Lab1 group 1

Henrik Karlsson, Milda Poceviciute and Ugurcan Lacin

11/13/2017

Assignment 1

Part 1 and 2

The task in assignment 1 is to build a spamfilter with the help of KNN-classification techniques. A data file is provided, containing 48 different columns with word frequencies and a variable telling if the mail was classified as spam or not where each row represents one email. The spam classification was made manually.

First step is to split the data in a training and test data set. The training set is being used to train the model and the test set is used to evaluate how good the classification is.

Next step is to start build the knearest neighbor function. When comparing words it's useful to compare the distances inbetween the words, therefore the function starts to compute a distance matrix. The matrix compares the how far each data point in the test data is to the train data, which enables you find out k closest neighbor, points in train data. The distance matrix is computed by:

$$c(X,Y) = \frac{X^T Y}{\sqrt{\sum_i X_i^2}\sqrt{\sum_i Y_i^2}}$$

```
# Import data
suppressWarnings(suppressMessages(library(readxl)))
data <- as.data.frame(read_xlsx("spambase.xlsx"))

# Devide data into testing and training sets

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

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)

# implemented steps ii)-iv)
  Xn=Xn/matrix(sqrt(rowSums(Xn^2)), nrow=n1, ncol=p-1)
  C <- X%*%t(Xn)
  D <- 1 - C
  Prob <- c()
  for (i in 1:n2 ){
```

```
# the computed distance matrix is used to find
# which observations are the nearest neighbors to case #i
      neighbours <- order(D[,i])[1:k]

# Prob[i] is derived by using the target values of the nearest neighbors
     is_spam <- data[,49][neighbours]
     Prob[i] <- length(is_spam[is_spam == 1])/ length(is_spam)
  }
  return(Prob)
}
```

Once the train data neighbors are known for each point in test data, a probability is computed to estimate the point in test data. We know the true classification for train data which is used for the computation. The probability is computed by number of nearest neighbor that is classified as spam divided by number of neighbors.

$$p(C_i) = \frac{N_i}{N}$$

The result of the function will be a vector with probabilities of being classified as spam for each data point in test data.

A function to calculate confusion matrix is built, called confusion_matrix, which crosstabulates the predicted classifications to the true classification.

```
# Function that builds the confussion matrix
confusion_matrix <- function(predicted, actual) {
    df <- data.frame("Pred" = predicted, "Real" = actual)
    confusion_mat <- with(df, table(Real, Pred))
    totals <- matrix(c(sum(confusion_mat[1, ]), sum(confusion_mat[2, ])), nrow = 2, ncol = 1)
    confusion_mat <- cbind(confusion_mat, totals)
    colnames(confusion_mat) <- c("0", "1", "Total")
    return(confusion_mat)
}
```

Part 3 and 4

k = 5 and k = 1, knearest model

The knearest model (from the section **Parts 1 and 2**) is fitted using training data, the return is the probability of the k-nearest neighbour being a spam. Then the clasification rule is applied: $\hat{Y} = 1$ *if* $p(Y = 1|X) = 0.5$ *else* $\hat{Y} = 0$.

```
# k = 5
k5_result <- knearest(train, 5, test)
# Classification rule applied
k5_predict <- ifelse(k5_result>0.5, 1, 0)
# Confusion matrix
confusion_mat_5 <- confusion_matrix(k5_predict, test[,49])
# Misclassification rate
misclass_rate_5 <- 100 * sum(test[,49] != k5_predict)/nrow(test)

# k = 1
k1_result <- knearest(train, 1, test)
# Classification rule applied
k1_predict <- ifelse(k1_result>0.5, 1, 0)
```

```
# Confusion matrix
confusion_mat_1 <- confusion_matrix(k1_predict, test[,49])
# Misclassification rate
misclass_rate_1 <- 100 * sum(test[,49] != k1_predict)/nrow(test)
```

Results:

- The confusion matrix when k = 5:

```
##     0   1 Total
## 0 695 242   937
## 1 193 240   433
```

- The misclassification rate when k = 5:

```
## [1] 31.75182
```

- The confusion matrix when k = 1:

```
##     0   1 Total
## 0 639 298   937
## 1 178 255   433
```

- The misclassification rate when k = 1:

```
## [1] 34.74453
```

As it can be seen, when we decrease K value it increases missclassification rate which means more values are wrongly predicted. We understand from this that we have to be careful when choosing k. For example, choosing a value of 1 would only take into account the nearest neighbor. However, when we choose the value of 5, we have a more consistent result because we consider the nearest 5 neighbors and so we have a better interpretation of neighbors. But increasing this value too much may not give a good result as well.

Part 5

k = 5 kknn model

Thereafter, the kknn package is loaded and the results are being compared between function knearest and kknn. The code used for this analysis is:

```
suppressWarnings(suppressMessages(library(kknn)))
# k = 5
kknn5 <- kknn(formula = Spam~.,train = train, test = test, k=5)
# Classification rule applied
kknn5_pred <- ifelse(kknn5$fitted.values>0.5, 1, 0)
# Confusion matrix
confusion_mat_kknn5 <- confusion_matrix(kknn5_pred, test[,49])
# Misclassification rate
misclass_rate_kknn5 <- 100 * sum(test[,49] != kknn5_pred)/nrow(test)
```

Results:

- The confusion matrix, k = 5:

```
##     0   1 Total
## 0 640 297   937
## 1 177 256   433
```

- The misclassification rate, k = 5:

```
## [1] 34.59854
```

Comparison of the four models

The *knearest* function provides better prediction with k = 5 compared to k = 1 based on the misclassification rates (31.75182 and 34.74453 respectively). The *kknn* function has a smaller misclassification (34.59854) than *knearest* with k=1, but performs worse than *knearest* function with k=5.

