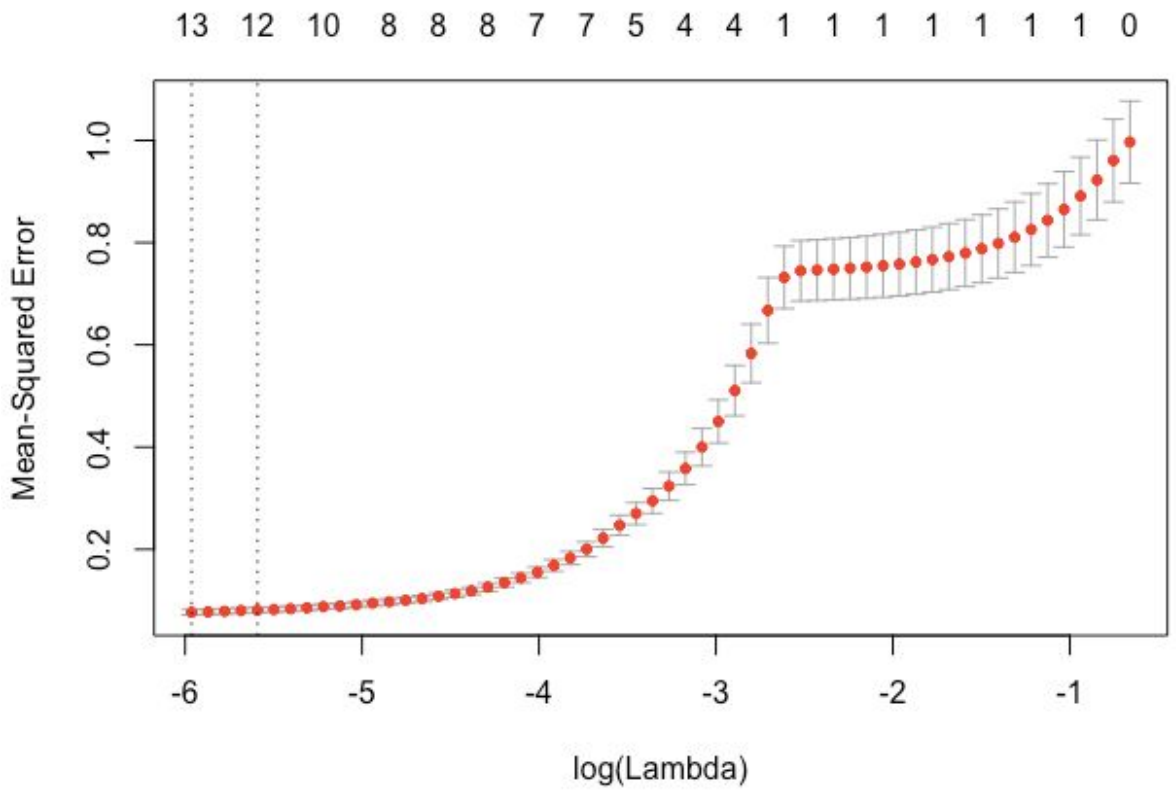
In this problem I used the LASSO insted of the ridge regression used in 2.5 and plotted the results.



As we can see in this plot LASSO will basically promote and turn off features. In my case feature 41 plays a big role and some features are turned off even on very low lamdbas. So LASSO actually does feature selection.

2.7

In this problem I was to use cross-validation to find the optimal LASSO model and the corresponding lamdba. When comparing MSE to lambda I got the following plot.



I can see in this plot that a lower lambda value gives a lower MSE. The optimal model have 13 features selected and the optimal lambda was 0.002571916.

2.8

In this final problem I was to compare the results from 2.4 and 2.7.

In 2.4 I used stepAIC for feature selection and in 2.7 I used LASSO. When looking at the MSE to evaulate how they performed we can conclude that LASSO was a lot better. Interesting to note that LASSO only selected 13 features compared to stepAIC that selected 63. From that we can make the conclusion that it's not always better with more features but what matters is to select the best features.

