Lab3

Christian von Koch

2019-12-17

#Lab 1

##Assignment 1

```
#1: Read data and divide into test and train sets

Dataframe=read.csv2("spambase.csv")
n=dim(Dataframe)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=Dataframe[id,]
test=Dataframe[-id,]

#2: Use logistic regression (functions glm(), predict()) to classify the training and test data by the

#Create model for prediction
spammodel = glm(Spam~., family='binomial', data=train)
summary(spammodel)

#Predict values and create confusion matrix for traindata
predicted_values_train = predict(spammodel, newdata=train, type='response')
confusion_matrix_train = table(train$Spam, predicted_values_train>0.5)
print(confusion_matrix_train)

#Predict values and create confusion matrix for testdata
predicted_values_test = predict(spammodel, newdata=test, type='response')
confusion_matrix_test = table(test$Spam, predicted_values_test>0.5)
print(confusion_matrix_test)

#Create function for misclassification rate
missclass=function(conf_matrix, fit_matrix){
  n=length(fit_matrix[,1])
  return(1-sum(diag(conf_matrix))/n)
}

#Calculate missclassification rate for train and test data
missclass_train = missclass(confusion_matrix_train, train)
print(missclass_train)
missclass_test = missclass(confusion_matrix_test, test)
print(missclass_test)

#Conclusion: It is reasonable that the model performs better on the train data compared to the test dat

#3: Use logistic regression to classify the test data by the classification principle: Same as above bu
#threshold 0.8. Compare the results. What effect did the new rule have.

#Create confusion matrix where classification is based on threshold 0.8
confusion_matrix_train2 = table(train$Spam, predicted_values_train>0.8)
```

```
confusion_matrix_test2 = table(test$Spam, predicted_values_test>0.8)
print(confusion_matrix_train2)
print(confusion_matrix_test2)

#Calculate missclassification rate for train and test data with threshold 0.8
missclass_train2 = missclass(confusion_matrix_train2, test)
print(missclass_train2)
missclass_test2 = missclass(confusion_matrix_test2, train)
print(missclass_test2)

#Conclusion: The misclassification rates have similar results. Showing us that model is well fitted, si

#4: Use standard classifier kknn() with K=30 from package kknn, report the misclassification rates for

#Fetch package kkm
#install.packages("kknn")
library("kknn")

#Classify according to kknn with k=30 for test and train data sets
kknn_30_train = kknn(formula = Spam~., train, train, k=30)
kknn_30_test = kknn(formula = Spam~., train, test, k=30)
confusion_matrix_kknn30_train = table(train$Spam, kknn_30_train$fitted.values>0.5)
missclass_kknn30_train = missclass(confusion_matrix_kknn30_train, train)
confusion_matrix_kknn30_test = table(test$Spam, kknn_30_test$fitted.values>0.5)
missclass_kknn30_test = missclass(confusion_matrix_kknn30_test, test)
print(confusion_matrix_kknn30_train)
print(missclass_kknn30_train)
print(confusion_matrix_kknn30_test)
print(missclass_kknn30_test)

#Conclusion: The misclassification values between predictions of the different sets differ alot. This s

#5: Repeat step 4 for K=1. Classify according to kknn with k=1 for test and train data sets. What does

kknn_1_train = kknn(formula = Spam~., train, train, k=1)
kknn_1_test = kknn(formula = Spam~., train, test, k=1)
confusion_matrix_kknn1_train = table(train$Spam, kknn_1_train$fitted.values>0.5)
missclass_kknn1_train = missclass(confusion_matrix_kknn1_train, train)
confusion_matrix_kknn1_test = table(test$Spam, kknn_1_test$fitted.values>0.5)
missclass_kknn1_test = missclass(confusion_matrix_kknn1_test, test)
print(confusion_matrix_kknn1_train)
print(missclass_kknn1_train)
print(confusion_matrix_kknn1_test)
print(missclass_kknn1_test)

#Conclusion: The misclassification rate for the training confusion matrix is zero since it compares eac
```

