Introduction to Machine Learning

TDDE01 - Lab1

Author:
David Tran - davtr766
Jakob Bertlin - jakbe457

I. Introduction

As part of the TDDE01 course, Introduction to Machine Learning, the students are obliged to complete a series of three labs in order to complete the course. The purpose of this lab is to learn more about K-nearest neighbor methods, the log-likelihood algorithm and linear regression.

II. Assignment 1

Spam classification with nearest neighbors
Given a data file *spambase.csv* containing information about the frequency of various words and characters for a total of 2740 e-mails. The data file can be interpreted as a matrix of size 2740x49 where every row is a specific mail and every column is words that corresponds to the mail. The data has been manually classified as spam or not spam by including an extra column where the values are either 1 (spam) or 0 (not spam).

The data is divided into two sets, one training set and one test set where the distribution is 50/50. To be able to fit the training data it is necessary to propose an appropriate K-nearest neighbor classifier for the distance metric between two points. In this case, the measure is based on cosine similarity which for two vectors is defined as:

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

Fig. 1. Cosine Similarity definition

$$C = \left\| c(X_i, Y_j) \right\| \text{ as } \hat{\boldsymbol{X}}\hat{\boldsymbol{Y}}^{\boldsymbol{T}}$$

Fig. 2. Definition of C matrix.

where *X* is the testing set and *Y* is the training set. The distance matrix is defined as *D = 1 - C*. Every row in the distance matrix is an e-mail and the values of the columns is the relative distance of each *Y* with respect to *X*. Therefore, by sorting each column in increasing order and pick *K* first elements results in the *K* closest e-mail from the training set *X* in respect to each testing set *Y*. Now the K-nearest neighbors can be found by adding the values from 0 to *K* and dividing by *K* to get the probability for given *K*. The result is then stored in a probability vector. This is the *knearest()* function, implemented from scratch.

After defining the function *knearest()* that can be used to compute K-nearest neighbor method then the next exercise is to classify the training and test data by using *K = 5* with given classification principle:

$$\hat{Y} = 1 \; if \; p(Y=1|X) > 0.5, otherwise \; \hat{Y} = 0$$

Fig. 3. Classification principle for exercise 4

.

The confusion matrix for *K = 5* following by this principle is the following for test data:

```
Truth    0    1
    0  692  245
    1  192  241
Missclassification rate: 0.3189781
```

The confusion matrix for *K = 5* following by this principle is the following for train data:

```
      Prediction
Truth    0    1
    0  777  168
    1  118  307
Missclassification rate: 0.2087591
```

This has to be repeated but for the case *K = 1*. The result of the confusion matrix and missclassification rate for test data:

```
      Prediction
Truth    0    1
    0  639  298
    1  178  255
Missclassification rate: 0.3474453
```

The result of the confusion matrix and missclassification rate for train data(*K = 1*):

```
      Prediction
Truth    0    1
    0  939    6
    1    2  423
Missclassification rate: 0.005839416
```

The next exercise is to use the standard classifier *kknn()* that can be installed as package in *R Studio* and compute the confusion matrix and missclassification rate for test data:

```
       Prediction
Truth     0    1
     0  893   44
     1  427    6
Missclassification  rate:  0.3437956
```

The last exercise of the assignment is to use *knearest()* and *kknn()* with *K = 5* and classify the test data by using the following principle:

$$\hat{Y} = 1 \; if \; p(Y = 1|X) > \pi, otherwise \; \hat{Y} = 0$$

Fig. 4. Classification principle for exercise 6

.

where π = *0.05, 0.10, 0.15,...,0.95* and then compute the sensitivity and the specificity values for the two methods and lastly plot the corresponding ROC curves. The sensitivity function can be computed with the following formula:

$$Sensitivity = \frac{TP}{TP + FN}$$

and the specificity can be computed with the following formula:

$$Specificity = 1 - \frac{FP}{FP + TN}$$

where *TP = true positive, TN = true negative, FN = false negative and FP = false positive* and can be retrieved from the confusion matrix. The resulting ROC plot is presented below:

