Lab3

Christian von Koch

2019-12-17

#2016-01-09

##Assignment 1

##Dataset crx.csv contains encrypted information about the customers of a bank and whether each individual has paid back the loan or not: Class 1=paid back, 0=not paid back  
  
##Divide the dataset into training and test sets (80/20), use seed 12345. Fit a decision tree with default settings to the training data and plot the resulting tree. Finally, remove the second observation from the training data, fit the tree model again and plot the tree. Compare the trees and comment why the tree structure changed so much although only one observation is deleted.  
  
#Read data  
RNGversion('3.5.1')  
library(tree)  
Dataframe=read.csv("crx.csv")  
n=dim(Dataframe)[1]  
set.seed(12345)  
id=sample(1:n, floor(n\*0.8))  
train=Dataframe[id,]  
test=Dataframe[-id,]  
  
treemodel=tree(Class~., data=train)  
summary(treemodel)  
plot(treemodel)  
text(treemodel, pretty=0)  
train\_new=train[-2,]  
treemodel\_new=tree(Class~., data=train\_new)  
plot(treemodel\_new)  
text(treemodel\_new, pretty=0)  
  
##Answer: Tree structure does not change at all.   
  
##Prune the tree fitted to the training data by using the cross-validation. Provide a cross-validation plot and comment how many leaves the optimal tree should have. Which variables were selected by the optimal tree?  
  
cv.res=cv.tree(treemodel)  
plot(cv.res$size, cv.res$dev, type="b", col="red")  
plot(log(cv.res$k), cv.res$dev, type="b", col="red")  
optimalTree=prune.tree(treemodel, best=3)  
summary(optimalTree)  
plot(optimalTree)  
text(optimalTree, pretty=0)  
  
##Answer: Two variables were selected for the optimal tree; A9 and A10. The best no of leaves is 3.   
  
##Use this kind of code to prepare the feature set to be used with a LASSO model (here 'train' is the training data):  
## x\_train = model.matrix(~ .-1, train[,-16])  
##Fit a LASSO model to the training data, carefully consider the choice of family parameter in the glmnet function. Report the cross-validation plot, find the optimal penalty parameter value and report the number of components selected by LASSO. By looking at the plot, comment whether the optimal model looks statistically significantly better than the model with the smallest value of the penalty parameter.  
  
x\_train = model.matrix( ~ .-1, train[,-16])  
library(glmnet)  
class=as.factor(train$Class)  
lassomodel=cv.glmnet(x\_train, class, alpha=1, family="binomial")  
lassomodel$lambda.min  
plot(lassomodel)  
coef(lassomodel, s="lambda.min")  
  
##Answer: Optimal penalty parameter is 0.01036912 and the number of components used are 23. The optimal model does not look significantly better than when the smallest value is used.   
  
##Use the following error function to compute the test error for the LASSO and tree models:   
##E=sum(Yi\*log(pi)+(1-Yi)\*log(1-pi)) where Yi is the target value and pi are predicted probabilities pf Yi=1. Which model is the best according to this criterion? Why is this criterion sometimes more reasonable to use than the misclassification rate?  
  
x\_test = model.matrix( ~ .-1, test[,-16])  
pred\_lasso=predict(lassomodel, s=lassomodel$lambda.min, newx=x\_test, type="response")  
  
errorfunction=function(classvector, predvector) {  
 return(sum(classvector\*log(predvector)+(1-classvector)\*log(1-predvector)))  
}  
  
pred\_tree=predict(optimalTree, newdata=test, type="vector")  
error\_tree=errorfunction(test$Class, pred\_tree)  
error\_lasso=errorfunction(test$Class, pred\_lasso)  
  
##The tree model is better according to this criterion. This criterion might be more suitable since it takes into account the probability of a class being classified and not just if it gets it right.

##Assignment 2

##In the following steps, you are asked to use the R package kernlab to learn a SVM for classifying the spam dataset that is included with the package. For the C parameter consider values 1 and 5. Consider the radial basis function kernel (also known as Gaussian) and the linear kernel. For the former, consider a width of 0.01 and 0.05. This implies that you have to select among six models.  
  