##Assignment 2

```
#1: Import data
Dataframe=read.csv2("machines_csv.csv")

#2: Assume probability model p(x|theta) = theta*e^(-theta*x) for x = Length in which observations are i
```

```
#Compute a function for calculating the maximum likelihood of a function
loglikelihood=function(theta, x){
  n = length(x[,1])
  return(n*log(theta)-theta*sum(x))
}

#Plot curve for different theta values
theta_curve = curve(-loglikelihood(x, Dataframe), xlab="Theta", from=min(Dataframe), to=max(Dataframe))

#Find maximum likelihood value of theta
theta_max = function(x){
  n=length(x[,1])
  return(n/sum(x))
}

#Find maxtheta
max_theta = theta_max(Dataframe)
print(max_theta)

#Conclusion: We can see from the probabilistic model that the distribution is of type exponential. The m
##likelihood value of theta is: 42.29453 The optimal theta for is: 1.126217

#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 solu
#each case?

#New vector with first 6 values
y = matrix(Dataframe[1:6,1], nrow=length(Dataframe[1:6,1]), ncol=1)
print(y)

#Plot new curve on top of each other
curve(-loglikelihood(x, Dataframe), xlab="Theta", from=0, to=20, add=FALSE, col="red", ylim=c(0,100))
curve(-loglikelihood(x, y), xlab="Theta", from=0, to=20, add=TRUE, col="blue", ylim=c(0,100))

#Conclusion: The graph is increasing at a much slower pace when only using the first six values compare

#4: Assume now a Bayesian model with p(x|theta)=theta*e^(-theta*x) and a prior p(theta)=lambda*e^(-lamb
#Write a function computing l(theta)=log(p(x|theta)*p(theta)). What kind of measure is actually compute
#function? Plot the curve showing the dependence of l(theta) on theta computed using the entire data an

#Compute a function for calculating the likelihood of the bayesian function
bayesian_likelihood=function(theta, lambda, x){
  n = length(x[,1])
  return(n*log(theta)-theta*sum(x)-lambda*theta)
}

#Find maximum likelihood value of theta
bayesian_theta_max = function(lambda, x){
  n=length(x[,1])
  return(n/(sum(x)+lambda))
}

#Find maxtheta
```

```
bayesian_max_theta = bayesian_theta_max(10, Dataframe)
print(bayesian_max_theta)

#Plot new curve on top of each other
curve(-bayesian_likelihood(x, 10, Dataframe), ylab="-Loglikelihood", xlab="Theta", from=0, to=10, add=F
      ylim=c(20,300))
curve(-loglikelihood(x, Dataframe), ylab="-Loglikelihood", xlab="Theta", from=0, to=10, add=TRUE, col="
      ylim=c(20,300))

#Conclusion: When using an bayesian model we have a prior that gives the model information beforehand w
#fitting the model. The optimal theta is now 0.91 which is close to the datasets meanvalue, which makes

#5: Use theta value found in step 2 and generate 50 new observations from p(x|theta)=theta*e^(-theta*x)

#Generate 50 new observation using theta value from step 2
set.seed(12345)
newdata = rexp(50, rate = max_theta)
print(newdata)

#Plot new data and old data in histogram
olddata = Dataframe$Length
print(olddata)
hist(newdata)
hist(olddata)

#Conclusion: The histogram shows us that the distribution is fairly similar between the actual and pred
```

##Assignment 4

```
#1: Read data and plot Moisture vs Protein
Dataframe=read.csv2("tecator_csv.csv")
n = length(Dataframe[,1])
print(Dataframe)
moisture = Dataframe$Moisture
protein = Dataframe$Protein
fat = Dataframe$Fat
plot(moisture, protein, type="p", ylab="Protein", xlab="Moisture", col="red")

#Conclusion: Looks like a linear relation so a linear regression model is appropriate

#2: Consider model Mi in  which Moisture is normally distributed, and the expected Moisture is a polyno
#of Protein including the polynomial terms up to power of i (i.e. M1 is a linear model, M2 is a quadrat

#Conclusion: A probabilistic model describing M(i) is: M(i) = w0 + w1 * X + w2 * X2 + ... + wi * Xi (3)
#criterion is a suitable method since it punishes outliers to a larger extent. This creates a better fi
#compared to when you punish the absolute value. This reduces the risk of an overfitted model.

#3: Divide the data into training and validation sets (50%/50%) and fit models Mi, i=1,...,6.  For each

colno_protein = which(colnames(Dataframe)=="Protein")
colno_moisture = which(colnames(Dataframe)=="Moisture")
moisture_protein = Dataframe[colno_protein:colno_moisture]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
```

```r
train=Dataframe[id,]
test=Dataframe[-id,]
moisture_train = train$Moisture
moisture_test = test$Moisture

#Create function for fitting linear regression models
fit_moisture_model = function(x) {
  return(lm(formula = Moisture ~ poly(Protein, degree=x), data=train))
}

#Create predict function for training set
predict_train = function(model){
  return(predict(model, newdata = train))
}

#Create predict function for test set
predict_test = function(model){
  return(predict(model, newdata = test))
}

#Create function for calculating MSE, input parameters are vectors containing original data and predict
calcMSE = function(y, yhat){
  return(sum((y-yhat)^2)/length(y))
}

#Create models and predictions and store MSE values in a vector for training and test data
vector_train = c()
vector_test = c()
for (i in 1:6){
  fit = fit_moisture_model(i)
  predicted_train = predict_train(fit)
  predicted_test = predict_test(fit)
  vector_train[i] = calcMSE(moisture_train, predicted_train)
  vector_test[i] = calcMSE(moisture_test, predicted_test)
}

#Create numeric vector 1 through 6
models = c(1:6)

#Plot MSE for each model
plot(models, vector_train, col="blue", xlab="Model", ylab="MSE")
par(new=TRUE)
plot(models, vector_test, col="red", xlab="Model", ylab="MSE")

#Print MSE values
print(vector_train)
print(vector_test)

#Conclusion: Shown in the graphs we can see that for the tested values, M(3) has the lowest MSE and the

#Use the entire data set in the following computations:

#4: Fat response and Channel1-100 are predictors. Use stepAIC. How many variables were selected?
```

```
#Fetch package stepAIC
#install.packages("MASS")
library("MASS")

#Perform stepAIC on fat
colno_fat = which(colnames(Dataframe)=="Fat")
channel_values = Dataframe[1:(colno_fat-1)]
fit_fat = lm(fat~., data = channel_values)
step = stepAIC(fit_fat, direction="both")
step$anova
summary(step)

#Conclusion: When we use StepAIC in total 63 variables were selected. These were chosen because not all

#5: Fit a Ridge regression model with same predictor and response variables. Plot model coefficient dep

#install.packages("glmnet")
library("glmnet")
covariates = scale(Dataframe[,2:(colno_fat-1)])
response = scale(Dataframe[,colno_fat])
ridge_model = glmnet(as.matrix(covariates), response, alpha=0, family="gaussian")
plot(ridge_model, xvar="lambda", label=TRUE)

#Conclusion: Coefficients goes towards 0 when lambda goes towards infinity

#6: Repeat last step but with LASSO instead of Ridge. Differences?

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

#Conclusion 5 and 6: The graphs below shows us that when we increase lambda fewer variables are selecte
#models, since the coefficients goes towards zero. In the Lasso model, the penalty is the absolute valu

#7: Choose the best model by cross validation for LASSO model. Report optimal lambda and how many varia
#chosen by the model and make conclusions. Present also a plot showing the dependence of the CV score a

lasso_model_optimal = cv.glmnet(as.matrix(covariates), response, alpha=1, family="gaussian", lambda=seq
lasso_model_optimal$lambda.min
plot(lasso_model_optimal)
coef(lasso_model_optimal, s="lambda.min")
print(lasso_model_optimal$lambda.min)

#Conclusion: When the lambda increases in value, the MSE seems to strictly increase. From this model, w
#the conclusion that the optimal lambda = 0. This means, that all variables should be included for a be
#predicted model. Compared to the results from stepAIC, all variables where chosen since lambda = 0 ins
#that were chosen.
```