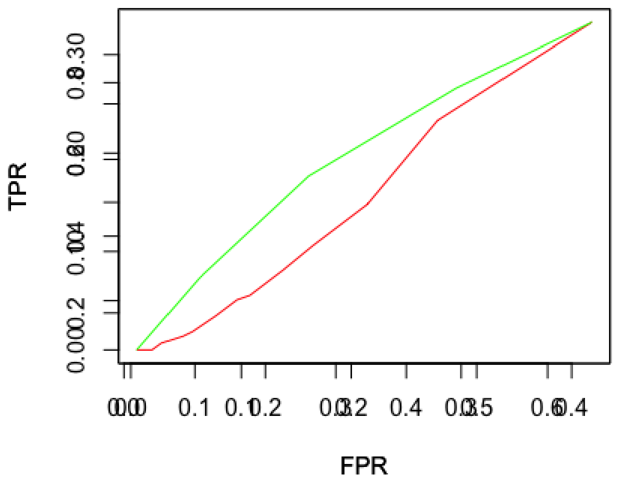

Fig. 5. ROC curves

.

where the green plot is the *knearest()* function and the red plot is *kknn()* from the standard package.

The conclusion from the plot is that the *knearest()* function that was implemented from scratch is a better K-nearest algorithm compared to *kknn()* from the standard package because the *knearest()* function cover a larger area compared to *kknn()*. Since the *kknn()* uses the Minkowski distance as distance metric and *knearest()* uses the Cosine Similarity, a conclusion can be made that the Cosine Similarity distance metric is better than the Minkowski distance for this case.

III. Assignment 2

Inference about lifetime of machines

Given a datafile *machines.csv* that contains information about the lifetime of machines. The company that manufactures these machines wants to know more about the underlying process in order to determine the appropriate warranty time. The variable *Length* equals to the lifetime of the machines.

One can assume the probability model below:

$$p(x|\theta) = \theta e^{-\theta x} \text{ for } \boldsymbol{x}\text{=}Length$$

Fig. 6. Probability model

.

The observations that has been done on the machines are independent and identically distributed. The first exercise of this assignment is to implement a function that computes the log-likelihood of the given model and plot the curve showing the dependence of log-likelihood θ where the entire data is used for fitting. The resulting curve is presented below:

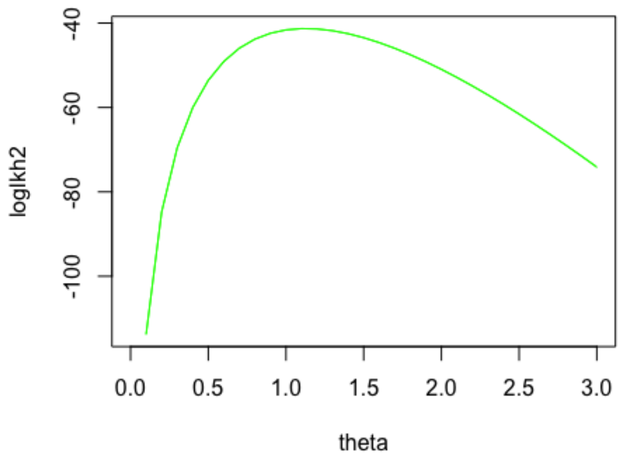

Fig. 7. Dependence plot of log-likelihood θ

.

where the sequence of θ is from 0 to 3 with step size 0.1. The maximum log-likelihood value of θ is 1.1. Now the above step is repeated but instead of observing the whole data, the next step is to observe the six first observations from the data and plot it together with the above plot. The result is presented below:

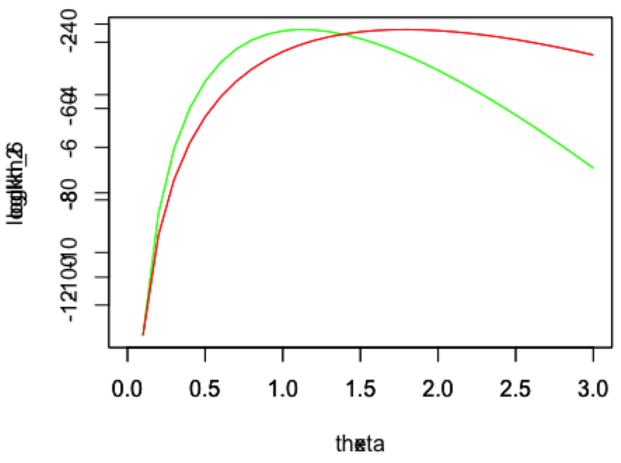

Fig. 8. Dependence plot of log-likelihood θ for whole data and for six first observations

where the red plot is the curve of six first observations and the green plot is the curve with all observations. The conclusion from the above plot is that the red curve is unreliable because it only observes the six first observations and therefore miss out on information. Comparing the curves, the red plot has a higher θ value compared to the green plot in e.g θ = 2.0 even though the green plot computes the curve with the whole data. Therefore, the reliability should be worse when the amount of observations is limited which is shown in the plot.