Part 6

Now, We will compare knearest() and kknn() functions with K=5 and will classify the test data by using following principle:

$$\hat{Y} = 1 \;\; if \;\; p(Y = 1|X) > \pi, \;\; otherwise \;\; \hat{Y} = 0, \;\; where \;\; \pi = 0.05, 0.1, 0.15, ..., 0.95.$$

The code used to compute sensitivity and the specificity values for the two methods, and to generate the plots is:

```
ROC=function(Y, Yfit, p){
  m=length(p)
  TPR=numeric(m)
  FPR=numeric(m)
  for(i in 1:m){
    t=table(Yfit>p[i], Y)
    TPR[i]= t[2,2]/(t[1,2]+t[2,2])
    FPR[i]= t[2,1]/(t[2,1]+t[1,1])
  }
  return (list(TPR=TPR,FPR=FPR))
}

pi <- seq(0.05,0.95, by=0.05)
TPR_FPR <- ROC(test[,49],k5_result,pi)
TPR_FPR_kknn5 <- ROC(test[,49],kknn5$fitted.values,pi)

# Specificity = 1 - FPR, knearest with k=5
spec_kn <- as.vector(unlist(lapply(TPR_FPR$FPR, function(x){1-x})))

# Sensitivity = TPR, knearest with k=5
sens_kn <- as.vector(unlist(TPR_FPR$TPR))

# Specificity = 1 - FPR, kknn with k=5
spec_kknn <- as.vector(unlist(lapply(TPR_FPR_kknn5$FPR, function(x){1-x})))

# Sensitivity = TPR, kkknn with k=5
sens_kknn <- as.vector(unlist(TPR_FPR_kknn5$TPR))

library(ggplot2)
plot_data1 <- as.data.frame(TPR_FPR)
plot_data2 <- as.data.frame(TPR_FPR_kknn5)
names(plot_data2) <- c("TPR2", "FPR2")
df <- cbind(plot_data1,plot_data2)
plot <- ggplot2::ggplot(df)+
        geom_line(aes(x=df$FPR, y = df$TPR, colour="blue"))+
        geom_point(aes(x=df$FPR, y = df$TPR, colour="blue"))+
        geom_line(aes(x=df$FPR2, y = df$TPR2, colour="red"))+
        geom_point(aes(x=df$FPR2, y = df$TPR2, colour="red"))+
```

```
        scale_color_manual(labels = c("knearest", "kknn"), values = c("blue", "red"))+
        labs(x="FPR",y="TPR", ttile="ROC curve",color = "Model")+
        geom_abline(intercept = 1, slope = -1)
```

Results

- When knearest model was used with k=5, these values were obtained:
- Sensitivity and average sensitivity

```
##  [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
```

```
## [1] 0.5131883
```

- Specificity and average specificity

```
##  [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
```

```
## [1] 0.7166208
```

- When kknn model was used with k=5, these values were obtained:
- Sensitivity and average sensitivity

```
##  [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
```

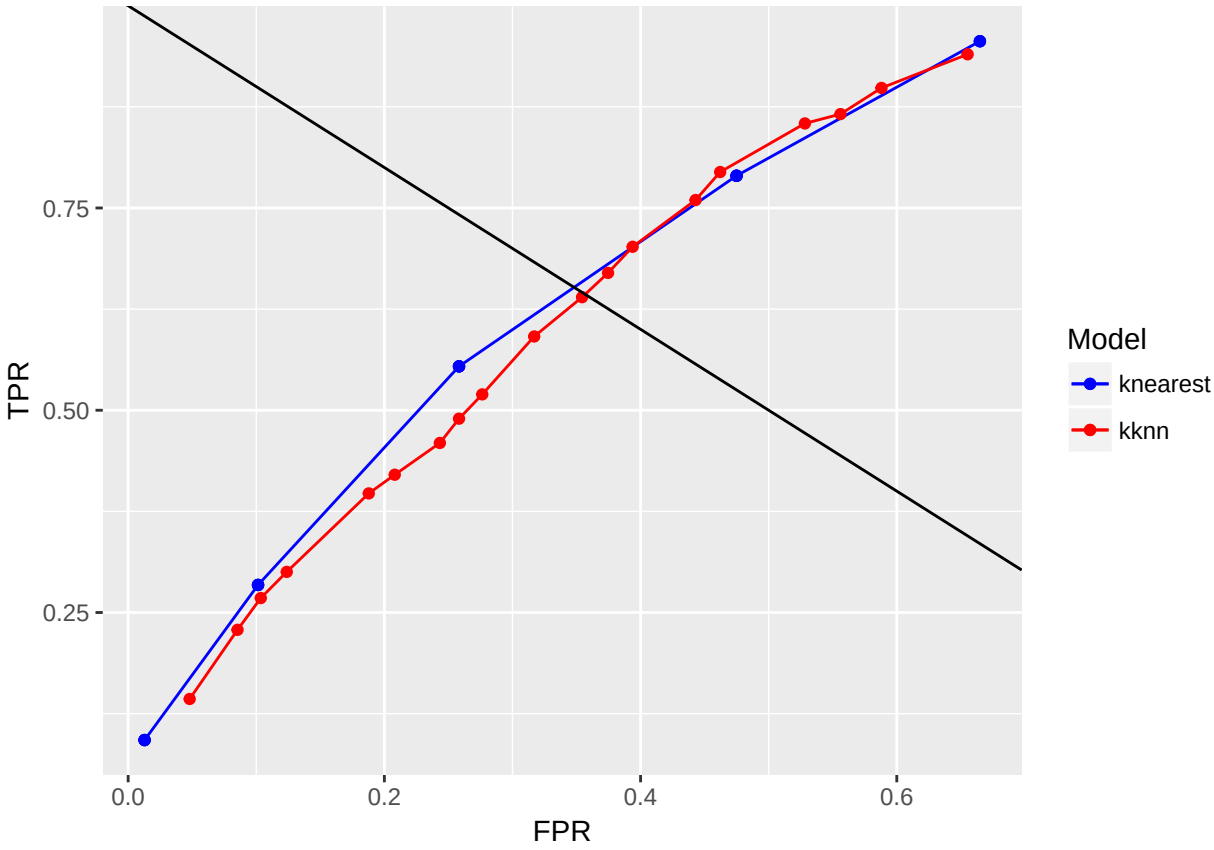
```
## [1] 0.5759086
```

- Specificity and average specificity

```
##  [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
```

```
## [1] 0.6733135
```

- ROC curve functions



Sensitivity is the probability of detecting the positives (in our case it is a probability of correctly detecting non-spam emails). Specificity is the probability of detecting the negatives (in our case it is a probability of correctly detecting spam emails). On average knearest method has lower sensitivity and higher specificity compared to kknn model. This implies that kneartest model is on average better at classifying spams than kknn model. For the theoretical best model area under ROC curve is 1, if the area under ROC curve is 0.5, this indicates that the model is no better than random guessing. When we analyze the graph, we see that the knearest function, indicated by the blue line, has more space than the package of the kknn under the curve. Hence, knearest function seem to be a better model in this case.

Assignment 3

The task is to build a Cross Validation for a linear model. In the code, it's assumed that there are 5 covariates the model.