#Lab2

##Assignment 1

```
#1: Read data and plot carapace length versus rear width (obs coloured by sex). Do you think that this

RNGversion('3.5.1')
```

```r
Dataframe=read.csv("australian-crabs.csv")
n = length(Dataframe[,1])
CL = Dataframe$CL
RW = Dataframe$RW
plot(CL, RW, main="Plot of carapace length versus rear width depending on sex", sub="Red = Female, Blue
     col=c("red", "blue")[Dataframe$sex], xlab="CL", ylab="RW")

#Create function for misclassification rate
missclass=function(conf_matrix, fit_matrix){
  n=length(fit_matrix[,1])
  return(1-sum(diag(conf_matrix))/n)
}

#Conclusion: Yes the classification seems to be linearly separable. However, the two clusters of data c

#2: LDA analysis with target Sex, and features CL and RW and proportional prior by using lda() function

library("MASS")
model = lda(sex ~ CL+RW, data=Dataframe)
predicted = predict(model, data=Dataframe)
confusion_matrix = table(Dataframe$sex, predicted$class)
misclass = missclass(confusion_matrix, Dataframe)
print(confusion_matrix)
print(misclass)
plot(CL, RW, main="Plot predicted values of CL and RW depending on sex", sub="Red = Female, Blue = Male
     col=c("red", "blue")[predicted$class], xlab="CL", ylab="RW")

#Conclusion: When comparing the graph from step 1 and the graph of the predicted values it is noteable
#classifications do not differ that much. With a misclassification rate of only 0.035 and 200 datapoint

#3: Repeat step 2 but use priors p(Male)=0.9 and p(Female)=0.1

model2 = lda(sex ~ CL+RW, data=Dataframe, prior=c(1,9)/10)
predicted2 = predict(model2, data=Dataframe)
confusion_matrix2 = table(Dataframe$sex, predicted2$class)
misclass2 = missclass(confusion_matrix2, Dataframe)
print(confusion_matrix2)
print(misclass2)
plot(CL, RW, main="Plot predicted values of CL and RW with priors p(Male)=0.9 and p(Female)=0.1"
     , sub="Red = Female, Blue = Male", col=c("red", "blue")[predicted2$class], xlab="CL", ylab="RW")

#Conclusion: From this graph we can see that a few more data points were classified incorrectly. This i

#4: Repeat step 2 but now with logistic regression (use function glm()). Compare with LDA results. Fina

model3 = glm(sex ~ CL+RW, data=Dataframe, family='binomial')
predicted3 = predict(model3, newdata=Dataframe, type='response')
sexvector = c()
for (i in predicted3) {
  if (i>0.9) {
    sexvector = c(sexvector, 'Male')
  } else {
    sexvector = c(sexvector, 'Female')
  }
}
print(sexvector)
sexvector_factor = as.factor(sexvector)
confusion_matrix3 = table(Dataframe$sex, sexvector_factor)
misclass3 = missclass(confusion_matrix3, Dataframe)
print(confusion_matrix3)
print(misclass3)
plot(CL, RW, main="Plot predicted values of CL and RW but with logistic regression",
     col=c("red", "blue")[sexvector_factor], xlab="CL", ylab="RW", xlim=c(0,50), ylim=c(0,20))

boundaryline = function(length, coefficientvector, prior) {
  return(-coefficientvector[1]/coefficientvector[3]-(coefficientvector[2]/coefficientvector[3])*length+
           log(prior/(1-prior))/coefficientvector[3])
}
par(new=TRUE)
curve(boundaryline(x, model3$coefficients, 0.9), xlab="CL", ylab="RW", col="green", from=0, to=50, xlim=
      ylim=c(0,20),
      sub="Red = Female, Blue = Male, Green = Boundaryline")

#Conclusion: When using logistic regression the results are similar as the first built model with LDA.
```