The next step of the assignment is to assume a bayesian model that is the same as in *fig.6*, however, a prior p(θ) = $\lambda\ e^{-\lambda x}$ is now taken into consideration where λ = *10*. The aim now is to compute *l*(θ) = *log*(*p*($x|\theta$)*p*(θ)) and plot the corresponding curve. The resulting plot is presented below:

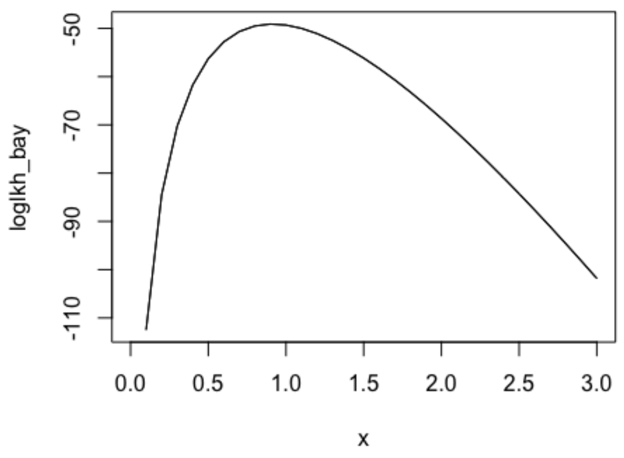

Fig. 9. Dependence plot of bayesian model

The maximum log-likelihood value of θ is 0.9 compared to previous 1.1. The kind of measure computed by this function is how good the model fits the data.

The last step of the assignment is to use the computed θ value 1.1 and generate 50 new observations and plot the histograms of the original and the new data. The following plots are given below:

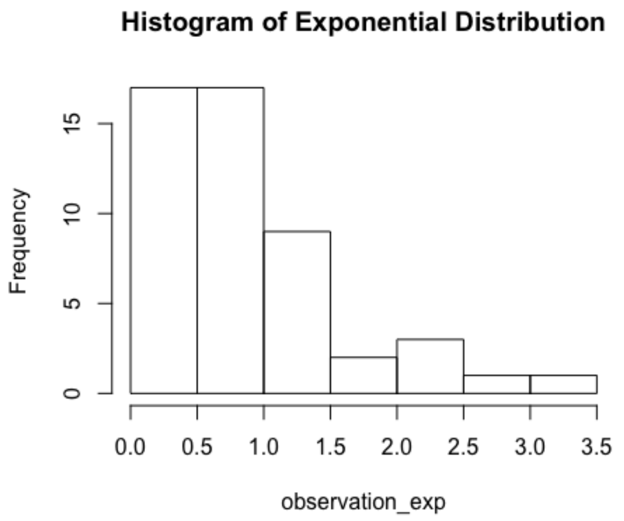

Fig. 10. Histogram of Exponential Distribution

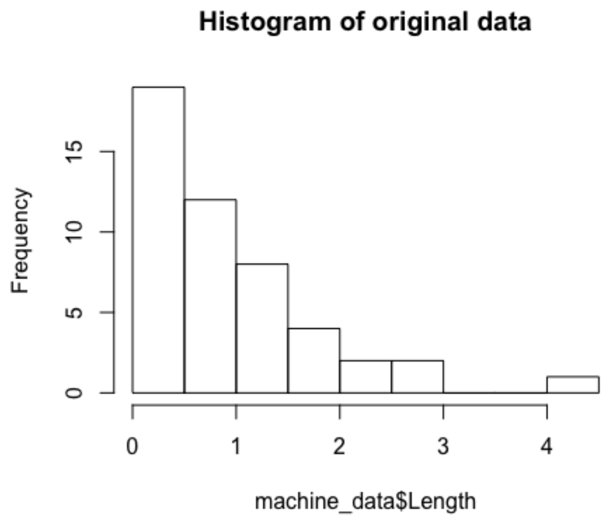

Fig. 11. Histogram of Original Data

A conclusion from these histograms is that the histograms are similar which implies that the model used is a good model.

