Helpful info

How to ensure that your model is not overfitting? Keep the design of the model simple. Try to reduce the noise in the model by considering fewer variables and parameters. Cross-validation techniques such as K-folds cross validation help us keep overfitting under control. Regularization techniques such as LASSO help in avoiding overfitting by penalizing certain parameters if they are likely to cause overfitting.

Code from labs with comments

Lab 1

Assignment 1

```r
library(kknn)
setwd("~/Programming/TDDE01/Lab 1")

#1.
#Import the data into R and divide it into training and test sets (50%/50%)
data <- read.csv2("spambase.csv")

#Split data into training and test set.

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

#2.
#Use logistic regression (functions glm(), predict()) to classify the training
#and test data by the classification principle
#Y_hat=1 if (Y=1|X) > 0.5, otherwise Y_hat=0
#and report the confusion matrices (use table()) and the misclassification rates for
#training and test data. Analyse the obtained results.

#GLM model for data with family binomial --> only 0s and 1s
model <- glm(Spam ~ ., family=binomial, data=train)
predictModel= predict(model, newdata=test, type="response")

probability <- ifelse(predictModel_test > 0.5, "1", "0") #Split up the model into spam and not spam

confMatrix <- table(probability, test[,"Spam"]) #Confusionmatrix from the model

modelDiag <- diag(confMatrix) #Diagonal of the confMa

#Missclassfication rate by dividing the diagonal
#from the confusionmatricx with the whole confusionmatrix
missClMa1 = 1-(sum(modelDiag)/sum(confMatrix))

#Same but for training data
predictModel_train = predict(model, newdata=train, type="response")

#Split up the model into spam and not spam
probability_train <- ifelse(predictModel_train > 0.5, "1", "0")
#Confusionmatrix from the model
```

```r
confMatrix_train <- table(probability_train, train[,"Spam"])
modelDiag_train <- diag(confMatrix_train) #Diagonal of the confMa
#Missclassfication rate by dividing the diagonal from the
#confusionmatricx with the whole confusionmatrix
missClMa1_train = 1-(sum(modelDiag_train)/sum(confMatrix_train))

#Prints results
print("Confusion matrix 2 test:")
print(confMatrix)
print("missclassification 2 test:")
print(missClMa1)

print("Confusion matrix 2 train:")
print(confMatrix_train)
print("missclassification 2 train:")
print(missClMa1_train)
#3.
#Use logistic regression to classify the test data by the classification principle
#Y_hat=1 if p(Y=1|X) > 0.8, otherwise Y_hat=theta
#and report the confusion matrices (use table()) and the misclassification rates for
#training and test data. Compare the results. What effect did the new rule have?
probability2 <- ifelse(predictModel > 0.8, "1", "0") #Split up the model into spam and not spam

confMatrix2 <- table(probability2, test[,"Spam"])

modelDiag2 <- diag(confMatrix2)

missClMa2 = 1-(sum(modelDiag2)/sum(confMatrix2))

#Same but with training data
#Split up the model into spam and not spam
probability2_train <- ifelse(predictModel_train > 0.8, "1", "0")
#Confusionmatrix from the model
confMatrix2_train <- table(probability2_train, train[,"Spam"])
modelDiag2_train <- diag(confMatrix2_train) #Diagonal of the

#Missclassfication rate by dividing the diagonal from the
#confusionmatricx with the whole confusionmatrix
missClMa2_train = 1-(sum(modelDiag2_train)/sum(confMatrix2_train))


print("Confusion matrix 3:")
print(confMatrix2)
print("missclassification 3:")
print(missClMa2)

print("Confusion matrix 3 train:")
print(confMatrix2_train)
print("missclassification 3 train:")
print(missClMa2_train)
#4.
#Use standard classifier kknn() with K=30 from package kknn, report the the
#misclassification rates for the training and test data and compare the results with step 2.
```

```r
#KKNN with K=30

kknn_K30 = kknn(Spam ~ ., train=train, test=test, k=30)
kknn_K30_pred = predict(kknn_K30)

kknn_K30_pred <- ifelse(kknn_K30_pred > 0.5, 1, 0) #Split up the model into spam and not spam


confMa_K30 = table(kknn_K30_pred, test$Spam)
misCl_K30 = 1-sum(diag(confMa_K30)/sum(confMa_K30))

#Training data
confMa_K30_train = table(kknn_K30_pred, train$Spam)
misCl_K30_train = 1-sum(diag(confMa_K30_train)/sum(confMa_K30_train))

print("Missclassification 4 testing:")
print(misCl_K30)

print("Missclassification 4 training:")
print(misCl_K30_train)


#5.
#Repeat step 4 for K=1 and compare the results with step 4.
#What effect does the decrease of K lead to and why?

#KKNN with K=1
kknn_K1 = kknn(Spam ~ ., train=train, test=test, k=1)
kknn_K1_pred = predict(kknn_K1)

#Split up the model into spam and not spam
kknn_K1_pred <- ifelse(kknn_K1_pred > 0.5, 1, 0)

confMa_K1 = table(kknn_K1_pred, test[,"Spam"])
misCl_K1 = 1-sum(diag(confMa_K1)/sum(confMa_K1))

#Training data
confMa_K1_train = table(kknn_K30_pred, train[,"Spam"])
misCl_K1_train = 1-sum(diag(confMa_K1_train)/sum(confMa_K1_train))

print("Missclassification 5 testing:")
print(misCl_K1)

print("Missclassification 5 training:")
print(misCl_K1_train)
```

Assignment 2

```r
library(readxl)
setwd("~/Programming/TDDE01/Lab 1")


set.seed(12345)
```

```r
#1
#imports file
machines <- read_excel("machines.xlsx")

#2
#Assume the probability model p(x|theta)=theta*exp(-theta*x) for x=Length in which
#observations are independent and identically distributed.
#What is the distribution type of x? Write a function that
#computes the log-likelihood log(p(x|theta )for a gven theta nd a given data vector x.
#Plot the curve showing the dependence of log-likelihood on theta
#here the entire data is used for fitting.
#What is the maximum likelihood value of theta rding to the plot?

#Comes from  p(x|theta) = theta*exp(-theta) then using thew likelihood
#function with it and then minimizing that.
likelihoodlog = function(x, theta) {
  return(-dim(x)[1] * log(theta) + theta * sum(x))
}
print(likelihoodlog(machines[1], 1))

#Plot curve
curve(
  likelihoodlog(machines, x),
  xlim = c(0, 4),
  ylim = c(0,60),
  col = "blue"
)

#Min theta. Comes from deriving the loglikelihood-function.
#The more data I have the more exactly I can pinpoint the exact point of failure
minvalue = dim(machines)/sum(machines)
print("Min theta:")
minthetalikelihood = function(x) {
  return (dim(x)[1]/sum(x))
}
print(minthetalikelihood(machines))

#3
#Repeat step 2 but use only 6 first observations from the data,
#and put the two log-likelihood curves (from step 2 and 3) in the same plot.
#What can you say about reliability of the maximum likelihood solution in each case?

curve(likelihoodlog(machines[1:6,],x), from=0, to=4, col="red", add = TRUE)
print((dim(machines[1:6,])[1])/sum(machines[1:6,]))

#4
#Assume now a Bayesian model with p(x|theta)=theta*exp(-theta*x) and
#a prior theta=lambda*exp(-labda*theta), lambda=10.
#Write a function computing l(theta)=log(p(x|theta)p(theta)).
#What kind of measure is actually computed by this function?
#Plot the curve showing the dependence of l(theta) on theta
#computed using the entire data and overlay it with a plot
#from step 2. Find an optimal theta and compare your result
```

```
#with the previous findings.

bayesianfunc = function(x, theta, lambda) {
  return(likelihoodlog(x, theta) - log(lambda) + lambda * theta)
}
print("Bayesianfunction:")
print(bayesianfunc(machines,1,10))

curve(bayesianfunc(machines,x,10), xlab="Theta", ylab="l(Theta)", from=0, to=4, col="green", add=TRUE)

#min theta for bayesianfunc. I get it from deriving the
#bayesianfunc with respect to theta and set it to = 0 to get the min
print("Min theta baythetasian func.:")
print(dim(machines)[1]/(sum(machines)+10))



#5
#Use theta value found in step 2 and generate 50 new observations
#from p(x|theta)=theta*exp(-theta*x)
#(use standard random number generators). Create the histograms
#of the original and the new data and make conclusions.

theta = minthetalikelihood(machines)

##Creaes 5 new data points with the rate from 2.
newdata = rexp(50, rate=theta)
print(newdata)
#Stores the old data from the Length col in the machines
olddata <- machines$Length

#Creages new window for plots
hist(olddata, col="red", xlim=c(0,5), ylim=c(0,20), xlab="x")
hist(newdata, col="blue", xlim=c(0,5), ylim=c(0,20), add=TRUE, breaks="FD")
```