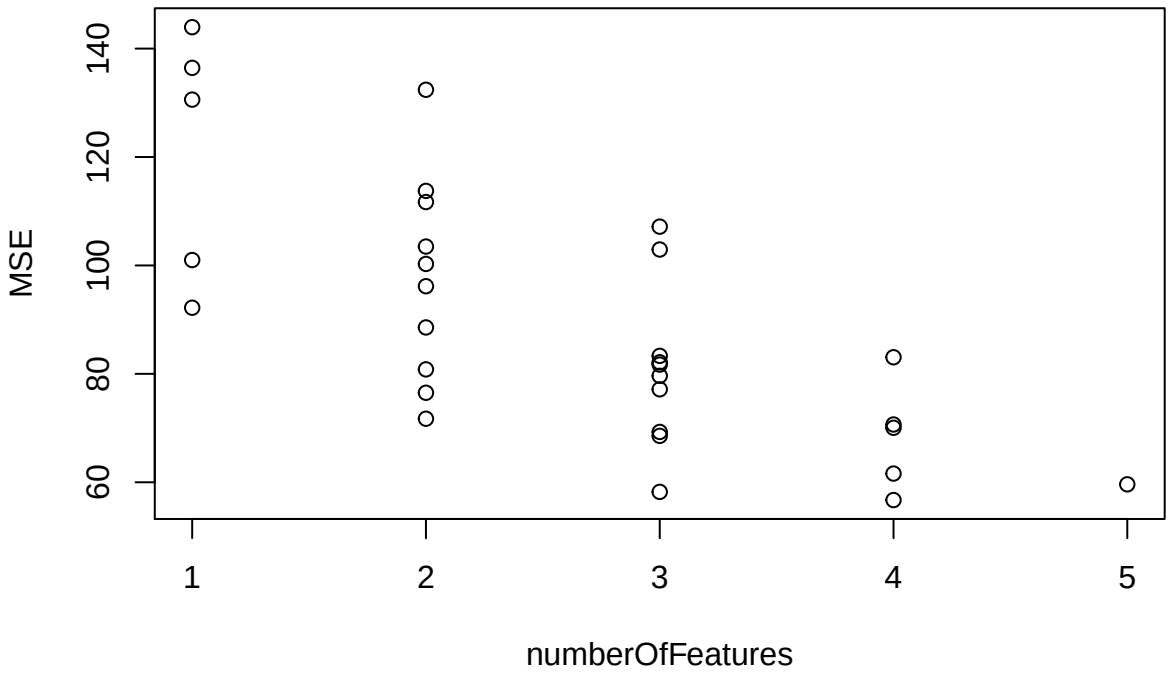
Swiss data from R is being used in this assigment.

mylin functions basically calculates linear model for given X and Y matrices and find beta values. After that, it multiply beta and test X matrix and return it.

```
#linear regression
mylin = function(X, Y, Xpred) {
  Xpred1 = cbind(1, Xpred)
  X <- cbind(1, X)
  beta <- solve(t(X) %*% X) %*% t(X) %*% Y
  Res = Xpred1 %*% beta
```

```
  return(Res)
}

myCV = function(X, Y, Nfolds) {
  n = length(Y)
  p = ncol(X)
  set.seed(12345)
  ind = sample(n, n)
  X1 = X[ind, ]
  Y1 = Y[ind]
  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

            for (k in 1:Nfolds) {
              startIndice <- (k * sF) - sF + 1
              endIndice <- 0
              if (k == Nfolds) {
                mod <- nrow(X1) %% Nfolds
                endIndice <- (k * sF) + mod
              } else{
                endIndice <- k * sF
              }

              selectedFeatures <- which(model == 1)
              Xvalidate <- as.matrix(X1[,selectedFeatures])
              Xvalidate <- as.matrix(Xvalidate[startIndice:endIndice,])
              Yvalidate <- as.matrix(Y1[startIndice:endIndice])

              Xtrain <- X1[-c(startIndice:endIndice), selectedFeatures]
              Ytrain <- Y1[-c(startIndice:endIndice)]
              Ypred <- mylin(X = Xtrain, Y = Ytrain, Xpred = Xvalidate)

              SSE = SSE + sum((Ypred - Yvalidate) ^ 2)
            }
            curr = curr + 1
            MSE[curr] = SSE / n
            Nfeat[curr] = sum(model)
            Features[[curr]] = model
```

```
        }
  numberOfFeatures <-lapply(Features,FUN = function(x) {return(length(which(x == 1)))})
  plot(x = numberOfFeatures, y = MSE)
  i = which.min(MSE)
  Features=Features[[i]]
  Features2 <- which(Features == 1)
  return(list(CV=MSE[i],
              Features=colnames(X1)[Features2],
              Features = Features))
}

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




```
print(res)
```

```
## $CV
## [1] 56.72245
##
## $Features
## [1] "Agriculture"      "Education"        "Catholic"
## [4] "Infant.Mortality"
##
## $Features
## [1] 1 0 1 1 1
```

In conclusion, it can be seen that the MSE tends to decrease as the number of features increases. But that does not mean that the highest number of features gives the lowest MSE rate. By evaluating the results it can be seen which features bring the best model. According to this result, one can conclude that the Examination feature doesn't influence the success of the model well.

Assignment 4

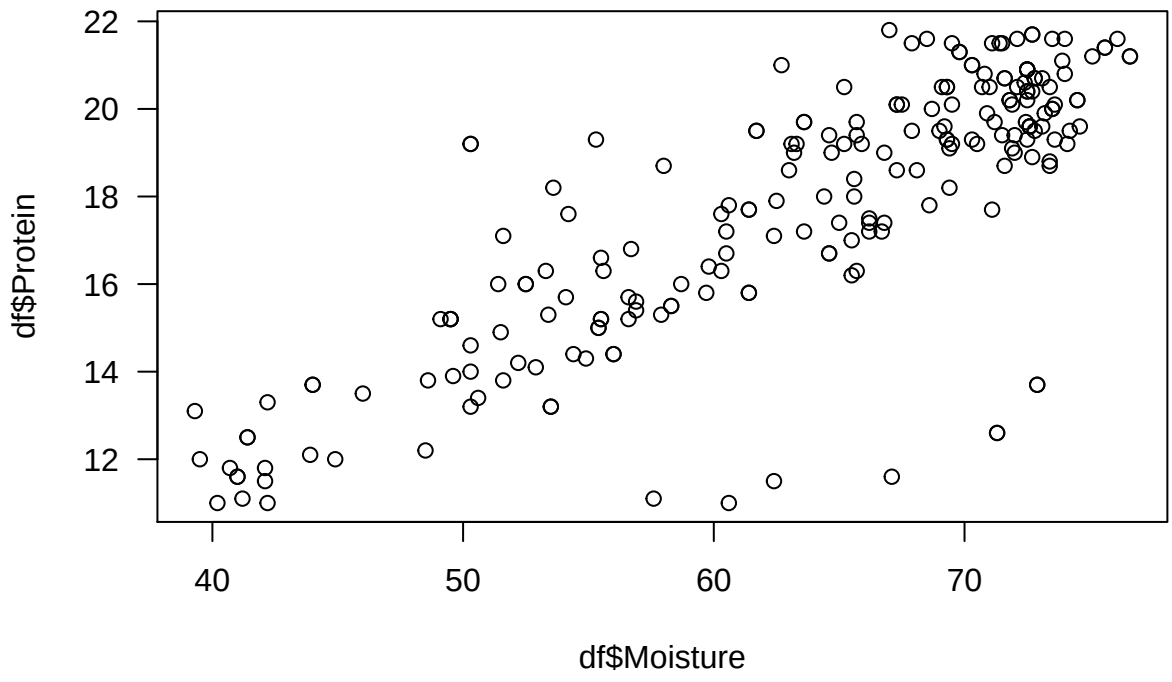
Part 1