##Use nested cross-validation to estimate the error of the model selection task described above. Use two folds for inner and outer cross-validation. Note that you only have to implement the outer cross-validation: The inner cross-validation can be performed by using the argument cross=2 when calling the function ksvm. Hint: Recall that inner cross-validation estimates the error of the different models and selects the best, which is then evaluated by the outer cross-validation. So, the outer cross-validation evaluates the model selection performed by the inner cross-validation  
  
RNGversion('3.5.1')  
library(kernlab)  
set.seed(12345)  
data(spam)  
n=dim(spam)[1]  
id=sample(1:n, floor(n\*0.5))  
fold1=spam[id,]  
fold2=spam[-id,]  
C=c(5,1)  
width=c(0.01,0.01,0.05,0.05,0,0)  
kernel=c("rbfdot", "rbfdot", "rbfdot", "rbfdot", "vanilladot", "vanilladot")  
  
missclass=function(conf\_matrix, fit\_matrix){  
 n=length(fit\_matrix[,1])  
 return(1-sum(diag(conf\_matrix))/n)  
}  
  
prediction=function(train, test, C, width, kernel) {  
 if (width == 0) {  
 svmmodel=ksvm(type~., data=train, kernel=kernel, C=C, cross=2)  
 } else {  
 svmmodel=ksvm(type~., data=train, kernel=kernel, C=C, cross=2, kpar=list(sigma=width))  
 }  
 predicted=predict(svmmodel, newdata=test)  
 confusion=table(test$type, predicted)  
 return(missclass(confusion, test))  
}  
  
scores=numeric(6)  
scores2=numeric(6)  
for (i in 1:6) {  
 scores[i]=prediction(fold1, fold2, C[(i %% 2)+1], width[i], kernel[i])  
 scores2[i]=prediction(fold2, fold1, C[(i %% 2)+1], width[i], kernel[i])  
}  
  
avgScore=(scores+scores2)/2  
bestModel=which(avgScore == min(avgScore))  
print(bestModel)  
  
##Answer: Optimal model is using a C-value of 5, gaussian kernel with width 0.01.  
  
##Produce the code to select the model that will be returned to the user.  
  
#Final model  
  
finalModel = ksvm(type~., data=spam, kernel="rbfdot", C=5, kpar=list(sigma=0.01), cross=2)

#2017-04-18

##Assignment 1

##Plot the dependence of CW versus BD where the points are colored by Species. Are CW and BD good predictors of the Species?  
  
#Read data  
RNGversion('3.5.1')  
Dataframe=read.csv("australian-crabs.csv")  
CW=Dataframe$CW  
BD=Dataframe$BD  
plot(CW, BD, col=c("blue", "orange")[Dataframe$species], main="Plot of CW vs BD", sub="Blue = blue, Orange=orange")  
  
##Answer: It seems like they are since it is a clear distinction between the two clusters of data. A linear predictor seems like a good way of classifying the data.  
  
##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  
  
#Create function for misclassification rate  
misclass=function(conf\_matrix, fit\_matrix){  
 n=length(fit\_matrix[,1])  
 return(1-sum(diag(conf\_matrix))/n)  
}  
  
  
library(MASS)  
library(e1071)  
fit\_naive=naiveBayes(species~CW+BD, data=Dataframe)  
pred=predict(fit\_naive, newdata=Dataframe, type="class")  
confusion\_naive=table(Dataframe$species, pred)  
print(confusion\_naive)  
misclass\_naive=misclass(confusion\_naive, Dataframe)  
  
##Answer: The quality of the fit seems to be rather bad since the model misclassifies 39,5 % of the data according to the misclassification rate. Since the data is strongly correlated and the method of using Naive bayes implies an assumption of independent features it is reasonable that the model does not perform that well, again since it is clear from the plot that the two features used, CW and BD are strongly correlated.   
  