Assignment 3

```
#Implement an R function that performs feature selection (best subset selection)
#in linear regression by using k-fold cross-validation without using any
#specialized function like lm() (use only basic R functions). Your function
#should depend on:
#• X: matrix containing X measurements.
#• Y: vector containing Y measurements
#• Nfolds: number of folds in the cross-validation.
#You may assume in your code that matrix X has 5 columns. The function
#should plot the CV scores computed for various feature subsets against
#the number of features, and it should also return the optimal subset of
#features and the corresponding cross-validation (CV) score. Before splitting
#into folds, the data should be permuted, and the seed 12345 should be used for
#that purpose.

#Function for linear model
mylin = function(X, Y, Xpred) {
    Xpred1 = cbind(1, Xpred)
    X1 = cbind(1, X)
    beta = solve(t(X1) %*% X1) %*% t(X1) %*% Y # obtained using the "training" data matrix
    Res = Xpred1 %*% beta # y_hat for the "test" data
    return(Res)
}
myCV = function(X, Y, Nfolds) {
    n = length(Y) # number of observations (rows)
    p = ncol(X) # number of covariates (variables or columns)
    set.seed(12345)
    ind = sample(n, n) # indexes are randomized
    X1 = X[ind, ] # randomize the order of the observations
    Y1 = Y[ind]
    sF = floor(n / Nfolds) # number of observations inside each fold
    MSE = numeric(2 ^ p - 1) # vector of the length of 2^p-1 combinations
    Nfeat = numeric(2 ^ p - 1)
    Features = list() # features that will be selected
    curr = 0 # current
    folds_obs <- cut(1:n, breaks = Nfolds, labels = FALSE)
    #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) {
                            indices <-
                                ind[which(folds_obs == k)] #indeces of the observations in fold k
                            X_mylin <-
                                X[-indices, which(model == 1)]
                            XPred_mylin <-
                                X[indices, which(model == 1)]
                            Y_mylin <- Y[-indices]
                            Ypred <-
                                mylin(X_mylin, Y_mylin, XPred_mylin)
                            Yp <- Y[indices]
                            SSE = SSE + sum((Ypred - Yp) ^ 2)
                        }
                        curr = curr + 1
                        MSE[curr] = SSE / n
                        Nfeat[curr] = sum(model)
                        Features[[curr]] =
                            model
                    }
    plot(Nfeat, MSE)
    abline(h = MSE[which.min(MSE)], col = "red")
    i = which.min(MSE)
    return(list(CV = MSE[i], Features = Features[[i]]))
}
```

```r
#Test your function on data set swiss available in the standard R repository:
#• Fertility should be Y
#• All other variables should be X
#• Nfolds should be 5
#Report the resulting plot and interpret it. Report the optimal subset
#of features and comment whether it is reasonable that these specific
#features have largest impact on the target.
data("swiss")
myCV(as.matrix(swiss[, 2:6]), swiss[[1]], 5)
```