```
df <- as.data.frame(read_xlsx("tecator.xlsx"))
```

When plotting Protein to Moisture, it seems to exist a linear pattern in between the two variables. There are a few outliers that by a peer view of the data, doesn't seem to follow the same pattern as the rest of the variables. However, this is something that not will be further investigated in this exercise.

A linear model seem to be reasonable to fit to this data.

```
plot(df$Moisture, df$Protein, las = 1)
```



Part 2

MSE is a suitable way of comparing the models because it shows the average squared distance that the regression model has from the real values. Squaring makes the large distrances have more weight than small ones. MSE incorporates the variance and bias of the estimator as it shows the averaged out SSE for each model. In our case, the models will have explanatory variables highly correlated to each other and MSE can capture the increasing variance as the complexity of the models increases.

The data is divided into test and training data, where we split 50% as train and 50% as test.

```
set.seed(12345)
index <- sample(nrow(df), floor((nrow(df)*0.5)))
train <- df[index, -grep("Channel", names(df))]
test <- df[-index, -grep("Channel", names(df))]
```

A function is defined to build a the polynomial function with the polynomial i

```
model <- function(i, data){
    m <- lm(Moisture ~ poly(Protein, i, raw = TRUE), data)
```

```
    return(m)
}
```

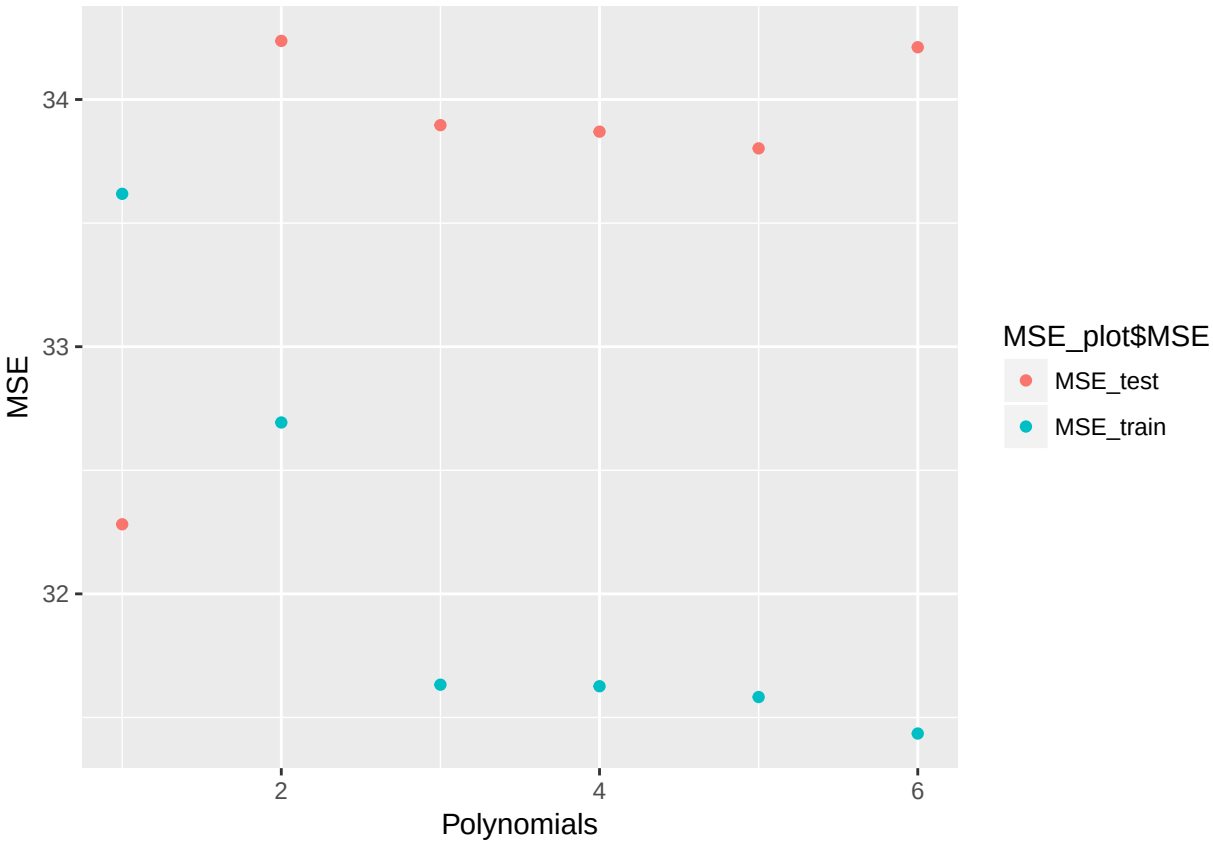
Another function is build to compute the Mean Square Error for each polynomial function for both the test and training data

```
generate_function <- function(train, test){
    output <- vector()

    for(i in 1:6){
        m <- model(i, train)
        pred <- predict(m, test)
        MSE_train <- mean(residuals(m)^2)
        MSE_test <- mean((pred - test$Moisture)^2)

        output <- rbind(output,c(MSE_train, MSE_test))
    }
    output <- cbind(output, c(1:6))
    colnames(output) <- c("MSE_train", "MSE_test", "model")

    return(output)
}
```

Part 3

We combine the two models to compute the MSE score and plot the MSE for each data set, train and test.

