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 classification Y(hat) = 1 if p(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.  
  
#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 data. However, the fact that the miscalculations are similar indicates that the model performs similarly on two different data sets which is generally how you want your model to behave. By analyzing the confusion matrices, it is notable that the model is wrong more times proportionally when trying to classify an email that is spam than an email that is not spam.  
  
#3: Use logistic regression to classify the test data by the classification principle: Same as above but with   
#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, since the model acts similar between trained and tested data. Although this classificiation principle gives us a higher misclassification rate, it lowered the risk of a non-spam being classified as spam substantially. Therefore we prefer this principle over the previous.  
  
#4: Use standard classifier kknn() with K=30 from package kknn, report the misclassification rates for the training and test data and compare the results with step 2.  
  
#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 shows us that our model is not well fitted. The misclassification is lower for the trained data, since the model is fitted after these values. Compared to the results from using logistic regression to classify the data, the results from the KKNN model were significally worse on the test data. This implies that KKNN classification with K=30 is worse than logistic regression in this case.  
  
#5: Repeat step 4 for K=1. Classify according to kknn with k=1 for test and train data sets. What does the decrease of K lead to and why?  
  
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 each data point to itself, predicting all correct. This explains the high misclassification rate for the confusion matrix made on the test data. Using K=1 is a very unreliable method because it does not imply a large statistical advantage.

##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 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 given theta and a given data vector x. Plot the curve showing the dependence of log-likelihood on theta where the entire data is used for fitting. What is the maximum likelihood value of theta according to plot?  
  
#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 maximum  
##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 solution in  
#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 compared with to the graph when we use all data. The model with more data is more reliable since there is a more certain min-value from the graph whereas the one with only six values, the theta value could be anything from the minimum value and forward.  
  
#4: Assume now a Bayesian model with p(x|theta)=theta\*e^(-theta\*x) and a prior p(theta)=lambda\*e^(-lambda\*x), 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.   
  
#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=FALSE, col="red",  
 ylim=c(20,300))  
curve(-loglikelihood(x, Dataframe), ylab="-Loglikelihood", xlab="Theta", from=0, to=10, add=TRUE, col="blue",  
 ylim=c(20,300))  
  
#Conclusion: When using an bayesian model we have a prior that gives the model information beforehand which helps  
#fitting the model. The optimal theta is now 0.91 which is close to the datasets meanvalue, which makes sense that this model gives a better predicted value.  
  
#5: Use theta value found in step 2 and generate 50 new observations from p(x|theta)=theta\*e^(-theta\*x) (use standard number generators). Create the histograms of the original and the new data and make conclusions.   
  
#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 predicted data. This concludes model was accurately fitted to the correct distribution model.

##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 polynomial function  
#of Protein including the polynomial terms up to power of 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?  
  
#Conclusion: A probabilistic model describing M(i) is: M(i) = w0 + w1 \* X + w2 \* X2 + ... + wi \* Xi (3) The MSE  
#criterion is a suitable method since it punishes outliers to a larger extent. This creates a better fitted model  
#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 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.  
  
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))  
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 predicted data  
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 therefore being the model with the smallest error. In terms of bias, what we can see is that in the training model the predicted error, and MSE, is descending for a more complex model, because bias is defined by the ability for a model to fit data. This gives us an overfitted model, where we can see that the variance increases the more complex models because the MSE increases for the tested data and the MSE decreases for the training data. Variance is defined by the difference in predictions on different data sets.  
  
#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 variables are needed to predict a dependent variable.  
  
#5: Fit a Ridge regression model with same predictor and response variables. Plot model coefficient depend on the log of the penalty factor lambda and report how the coefficients change with lambda.   
  
#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 selected to the  
#models, since the coefficients goes towards zero. In the Lasso model, the penalty is the absolute value, and in the Ridge model the penalty is squared (penalizes large values more).   
  
#7: Choose the best model by cross validation for LASSO model. Report optimal lambda and how many variables that  
#chosen by the model and make conclusions. Present also a plot showing the dependence of the CV score and comment how the CV score changes with lambda.   
  
lasso\_model\_optimal = cv.glmnet(as.matrix(covariates), response, alpha=1, family="gaussian", lambda=seq(0,1,0.001))  
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, we can make  
#the conclusion that the optimal lambda = 0. This means, that all variables should be included for a better  
#predicted model. Compared to the results from stepAIC, all variables where chosen since lambda = 0 instead of 63  
#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 data is easy to classify by LDA? Motivate answer.   
  