##Assignment 2

```
#1: Read data and divide into train, validation and test sets as 50/25/25.

library("tree")
RNGversion('3.5.1')

data=read.csv2("creditscoring.csv")
n=dim(data)[1]
set.seed(12345)
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,]

#Create function for misclassification rate
misclass=function(conf_matrix, fit_matrix){
  n=length(fit_matrix[,1])
  return(1-sum(diag(conf_matrix))/n)
}

#2: Fit a decision tree to train data using the measures of impurity gini and deviance. Report misclass

fit_deviance=tree(good_bad~., data=train, split="deviance")
predicted_deviance=predict(fit_deviance, newdata=test, type="class")
confusionmatrix_deviance=table(test$good_bad, predicted_deviance)
misclass_deviance=misclass(confusionmatrix_deviance, test)
print(confusionmatrix_deviance)
```

```r
print(misclass_deviance)
fit_gini=tree(good_bad~., data=train, split="gini")
predicted_gini=predict(fit_gini, newdata=test, type="class")
confusionmatrix_gini=table(test$good_bad, predicted_gini)
misclass_gini=misclass(confusionmatrix_gini, test)
print(confusionmatrix_gini)
print(misclass_gini)
#Deviance has best misclass score

#Conclusion: It can be concluded from the misclassification rates that the split method deviance, class

#3: Use training and valid data to choose optimal tree depth. Present graphs of the dependence of devia
#training and validation data on the number of leaves. Report optimal tree, report it's depth and varia

fit_optimaltree=tree(good_bad~., data=train, split="deviance")
summary(fit_optimaltree)
trainScore=rep(0,15)
testScore=rep(0,15)
for(i in 2:15){
  prunedTree=prune.tree(fit_optimaltree, best=i)
  pred=predict(prunedTree, newdata=valid, type="tree")
  #Divide by two since double of data points
  trainScore[i]=deviance(prunedTree)/2
  testScore[i]=deviance(pred)
}
plot(2:15, trainScore[2:15], type="b", col="red", ylim=c(200,500))
points(2:15, testScore[2:15], type="b", col="blue")
min_deviance=min(testScore[2:15])
print(min_deviance)
optimal_leaves=which(testScore[1:15] == min_deviance)
print(optimal_leaves)
#Optimal no of leaves is 4
finalTree=prune.tree(fit_optimaltree, best=4)
summary(finalTree)
plot(finalTree)
text(finalTree, pretty=0)
#Final tree contains variables savings, duration and history. Since 3 vars => Depth of tree is 3.
predicted_test=predict(finalTree, newdata=test, type="class")
confusionmatrix_test=table(test$good_bad, predicted_test)
misclass_test=misclass(confusionmatrix_test, test)
print(confusionmatrix_test)
print(misclass_test)

#Conclusion: The tree with the lowest deviance used 4 leaves which is the optimal tree. The variables u

#4: Use traning data to perform classification using Naives bayes and report the confusion matrices and
#misclassification rates for the traning and for the test data. Compare with results from previous step

#Load libraries
library(MASS)
library(e1071)
fit_naive=naiveBayes(good_bad~., data=train)
#Create function for predicting and creating confusion matrice and printing misclassification rate
```

```r
compute_naive=function(model,data){
  predictedNaive=predict(model, newdata=data, type="class")
  confusionmatrixNaive=table(data$good_bad,predictedNaive)
  misclass = misclass(confusionmatrixNaive, data)
  print(confusionmatrixNaive)
  print(misclass)
  return(predictedNaive)
}
predictedNaive_train=compute_naive(fit_naive,train)
predictedNaive_test=compute_naive(fit_naive, test)

#Conclusion: With the naive bayes method the misclassification rate is higher than what was concluded i
#The misclassification rate for test data for the naive bayes method is 0.316 and the misclassification

#5: Use optimal tree and Naives Bayes to classify the test data by using principle: classified as 1 if

#Writing function for classifying data
class=function(data, class1, class2, prior){
  vector=c()
  for(i in data) {
    if(i>prior){
      vector=c(vector,class1)
    } else {
      vector=c(vector,class2)
    }
  }
  return(vector)
}

x_vector=seq(0.05,0.95,0.05)
tpr_tree=c()
fpr_tree=c()
tpr_naive=c()
fpr_naive=c()
treeVector=c()
treeConfusion = c()
naiveConfusion = c()
treeClass = c()
naiveClass = c()
#Reusing optimal tree found in task 3 but returntype is response instead
set.seed(12345)
predictTree=data.frame(predict(finalTree, newdata=test, type="vector"))
predictNaive=data.frame(predict(fit_naive, newdata=test, type="raw"))
for(prior in x_vector){
  treeClass = class(predictTree$good, 'good', 'bad', prior)
  treeConfusion=table(test$good_bad, treeClass)
  if(ncol(treeConfusion)==1){
    if(colnames(treeConfusion)=="good"){
      treeConfusion=cbind(c(0,0), treeConfusion)
    } else {
      treeConfusion=cbind(treeConfusion,c(0,0))
    }
  }
  totGood=sum(treeConfusion[2,])
  totBad=sum(treeConfusion[1,])
  tpr_tree=c(tpr_tree, treeConfusion[2,2]/totGood)
  fpr_tree=c(fpr_tree, treeConfusion[1,2]/totBad)
  print(fpr_tree)
  naiveClass=class(predictNaive$good, 'good', 'bad', prior)
  naiveConfusion=table(test$good_bad, naiveClass)
  if(ncol(naiveConfusion)==1){
    if(colnames(naiveConfusion)=="good"){
      naiveConfusion=cbind(c(0,0), naiveConfusion)
    } else {
      naiveConfusion=cbind(naiveConfusion,c(0,0))
    }
  }
  totGood=sum(naiveConfusion[2,])
  totBad=sum(naiveConfusion[1,])
  tpr_naive=c(tpr_naive, naiveConfusion[2,2]/totGood)
  fpr_naive=c(fpr_naive, naiveConfusion[1,2]/totBad)
}
#Plot the ROC curves
plot(fpr_naive, tpr_naive, main="ROC curve", sub="Red = Naive Bayes, Blue = Tree", type="l", col="red",
     ylim=c(0,1), xlab="FPR", ylab="TPR")
points(fpr_tree, tpr_tree, type="l", col="blue")
#Naive has greatest AOC => should choose Naive

#Conclusion: From the ROC-curve we cam see that the total area under the curve (AOC) is the biggest for
#bayes method. Therefore this method should be the one to use instead of the decision tree model.

#6: Repeat Naive Bayes with loss matrix punishing with factor 10 if predicting good when bad and 1 if p
#bad when good.

naiveModel=naiveBayes(good_bad~., data=train)
train_loss=predict(naiveModel, newdata=train, type="raw")
test_loss=predict(naiveModel, newdata=test, type="raw")
confusion_trainLoss=table(train$good_bad, ifelse(train_loss[,2]/train_loss[,1]>10, "good", "bad"))
misclass_trainLoss=misclass(confusion_trainLoss, train)
print(confusion_trainLoss)
print(misclass_trainLoss)
confusion_testLoss=table(test$good_bad, ifelse(test_loss[,2]/test_loss[,1]>10, "good", "bad"))
misclass_testLoss=misclass(confusion_testLoss, test)
print(confusion_testLoss)
print(misclass_testLoss)

#Conclusion: The misclassification rates have changed since a higher punishment is given when predictin
#creditscore when in fact it was bad (reasonable since bank loses money then). It is less worse to pred
#creditscore but turns out to be good (just a loss of customer). Due to this more errors occur mainly b
```