IV. Assignment 4

Linear regression and regularization The last assignment of this lab is about linear regression and regularization. Given a data file *tecator.csv* which contains the result of study aimed to investigate if near infrared absorbance spectrum can be used to predict fat content of meat. The data can be interpreted as a matrix where every row is a sample and the column for the given row contains a 100 channel spectrum of absorbance records and the level of moisture, fat and protein.

The first exercise is to create a plot of moisture vs protein which is shown below:

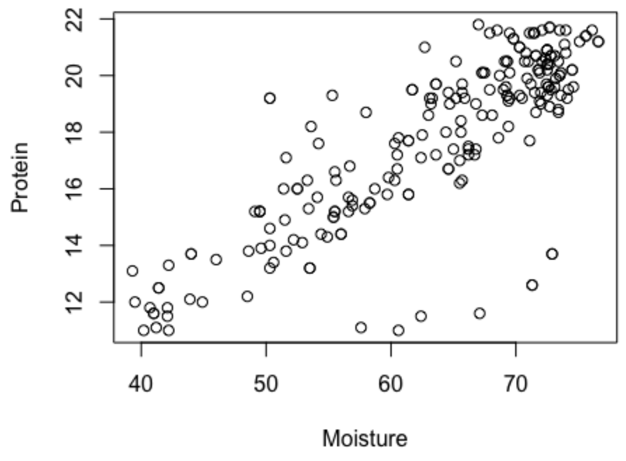

Fig. 12. Moisture vs Protein

By looking at the plot above, one can conclude that the model is a linear model. Therefore, the data are described well by a linear model.
Now let's consider a model M_i where moisture is normally distributed and the expected moisture is a polynomial function of protein. A probalistic model that describes M_i is given below:

```
X = Protein
Y = Moisture
Y ~ N(M0 + M1*x + M2*x^2 + ....., Mn*x^n,
     sigma^2)
```

It is appropriate to use the MSE criterion when fitting this model to the training data because it can handle multiple features which will behave like a polynomial function.

The next step is to divide the data into train set and validation set similarly to assignment 1 and fit model M_i, where *i= 1...6* and record the training and validation MSE. The plots are presented below:

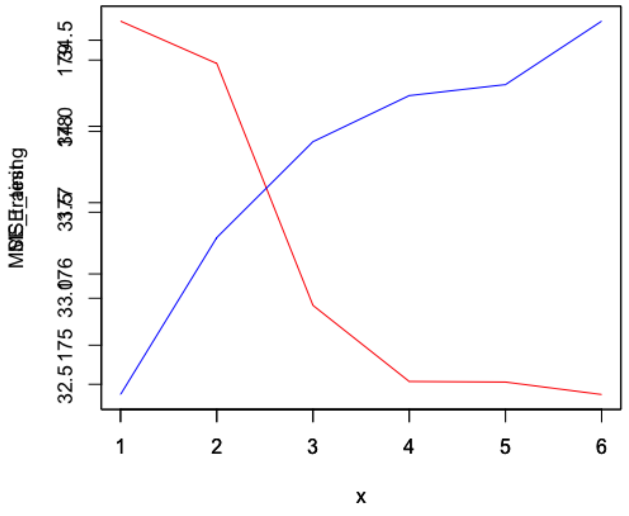

Fig. 13. Moisture vs Protein

where the red is the training curve and the blue is the test curve. The above plot indicates that using a higher polynomial degree for the test set increases the MSE and the opposite for the training set.

The next step is to use the function *stepAIC* to perform a variable selection of a linear model in which Fat is response and Channel1-Channel100 are predictors. The function returned 63 features which indicates about half of all features contributes to Fat.

Now we fit a Ridge regression model with same predictors and response variables.

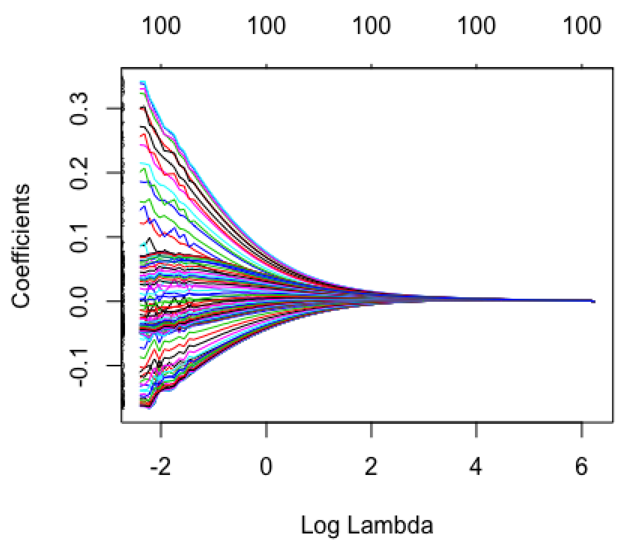

Fig. 14. Ridge Regression

Here one can conclude that all coefficients go to zero when lambda increases.