##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  
  
glmmodel=glm(species~CW+BD, data=Dataframe, family="binomial")  
pred\_glm=predict(glmmodel, newdata=Dataframe, type='response')  
species\_vector=as.factor(c(ifelse(pred\_glm>0.5, 'Orange', 'Blue')))  
confusion\_glm=table(Dataframe$species, species=ifelse(pred\_glm>0.5, 'Orange', 'Blue'))  
print(confusion\_glm)  
plot(CW, BD, main="Plot predicted values of CW and BD but with logistic regression",  
 col=c("blue", "orange")[species\_vector], xlab="CW", ylab="BD", xlim=c(18,53), ylim=c(5,22),  
 sub="Red = Female, Blue = Male, Green = Boundaryline")  
  
boundaryline = function(length, coefficientvector, prior) {  
 return(-coefficientvector[1]/coefficientvector[3]-(coefficientvector[2]/coefficientvector[3])\*length+  
 log(prior/(1-prior))/coefficientvector[3])  
}  
  
curve(boundaryline(x, glmmodel$coefficients, 0.5), col="green", add=TRUE)  
glmmodel$coefficients  
  
##Answer: The quality of the classification is as expected really good (since looking at the plot, a linear classifier seemed appropriate). The model only misclassified 4 points which yielded a low misclassification rate. The equation for the boundary line is as follows: BD=-0.484810/9.432817+CW\*3.624760/9.432817  
  
##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.   
  
pcaData=Dataframe[,7:8]  
pcaData=scale(pcaData)  
response=as.vector(Dataframe$species)  
pcaAnalysis=prcomp(pcaData)  
lambda=pcaAnalysis$sdev^2  
#Eigenvalues  
print(lambda)  
#Proportion of variation  
propVar= lambda/sum(lambda)\*100  
screeplot(pcaAnalysis)  
print(propVar)  
summary(pcaAnalysis)  
X=pcaData  
coeff=pcaAnalysis$rotation  
Z=X%\*%coeff  
  
##Answer: Since we could se a clear pattern in the plot from step 1, PCA analysis will select the angle of the line as its first principle component since across this line we can find the most variation in the data. Since the data was so strongly correlated, almost all of the variation in the data can be explained by just one PCA component. The equation expressing principle component coordinates through the original ones is as follows: PCACoordinates=X\*coefficientsMatrix, i.e. PCACoordinates=X\*pcaAnalysis$rotation.   
##Equations separately: PC1=CW\*0.7071068+BD\*0.7071068, PC2=CW\*(-0.7071068)+BD\*0.7071068  
  
##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.  
  
naiveData=as.data.frame(cbind(pcaAnalysis$x, species=response))  
naiveModelPCA=naiveBayes(species~PC1+PC2, data=naiveData)  
predPCA=predict(naiveModelPCA, newdata=naiveData, type="class")  
confusion\_naivePCA=table(Dataframe$species, predPCA)  
print(confusion\_naivePCA)  
misclass\_naivePCA=misclass(confusion\_naivePCA, Dataframe)  
  
##Answer: The classification is now 100 % correct. This is due to the fact that the two PCA components derived are mutually independent of each other which makes naive bayes classifier a perfect fit since this is exactly what the model is assuming.

##Assignment 2

##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.   
  
##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)  
  
#Read data  
RNGversion('3.5.1')  
Dataframe=read.csv2("bank.csv")  
  
linear\_model=glm(Visitors~., data=Dataframe, family="poisson")  
linear\_model$coefficients  
control=exp(0.1742155+0.4017126\*seq(9,12,0.1))  
print(control)  
  
##Answer: The probabilistic expression for the target is Visitors=e^(0.1742155+0.4017126\*Time). The control vector shows that the response variable resulted from the equation is fairly reasonable and resembles the original data.  
  