Assignment 4

```r
library(readxl)
library(Metrics)
library(MASS)
library(ggplot2)
library(glmnet)
setwd("~/Programming/TDDE01/Lab 1")

set.seed(12345)
RNGversion('3.5.1')

#1
#Import data to R and create a plot of Moisture versus Protein.
#Do you think that these data are described well by a linear model?

#Imports file
tecator <- read_excel("tecator.xlsx")
moisture = tecator$Moisture;
protein = tecator$Protein;

#Plots the moisture and protein
plot(protein, moisture, xlab="Moisture", ylab="Protein", main="Moisture vs. Protein")
#Quite well with a linear model. Prob quite large deviations sometimes though

#Below is just to see the linear model
data_moisture_protein = tecator[,103:104]

fit1 <- lm(formula = moisture ~ protein, data=data_moisture_protein)
summary(fit1)
#Gives the
fitted1 = predict(fit1, interval="confidence")
#Plots line with protein as x and the new predicted fitted values as y
lines(protein, fitted1[, "fit"])
#This shows the line M1

#2
#Consider model Mi in which Moisture is normally distributed, and the expected Moisture
#is a polynomial function of Protein including the polynomial
#terms up to power i(i.e M1 is a linear model, M2 is a quadratic model and so on).
#Report a probabilistic model that describes Mi . Why is it appropriate to use
#MSE criterion when fitting this model to a training data?

#Consider the functions are:
```

```r
#M1 = w0 + w1*x1 + e
#M2 = w0 + w1*x1 + w2*x2^2 + e
#and so on

#3
#Divide the data into training and validation
#sets( 50%/50%) and fit models Mi , i = 1 ... 6.
#For each model, record the training and the validation
#MSE and present a plot showing how
#training and validation MSE depend on i (write some
#R code to make this plot).
#Which model is best according to the plot? How do the
#MSE values change and why?
#Interpret this picture in terms of bias-variance tradeoff.

#Splits the data into training set and test set
#using only moisture and protein cols
n=dim(tecator[,103:104])
set.seed(12345)
id=sample(1:n[1], floor(n*0.5))
train=tecator[id,103:104]
test=tecator[-id,103:104]

#Splits up the test and training data to protein and moisture respectively
p_train = train$Protein
m_train = train$Moisture
p_test = test$Protein
m_test = test$Moisture

#Models below for M1-M6, regression
m1_model = lm(formula = Moisture ~ Protein, data=train)
m2_model = lm(formula = Moisture ~ Protein + I(Protein^2), data = train)
m3_model = lm(formula = Moisture ~ Protein + I(Protein^2) +
                I(Protein^3), data = train)
m4_model = lm(formula = Moisture ~ Protein + I(Protein^2) +
                I(Protein^3) + I(Protein^4), data = train)
m5_model = lm(formula = Moisture ~ Protein + I(Protein^2) +
                I(Protein^3) + I(Protein^4) + I(Protein^5), data = train)
m6_model = lm(formula = Moisture ~ Protein + I(Protein^2) +
                I(Protein^3) + I(Protein^4) + I(Protein^5) + I(Protein^6),
              data = train)

#Predictions with the help of the training data
m1_model_pred = predict(m1_model, newdata=train)
m2_model_pred = predict(m2_model, newdata=train)
m3_model_pred = predict(m3_model, newdata=train)
m4_model_pred = predict(m4_model, newdata=train)
m5_model_pred = predict(m5_model, newdata=train)
m6_model_pred = predict(m6_model, newdata=train)

#Predictions with the help of the test data
m1_model_pred_test = predict(m1_model, newdata=test)
m2_model_pred_test = predict(m2_model, newdata=test)
```

```r
m3_model_pred_test = predict(m3_model, newdata=test)
m4_model_pred_test = predict(m4_model, newdata=test)
m5_model_pred_test = predict(m5_model, newdata=test)
m6_model_pred_test = predict(m6_model, newdata=test)

#Mean squared error for the training data
mse_train <- vector()
mse_train[1] <- mse(m_train, m1_model_pred)
mse_train[2] <- mse(m_train, m2_model_pred)
mse_train[3] <- mse(m_train, m3_model_pred)
mse_train[4] <- mse(m_train, m4_model_pred)
mse_train[5] <- mse(m_train, m5_model_pred)
mse_train[6] <- mse(m_train, m6_model_pred)

for (i in 1:6) {
  cat("MSE for training data with i=", i)
  print(mse_train[i])
}

#Mean squared error for the testing data
mse_test <- vector()
mse_test[1] <- mse(m_test, m1_model_pred_test)
mse_test[2] <- mse(m_test, m2_model_pred_test)
mse_test[3] <- mse(m_test, m3_model_pred_test)
mse_test[4] <- mse(m_test, m4_model_pred_test)
mse_test[5] <- mse(m_test, m5_model_pred_test)
mse_test[6] <- mse(m_test, m6_model_pred_test)

for (i in 1:6) {
  cat("MSE for testing data with i=", i)
  print(mse_test[i])
}

plot(1:6, mse_train, col="blue", type="l", xlab="i",
     ylab="MSE", ylim=c(23,45), main="MSE dependencie of i. Where green = test, blue= train")
lines(1:6, mse_test,col="green")

fitted_fat <- lm(tecator$Fat ~ ., data=tecator[,2:101])
steps <- stepAIC(fitted_fat, direction="both")

steps$anova

print("Number of selected variables:")
print(length(steps$coefficients)-1)
#5
#Fit a Ridge regression model with the same predictor and response variables.
#Present a plot showing how model coefficients depend on
#the log of the penalty factor lambda and report how the coefficients change with lambda.
#Takes the scaled tecator of Channel1-100

## Scale takes (x - mean(x)) / sd(x))
covariates=scale(tecator[,2:101])
#Scales all varables other than ponse-variable, which in this case is the fat
```

```r
response=scale(tecator[,102])
#Using glmnet with alpha=0 gives the Ridge-Regression
model_ridge=glmnet(as.matrix(covariates), response, alpha=0,family="gaussian")
plot(model_ridge, xvar="lambda", label=TRUE, main="Ridge Regression\n")

#6
#Repeat step 5 but fit LASSO instead of the Ridge regression
#and compare the plots from steps 5 and 6. Conclusions?
model_lasso = glmnet(as.matrix(covariates), response, alpha = 1, family="gaussian")
plot(model_lasso, xvar="lambda", label=TRUE, main="Lasso Regression\n")

#7
#Use cross-validation to find the optimal LASSO model (make sure that case
#ambda 0s also considered by the procedure) , report the optimaal
#bda how many variables were chos
#senn the model and make conclusions. Pres
#ent also a plot showing the dependence of the CV sco comment
#how the CV score changes with lambda.

cv_lasso_model = cv.glmnet(as.matrix(covariates), response, alpha=1, family="gaussian", nfolds=40)
plot(cv_lasso_model, main="Plot CV")
coef(cv_lasso_model, s="lambda.min")
print(cv_lasso_model$lambda.min)
```

Lab 2

Assignment 1

```r
setwd("~/Programming/TDDE01/Lab 2")
australian_crabs <- read.csv("australian-crabs.csv")
set.seed(12345)

#1
library(MASS)
library(ggplot2)
str(australian_crabs)

australian_crabs.gender = split(australian_crabs, australian_crabs$sex)

#Plots matrix in different colors in regards to gender
plot_crabs <-
  ggplot(australian_crabs, aes(x = CL, y = RW, color = australian_crabs$sex)) + geom_point()
plot_crabs

#2
#LDA-model with target sex and features CL and RW. Gives proportional prior
lda_crabs <- lda(sex ~ RW + CL, data = australian_crabs)
print(lda_crabs)

#Predicts the LDA-model
lda_crabs.predicted = predict(lda_crabs)

#Scatterplots the lda-model in different colors in regard to sex
plot_crabs_lda <-
```

```r
  ggplot(australian_crabs,
         aes(x = CL, y = RW, color = lda_crabs.predicted$class)) + geom_point()
plot_crabs_lda

confusionmatrix.lda <-
  table(lda_crabs.predicted$class, australian_crabs$sex)
print(confusionmatrix)

missclassification.lda <-
  1 - (sum(diag(confusionmatrix.lda)) / sum(confusionmatrix.lda))
print(missclassification.lda)
#It is good

#3
#Sets the priors to 0.1=female and 0.9=male
lda_crabs_prior.predicted = predict(lda_crabs, prior = c(0.1, 0.9))
#Plots like #2
plot_crabs_lda_prior <-
  ggplot(australian_crabs,
         aes(x = CL, y = RW, color = lda_crabs_prior.predicted$class)) + geom_point()
plot_crabs_lda_prior

confusionmatrix_prior.lda <-
  table(lda_crabs_prior.predicted$class, australian_crabs$sex)

print(confusionmatrix_prior.lda)

missclassification_prior.lda <-
  1 - (sum(diag(confusionmatrix_prior.lda)) / sum(confusionmatrix_prior.lda))
print(missclassification_prior.lda)


#4

#Gives the GLM for the target sex with features RW and CL
glm_crabs = glm(sex ~ RW + CL, family = binomial, data = australian_crabs)
glm_crabs.predicted = predict(glm_crabs, type = "response")
print(glm_crabs.predicted)
#Split data into Male and Female depending on the predicted value
glm_crabs.predicted <-
  ifelse(glm_crabs.predicted > 0.5, "Male", "Female")
print(glm_crabs.predicted)

plot_glm_crabs <-
  ggplot(australian_crabs, aes(x = CL, y = RW, color = glm_crabs.predicted)) + geom_point()
plot_glm_crabs

confusionmatrix.glm <-
  table(glm_crabs.predicted, australian_crabs$sex)
print(confusionmatrix.glm)

missclassification.glm <-
  1 - (sum(diag(confusionmatrix.glm)) / sum(confusionmatrix.glm))
```

```r
print(missclassification.glm)


plot_glm_line <-
  plot_glm_crabs + stat_function(
    fun = function(x) {
      (glm_crabs[["coefficients"]][["(Intercept)"]]
       + glm_crabs[["coefficients"]][["CL"]] *
         x) / (-glm_crabs[["coefficients"]][["RW"]])
    },
    xlim = c(5, 50)
  )
plot_glm_line
```