##Assignment 3

```
#1: Reorder your data with respect to the increase of MET and plot EX versus MET. Discuss what kind of

RNGversion('3.5.1')
#Read data
set.seed(12345)
```

```
Dataframe=read.csv2("State.csv")
Dataframe=Dataframe[order(Dataframe$MET),]
MET=Dataframe$MET
EX=Dataframe$EX

plot(MET, EX, xlab="EX", ylab="MET", type="p", main="Plot of EX vs MET")

#Conclusion: Some kind of squared model might be useful here.

#2: Use package tree  and fit a regression tree model with target EX and feature MET in which the numbe

library(tree)
treemodel=tree(EX~MET, data=Dataframe, control=tree.control(48, mincut=8))
summary(treemodel)
plot(treemodel)
text(treemodel, pretty=0)
set.seed(12345)
cvTreeModel = cv.tree(treemodel)
plot(cvTreeModel$size, cvTreeModel$dev, type="b", col="red", xlab="Size", ylab="Dev")
bestSize = cvTreeModel$size[which.min(cvTreeModel$dev)]
bestTree=prune.tree(treemodel, best=bestSize)
plot(bestTree)
text(bestTree, pretty=0)
title("Optimal tree")
predData=predict(bestTree, newdata=Dataframe)
plot(MET, EX, xlab="EX", ylab="MET", type="p", col="red", main="Plot original vs predicted data")
points(MET, predData, col="blue")
summaryfit=summary(bestTree)
hist(summaryfit$residuals, breaks=10)

#Conclusion: The distribution of the residuals seems to be fairly normally distributed with no bias. Th

library(boot)
# computingbootstrapsamples
f=function(data, ind){
  data1=data[ind,]# extractbootstrapsample
  treeModel=tree(EX~MET, data=data1, control=tree.control(48, mincut=8))
  prunedtree=prune.tree(treeModel, best=3)
  predData=predict(prunedtree,newdata=Dataframe)
  return(predData)
}
res=boot(Dataframe, f, R=1000) #make bootstrap
confIntNPBoot=envelope(res)
plot(MET, EX, xlab="EX", ylab="MET", pch=21, bg="orange", main="Plot original vs predicted data", ylim=
points(MET, predData, type="l", col="blue")
points(MET, confIntNPBoot$point[2,], type="l")
points(MET, confIntNPBoot$point[1,], type="l")

#Conclusion: The confidence bands are bumpy. This is due to the fact that no distribution is assumed fo

mle=prune.tree(treemodel, best=3)
summaryMLE = summary(mle)
rng=function(data, mle) {
  data1=data.frame(EX=data$EX, MET=data$MET)
  n=length(data$EX)
  #generatenew EX
  data1$EX=rnorm(n,predict(mle, newdata=data1), sd(summaryMLE$residuals))
  return(data1)
}

f1=function(data1){
  treemodel=tree(EX~MET, data=data1, control=tree.control(48,mincut=8)) #fit linearmodel
  prunedtree=prune.tree(treemodel, best=3)
  n=length(Dataframe$EX)
  #predictvaluesfor all EX values from the original data
  predData=predict(prunedtree,newdata=Dataframe)
  predictedEX=rnorm(n, predData, sd(summaryMLE$residuals))
  return(predictedEX)
}
res=boot(Dataframe, statistic=f1, R=1000, mle=mle, ran.gen=rng, sim="parametric")
predIntPBoot=envelope(res)
points(MET, predIntPBoot$point[2,], type="l", col="green")
points(MET, predIntPBoot$point[1,], type="l", col="green")

#Conclusion: NOTE: This code above is wrong. The confidence bands for parametric bootstrap shold be com
```