##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?  
  
library(boot)  
rng=function(data, mle) {  
 data1=data.frame(Visitors=data$Visitors, Time=data$Time)  
 n=length(data$Visitors)  
 #generate new Price  
 data1$Visitors=rnorm(n,predict(mle, newdata=data1), sd(mle$residuals))  
 return(data1)  
}  
  
f1=function(data1){  
 res=lm(Visitors~., data=data1) #fit linearmodel  
 #predictvaluesfor all Visitor valuesfrom the original data  
 Visitors=predict(res,newdata=data.frame(Time=seq(12,13,0.05)))  
 n=length(seq(12,13,0.05))  
 predictedVisitors=rnorm(n, Visitors, sd(linear\_model$residuals))  
 return(predictedVisitors)  
}  
res=boot(Dataframe, statistic=f1, R=1000, mle=linear\_model, ran.gen=rng, sim="parametric")  
e=envelope(res)  
plot(Dataframe$Time, Dataframe$Visitors, main="Forecasting of visitors depending on time", xlab="Time",  
 ylab="Visitors", xlim=c(9,13), ylim=c(30,500))  
points(seq(12,13,0.05), exp(e$point[2,]), type="l", lty=21, col="gray")  
points(seq(12,13,0.05), exp(e$point[1,]), type="l", lty=21, col="gray")  
  
min\_value\_13=exp(e$point[2,21])  
max\_value\_13=exp(e$point[1,21])  
cat("The bank should expect between", min\_value\_13, "and", max\_value\_13, "customers", sep=" ")  
  
##Answer: The band seems to give a correct forecasting. The bank should expect between approx 177 and 281  
##customers at 13:00.

#2018-01-11

##Assignment 1

#Read data and divide randomely into train and test  
Dataframe=read.csv("video.csv")  
Dataframe$codec = c()  
n=dim(Dataframe)[1]  
set.seed(12345)  
id=sample(1:n, floor(n\*0.5))  
train=Dataframe[id,]  
test=Dataframe[-id,]  
  
##Perform principle component analysis with and without scaling. How many vars for 95 % of variation in both cases. Explain why so few components are needed when scaling is not done.   
  
PCAdata=subset(train, select=-utime)  
pcaAnalysis\_noScale=prcomp(PCAdata, scale=FALSE)  
screeplot(pcaAnalysis\_noScale)  
lambda=pcaAnalysis\_noScale$sdev^2  
print(lambda)  
#Proportion of variation  
propVar=lambda/sum(lambda)\*100  
print(propVar)  
#Function for calculating the number of components needed for explaining at least 95% of the variation.  
calcNoVars = function(data){  
 noOfVars=1  
 sumOfVariation=data[noOfVars]  
 while(sumOfVariation<95){  
 noOfVars=noOfVars+1  
 sumOfVariation=sumOfVariation+data[noOfVars]  
 }  
 return(noOfVars)  
}  
print(calcNoVars(propVar))  
pcaAnalysis\_scale=prcomp(PCAdata, scale=TRUE)  
lambda\_scale=pcaAnalysis\_scale$sdev^2  
propVar\_scale=lambda\_scale/sum(lambda\_scale)\*100  
print(propVar\_scale)  
screeplot(pcaAnalysis\_scale)  
print(calcNoVars(propVar\_scale))  
  
##Answer: Fewer components are needed since outliers of the different parameters have a higher impact when they are not scaled accordingly. When scaled outliers have less impact and therefore the percentage of the variation for each component decreases.  
  
##Write a code that fits a principle component regression ("utime" as response and all scaled numerical variables as features) with M components to the training data and estimates the training and test errors, do this for all feasible M values. Plot dependence of the training and test errors on M and explain this plot in terms of bias-variance tradeoff.  
  
trainscore=c()  
testscore=c()  
library(pls)  
pcamodel=pcr(utime~., 17, data=train, scale=TRUE)  
for (i in 1:17) {  
 pred\_train=predict(pcamodel, ncomp=i)  
 pred\_test=predict(pcamodel, newdata=test, ncomp=i)  
 trainscore[i]=mean((train$utime-pred\_train)^2)  
 testscore[i]=mean((test$utime-pred\_test)^2)  
}  
  
plot(trainscore, xlab="Index", ylab="Error", col="blue", type="b", ylim=c(100,300))  
points(testscore, xlab="Index", ylab="Error", col="red", type="b", ylim=c(100,300))  
noOfPCA=which(testscore == min(testscore))  
print(noOfPCA)  
  
##Answer: When using more and more components the bias decreases and the variance goes up. The model performs betterand better on training data. However, at one point the model becomes overfitted and performs worse on the test data as more components are added. The point where the model performs best on test data is when using 14 PC:s.   
  