```
MSE <- generate_function(train, test)

suppressWarnings(suppressMessages(library(tidyverse)))
MSE_plot <- MSE %>%
    as.data.frame(.) %>%
    gather(MSE, value, -model) %>%
    mutate(MSE = as.factor(.$MSE))

ggplot(MSE_plot, aes(x = model, y = value)) +
    geom_point(aes(color = MSE_plot$MSE)) +
    ylab("MSE") +
    xlab("Polynomials")
```



From the plot, it can be interpreted that our train data goes from the first polynomial (linear regression) with a high bias and low variance towards the 6th polynomial with low bias and high variance in. The MSE is reduced for each added polynomial in the training data with the risk of overfitting. If the MSE for training data would be 0, the curve would have an perfect fit the data and the model would have captured the error term in the training data, and the model would be overfitted.

If we compare the MSE values for the test data, the data that haven't been used when the model have been trained, we can see that the lowest MSE in the first polynomial, the linear regression. It seems like the additional polynomials have captured more of the random error in the train data than the actual underlying pattern.

The best model is the linear regressions, which makes sense since when we first plotted the variables we could see a linear relationship.

The graph below illustrates that the more parameters are included (the higher polynomial is used) when modelling traning data, a smaller MSE is received. However, when the same models are used on the testing data, it is obvious that the performance decreases (MSE increases) as the number of the elements are increased in the regression model. So considering the validation data, we should choose the linear model as it produces the smallest risk (MSE). But considering Training models, the model of 6 degree polynomial regression has the lowest risk. Models of 6 degree polynomial regression has low bias as they have high complexity. The Variance in Training model is decreasing as model gets more complex, but for the validation models variance is increasing as models get more complex (MSE is increasing). The variance of the models is high, due to a high correlation between the independent variables.

Part 4

Now we try to predict the fat level using the 100 infrared absorbance spectrum variables.

Here an analytical method, stepwise AIC, is being used on a linear regression for finding the optimal amount of parameters for the model.

```
suppressWarnings(suppressMessages(library(MASS)))
df4 <- df[, 2:102]
mod <- lm(Fat ~., df4)
aic <- stepAIC(mod, direction = "both")
```

The stepwise AIC model selects 64 out of the 100 variables for the model prediction.

```
length(aic$coefficients)
```

```
## [1] 64
```

Part 5

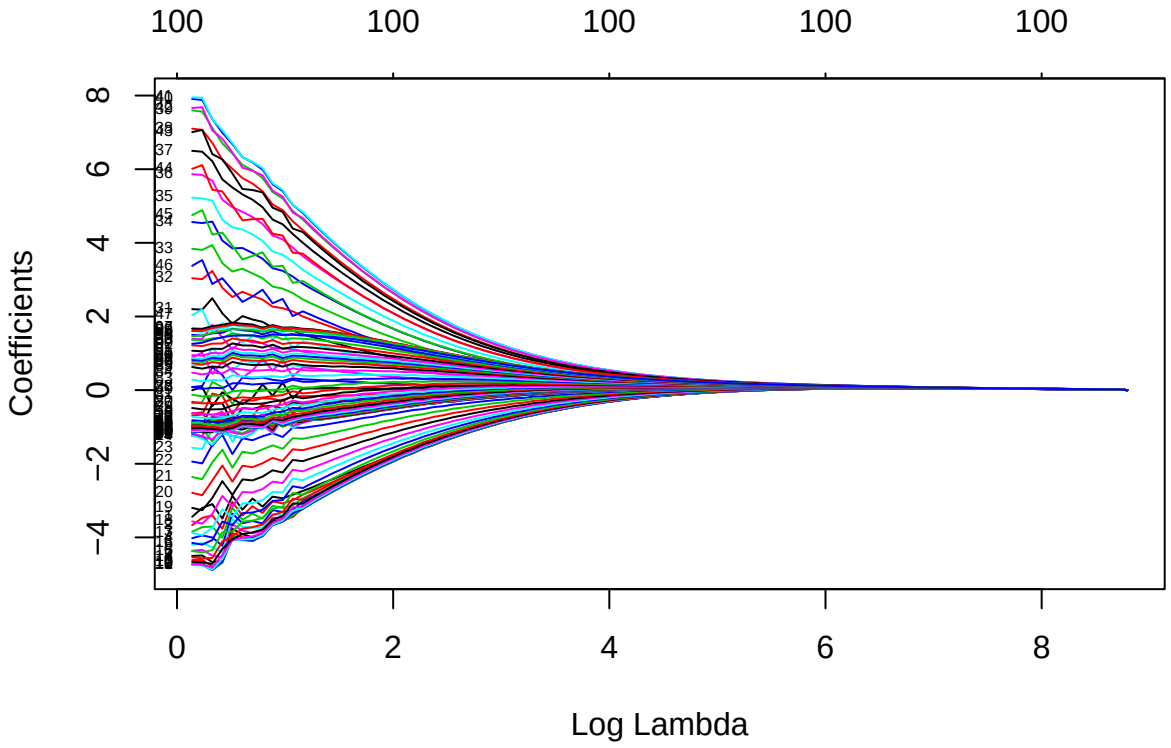
Instead of using the stepAIC we can use a ridge regression to penalize each variable to avoid overfitting of the model. The loss function for the ridge regression is defined as:

$$\lambda * \sum w_j^2$$

The exponential of the w values makes the loss function to penalize outliers hard. When the Lambda is 0 the ridge regression is a linear regression and as lambda increases the coeficients converging towards zero and have no effect on the regression. Each line represents one parameter and all parameters are kept in the model compared to stepAIC or Lasso which removes variables.

The acceptance region for the parameter being considered 0 for exponential loss functions is circular. That is causing the coeficents to smoothly decrease as the log lambda increases. This circular acceptance region makes the parameters w go towards 0 but not exactly zero.