We do the same but for Lasso:

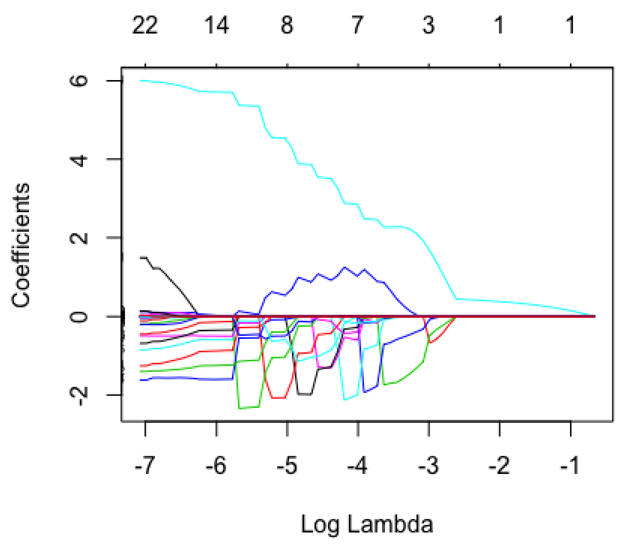

Fig. 15. Ridge Regression

We can see that some features are completely ignored and some features have a bigger impact than others.

The next step is to use cross-validation and find the optimal LASSO model. The function *cv.glmnet()* is used and the following plots are then computed:

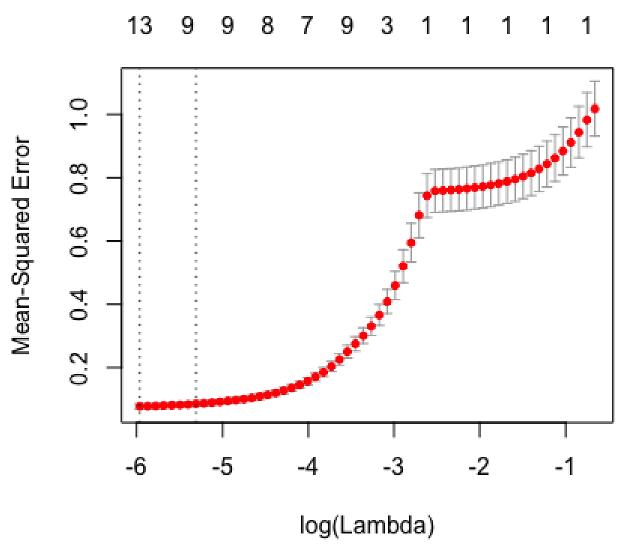

Fig. 16. Ridge Regression

By interpreting the plot, one can conclude that a lower lambda value indicates a smaller MSE. Therefore, the smallest lambda is most optimal which is 0.002571916. The amount of features selected were 13 according to the plot for that specific lambda value.

Lastly, LASSO gives a lower MSE than stepAIC even though stepAIC used 63 features compared to LASSO with 13 features. The conclusion is that is it not always better to use more features as we can see in the comparison, because it depends on the dataset and what type of features we select.