##Assignment 4

```
#1: Read data
RNGversion('3.5.1')

data=read.csv2("NIRspectra.csv")
data$Viscosity=c()
n=dim(data)[1]

#1: Conduct standard PCA using the feature space and provide a plot explaining how much variation is ex

pcaAnalysis=prcomp(data)
lambda=pcaAnalysis$sdev^2
#Eigenvalues
print(lambda)
#Proportion of variation
propVar= lambda/sum(lambda)*100
screeplot(pcaAnalysis)
print(propVar)
noOfVars=1
sumOfVariation=propVar[noOfVars]
while(sumOfVariation<99){
  noOfVars=noOfVars+1
  sumOfVariation=sumOfVariation+propVar[noOfVars]
}
#Print number of variables used
print(noOfVars)
#Print PC1 and PC2 in plot
plot(pcaAnalysis$x[,1],pcaAnalysis$x[,2], ylim=c(-10,10), type="p", col="blue", main="PC1 vs PC2", xlab
     ylab="PC2")
#We can see from the graph that the data is very accurately described by PC1.
```

```
#Conclusion: From the screeplot it can be conlcuded that the two components captures almost all of the

#2: Make trace plots of the loadings of the components selected in step 1. Is there any principle compo

U=pcaAnalysis$rotation
plot(U[,1], main="Traceplot, PC1", xlab="index", ylab="PC1", type="b")
plot(U[,2], main="Traceplot, PC2", xlab="index", ylab="PC2", type="b")

#Conclusion: We can see from graph that PC2 is not described by so many original features since it is c

#3: Perform independent Component Analysis (ICA) with no of components selected in step1 (set seed 1234
# Compute W'=K*W and present columns of W' in form of the trace plots. Compare with trace plots in step
# Make a plot of the scores of the first two latent features and compare it with the score plot from st

#Install package fastICa
#install.packages("fastICA")
library("fastICA")

set.seed(12345)
icaModel = fastICA(data, n.comp=2, verbose=TRUE)
W=icaModel$W
K=icaModel$K
W_est=K%*%W
plot(W_est[,1], main="Traceplot, ICA1", xlab="index", ylab="ICA1", type="b", col="red")
plot(W_est[,2], main="Traceplot, ICA2", xlab="index", ylab="ICA2", type="b", col="red")
plot(icaModel$S[,1], icaModel$S[,2], main="ICA1 vs ICA2", xlab="ICA1", ylab="ICA2", type="p", col="blue

#Conclusion: When comparing the trace plots of ICA1 and ICA2 with PC1 and PC2 from step 2, it is noteab
```