```
suppressWarnings(suppressMessages(library(glmnet)))
ridge <- glmnet(x = as.matrix(df4[,1:100]),
                y = as.matrix(df4[, 101]),
                alpha = 0,
                family = "gaussian")
plot(ridge, xvar = "lambda", label = TRUE)
```



Part 6

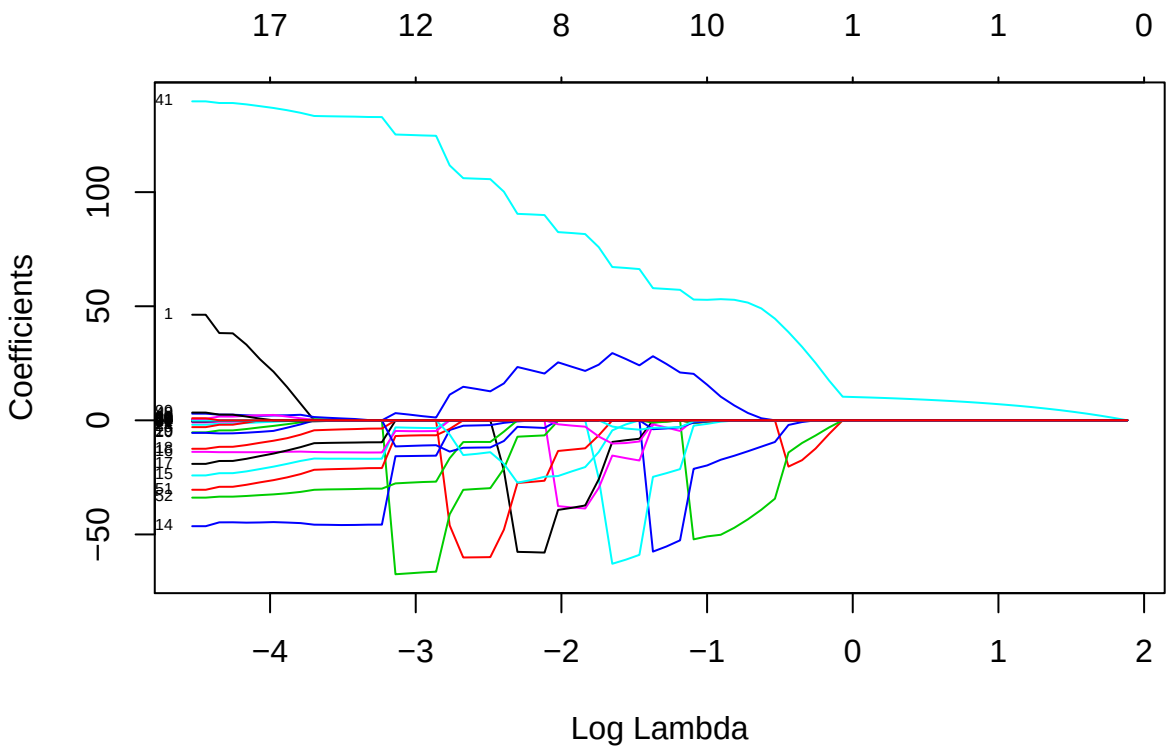
The lasso regression uses the *absolut* error of w and is calculated by:

$$\lambda * \sum_{j=1}^{p} |w_j|$$

The lasso regression is widely used when the number of parameters exceed the number of observations.

Instead of using ordinary lasso, we add cross validation to the lasso model to improve it. In this case, we consider $\lambda = 0$ among other λ options. When $\lambda = 0$, the Lasso regression is equal to a normal linear regression with OLS estimation. The purpose of OLS estimation is minimize the SSE, which is used for calculating MSE, therefore it's intuitive that the lowest MSE is the linear regression including all parameters. However, this is minimizing the MSE and not necessary the best for predicting the new data.

```
lasso <- glmnet(x = as.matrix(df4[,1:100]),
                y = as.matrix(df4[, 101]),
                alpha = 1, family = "gaussian")
plot(lasso, xvar = "lambda", label = TRUE)
```



Part 7

Instead of using ordinary lasso, we add cross validation to the lasso model to improve it. In this case, the loss function is forced to include $\lambda = 0$ which implies that the entire loss function is cancelled out.

```
CVlasso <- cv.glmnet(x = as.matrix(df4[,1:100]),
                     y = as.matrix(df4[, 101]),
                     alpha = 1,
                     lambda = seq(0, 100, by = 0.05),
                     family = "gaussian")
CVlasso$lambda.min
```

```
## [1] 0
```

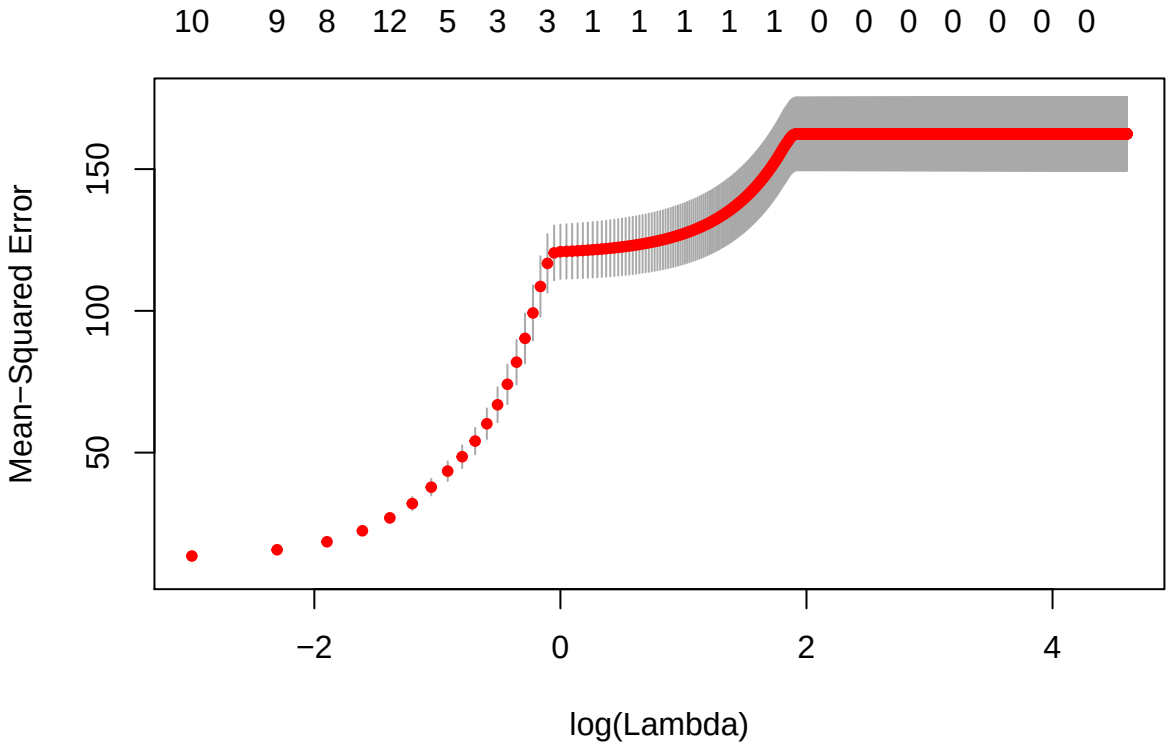
```
coef(CVlasso, s = "lambda.min")
```

```
## 101 x 1 sparse Matrix of class "dgCMatrix"
##                        1
## (Intercept)  13.4057080
## Channel1    -28.2044200
## Channel2    165.3605940
## Channel3      4.2183367
## Channel4      2.1017688
## Channel5      0.5031147
## Channel6     -0.1696027
## Channel7     -0.7735280
## Channel8     -0.5701405
## Channel9     -2.9892088
## Channel10   -71.4211306
## Channel11    -5.3569710
## Channel12    -6.8424803
```