APPENDIX

Listing 1. Assignment 1

```
data = read.csv2("spambase.csv")

#Divide the data into two set, training set and test set.
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] #Dimension of data in row.
  n2=dim(newdata)[1] #Dimension newdata in row.
  p1=dim(data)[2] #Dimension of data in column.
  p2=dim(newdata)[2] #Dimension of newdata in column.
  Prob=numeric(n2) #Probability vector of size n2.
  X=as.matrix(data[,-p1]) #Removes last column which is the solution. (Spam)
  Xn=as.matrix(newdata[,-p1]) #Removes last column which is the solution. (Spam)
  X=X/matrix(sqrt(rowSums(X^2)), nrow=n1, ncol=p1-1)

  Y=as.matrix(newdata[,-p2])
  Y=Y/matrix(sqrt(rowSums(Y^2)), nrow=n2, ncol=p2-1)

  C = X %*% t(Y)
  D = 1 - C
  Dsort = apply(D,2,sort)

  for (i in 1:n2 ){
    probval = 0
    for(n in 1:k){
      pos = match(Dsort[n,i], D[,i])
      probval = probval + data[pos,p1]
    }
    print(probval/k)
    Prob[i]=probval/k
  }
  return(Prob)
}


ROC=function(p, input){
  m=length(p)
  TPR=numeric(m)
  FPR=numeric(m)
  for(i in 1:m){
    t = table(test[,ncol],as.numeric(input[i,]))
    TP = t[2,2]
    FP = t[1,2]
    sum_TP_FN = TP + t[2,1] #TP + FN
    sum_FP_TN = FP + t[1,1] #FP + TN
    TPR[i]= TP/sum_TP_FN
    FPR[i]= FP/sum_FP_TN
  }
  return (list(TPR=TPR,FPR=FPR))
}

#Exercise 3
knn5 = knearest(train,5,test)
knn5 = round(knn5)
ncol = dim(test)[2]
confMat5 = table(test[,ncol],knn5,dnn = list("Truth","Prediction")) #Confusion matrix
miss_class_rate = (confMat5[1,2] + confMat5[2,1])/1370

#3b. Train data
knn5b = knearest(train,5,train)
knn5b = round(knn5b)
ncol = dim(test)[2]
confMat5b = table(train[,ncol],knn5b,dnn = list("Truth","Prediction")) #Confusion matrix
miss_class_rate1 = (confMat5b[1,2] + confMat5b[2,1])/1370

#Exercise 4
knn1 = knearest(train,1,test)
knn1 = round(knn1)
```

```
ncol = dim(test)[2]
confMat1 = table(test[,ncol],knn1, dnn = list("Truth","Prediction")) #Confusion matrix
miss_class_rate2 = (confMat1[1,2] + confMat1[2,1])/1370

#4b Test data
knn1b = knearest(train,1,train)
knn1b = round(knn1b)
ncol = dim(test)[2]
confMat1b = table(train[,ncol],knn1b, dnn = list("Truth","Prediction")) #Confusion matrix
miss_class_rate3 = (confMat1b[1,2] + confMat1b[2,1])/1370

#Exercise 5
kknn5 = kknn(formula = formula(train), train, test, k = 5)
kknn5 = kknn5[["fitted.values"]] #Convert to vector.
kknn5_ex5 = round(kknn5)
confMatKKNN = table(test[,ncol],kknn5_ex5,dnn = list("Truth","Prediction")) #Confusion matrix
miss_class_rate4 = (confMatKKNN[1,2] + confMatKKNN[2,1])/1370


#Matrix where every row is probability vector of pi[i].
#Function to convert vector to matrix.
vecToMat = function(vec){
  pi = seq(from = 0.05, to = 0.95, by = 0.05)
  mat = matrix(nrow = length(pi), ncol = length(kknn5), byrow = TRUE)
  for(i in 1:length(pi)){
    prob = numeric(length(vec))
    for(j in 1:length(prob)){
      if(vec[j] > pi[i])
        prob[j] = 1
      else
        prob[j] = 0
    }
    mat[i,] = prob;
  }
  return(mat)
}

mat_kknn5 = vecToMat(kknn5)
knn5 = knearest(train,5,test)
mat_knn5 = vecToMat(knn5)

pi = seq(from = 0.05, to = 0.95, by = 0.05)
ROCkknn <- ROC(pi,mat_kknn5)
ROCknn5 <- ROC(pi, mat_knn5)
plot(ROCkknn$FPR, ROCkknn$TPR,type="l",col="red", xlab = "FPR", ylab = "TPR", xlim = c(0,0.4), ylim = c
    (0,0.8))
par(new = TRUE)
plot(ROCknn5$FPR, ROCknn5$TPR,type="l",col="green", xlab = "FPR", ylab = "TPR", xlim = c(0,0.4), ylim = c
    (0,0.8))
```