Assignment 2

```r
setwd("~/Programming/TDDE01/Lab 2")
library(tree)
library(readxl)
library(MASS)
library(e1071)

data <- read_excel("creditscoring.xls")
set.seed(12345)

#Splits data into training, validation and test
n=dim(data)[1]
id=sample(1:n, floor(n*0.5))
train=data[id,]
id1=setdiff(1:n, id)
set.seed(12345)
id2=sample(id1, floor(n*0.25))
valid=data[id2,]
id3=setdiff(id1,id2)
test=data[id3,]


#Train and test for deviance
fit.train.dev <- tree(ifelse(good_bad == "good", 1, 0) ~ . , data=train, split="deviance")

#Train and test for gini
fit.train.gini <- tree(ifelse(good_bad == "good", 1, 0)  ~ . , data=train, split="gini")

#Predictions
fit_pred.train.dev <- predict(fit.train.dev, newdata=train)

fit_pred.train.gini <- predict(fit.train.gini, newdata=train)

fit_pred.test.dev <- predict(fit.train.dev, newdata=test)

fit_pred.test.gini <- predict(fit.train.gini, newdata=test)

#Conf matrix
```

```r
confMa.train.dev <- table(train$good_bad,(ifelse(fit_pred.train.dev > 0.5, "good", "bad")))
confMa.test.dev <- table(test$good_bad,(ifelse(fit_pred.test.dev > 0.5, "good", "bad")))
confMa.train.gini <- table(train$good_bad,(ifelse(fit_pred.train.gini > 0.5, "good", "bad")))
confMa.test.gini <- table(test$good_bad,(ifelse(fit_pred.test.gini > 0.5, "good", "bad")))

#Misclassification rates
misCl.train.dev <- 1-(sum(diag(confMa.train.dev))/sum(confMa.train.dev))
misCl.test.dev <- 1-(sum(diag(confMa.test.dev))/sum(confMa.test.dev))
misCl.train.gini <- 1-(sum(diag(confMa.train.gini))/sum(confMa.train.gini))
misCl.test.gini <- 1-(sum(diag(confMa.test.gini))/sum(confMa.test.gini))

#Prints the miscl. rates
misCl.train.dev
misCl.test.dev
misCl.train.gini
misCl.test.gini

#I choose dev as it has the best miscl-rate

#3
trainScore=rep(0,9)
validationScore=rep(0,9)

for (i in 2:9) {
  prunedTree <- prune.tree(fit.train.dev, best=i)
  prediction <- predict(prunedTree, newdata = valid, type = "tree") # Predict with the pruned tree and
  trainScore[i] <- deviance(prunedTree) # Calculate deviance of test set
  validationScore[i] <- deviance(prediction) # Calculate deviance of val set
}

plot(2:9, trainScore[2:9], type="b", col="green", ylim=c(40,100), ylab="Deviance", xlab="No. of leaves"
points(2:9, validationScore[2:9], type="b", col="blue", ylim=c(40,100))
legend("top", legend=c("Training score (green)", "Validation score (blue)"))

#Optimal number of leaves
match(min(validationScore[2:9]),validationScore)
optPrunedTree <- prune.tree(fit.train.dev, best=match(min(validationScore[2:9]),validationScore))
plot(optPrunedTree, sub="asd")
text(optPrunedTree, pretty=0)
title("Pruned tree with 6 leaves")

#Estimates classification
predict.tree <- predict(optPrunedTree, newdata = test)
confMa.prune <- table(ifelse(predict.tree>0.5, "good", "bad"), test$good_bad)
testMisClass <- 1-sum(diag(confMa.prune)/sum(confMa.prune))
print(testMisClass)


#4
####NAIVE BAYES#######
#List the main advantage of Navie Bayes?
#A Naive Bayes classifier converges very quickly as compared to other models like logistic regression.
#As a result, we need less training data in case of naive Bayes classifier .
```

```
set.seed(12345)


naive.model = naiveBayes(good_bad ~ ., data=train)
naive.model$levels<-c('bad', 'good')

#Sets training data for naive
set.seed(12345)
naive.train = predict(naive.model, newdata=train)
confMa.naive.train = table(naive.train, train$good_bad)
misCl.naive.train = 1-(sum(diag(confMa.naive.train))/sum(confMa.naive.train))
print(confMa.naive.train)
print(misCl.naive.train)

#Sets test for naive
set.seed(12345)
naive.test = predict(naive.model, newdata=test)
confMa.naive.test = table(naive.test, test$good_bad)
misCl.naive.test = 1-(sum(diag(confMa.naive.test))/sum(confMa.naive.test))
print(confMa.naive.test)
print(misCl.naive.test)


#5
pi = seq(0.05, 0.95, by=0.05)
tree.model.dataframe = data.frame("0.05" = c(rep(0,250)))
tree.model.tpr = c(rep(0,length(pi)))
tree.model.fpr = c(rep(0,length(pi)))

for(k in 1:length(pi)) {
  tree.model.dataframe[, k] = ifelse(predict.tree > pi[k], 1, 0)
  confMa.tmp = table(ifelse(test$good_bad == 'good', 1, 0), tree.model.dataframe[, k])
  print(confMa.tmp)



  if (dim(confMa.tmp)[2] == 1) {
    if (colnames(confMa.tmp) == "1") {
      confMa.tmp <- cbind(as.matrix(c(0, 0)), confMa.tmp)
    } else {
      confMa.tmp <- cbind(confMa.tmp, as.matrix(c(0, 0)))
    }
  }

  tree.model.tpr[k] = confMa.tmp[2, 2] / (confMa.tmp[2, 1] + confMa.tmp[2, 2])
  tree.model.fpr[k] = confMa.tmp[1, 2] / (confMa.tmp[1, 1] + confMa.tmp[1, 2])

}
plot(x = tree.model.fpr, y = tree.model.tpr, type = "l", col="green", main="Green = tree, Blue = naive"


set.seed(12345)
#ROC for naive
```

```r
naive.test.raw = predict(naive.model, newdata = test, type="raw")

naive.model.dataframe = data.frame("0.05" = c(rep(0,250)))
naive.model.tpr = c(rep(0,length(pi)))
naive.model.fpr = c(rep(0,length(pi)))


for (k in 1:length(pi)) {
  naive.model.dataframe[, k] = ifelse(naive.test.raw[, 2] > pi[k], 1, 0)
  confMa.tmp = table(ifelse(test$good_bad == 'good', 1, 0), naive.model.dataframe[, k])

  if (dim(confMa.tmp)[2] == 1) {
    if (colnames(confMa.tmp) == "1") {
      confMa.tmp <- cbind(as.matrix(c(0, 0)), confMa.tmp)
    } else {
      confMa.tmp <- cbind(confMa.tmp, as.matrix(c(0, 0)))
    }
  }

  naive.model.tpr[k] = confMa.tmp[2, 2] / (confMa.tmp[2, 1] + confMa.tmp[2, 2])
  naive.model.fpr[k] = confMa.tmp[1, 2] / (confMa.tmp[1, 1] + confMa.tmp[1, 2])

}
lines(x = naive.model.fpr, y = naive.model.tpr, col="blue")

#6
naive.train.raw = predict(naive.model, newdata = train, type="raw")


#1*p(bad|x) > 10*p(good|x) -> bad, L12=1, L21=10, L12/L21=10
confMa.train.naive.loss = table(train$good_bad,
                                ifelse(naive.train.raw[,1]/naive.train.raw[,2] > 0.1, "bad", "good"))
print(confMa.train.naive.loss)
misCl.train.naive.loss <- 1-(sum(diag(confMa.train.naive.loss))/sum(confMa.train.naive.loss))
print(misCl.train.naive.loss)

confMa.test.naive.loss = table(test$good_bad,
                               ifelse(naive.test.raw[,1]/naive.test.raw[,2] > 0.1, "bad", "good"))


print(confMa.test.naive.loss)
misCl.test.naive.loss <- 1-(sum(diag(confMa.test.naive.loss))/sum(confMa.test.naive.loss))
print(misCl.test.naive.loss)
```