```
## Channel13   -59.8436791
## Channel14   -46.3295421
## Channel15    38.9634485
## Channel16   -18.1790933
## Channel17   -17.8862241
## Channel18    39.8965162
## Channel19   -12.8605177
## Channel20   -13.4365148
## Channel21   -11.7465540
## Channel22    -9.1662511
## Channel23    -5.3297011
## Channel24    -0.0675367
## Channel25     4.0707272
## Channel26     5.1170204
## Channel27     3.2808438
## Channel28     0.1217732
## Channel29    -2.3161208
## Channel30    -6.0183890
## Channel31   -10.4007828
## Channel32   -12.7348628
## Channel33   -11.6492660
## Channel34    -6.5619658
## Channel35    -0.8967759
## Channel36     3.7507062
## Channel37     7.0132998
## Channel38     7.8938834
## Channel39     5.0968378
## Channel40    13.1134184
## Channel41   107.6170369
## Channel42     7.5716731
## Channel43     6.9567259
## Channel44     6.0780951
## Channel45     2.2146611
## Channel46    -2.7231975
## Channel47    -5.7742816
## Channel48    -4.1734620
## Channel49     4.9130549
## Channel50    13.5661829
## Channel51   -39.7593963
## Channel52   -87.3613723
## Channel53   -40.6764108
## Channel54    33.9068963
## Channel55    30.0167828
## Channel56    23.9088076
## Channel57    16.8802085
## Channel58     9.5633915
## Channel59     1.3092854
## Channel60     2.4211066
## Channel61    -1.1511903
## Channel62    -2.6264139
## Channel63    -7.5983578
## Channel64   -10.6103507
## Channel65    90.1239488
## Channel66   -13.3757782
```

```
## Channel67   -10.4913858
## Channel68    -9.3765287
## Channel69   -12.4881645
## Channel70   -14.8892651
## Channel71   -13.0865426
## Channel72    -9.7547479
## Channel73   -11.3819681
## Channel74   -13.7424100
## Channel75   -10.8157544
## Channel76    -4.5367048
## Channel77    -4.4008924
## Channel78    -3.7923308
## Channel79    -3.3140243
## Channel80    -1.4863669
## Channel81    -1.6876310
## Channel82    -1.2652896
## Channel83    -1.4686360
## Channel84    -1.7643266
## Channel85    -0.1951605
## Channel86     1.0005805
## Channel87     2.8718402
## Channel88     3.7631528
## Channel89     4.1732807
## Channel90     3.3897751
## Channel91     2.7598670
## Channel92     3.3394940
## Channel93     3.3575244
## Channel94     3.5752516
## Channel95     3.8570431
## Channel96     3.9505446
## Channel97     4.6451638
## Channel98     4.7915287
## Channel99     4.3397991
## Channel100    3.0455692
```

```
plot(CVlasso)
```



According to coefficients result, it is observed that if lambda is equal to zero it does not affect model improvement. As mentioned before, following equation is used to minimize loss function in Lasso.

$$\lambda * \sum_{j=1}^{p} |w_j|$$

When the equation is taken into consideration, the result gives the zero value when the lambda value equals zero.

$$0 * \sum_{j=1}^{p} |w_j| = 0$$

It can be interpreted as there is no need for feature selection.

Part 8

All three models present different aproaches to fitting same data. The stepAIC selects models based on Akaike Information Criterion and it selected model with 63 variables. While LASSO model with optimal λ selected all possible parameters (100). This indicates that in Lasso regresssion the lowest MSE is reached when no penalty is included ($\lambda = 0$). But then you have a OLS regression solution and the complexity and the variance of the models are not decreased (as you select all to include all β into the model). The reason why you would want to reduce the variance of the models is - even if you select the best model for the training data (with lowest MSE), it may not fit the new data well (due to the variance in models).

Ridge and LASSO regressions penalises large β values and reduces the variance this way. However, Ridge and LASSO regressions have different shapes of the beta acceptance regions: in Ridge regression the region has circular shape which means betas are slowly converging to 0 (but for many smaller lambdas they are not exactly 0). In Lasso regression the acceptance region has a diamond shape that implies that for different

lambdas different betas have non zero values while other betas have exact 0 values. The optimal λ value in Ridge regression is 0.1 (when $\lambda = 0$ is excluded) while in LASSO regressions it is 0 (when $\lambda = 0$ is excluded).

Appendix

```
knitr::opts_chunk$set(echo = TRUE)
# Import data
suppressWarnings(suppressMessages(library(readxl)))
data <- as.data.frame(read_xlsx("spambase.xlsx"))

# Devide data into testing and training sets

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

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)

# implemented steps ii)-iv)
  Xn=Xn/matrix(sqrt(rowSums(Xn^2)), nrow=n1, ncol=p-1)
  C <- X%*%t(Xn)
  D <- 1 - C
  Prob <- c()
  for (i in 1:n2 ){
# the computed distance matrix is used to find
# which observations are the nearest neighbors to case #i
      neighbours <- order(D[,i])[1:k]

# Prob[i] is derived by using the target values of the nearest neighbors
     is_spam <- data[,49][neighbours]
     Prob[i] <- length(is_spam[is_spam == 1])/ length(is_spam)
  }
  return(Prob)
}
# Function that builds the confussion matrix
confusion_matrix <- function(predicted, actual) {
    df <- data.frame("Pred" = predicted, "Real" = actual)
    confusion_mat <- with(df, table(Real, Pred))
    totals <- matrix(c(sum(confusion_mat[1, ]), sum(confusion_mat[2, ])), nrow = 2, ncol = 1)
    confusion_mat <- cbind(confusion_mat, totals)
    colnames(confusion_mat) <- c("0", "1", "Total")
    return(confusion_mat)
```

```
}
# k = 5
k5_result <- knearest(train, 5, test)
# Classification rule applied
k5_predict <- ifelse(k5_result>0.5, 1, 0)
# Confusion matrix
confusion_mat_5 <- confusion_matrix(k5_predict, test[,49])
# Misclassification rate
misclass_rate_5 <- 100 * sum(test[,49] != k5_predict)/nrow(test)

# k = 1
k1_result <- knearest(train, 1, test)
# Classification rule applied
k1_predict <- ifelse(k1_result>0.5, 1, 0)
# Confusion matrix
confusion_mat_1 <- confusion_matrix(k1_predict, test[,49])
# Misclassification rate
misclass_rate_1 <- 100 * sum(test[,49] != k1_predict)/nrow(test)

confusion_mat_5
misclass_rate_5
confusion_mat_1
misclass_rate_1
suppressWarnings(suppressMessages(library(kknn)))
# k = 5
kknn5 <- kknn(formula = Spam~.,train = train, test = test, k=5)
# Classification rule applied
kknn5_pred <- ifelse(kknn5$fitted.values>0.5, 1, 0)
# Confusion matrix
confusion_mat_kknn5 <- confusion_matrix(kknn5_pred, test[,49])
# Misclassification rate
misclass_rate_kknn5 <- 100 * sum(test[,49] != kknn5_pred)/nrow(test)

confusion_mat_kknn5
misclass_rate_kknn5
ROC=function(Y, Yfit, p){
  m=length(p)
  TPR=numeric(m)
  FPR=numeric(m)
  for(i in 1:m){
    t=table(Yfit>p[i], Y)
    TPR[i]= t[2,2]/(t[1,2]+t[2,2])
    FPR[i]= t[2,1]/(t[2,1]+t[1,1])
  }
  return (list(TPR=TPR,FPR=FPR))
}

pi <- seq(0.05,0.95, by=0.05)
TPR_FPR <- ROC(test[,49],k5_result,pi)
TPR_FPR_kknn5 <- ROC(test[,49],kknn5$fitted.values,pi)