##Use PCR model with M=8 and report a fitted probabilistic model that shows the connection between the target and the principal components.  
  
pcamodel\_new=pcr(utime~., 8, data=train, scale=TRUE)  
pcamodel\_new$Yloadings  
mean(pcamodel\_new$residuals^2)  
  
##Answer: The formula is given by the loadings of the model and the variance is given by taking the average of the sum of squared residuals.  
  
##Use original data to create variable "class" that shows "mpeg" if variable "codec" is equal to "mpeg4", and "other" for all other values of "codec". Create a plot of "duration" versus "frames" where cases are colored by "class". Do you think that the classes are easily separable by a linear decision boundary?  
  
Dataframe2=read.csv("video.csv")  
Dataframe2=subset(Dataframe2, select=c(codec, frames, duration))  
Dataframe2=cbind(Dataframe2, class=ifelse(Dataframe2$codec == 'mpeg4', 'mpeg', 'other'))  
plot(Dataframe2$duration, Dataframe2$frames, col=c("red", "blue")[Dataframe2$class], xlab="Duration", ylab="Frames",  
 main="Plot of duration vs frames")  
  
##Answer: It seems that a linear decision boundary could separate the two classes rather well with exception of a few cases near the origin of the plot.   
  
##Fit a Linear Discriminant Analysis model with "class" as target and "frames" and "duration" as features to the  
##entire dataset (scale features first). Produce the plot showing the classified data and report the training error. Explain why LDA was unable to achieve perfect (or nearly perfect) classification in this case.  
  
#Create function for misclassification rate  
missclass=function(conf\_matrix, fit\_matrix){  
 n=length(fit\_matrix[,1])  
 return(1-sum(diag(conf\_matrix))/n)  
}  
  
library(MASS)  
LDAData=Dataframe2  
LDAData$duration=scale(LDAData$duration)  
LDAData$frames=scale(LDAData$frames)  
ldamodel=lda(class~duration+frames, data=LDAData)  
predicted\_lda=predict(ldamodel, data=LDAData)  
confusion\_matrix=table(LDAData$class, predicted\_lda$class)  
misclass=missclass(confusion\_matrix, LDAData)  
print(confusion\_matrix)  
print(misclass)  
plot(Dataframe2$duration, Dataframe$frames, col=c("red", "blue")[predicted\_lda$class], xlab="Duration", ylab="Frames",  
 main="Plot of duration vs frames after LDA")  
  
##Answer: Because the two clusters of data don't have the same covariance matrix which can also be seen in the plot. The linear patterns are different for the two classes and have clearly different slopes.   
  
##Fit a decision tree model with "class" as target and "frames" and "duration" as features to the entire dataset,   
##Choose an appropriate tree size by cross-validation. Report the training error. How many leaves are there in the final tree? Explain why such a complicated tree is needed to describe such a simple decision boundary.  
  
library(tree)  
treemodel=tree(class~duration+frames, data=Dataframe2)  
summary(treemodel)  
#Since number of terminal nodes is 11 we will check which number of leaves that is optimal in terms of lowest deviance  
trainscore=rep(0,11)  
for (i in 2:11) {  
 prunedTree=prune.tree(treemodel, best=i)  
 trainscore[i]=deviance(prunedTree)  
}  
plot(2:11, trainscore[2:11], type="b", col="red", ylim=c(0,700))  
finalTree=prune.tree(treemodel, best=11)  
temp=predict(treemodel, type="class")  
confusion\_matrix\_tree=table(Dataframe2$class, temp)  
tree\_misclass= missclass(confusion\_matrix\_tree, Dataframe2)  
print(confusion\_matrix\_tree)  
print(tree\_misclass)  
  
##Answer: As seen in the plot the optimal number of leaves is the maximal one which is 11.