#Lab3

##Assignment 1

```
RNGversion('3.5.1')
## Assignment 1:
## Implement a kernel method to predict the hourly temperatures for a date and place in Sweden.
## To do so, you are provided with the files stations.csv and temps50k.csv. These
## files contain information about weather stations and temperature measurements in the stations
## at different days and times. The data have been kindly provided by the Swedish Meteorological
## and Hydrological Institute (SMHI).
## You are asked to provide a temperature forecast for a date and place in Sweden. The
## forecast should consist of the predicted temperatures from 4 am to 24 pm in an interval of 2
## hours. Use a kernel that is the sum of three Gaussian kernels:
##  The first to account for the distance from a station to the point of interest.
##  The second to account for the distance between the day a temperature measurement
##    was made and the day of interest.
##  The third to account for the distance between the hour of the day a temperature measurement
##    was made and the hour of interest.
## Choose an appropriate smoothing coefficient or width for each of the three kernels above.
## Answer to the following questions:
##  Show that your choice for the kernels' width is sensible, i.e. that it gives more weight
##    to closer points. Discuss why your of definition of closeness is reasonable.
##  Instead of combining the three kernels into one by summing them up, multiply them.
##    Compare the results obtained in both cases and elaborate on why they may differ.
```

```r
## Note that the file temps50k.csv may contain temperature measurements that are posterior
## to the day and hour of your forecast. You must filter such measurements out, i.e. they cannot
## be used to compute the forecast. Feel free to use the template below to solve the assignment.

set.seed(1234567890)
#install.packages("geosphere")
library(geosphere)
stations <- read.csv("stations.csv")
temps <- read.csv("temps50k.csv")
#A join operation on "station_number"
st <- merge(stations,temps,by="station_number")
n = dim(st)[1]
#Kernel weighting factors
h_distance <- 100000
h_date <- 20
h_time <- 2
#Latitude of interest
a <- 59.4059
#Longitude of interest
b <- 18.0256
#Coordinates for Danderyd
#Create a vector of the point of interest
placeOI = c(a, b)
dateOI <- as.Date("1995-07-29") # The date to predict (up to the students), my birth date
timesOI = c("04:00:00", "06:00:00", "08:00:00", "10:00:00", "12:00:00", "14:00:00", "16:00:00", "18:00:0
            "20:00:00",
          "22:00:00", "24:00:00")

plotDist = function(dist, h){
  u = dist/h
  plot(dist, exp(-u^2), type="l", main="Plot of kernel wights for distances", xlab="Distance")
}

dist = seq(0, 100000, 1)
plotDist(dist, h_distance)

plotDate = function(date, h){
  u = date/h
  plot(date, exp(-u^2), type="l", main="Plot of kernel wights for dates", xlab="Days")
}

date = seq(-182,182,1)
plotDate(date, h_date)

plotTime = function(time, h){
  u = time/h
  plot(time, exp(-u^2), type="l", main="Plot of kernel wights for time", xlab="Hours")
}

time = seq(-12,12,1)
plotTime(time, h_time)

#Remove posterior data
```

```
filter_posterior = function(date, time, data){
  return(data[which(as.numeric(difftime(strptime(paste(date, time, sep=" "), format="%Y-%m-%d %H:%M:%S")
                     strptime(paste(data$date, data$time, sep=" "),format="%Y-%m-%d %H:%M:%S")))>0),
}

#A gaussian function for the difference in distance
gaussian_dist = function(place, data, h) {
  lat = data$latitude
  long = data$longitude
  points = data.frame(lat,long)
  u = distHaversine(points, place)/h
  return (exp(-u^2))
}

xy = gaussian_dist(placeOI, st, h_distance)

#A gaussian function for difference in days
gaussian_day = function(date, data, h){
  compare_date = as.Date(data$date)
  diff = as.numeric(date-compare_date)
  for (i in 1:length(diff)) {
    if (diff[i] > 365) {
      diff[i] = diff[i] %% 365
      if(diff[i]>182){
        diff[i]=365-diff[i]
      }
    }
  }
  u = diff/h
  return (exp(-u^2))
}

#A gaussian function for difference in hours
gaussian_hour = function(hour, data, h){
  compare_hour = strptime(data$time, format="%H:%M:%S")
  compare_hour = as.numeric(format(compare_hour, format="%H"))
  hour = strptime(hour, format="%H:%M:%S")
  hour = as.numeric(format(hour, format="%H"))
  diff = abs(hour-compare_hour)
  for (i in 1:length(diff)){
    if(diff[i]>12){
      diff[i] = 24-diff[i]
    }
  }
  u=diff/h
  return(exp(-u^2))
}

#Defining values that will be used in loop below
kernel_sum = c()
kernel_mult = c()

#Looping through time array and data points in nested loop to calculate the 11 kernel values
```

```r
for (time in timesOI) {
  filtered_data = filter_posterior(dateOI, time, st)
  kernel_dist = gaussian_dist(placeOI, filtered_data, h_distance)
  kernel_day = gaussian_day(dateOI, filtered_data, h_date)
  kernel_time = gaussian_hour(time, filtered_data, h_time)
  sum_kernel = kernel_dist+kernel_day+kernel_time
  temp_sum = sum(sum_kernel * filtered_data$air_temperature)/sum(sum_kernel)
  mult_kernel = kernel_dist*kernel_day*kernel_time
  temp_mult = sum(mult_kernel * filtered_data$air_temperature)/sum(mult_kernel)
  kernel_sum = c(kernel_sum, temp_sum)
  kernel_mult = c(kernel_mult, temp_mult)
}


plot(kernel_sum, type="o", main ="Temperature estimate through sum of factors", xlab="Time",
     ylab="Est. temperature")
axis(1, at=1:length(timesOI), labels=timesOI)
plot(kernel_mult, type="o", main="Temperature estimate through product of factors", xlab="Time",
     ylab="Est. temperature")
axis(1, at=1:length(timesOI), labels=(timesOI))

#Conclusion: When studying the graphs above further, the h values can be motivated. Finally, the estima
```