# Specificity = 1 - FPR, knearest with k=5
spec_kn <- as.vector(unlist(lapply(TPR_FPR$FPR, function(x){1-x})))
```

```r
# Sensitivity = TPR, knearest with k=5
sens_kn <- as.vector(unlist(TPR_FPR$TPR))

# Specificity = 1 - FPR, kknn with k=5
spec_kknn <- as.vector(unlist(lapply(TPR_FPR_kknn5$FPR, function(x){1-x})))

# Sensitivity = TPR, kknn with k=5
sens_kknn <- as.vector(unlist(TPR_FPR_kknn5$TPR))

library(ggplot2)
plot_data1 <- as.data.frame(TPR_FPR)
plot_data2 <- as.data.frame(TPR_FPR_kknn5)
names(plot_data2) <- c("TPR2", "FPR2")
df <- cbind(plot_data1,plot_data2)
plot <- ggplot2::ggplot(df)+
        geom_line(aes(x=df$FPR, y = df$TPR, colour="blue"))+
        geom_point(aes(x=df$FPR, y = df$TPR, colour="blue"))+
        geom_line(aes(x=df$FPR2, y = df$TPR2, colour="red"))+
        geom_point(aes(x=df$FPR2, y = df$TPR2, colour="red"))+
        scale_color_manual(labels = c("knearest", "kknn"), values = c("blue", "red"))+
        labs(x="FPR",y="TPR", ttile="ROC curve",color = "Model")+
        geom_abline(intercept = 1, slope = -1)


sens_kn
mean(sens_kn)
spec_kn
mean(spec_kn)
sens_kknn
mean(sens_kknn)
spec_kknn
mean(spec_kknn)
plot

#linear regression
mylin = function(X, Y, Xpred) {
  Xpred1 = cbind(1, Xpred)
  X <- cbind(1, X)
  beta <- solve(t(X) %*% X) %*% t(X) %*% Y
  Res = Xpred1 %*% beta
  return(Res)
}

myCV = function(X, Y, Nfolds) {
  n = length(Y)
  p = ncol(X)
  set.seed(12345)
  ind = sample(n, n)
  X1 = X[ind, ]
  Y1 = Y[ind]
  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

            for (k in 1:Nfolds) {
              startIndice <- (k * sF) - sF + 1
              endIndice <- 0
              if (k == Nfolds) {
                mod <- nrow(X1) %% Nfolds
                endIndice <- (k * sF) + mod
              } else{
                endIndice <- k * sF
              }

              selectedFeatures <- which(model == 1)
              Xvalidate <- as.matrix(X1[,selectedFeatures])
              Xvalidate <- as.matrix(Xvalidate[startIndice:endIndice,])
              Yvalidate <- as.matrix(Y1[startIndice:endIndice])

              Xtrain <- X1[-c(startIndice:endIndice), selectedFeatures]
              Ytrain <- Y1[-c(startIndice:endIndice)]
              Ypred <- mylin(X = Xtrain, Y = Ytrain, Xpred = Xvalidate)

              SSE = SSE + sum((Ypred - Yvalidate) ^ 2)
            }
            curr = curr + 1
            MSE[curr] = SSE / n
            Nfeat[curr] = sum(model)
            Features[[curr]] = model

          }
  numberOfFeatures <-lapply(Features,FUN = function(x) {return(length(which(x == 1)))})
  plot(x = numberOfFeatures, y = MSE)
  i = which.min(MSE)
  Features=Features[[i]]
  Features2 <- which(Features == 1)
  return(list(CV=MSE[i],
              Features=colnames(X1)[Features2],
              Features = Features))
}

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

```
print(res)
df <- as.data.frame(read_xlsx("tecator.xlsx"))

plot(df$Moisture, df$Protein, las = 1)
set.seed(12345)
index <- sample(nrow(df), floor((nrow(df)*0.5)))
train <- df[index, -grep("Channel", names(df))]
test <- df[-index, -grep("Channel", names(df))]
model <- function(i, data){
    m <- lm(Moisture ~ poly(Protein, i, raw = TRUE), data)
    return(m)
}
generate_function <- function(train, test){
    output <- vector()

    for(i in 1:6){
        m <- model(i, train)
        pred <- predict(m, test)
        MSE_train <- mean(residuals(m)^2)
        MSE_test <- mean((pred - test$Moisture)^2)

        output <- rbind(output,c(MSE_train, MSE_test))
    }
    output <- cbind(output, c(1:6))
    colnames(output) <- c("MSE_train", "MSE_test", "model")

    return(output)
}
MSE <- generate_function(train, test)

suppressWarnings(suppressMessages(library(tidyverse)))
MSE_plot <- MSE %>%
    as.data.frame(.) %>%
    gather(MSE, value, -model) %>%
    mutate(MSE = as.factor(.$MSE))

ggplot(MSE_plot, aes(x = model, y = value)) +
    geom_point(aes(color = MSE_plot$MSE)) +
    ylab("MSE") +
    xlab("Polynomials")
suppressWarnings(suppressMessages(library(MASS)))
df4 <- df[, 2:102]
mod <- lm(Fat ~., df4)
aic <- stepAIC(mod, direction = "both")
length(aic$coefficients)
suppressWarnings(suppressMessages(library(glmnet)))
ridge <- glmnet(x = as.matrix(df4[,1:100]),
                y = as.matrix(df4[, 101]),
                alpha = 0,
                family = "gaussian")
plot(ridge, xvar = "lambda", label = TRUE)
lasso <- glmnet(x = as.matrix(df4[,1:100]),
                y = as.matrix(df4[, 101]),
```

```
                alpha = 1, family = "gaussian")
plot(lasso, xvar = "lambda", label = TRUE)

CVlasso <- cv.glmnet(x = as.matrix(df4[,1:100]),
                     y = as.matrix(df4[, 101]),
                     alpha = 1,
                     lambda = seq(0, 100, by = 0.05),
                     family = "gaussian")
CVlasso$lambda.min
coef(CVlasso, s = "lambda.min")
plot(CVlasso)
```