##Assignment 2

##Train a neural network (NN) 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. Consider threshold values i/1000 with i=1,...,10. Initialize the weights of the neural network to random values in the interval [-1,1]. Consider two NN architectures: A single hidden layer of 10 units, and two hidden layers with 3 units each. Choose the most appropriate NN architecture and threshold value. Motivate your choice. Feel free to reuse the code of the corresponding lab.  
##Estimate the generalization error of the NN selected above (use any method of your choice).  
##In the light of the results above, would you say that the more layers the better ? Motivate your answer.  
  
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] for model with 1 hidden layer  
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)  
}  
  
#Random initialization of the weights in the interval [-1, 1] for model with two hidden layers  
winit2=runif(22,-1,1)  
trainScore2=rep(0,10)  
validScore2=rep(0,10)  
#R could not perform neuralnet analysis with thresholds smaller than 7/10. That is why the loop starts at 7.  
for(i in 7:10) {  
 nn\_temp2 <- neuralnet(Sin~Var, data=train, hidden=c(3,3), threshold=i/1000, startweights=winit2)  
 nn2 = as.data.frame(nn\_temp2$net.result)  
 pred2=predict(nn\_temp2, newdata=valid)  
 trainScore2[i] = 1/n\*sum((nn2[,1]-train$Sin)^2)  
 validScore2[i] = 1/n\*sum((pred2-valid$Sin)^2)  
}  
  
plot(1:10, validScore[1:10], type="b", col="red", xlab="Threshold index", ylab="MSE")  
plot(7:10, validScore2[7:10], type="b", col="blue", xlab="Threshold index", ylab="MSE")  
min1=min(validScore[1:10])  
min2=min(validScore2[7:10])  
finalModel=ifelse(min1<min2, "1", "2")  
optimal\_i=ifelse(finalModel == '1', which(validScore[1:10] == min1, which(validScore2[7:10] == min2)))  
print(finalModel)  
print(optimal\_i)  
  
##Answer: The most appropriate model is using a one layer architecture with 10 units and using a threshold index of 4/1000. This is because this model yields the lowest MSE when applied to the validation data. No, it is not always best to use multiple layers as seen in this example.   
  
#Generating new data for testing.  
  
Var = runif(50, 0, 10)  
test = data.frame(Var, Sin=sin(Var))  
n=dim(test)[1]  
winit = runif(31, -1, 1)  
finalModel = neuralnet(Sin~Var, data=trva, hidden=10, threshold=4/1000, startweights=winit)  
results=as.data.frame(finalModel$net.result)  
pred = predict(finalModel, newdata = test)  
generror = 1/n\*sum((pred-test$Sin)^2)  
print(generror)  
plot(prediction(finalModel)$rep1)   
points(test, col = "red")

#2019-01.16

##Assignment 1

##The data file Influenza.csv contains contains the number of registered cases of influenza and mortality.  
  
##Assume that mortality y is poisson distributed. Write an R code computing the minus-loglikelihood of Mortality values for a given 𝜆𝜆 (use only basic R functions, do not use implemented Poisson distribution in R). Compute the minus log-likelihood values for 𝜆𝜆 = 10,110,210, … , 2910 and produce a plot showing the dependence of the minus log-likelihood on the value of 𝜆𝜆. Define the optimal value of 𝜆𝜆 by means of visual inspection (i.e. approximately).  
  