RNGversion('3.5.1')  
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 = Male",   
 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 clearly have different covariance matrices (since angle of trend is different) which is not optimal for the LDA method.  
  
#2: LDA analysis with target Sex, and features CL and RW and proportional prior by using lda() function in package MASS Make a Scatter plot of CL versus RW colored by the predicted Sex and compare it with the plot in step 1. Compute the misclassification error and comment on the quality of fit.  
  
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 that the  
#classifications do not differ that much. With a misclassification rate of only 0.035 and 200 datapoints it can be concluded that 7 observations were classified incorrectly. When comparing the graphs it is difficult to find the points which have changed color (since they have been classified incorrectly) but one example is the point farthest to the let which was classified as male but should have been classified as female. The model classifies the data accurately.   
  
#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 is due to the higher prior set on classifying a data point as male, i.e. 0.9. It is noteable in the confusion matrix that no males classified incorrectly. This is also due to the high prior which basically says that is is not that likely that a datapoint will be classified as a female. When the model in fact classify a data point as female it has to be sure of it, and this can be seen as stated above in the confusion matrix. On the other hand, more females are classified as males inaccurately since the higher prior. This also results in a higher misclassification rate of 0.08.   
  
#4: Repeat step 2 but now with logistic regression (use function glm()). Compare with LDA results. Finally, report the equation of the decision boundary and draw the decision boundary in the plot of the classified data.  
  
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=c(0,50),   
 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. This is simply a coincident and no real conclusion can be drawn regarding the exact same misclassification rate except from that the models seem to classify the data in a similar way. When comparing which data points that are classified as females and males in the two models it can be concluded that the model using logistic regression classifies the data in a way which enables a boundary line more distinctly. This is due to the characteristics of the logistic regression model. The equation for the decision boundary line is as follows: RW(hat)=-(beta0+beta1\*CL)beta2

##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 rates and choose optimal measure moving forward.   
  
fit\_deviance=tree(good\_bad~., data=train, split="deviance")  
predicted\_deviance=predict(fit\_deviance, newdata=test, type="class")  
confusionmatrix\_deviance=table(test$good\_bad, predicted\_deviance)  
misclass\_deviance=misclass(confusionmatrix\_deviance, test)  
print(confusionmatrix\_deviance)  
print(misclass\_deviance)  
fit\_gini=tree(good\_bad~., data=train, split="gini")  
predicted\_gini=predict(fit\_gini, newdata=test, type="class")  
confusionmatrix\_gini=table(test$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, classifies the data in a better way than the split method gini. Since the method deviance performed better it will be the chosen splitting method in the following steps.   
  
#3: Use training and valid data to choose optimal tree depth. Present graphs of the dependence of deviances for   
#training and validation data on the number of leaves. Report optimal tree, report it's depth and variables used by tree. Estimate misclassification rate for the test data.   
  
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 used by the tree is savings, duration and history, and the depth of the tree is 3. The misclassification rate for the test data is 0.256.   
  
#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 steps.  
  
#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  
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 in step 3.  
#The misclassification rate for test data for the naive bayes method is 0.316 and the misclassification rate for the decision tree from step 3 is 0.256. This indicates that the decision tree method classifies the data more accurately than what the model which uses the naive bayes method does.   
  
#5: Use optimal tree and Naives Bayes to classify the test data by using principle: classified as 1 if 'good' bigger than 0.05, 0.1, 0.15, ..., 0.9, 0.95. Compute the TPR and FPR for two models and plot corresponsing ROC curves.  
  
#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", xlim=c(0,1),   
 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 the naive  
#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 predicting  
#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 predicting good   
#creditscore when in fact it was bad (reasonable since bank loses money then). It is less worse to predict bad  
#creditscore but turns out to be good (just a loss of customer). Due to this more errors occur mainly because fewer people are classified to have good creditscores.

##Assignment 3

#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.  
  
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 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 mincut 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.  
  
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. The fit is quite good considering the simple model that it is.   
  