##Assignment 2

```r
##Use the function ksvm from the R package kernlab to learn a SVM for classifying the spam dataset that
# Perform model selection, i.e. select the most promising of the three models (use any method of your c
# Estimate the generalization error of the SVM selected above (use any method of your choice except cro
# Produce the SVM that will be returned to the user, i.e. show the code
# What is the purpose of the parameter C?

library(kernlab)
set.seed(1234567890)
data(spam)

#Create function for misclassification rate
missclass=function(conf_matrix, fit_matrix){
  n=length(fit_matrix[,1])
  return(1-sum(diag(conf_matrix))/n)
}


index=sample(1:4601)
train=spam[index[1:2500],]
valid=spam[index[2501:3501],]
test=spam[index[3502:4601],]

svmmodel1=ksvm(type~., data=train, kernel="rbfdot", kpar=list(sigma=0.05), C=0.5)
pred1=predict(svmmodel1, newdata=valid)
confusion1=table(valid$type, pred1)
misclass1=missclass(confusion1, valid)
print(confusion1)
print(misclass1)
```

```r
svmmodel2=ksvm(type~., data=train, kernel="rbfdot", kpar=list(sigma=0.05), C=1)
pred2=predict(svmmodel2, newdata=valid)
confusion2=table(valid$type, pred2)
misclass2=missclass(confusion2, valid)
print(confusion2)
print(misclass2)

svmmodel3=ksvm(type~., data=train, kernel="rbfdot", kpar=list(sigma=0.05), C=5)
pred2=predict(svmmodel3, newdata=valid)
confusion3=table(valid$type, pred2)
misclass3=missclass(confusion3, valid)
print(confusion3)
print(misclass3)

##Conclusion: The model with the C value of 1 is the best since it has the lowest misclassification rat

finalmodel=ksvm(type~., data=spam[index[1:3501],], kernel="rbfdot", kpar=list(sigma=0.05), C=1)
finalpred=predict(finalmodel, newdata=test)
finalconfusion=table(test$type, finalpred)
finalmisclass=missclass(finalconfusion, test)
print(finalconfusion)
print(finalmisclass)

##Answer: The purpose of the parameter C is to put a weight to the cost function. The higher C the more

#Final model

finalmodel=ksvm(type~., data=spam, kernel="rbfdot", kpar=list(sigma=0.05), C=1)
```

##Assignment 3

```r
## Assignment3:
## Train a neural network to learn the trigonometric sine function. To do so, sample 50 points
## uniformly at random in the interval [0,10]. Apply the sine function to each point. The resulting
## pairs are the data available to you. Use 25 of the 50 points for training and the rest for validatio
## The validation set is used for early stop of the gradient descent. That is, you should
## use the validation set to detect when to stop the gradient descent and so avoid overfitting.
## Stop the gradient descent when the partial derivatives of the error function are below a given
## threshold value. Check the argument threshold in the documentation.Consider threshold
## values i/1000 with i = 1,...,10. Initialize the weights of the neural network to random values in
## the interval [-1, 1].  Use a neural network with a single hidden layer of 10 units. Use the default
## for the arguments not mentioned here. Choose the most appropriate value for
## the threshold. Motivate your choice. Provide the final neural network learned with the chosen
## threshold. Feel free to use the following template.

RNGversion('3.5.1')
#install.packages("neuralnet")
library(neuralnet)
set.seed(1234567890)
Var <- runif(50, 0, 10)
trva <- data.frame(Var, Sin=sin(Var))
train <- trva[1:25,] # Training
valid <- trva[26:50,] # Validation
n = dim(valid)[1]
```

```
# Random initialization of the weights in the interval [-1, 1]
winit <- runif(31, -1, 1)
trainScore = rep(0,10)
validScore = rep(0,10)
for(i in 1:10) {
  nn_temp <- neuralnet(Sin~Var, data=train, hidden=10, threshold=i/1000, startweights=winit)
  nn = as.data.frame(nn_temp$net.result)
  pred=predict(nn_temp, newdata=valid)
  trainScore[i] = 1/n*sum((nn[,1]-train$Sin)^2)
  validScore[i] = 1/n*sum((pred-valid$Sin)^2)
}
plot(1:10, trainScore[1:10], type="b", col="red", xlab="Threshold index", ylab="MSE")
points(1:10, validScore[1:10], type="b", col="blue")
min_error=min(validScore[1:10])
print(min_error)
optimal_i=which(validScore[1:10] == min_error)
print(optimal_i)

##Conclusion: As seen in the graph, naturally the train data performs the best when the threshold value

optimal_nn = neuralnet(Sin~Var, data=train, hidden=10, threshold=optimal_i/1000, startweights=winit)
plot(optimal_nn)
# Plot of the predictions (black dots) and the data (red dots)
plot(prediction(optimal_nn)$rep1)
points(trva, col = "red")

##Conclusion: The optimal neural network with threshold 4/1000 is chosen which results in the neural ne
```