Assignment 3

```r
setwd("~/Programming/TDDE01/Lab 2")
library(ggplot2)
library(tree)
library(boot)
RNGversion('3.5.1')

#The data file State.csv contains per capita state and local public expenditures
```

```
#and associated state demographic and economic characteristics, 1960, and there are variables
#• MET: Percentage of population living in standard metropolitan areas
#• EX: Per capita state and local public expenditures ($)

#1
#Reorder your data with respect to the increase of MET and plot EX versus MET.
#Discuss what kind of model can be appropriate here. Use the reordered data in steps 2-5.
data <- read.csv2("State.csv")

#Orders data in ascending order
ordered_data <- data[order(data$EX), ]

plot <- ggplot() + geom_point(aes(x = EX, y = MET), data = data)
plot

#2
#Use package tree and fit a regression tree model with target EX and feature MET in which the number
#of the leaves is selected by cross-validation, use the entire data set and set minimum number of
#observations in a leaf equal to 8 (setting minsize in tree.control). Report the selected tree.
#Plot the original and the fitted data and histogram of residuals. Comment on the distribution of
#the residuals and the quality of the fit.

#Nobs is the number of rows to be used from the data. In this case it is everything
model.tree <-
  tree(EX ~ MET,
       data = ordered_data,
       control = tree.control(nobs = 48, minsize = 8))
model.tree.cv <- cv.tree(model.tree, FUN = prune.tree)
#Can plots dev vs size and can see that 3 is the best tree
plot_cv_tree <-
  ggplot() + geom_line(aes(x = model.tree.cv$size, y = model.tree.cv$dev)) + labs(x =
                                                                                     "Tree size")
plot_cv_tree

model.tree.pruned <- prune.tree(model.tree, best = 3)

#Plots the pruned tree with 3 leaves
plot(model.tree.pruned)
text(model.tree.pruned)

#Plots the residuals of the tree model
histo_tree_res <-
  ggplot(data.frame(residuals(model.tree)), aes(residuals(model.tree))) +
  geom_histogram(bins = 25, color = "red", fill = "red") +
  labs(x = "Residuals") + geom_density(alpha = .2, fill =
                                         "#FF6666")
histo_tree_res

#Predicts the tree with same ordered data with the pruned tree with 3 leaves
model.tree.predicted <- predict(model.tree.pruned, ordered_data)

#Plots the tree data with the original data as blue dots and the
plot_tree_data <-
```

```r
  ggplot() + geom_line(aes(x = ordered_data$MET, y = data.frame(model.tree.predicted)[, 1]), color =
                         "green") +
  geom_point(aes(x = ordered_data$MET, y = ordered_data$EX), color =
               "black") + labs(title = "Plotted tree data", x = "MET", y = "Predicted EX")
plot_tree_data

#3
#Compute and plot the 95% confidence bands for the regression tree model from step 2
#(fit a regression tree with the same settings and the same number of leaves as in step 2 to
#the resampled data) by using a non-parametric bootstrap. Comment whether the band is smooth or
#bumpy and try to explain why. Consider the width of the confidence band and comment whether
#results of the regression model in step 2 seem to be reliable.

#Function to be used in the bootstrapping function for the bootstrapping.
#Set after the statics word
boot_function <- function(data_boot, id) {
  data_boot <- data_boot[id, ]
  model.tree.boot <-
    tree(EX ~ MET,
         data = data_boot,
         control = tree.control(nobs = 48, minsize = 8))
  model.tree.boot.pruned <- prune.tree(model.tree.boot, best = 3)
  return(predict(model.tree.boot.pruned, newdata = ordered_data))
}

set.seed(12345)
bootres <- boot(ordered_data, boot_function, R = 1000)
bootres.envelope <- envelope(bootres)

#Plots everything including the confidence intervals
plot_boot <-
  ggplot() + geom_point(aes(
    x = ordered_data$MET,
    y = ordered_data$EX,
    color = "Original data-points"
  )) +
  geom_line(aes(
    x = ordered_data$MET,
    y = data.frame(model.tree.predicted)[, 1],
    color = "Predicted tree data"
  )) +
  scale_color_manual(
    name = '',
    values = c(
      'Confidence level 95%' = 'red',
      'Original data-points' = 'black',
      'Predicted tree data' = 'green',
      'Confidence level 95% parametric' = 'blue',
      'Confidence level 95% predicted' = 'black'
    )
  )

plot_boot_3 <- plot_boot +
  geom_line(aes(
    x = ordered_data$MET,
    y = bootres.envelope$point[1, ],
    color = "Confidence level 95%"
  )) +
  geom_line(aes(
    x = ordered_data$MET,
    y = bootres.envelope$point[2, ],
    color = "Confidence level 95%"
  )) + labs(title = "Non-parametric bootstrap", x = "MET", y = "EX")

plot_boot_3

#4
#Compute and plot the 95% confidence and prediction bands the regression tree model
#from step 2 (fit a regression tree with the same settings and the same number of
#leaves as in step 2 to the resampled data) by using a parametric bootstrap,
#assume Y~N(myi, sigma^2) (normal distribution) where myi are labels in the tree leaves
# and sigma^2 is the residual variance. Consider the width of the confidence band and
#comment whether results of the regression model in step 2 seem to be reliable.

boot_function2 <- function(data_boot2) {
  model.tree.boot2 <-
    tree(EX ~ MET,
         data = data_boot2,
         control = tree.control(nobs = 48, minsize = 8))
  model.tree.pruned.boot2 <- prune.tree(model.tree.boot2, best = 3)
  return(predict(model.tree.pruned.boot2, data_boot2))
}

#Random generating function for generating new EX values
boot.random <- function(data, mle) {
  dat <- data.frame(EX = data$EX, MET = data$MET)
  n = length(data$EX)
  dat$EX <- rnorm(n, predict(mle, dat), sd(resid(mle)))
  return(dat)
}


set.seed(12345)
bootres2 <-
  boot(
    data = ordered_data,
    statistic = boot_function2,
    mle = model.tree.pruned,
    ran.gen = boot.random,
    R = 1000,
    sim = "parametric"
  )
bootres2.envelope <- envelope(bootres2)

plot_boot4 <- plot_boot +
  #Plots the confidence interval
```

```r
  geom_line(
    aes(
      x = ordered_data$MET,
      y = bootres2.envelope$point[1, ],
      color = "Confidence level 95% parametric"
    )
  ) +
  geom_line(
    aes(
      x = ordered_data$MET,
      y = bootres2.envelope$point[2, ],
      color = "Confidence level 95% parametric"
    )
  ) + geom_line(
    #Plots the predicted bands
    aes(
      x = ordered_data$MET,
      y = bootres2.envelope$overall[2, ],
      color = "Confidence level 95% predicted"
    )
  ) + geom_line(
    aes(
      x = ordered_data$MET,
      y = bootres2.envelope$overall[1, ],
      color = "Confidence level 95% predicted"
    )
  ) + labs(title = "Parametric bootstrap", x = "MET", y = "EX")

plot_boot4
#This graph shows the confidence band (blue) and the prediction band (black)
#for the parametric bootstrap. The confidence band shows the confidence interval
#for the sample and the prediction band shows the prediction interval for the
#estimations. As one can see the prediction interval is quite good,
#but the confidence interval isnt that good as it misses more than 5 % of the
#data points.

#5
#Consider the histogram of residuals from step 2 and suggest what
#kind of bootstrap is actually more appropriate here.

#What one can see in the residual plot from step 2 is that there isnt a
#predictable distribution and it is thereby better to use the non-parametric
#bootstrap solution as that one doesnt depend on a certain distribution
```