Listing 2. Assignment 2

```
machine_data = read.csv2("machines.csv")

#Exercise 2.
loglkh = function(theta, nr_obs){
  if(nr_obs == dim(machine_data)[1]){
    len_machine = dim(machine_data)[1]
  }
  else
    len_machine = nr_obs

  machine_vec = as.numeric(unlist(machine_data)) #Convert machine_data to vector.
  samples = 10
  len = length(theta)
  prob = rep(1,len) #Create probability vector of 1's.
  for(i in theta){
    for(j in 1:len_machine){
      index = i / (1/samples)+1 #Convert to real indices.
      prob[index] = prob[index] + log(i*exp(-i*machine_vec[j])) #Sum of loglikelihood formula
    }
  }
  return (prob)
}

#Exercise 3
theta = seq(from = 0, to = 3, by = 0.1)
nr_obs = dim(machine_data)[1]
loglkh2 = loglkh(theta,nr_obs)
thetamax = theta[which(loglkh2==max(loglkh2))]

#Plot
theta<-theta
plot(theta,loglkh2,type="l",col="green")
lines(theta,loglkh2,col="green", xlab = theta)

par(new=TRUE) #Hold on in MATLAB.

nr_obs6 = 6
loglkh_6 = loglkh(theta, nr_obs6)
thetamax_6 = theta[which(loglkh_6==max(loglkh_6))]

#Plot
plot(x,loglkh_6,type="l",col="red")
lines(x,loglkh_6,col="red", xlab = theta)

#Exercise 4
loglkh_bayesian = function(theta, nr_obs){
  p = loglkh(theta,nr_obs)
  len = length(theta)
  prob_bay = rep(0,len) #Create probability vector of 0's.
  j = 1;
  for(i in theta){
    prior = 10*exp(-10*i)
    prob_bay[j] = prob_bay[j] + p[j] + log(prior)
    j = j+1
  }
  return (prob_bay)
}

par(new = TRUE)
loglkh_bay <- loglkh_bayesian(theta,nr_obs)
plot(x,loglkh_bay,type="l",col="black")
lines(x,loglkh_bay,col="black")
thetamax_bay = theta[which(loglkh_bay==max(loglkh_bay))]

#Exercise 5
observation_exp = rexp(50, thetamax)
hist(observation_exp,main = "Histogram of Exponential Distribution")
hist(machine_data$Length, main = "Histogram of original data")
```

Listing 3. Assignment 4

```r
teactor_data = read.csv2("tecator.csv")
plot(teactor_data$Moisture, teactor_data$Protein, xlab = "Moisture", ylab = "Protein")

#Exercise 2

#Exercise 3
#Divide the data into training and validation set(50%/50%).
n=dim(teactor_data)[1]
id=sample(1:n, floor(n*0.5))
train=teactor_data[id,]
test=teactor_data[-id,]

#Fit model Mi, i = 1,...,6.
MSE = function(predict, obs){ #Prediction vector, Observation vector.
  return(mean((predict - obs)^2))
}

MSE_training = numeric(6)
MSE_test = numeric(6)
Moisture = teactor_data$Moisture
Protein = teactor_data$Protein
for(i in 1:6){
  model = lm(Moisture~poly(Protein,i), data = train)
  model_predict_train = predict.lm(model)
  model_predict_test = predict.lm(model, data = test)
  MSE_training[i] = MSE(unname(model_predict_train), train$Moisture)
  MSE_test[i] = MSE(unname(model_predict_test), test$Moisture)
}
x = seq(from = 1, to = 6)
plot(x, MSE_training, col = "red", type = "l")
par(new = TRUE)
plot(x, MSE_test, col = "blue", type = "l")

#Ex4
library(MASS)
lm_fat = lm(Fat~., data=teactor_data[, 2:102]) #First column and last columns are not features.
step = stepAIC(lm_fat)

#Ex5
library(glmnet)
covariates=scale(teactor_data[,2:101]) #Features
response=scale(teactor_data[, 102]) #Fat
model0=glmnet(as.matrix(covariates), response, alpha=0,family="gaussian")
plot(model0, xvar="lambda", label=TRUE)

#Ex6
lasso = glmnet(as.matrix(covariates), response, alpha=1,family="gaussian") #alpha = 1, gaussian => LASSO
plot(lasso, xvar="lambda", label=TRUE)

#Ex7
cv_lasso = cv.glmnet(as.matrix(covariates), response, alpha=1,family="gaussian")
cv_lasso$lambda.min #Get minimum lambda
plot(cv_lasso)
coef(cv_lasso, s="lambda.min")
```