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=c(100,500))  
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 for the data. The model will try to accostom as best it can from the data given. The width of the confidence band is rather high which indicates that the model used is not that reliable. Furthermore we can almost draw a straight line between the whole band which would mean that each EX value would yield the same MET value which again implies that the model is not that good.   
  
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 computed separately The confidence bands retrieved are more smooth. However when looking at the residuals they do not seem to be normally distributed as assumed. Therefore a parametric bootstrap model is not preffered => choose non-parametric.

##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 explained by each feature. Provide plot that show the scores of PC1 vs PC2. Are there unusual diesel fuels according to this plot.   
  
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="PC1",   
 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 variation in the data. Therefore PCA analysis is suitable for the data. Two components capture 99.5957 % of the variation and therefore these two components will be used in the following steps. Most of the data points are around 0 for PC1 but there are some data points which can be described as outliers located to the farthest right in the score plot.   
  
#2: Make trace plots of the loadings of the components selected in step 1. Is there any principle component that is explaines by mainly a few original features?  
  
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 close to zero for many of the features. The last 30 or so variables have an effect on PC2.   
  
#3: Perform independent Component Analysis (ICA) with no of components selected in step1 (set seed 12345). Check the documentation of R for fastICA method and do following:  
# Compute W'=K\*W and present columns of W' in form of the trace plots. Compare with trace plots in step 2 and make conclusions. What kind of measure is represented by the matrix W'.  
# Make a plot of the scores of the first two latent features and compare it with the score plot from step 1.   
  
#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 noteable that for the first component the dependency on the features increases as the index increases. It is also noteable that the plots for the different components appear to be each others mirrors. This is reasonable because PCA tries to maximize the variance, i.e. look for correlation between the different features, whereas ICA tries to do the exact opposite, i.e. maximizing the independence between the different features by creating an orthogonal coordinate system. The parameter W' which is computed by W(hat)=K\*W describes how the features explain the principal components ICA1 and ICA2. When comparing the last score plot with the score plot from step 1 it can also be concluded that the score plot of ICA is mirroring the score plot of PCA. However, the axis of the coordinate system of ICA have been standardized wh ich is the difference between the plots.

#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.  
## 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", "18:00:00",   
 "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  
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 estimations for the temperatures are made through the summation of different kernel functions as well as multiplication of the different Kernel functions. The summation of kernel functions provides estimates closer to the mean of all temperatures (4.62) than what the multiplication of kernel functions has provided. This can be due to that data points which have received a high weight through the kernel functions will have more impact in the multiplication of kernel functions than with the summation of kernel functions, and similarily data points which have received a low weight through the kernel functions will have more impact in the multiplication of kernel functions than with the summation of kernel functions. To conclude, the three different weights in the multiplication of kernel functions all has to be quite high in order for the total weight to be high. On the other hand if one weight is low the whole weight is going to be low even though the other two are high. The result of this is that the data points with high weight is more significant and perhaps more similar to the point of interest and time of interest for the multiplication of kernels than for the summation of kernels. This can also be seen in the graphs where the temperatures for the multiplication of kernels seem more reasonable intuitively than for the summation of kernels and seem like a more accurate model.

##Assignment 2

##Use the function ksvm from the R package kernlab to learn a SVM 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 parameter C, consider values 0.5, 1 and 5. This implies that you have to consider three models.   
# Perform model selection, i.e. select the most promising of the three models (use any method of your choice except cross-validation or nested-cross-validation)  
# Estimate the generalization error of the SVM selected above (use any method of your choice except cross-validation or nested cross validation)  
# 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)  
  
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 rate. However, since the application is classification of spam emails, the value of C=0.5 is the best since it classified the least nonspam emails as spam.   
  
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 cost will a constraint violation yield.   
  
#Final model  
  
finalmodel=ksvm(type~., data=spam, kernel="rbfdot", kpar=list(sigma=0.05), C=1)

##Assignment 3

## 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 validation.   
## 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 values  
## 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 is as small as possible, i.e. 1/1000, and the performance decreases as this threshold increases for the train data. From the graph we can see that the threshold value 4/1000 performs the best on the validation data (since it results in the lowest MSE) and therefore this threshold will be used moving forward in the assignment.   
  
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 network shown above. The last two graphs briefly show how similar the predicted values from the model are in comparison to the real sinus curve. One can conclude that the neural network created resembles the shape of the sinus curve with quite a precision.