Assignment 4

```r
setwd("~/Programming/TDDE01/Lab 2")
library(fastICA)

data <- read.csv2("NIRSpectra.csv")

data1 <- data
```

```r
######SUMMARY########
#######PCA##########
#It reduces the dimensions to avoid the problem of overfitting.
#It deals with the Principal Components.
#It focuses on maximizing the variance.
#It focuses on the mutual orthogonality property of the principal components.
#It doesn't focus on the mutual independence of the components.

#######ICA##########
#It decomposes the mixed signal into its independent sources' signals.
#It deals with the Independent Components.
#It doesn't focus on the issue of variance among the data points.
#It doesn't focus on the mutual orthogonality of the components.
#It focuses on the mutual independence of the components.

#1
#Conduct a standard PCA by using the feature space and provide
#a plot explaining how much variation is explained by each feature.
#Does the plot show how many PC should be extracted? Select the minimal
#number of components explaining at least 99% of the total variance.
#Provide also a plot of the scores in the coordinates (PC1, PC2).
#Are there unusual diesel fuels according to this plot?
data1$Viscosity = c()
#Used to reset the Viscocity

res = prcomp(data1)

lambda = res$sdev^2

lambda

#Prints the effect of each in percentage
sprintf("%2.3f", (lambda)/sum(lambda)*100)

#Histogram of variance
screeplot(res)
#As can be seen in the plot above in combination with the
#lambda there are two features that explains $99\%$ of the total variance.
plot(res$x[,1], res$x[,2], main = "PC1 vs. PC2", xlab = "PC1", ylab = "PC2")
#The plot above shows the diesel fuels according to
#the features PC1 and PC2. There are a few outliers in this plot.
#Mainly the ones with a high value of the PC1.


#2
#Make trace plots of the loadings of the components selected in step 1.
#Is there any principle component that is explained by mainly a
#few original features?
U <- res$rotation
plot(U[,1], main="Traceplot for PC1")
#The plot above shows the traceplot for PC1. As can be seen
#in the plot it is not mainly explained by just a few few original
#features, but by a lot of them, but not so much by the ones around index 105.
```

```r
plot(U[,2], main="Traceplot for PC2")
#The plot abovee shows the traceplot for PC2. As can be
#seen in the plot it is mainly explained by a feew features
#around the higher index around 120.

#3
#Perform Independent Component Analysis with the number of
#components selected in step 1 (set seed 12345).
#Check the documentation for the fastICA method in R and do the following:
#  a. Compute W'= K * W and present the columns of W' in
#  form of the trace plots. Compare with the trace plots in step 2
#  and make conclusions. What kind of measure is represented by the matrix W'?
#  b. Make a plot of the scores of the first two latent features
#  and compare it with the score plot from step 1.

set.seed(12345)
ICA <- fastICA(data1, n.comp = 2, alg.typ = "parallel", fun = "logcosh", alpha = 1,
               method = "R", row.norm = FALSE, maxit = 200, tol = 0.0001,
               verbose = TRUE)

WTICK <- ICA$K %*% ICA$W

# Trace plot results from ICAs
plot(WTICK[,1], main= "Latent feature 1")
plot(WTICK[,2], main= "Latent feature 2")

#The two plots above both shows the latent features for the the W'
#columns 1 and 2. As can be seen they are inverted along
#the y-axis compared to the plots for PC1  and PC2.

# Plot of scores
plot(ICA$S[,1], ICA$S[,2], main = "Score", ylab = "Latent 2", xlab = "Latent 1")
#This plot is also a inverted version of the corresponding
#plot, PC1 vs PC2, but along the x-axis.
```

Lab 3

Assignment 1

```r
setwd("~/Programming/TDDE01/Lab 3")
RNGversion('3.5.1')
set.seed(1234567890)

library("geosphere")

stations <- read.csv("stations.csv")
temps <- read.csv("temps50k.csv")
st <- merge(stations, temps, by = "station_number")

## These values are up to the user. ud = user defined
ud.lat <- 59.325 # The lat of the point to predict
ud.long <- 18.071 # The long of the point to predict
ud.date <-"2010-05-24" # The date to predict (up to the students)
ud.h_distance <- 100000
ud.h_date <- 20
ud.h_time <- 4
##End input

ud.times <- c("04:00:00", "06:00:00","08:00:00","10:00:00","12:00:00","14:00:00",
           "16:00:00","18:00:00","20:00:00","22:00:00", "24:00:00")

#Filters posterior date
filterPosteriorDate <- function(data,date) {
  return(data[!(as.Date(data$date)-as.Date(date))>0,])
}

#Filters posterior time
filterPosteriorTime <- function(data, date, time) {
  return (data[!(as.Date(data$date) == as.Date(date) &
                 as.numeric(difftime(strptime(data$time, format="%H:%M:%S"),
                 strptime(time, format="%H:%M:%S"))) > 0 ),])
}
#Kernel for distance computing to point of interest
distGaussian <- function(data, target, h) {
  dist <- distHaversine(data.frame(data$longitude,data$latitude), target)
  u <- dist/h
  return (exp(-(u)^2));
}

#Kernel for date
dateGaussian <- function(data, target, h) {
  date_diff <- as.numeric(as.Date(data$date)-as.Date(target), unit="days")
  date_diff <- date_diff %% 365
  date_diff <- ifelse(date_diff > 182, 365-date_diff, date_diff)
  date_diff <- sort(date_diff)
  u <- date_diff/h
  return(exp(-(u)^2))
}

#Kernel for time
timeGaussian <- function(data,target,h) {
  time_difference <- difftime(strptime(data$time, format="%H:%M:%S"),
                              strptime(target, format="%H:%M:%S"))
  u <- as.numeric(time_difference/3600)/h
  return(exp(-(u)^2))
}

filtered_data <- filterPosteriorDate(st, ud.date)
length = length(ud.times)
t_sum <- vector(length = length)
t_multi <- vector(length = length)
t_summa_test <- vector(length = length)
t_multi_test<- vector(length = length)

for (i in 1:length) {
  print(i)
  filtered_data_by_time <-
    filterPosteriorTime(filtered_data, ud.date, ud.times[i])
  time <-
    timeGaussian(filtered_data_by_time, ud.times[i], ud.h_time)
  distance <-
    distGaussian(filtered_data_by_time,
                 data.frame(ud.long, ud.lat),
                 ud.h_distance)
  day <- dateGaussian(filtered_data_by_time, ud.date, ud.h_date)
  kernel_sum <- distance + day + time
  kernel_multi <- distance * day * time
  t_summa_test <- kernel_sum
  t_multi_test <- kernel_multi
  t_sum[i] <-
    sum(kernel_sum %*% filtered_data_by_time$air_temperature) / sum(kernel_sum)
  t_multi[i] <-
    sum(kernel_multi %*% filtered_data_by_time$air_temperature) / sum(kernel_multi)
}

temps <- list(t_sum=t_sum, t_multi=t_multi)

plot(temps$t_sum,xaxt='n', xlab="Time",
     ylab="Temperature", type="o", main = "Sum of kernels")
axis(1, at=1:length(ud.times), labels=ud.times)

plot(temps$t_multi,xaxt='n', xlab="Time",
     ylab="Temperature", type="o", main = "Multiplication of kernels")
axis(1, at=1:length(ud.times), labels=ud.times)


#To evaaluate h-values
plot(1:length(distance), distance, main = "Distance")
plot(1:length(day), day, main = "Day")
plot(1:length(time), distance, main = "Time")


#Below is the h-tests
h_test.plotKernalDistance <- function(distances, h) {
  u <- distances/h
  k <- exp(-u^2)
  plot(k, type="l", main = "H value for distance", xlab="Distance",)
}

h_test.plotKernalDate <- function(date_diff, h) {
  u <- date_diff/h
  date_diff <- date_diff %% 365
  date_diff <- ifelse(date_diff > 182, 365-date_diff, date_diff)
  k <- exp(-u^2)
  plot(k, type="l", main = "H value for date", xlab="Days",)
}

h_test.plotKernalHour <- function(time_diff, h) {
  u <- time_diff/h
  k <- exp(-u^2)
  plot(k, type="l", main = "H value for hour", xlab="Hours", xlim=c(0,12))
}

h_test.distance <- seq(0,300000,1)
h_test.date_diff <- seq(0,300,1)
h_test.time_diff <- seq(0,50,1)

h_test.plotKernalDistance(h_test.distance, ud.h_distance)
h_test.plotKernalDate(h_test.date_diff, ud.h_date)
h_test.plotKernalHour(h_test.time_diff, ud.h_time)
```