I'd also assume that LASSO works better than stepAIC on datasets where p >> n.

```
setwd("~/code/skola/tdde01/adam/lab2")

data = read.csv("tecator.csv")

set.seed(12345)


n=dim(data)[1]
id=sample(1:n, floor(n*0.5))
train=data[id,]
test=data[-id,]

plot(data$Moisture, data$Protein, ylab = "Protein", xlab = "Moisture")
# Yes, the data could be described by a linear model

mse_train_c = c()
mse_test_c = c()

for(i in 1:6) {
  M = lm(Moisture ~ poly(Protein, i), data=train)

  mse_train = mean(M$residuals^2)
  mse_train_c[i] = mse_train

  pred_test = predict(M, test)

  diff = pred_test-test[, "Moisture"]

  mse_test = mean(diff^2)
  mse_test_c[i] = mse_test

}

plot(seq(1,6), mse_train_c, col = "red", ylim=range(min(mse_train_c, mse_test_c),
max(mse_train_c, mse_test_c)), type="o")
points(seq(1,6), mse_test_c, col="blue", type="o")

#2.4 63 variables # Residual standard error: 1.107
library(MASS)
fat_m = lm(Fat ~., data=data[, 2:102])
step = stepAIC(fat_m)



#2.5 coefficients decrease with bigger lamnda, panalty effect
library(glmnet)
covariates=scale(data[,2:101])
response=scale(data[, 102])
r_m=glmnet(as.matrix(covariates), response, alpha=0,family="gaussian")
plot(r_m, xvar="lambda", label=TRUE)



#2.6 optimizes better by removing channels
```

```
l_m=glmnet(as.matrix(covariates), response, alpha=1,family="gaussian")
plot(l_m, xvar="lambda", label=TRUE)



#2.7 Choose 13 variables
l_cv=cv.glmnet(as.matrix(covariates), response, alpha=1,family="gaussian")
l_cv$lambda.min
plot(l_cv)
coef(l_cv, s="lambda.min")



#2.8
#2.4 MSE = 32, 2.7 MSE < 1 Much better



setwd("~/code/skola/tdde01/adam/lab2")

//linear regression
mylin=function(X,Y, Xpred){
  Xpred1=cbind(1,Xpred)
  X1 = cbind(1,X)
  beta = solve((t(X1) %*% as.matrix(X1))) %*% (t(X1)%*%Y)
  Ypred1=Xpred1%*%beta
  return(Ypred1)
}

myCV=function(X,Y,Nfolds){
  n=length(Y)
  p=ncol(X)
  set.seed(12345)
  ind=sample(n,n)
  X1=X[ind,]            # Shuffle
  Y1=Y[ind]             # Shuffle
  sF=floor(n/Nfolds)    # Rows per folds
  MSE=numeric(2^p-1)    # Empty vector with length 31
  Nfeat=numeric(2^p-1)  # Empty vector with length 31
  Features=list()
  curr=0

  #we assume 5 features.
  feats = c("Fertility", "Agriculture", "Examination", "Education", "Catholic",
"Infant.Mortality")
  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
```

```
            Xcol=X1
            for(i in 1:5) {
              if(model[i] == 0) {
                Xcol=Xcol[, !(colnames(Xcol) %in% c(feats[i+1])), drop = F]
              }
            }

            for (k in 1:Nfolds){
              #MISSING: compute which indices should belong to current fold
              start_i = k*9-8
              if(start_i==0) start_i=1
              end_i = k*9
              if(end_i>n) end_i=n

              x_test = Xcol[start_i:end_i,]
              y_test = Y1[start_i:end_i]

              x_train = Xcol[-(start_i:end_i),]
              y_train = Y1[-(start_i:end_i)]

                              #MISSING: implement cross-validation for model with features
in "model" and iteration i.
              m = mylin(x_train, y_train, x_test)

                              #MISSING: Get the predicted values for fold 'k', Ypred, and
the original values for fold 'k', Yp.
              Yp = y_test
              Ypred = m
              SSE=SSE+sum((Ypred-Yp)^2)
            }
            curr=curr+1
            MSE[curr]=SSE/n
            Nfeat[curr]=sum(model)
            Features[[curr]]=model

          }
  #MISSING: plot MSE against number of features
  for(i in 1:curr) {
    length_vector[i] = sum(Features[[i]])
  }

  plot(length_vector, MSE, xlab="# of features", ylab="CV score")

  i=which.min(MSE)
  return(list(CV=MSE[i], Features=Features[[i]]))
}

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