Assignment 2

```
setwd("~/Programming/TDDE01/Lab 3")
RNGversion('3.5.1')
library("kernlab")

data(spam)
data <- spam

n = dim(data)[1]
set.seed(12345)
id = sample(1:n, floor(n * 0.7))
train = data[id, ]
test = data[-id, ]

width <- 0.05

filter_0_5 <-
  ksvm(type ~ .,
       data = train,
       kernel = rbfdot(sigma = width),
       C = 0.5)
filter_3 <-
  ksvm(type ~ .,
       data = train,
       kernel = rbfdot(sigma = width),
       C = 3)
filter_5 <-
  ksvm(type ~ .,
       data = train,
       kernel = rbfdot(sigma = width),
       C = 5)
missCl <- function(data) {
  return(1 - sum(diag(data)) / sum(data))
}

filter_0_5.pred.train <- predict(filter_0_5, newdata = train)
filter_0_5.pred.test <- predict(filter_0_5, newdata = test)
filter_0_5.confMa.train <-
  table(train$type  , filter_0_5.pred.train)
print("Confma for C=0.5 with training data: ")
```

```r
filter_0_5.confMa.train
filter_0_5.confMa.test <- table(test$type  , filter_0_5.pred.test)
print("Confma for C=0.5 with training data: ")
filter_0_5.confMa.test

filter_0_5.missCl.train <- missCl(filter_0_5.confMa.train)
print("Misscl for C=3 with training data: ")
filter_0_5.missCl.train
filter_0_5.missCl.test <- missCl(filter_0_5.confMa.test)
print("Misscl for C=3 with testing data: ")
filter_0_5.missCl.test

filter_3.pred.train <- predict(filter_3, newdata = train)
filter_3.pred.test <- predict(filter_3, newdata = test)
filter_3.confMa.train <- table(train$type  , filter_3.pred.train)
print("Confma for C=3 with training data: ")
filter_3.confMa.train
filter_3.confMa.test <- table(test$type  , filter_3.pred.test)
print("Confma for C=3 with training data: ")
filter_3.confMa.test

filter_3.missCl.train <- missCl(filter_3.confMa.train)
print("Misscl for C=3 with training data: ")
filter_3.missCl.train
filter_3.missCl.test <- missCl(filter_3.confMa.test)
print("Misscl for C=3 with testing data: ")
filter_3.missCl.test

filter_5.pred.train <- predict(filter_5, newdata = train)
filter_5.pred.test <- predict(filter_5, newdata = test)
filter_5.confMa.train <- table(train$type  , filter_5.pred.train)
print("Confma for C=5 with training data: ")
filter_5.confMa.train
filter_5.confMa.test <- table(test$type  , filter_5.pred.test)
print("Confma for C=5 with training data: ")
filter_5.confMa.test

filter_5.missCl.train <- missCl(filter_5.confMa.train)
print("Misscl for C=5 with training data: ")
filter_5.missCl.train
filter_5.missCl.test <- missCl(filter_5.confMa.test)
print("Misscl for C=5 with testing data: ")
filter_5.missCl.test

print("Summary: ")
filter_0_5.missCl.train
filter_3.missCl.train
filter_5.missCl.train

print("Summary: ")
filter_0_5.missCl.test
filter_3.missCl.test
filter_5.missCl.test
```

```r
#Best is C=5.
filter_5
#Above prints the model
```

Assignment 3

```r
library(neuralnet)
library(ggplot2)

set.seed(1234567890)
#50 values between 0 and 10
Var <- runif(50, 0, 10)

# Create dataset
trva <- data.frame(Var, Sin=sin(Var))

# Divide dataset into training and validation set
tr <- trva[1:25,] # Training
va <- trva[26:50,] # Validation

# Random initialization of the weights in the interval [-1, 1]
# 31 weights are used
# Nw = (I+1)*H1 +(H1+1)*H2 +(H2+1)*O, where H is hidden
# layer, I is inputs and O is outputs.
# So in this case it is Nw=(1+1)*10+(10+1)*1=31
winit <- runif(31, -1, 1)

# Function predicting MSE
mse <- function(prediction, observation) {
  return (mean((observation - prediction)^2))
}

mse_val <- numeric()
mse_train <- numeric()
threshold <- numeric()
m_sq_err <- function(pred, obs) {
  return (mean((observation - prediction)^2))
}

for(i in 1:10) {
  nn <- neuralnet(Sin ~ Var, data = tr, startweights = winit, hidden = c(10),
                  threshold = i/1000)
  pred_train <- compute(nn, covariate=tr)$net.result
  pred_val <- compute(nn, covariate=va)$net.result
  threshold[i] <- i/1000
  mse_val[i] <- mse(pred_val, va$Sin)
  mse_train[i] <- mse(pred_train, tr$Sin)
  print(i)
}

plot(threshold, mse_val, type="o", ylab="MSE", main = "Validation dataset")
plot(threshold, mse_train, type="o", ylab="MSE", main = "Training dataset")
```

```r
# Your code here
nn <- neuralnet(Sin ~ Var, data = trva, startweights = winit, hidden = c(10),
                    threshold = 4/1000)
plot(nn)

pred_nn <- prediction(nn)$rep1
plot_net <- ggplot() + geom_point(aes(pred_nn[,1], pred_nn[,2])) +
  geom_point(aes(trva$Var, trva$Sin), colour="red")
plot_net
```

Exam 2017-04-17

Assignment 1

```r
setwd("~/Programming/TDDE01/Exam_2017-04-17")

data <- read.csv("australian-crabs.csv")
library(ggplot2)
#1
#Plot the dependence of CW versus BD where the points are colored by Species.
#Are CW and BD good predictors of the Species?
plot1 <-
    ggplot(aes(
        x = data$CW,
        y = data$BD,
        color = data$sex
    ), data = data) + geom_point()
plot1
#No, it is not that good of a classifier for the data

#2
#Create a Naïve Bayes classifier model with Species as target and CW and BD as predictors.
#Present the confusion matrix and comment on the quality of the classification.
#Based on the assumptions of the Naïve Bayes, explain why this model
#is not appropriate for these data
library(e1071)
model <- naiveBayes(sex ~ CW + BD, data = data)
model.pred <- predict(model, newdata = data)
confMa.naive <- table(model.pred, data$sex)
confMa.naive
missCla.naive <- 1 - (sum(diag(confMa.naive)) / sum(confMa.naive))
missCla.naive
#This model is not good for the data as it uses the


#3
#Fit the logistic regression now with Species as target and CW and BD as predictors
#and present the equation of the decision boundary. Plot the classified data and the
#decision boundary and comment on the quality of the classification
model.logistic <- glm(species ~ CW + BD, data = data, family = binomial)
model.logistic.pred <- predict(model.logistic, type = "response")

model.logistic.pred.classified <-
```

```r
    ifelse(model.logistic.pred > 0.5, "orange", "blue")

plot(model.logistic.pred)

plot2 <-
    ggplot(aes(x = (seq(1, 200, 1)), y = model.logistic.pred, color = data$species),
                 data = data) + geom_point() + geom_hline(yintercept = 0.5)
plot2

#The classification is quite good! There are only a few missclassified samples.


#4
#Scale variables CW and BD and perform principal component analysis with these two variables.
#Present the proportion of variation explained by PC1 and PC2 and based on results from step 1
#explain why the first principal component contains so much variation. Present the equations
#expressing principal component coordinates through the original coordinates.
res = prcomp(~CW+BD, data = data, scale = TRUE)

pcdata = data.frame(species = data$species, PC1 = res$x[,1], PC2 = res$x[,2])

#5
#Create a Naïve Bayes classifier model with Species as target and PC1 and PC2 as predictors.
#Compute the confusion matrix and explain how much the classification quality has changed and why.
model5 <- naiveBayes(species ~ PC1+PC2, data=pcdata)
model5.pred <- predict(model5, newdata=pcdata)
ConfMa <- table(model5.pred, data$species)
missCl <- 1-sum(diag(ConfMa))/sum(ConfMa)
```

Assignment 2

```r
#File bank.csv shows the number of customers (Visits) that arrived to a
#bank during various time slots (Time) between 9.00 and 12.00.
#1. Fit a generalized linear model in which response is Poisson distributed,
#and the canonical link (log) is used for regression. Report the probabilistic
#expression for the fitted model (how the target is distributed based on the feature) (1p)
#2. Compute a prediction band for the values of Time=12,12.05,12.1,...,13.0
#by using the model from step 1 and the parametric bootstrap with B=1000.
#Plot the original data values and the prediction band into one figure and
#comment whether the band seems to give a correct forecasting.
#How many customers (report a range) should the bank expect at 13.00? (3p)
setwd("~/Programming/TDDE01/Exam_2017-04-17")
RNGversion('3.5.1')

data <- read.csv2("bank.csv")

model <-
    glm(Visitors ~ Time, data = data, family = poisson(link = log))

fit <- fitted(model)
fit
plot(fit, xlab = "Time", xaxt = "n")
axis(1, at = 1:length(data$Time), labels = data$Time)
```

```r
data2 <- data.frame(Time = data$Time, Visitors = fit)
plot <-
    ggplot(aes(x = data2$data.Time, y = data2$fit), data = data2) + geom_point()
plot

#2
library(boot)

bootfunc <- function(databoot) {
    modelboot<-glm(Visitors ~ Time,
            data = databoot,
            family = poisson(link = log))
    return(fitted(modelboot))
}

bootstrapped <-
    boot(
        data = data2,
        statistic = bootfunc,
        R = 1000,

        sim = "parametric"
    )
```

Assignment 3

```r
setwd("~/Programming/TDDE01/Exam_2017-04-17")
RNGversion('3.5.1')

library(kernlab)
#In this assignment, you are asked to use the R package kernlab to learn SVMs for
#classifying the spam dataset that is included with the package. Consider the radial
#basis function kernel (also known as Gaussian) with a width of 0.05. For the C
#parameter, consider values 1, 10 and 100.
#(2p) Estimate the error for the three values of C. Use cross-validation with 2 folds.
#Hint: Use the argument cross=2 when calling the function ksvm. Use the function cross()
#to print out the error estimate. Use set.seed(1234567890).
#(2p) In the previous question, the error estimate may not be mono- tone with respect to
#the value of C. Explain why this happens.
data(spam)

data <- spam
set.seed(1234567890)
model1 <-
  ksvm(
    type ~ .,
    C = 1,
    kernel = rbfdot(sigma = 0.05),
    cross = 2,
    data = data
  )
set.seed(1234567890)
```

```
model10 <-
  ksvm(
    type ~ .,
    C = 10,
    kernel = rbfdot(sigma = 0.05),
    cross = 2,
    data = data
  )
set.seed(1234567890)

model100 <-
  ksvm(
    type ~ .,
    C = 100,
    kernel = rbfdot(sigma = 0.05),
    cross = 2,
    data = data
  )

cross(model1)
cross(model10)
cross(model100)

#NN
#In this assignment,you are asked to use the Rpackage neuralnet to train a NN to learn the trigonometri
#To produce the learning data, sample 50 points uniformly at random in the interval [0, 10] and, then,
#apply the sine function to each point.Your task is to estimate the mean squared error of a
#NN with a single hidden layer of 10 units for the regression task described above. Use cross-validatio
#with 2 folds. For the training, initialize the weights of the NN to random values in the interval [-1,
#Stop the training when the partial derivatives of the error function are below a threshold value of 0.
library(neuralnet)
set.seed(1234567890)
Var <- runif(50, 0, 10)
tr <- data.frame(Var, Sin = sin(Var))
tr1 <- tr[1:25, ] # Fold 1
tr2 <- tr[26:50, ] # Fold 2

set.seed(1234567890)

winit <- runif(31,-1, 1)
set.seed(1234567890)

nn1 <-
  neuralnet(
    Sin ~ Var,
    data = tr1,
    startweights = winit,
    hidden = c(10),
    threshold = 1 / 1000
  )
nn1.pred <- predict(nn1, newdata = tr2)
mse1 <- mse(tr1$Sin, nn1.pred)
set.seed(1234567890)
```

```r
nn2 <-
  neuralnet(
    Sin ~ Var,
    data = tr2,
    startweights = winit,
    hidden = c(10),
    threshold = 1 / 1000
  )
nn2.pred <- predict(nn2, newdata = tr1)
mse2 <- mse(tr2$Sin, nn2.pred)

mean(c(mse1, mse2))
```