RNGversion('3.5.1')  
  
Dataframe=read.csv("Influenza.csv")  
Mortality=Dataframe$Mortality  
  
like=function(y, lambda){  
 n=length(y)  
 return(lambda\*n-log(lambda)\*sum(y)+sum(log(factorial(y))))  
}  
  
#Find maximum likelihood value of theta  
lambda\_max = function(y){  
 n=length(y)  
 return(sum(y)/n)  
}  
  
lambda\_max=lambda\_max(Mortality)  
lambda=seq(10,2910,100)  
plot(like(Mortality, lambda), lambda, main="The minus loglike function of mortality depending on Lambda",   
 xlim=c(10,2910))  
  
##Answer: Towards infinity, don't know how to fix  
  
##Scale all variables except of Mortality. Divide the data randomly (50/50) into training and test sets and fit a LASSO regression with Mortality as a Poisson distributed target and all other variables as features. Select the optimal parameters in the LASSO regression by the crossvalidation and report the optimal LASSO penalization parameter and also the test MSE. Is the MSE actually the best way of measuring the error rate in this case? Report also the optimal LASSO coefficients and report which variables seem to have the biggest impact on the target. Check the value of intercept 𝛼𝛼, compute exp(𝛼𝛼) and compare it with the optimal 𝜆𝜆 in step 1. Are these quantities similar and should they be?  
  
library(cvTools)  
library(glmnet)  
  
features=scale(Dataframe[,-3])  
data=cbind(features, Mortality)  
  
n=dim(data)[1]  
set.seed(12345)  
id=sample(1:n, floor(n\*0.5))  
train=data[id,]  
test=data[-id,]  
covariates=train[,1:8]  
response=train[,9]  
  
lassomodel=cv.glmnet(as.matrix(covariates), response, alpha=1, family="poisson")  
opt\_lambda=lassomodel$lambda.min  
plot(lassomodel)  
coef(lassomodel, s="lambda.min")  
y=test[,9]  
ynew=predict(lassomodel, newx=as.matrix(test[,1:8]), type="response", s="lambda.min")  
MSE=mean((ynew-y)^2)  
interceptval=exp(coef(lassomodel, s="lambda.min")[1])  
  
##Answer: Optimal lasso coefficients shown above. The feature year is not used. The features Influenza,   
##temperature.deficit and influenza\_lag2 seem to have the biggest impact on the target. The exponential of the  
##intercept alpha is similar to the optimal lambda in step 1. This is due to the fact that the link between a poisson distribution and the calculated mean is the log-function. By applying exp to the intercept we thereby receive a value which corresponds to the calculated mean of lambda.   
  
##Fit a regression tree with Mortality as a target and all variables as a features and select the optimal size of the tree by cross-validation. Report the test MSE and compare it with the MSE of the LASSO regression in step 2. Why is it not reasonable to do variable selection by applying LASSO penalization also to the tree models?   
  
library(tree)  
train=as.data.frame(train)  
test=as.data.frame(test)  
treemodel=tree(Mortality~., data=train)  
set.seed(12345)  
cv.res=cv.tree(treemodel)  
plot(cv.res$size, cv.res$dev, type="b", col="red")  
bestSize=cv.res$size[which(min(cv.res$dev) == cv.res$dev)]  
finalTree=prune.tree(treemodel, best=bestSize)  
plot(finalTree)  
text(finalTree, pretty=0)  
yFit=predict(finalTree, newdata=test, type="vector")  
MSE\_tree=mean((yFit-test$Mortality)^2)  
  
##Answer: The MSE for the tree model is higher than for the LASSO-model which implies that the LASSO model should be used since it performs better. It is not reasonable to do LASSO penalization to tree model because the tree model is not continuous but discrete.   
  
##Perform principal component analysis using all the variables in the training data except of Mortality and report how many principal components are needed to capture more than 90% of the variation in the data. Use the coordinates of the data in the principal component space as features and fit a LASSO regression with Mortality as a Poisson distributed target by crossvalidation, check penalty factors 𝜆𝜆 = 0, 0.1, 0.2, … , 50. Provide a plot that shows the dependence of the cross-validation error on log(𝜆𝜆). Does complexity of the model increase when 𝜆𝜆 increases? How many features are selected by the LASSO regression? Report a probabilistic model corresponding to the optimal LASSO model.   
  
PCAdata=subset(train, select=-Mortality)  
pcaAnalysis=prcomp(PCAdata, scale=FALSE)  
screeplot(pcaAnalysis)  
lambda=pcaAnalysis$sdev^2  
print(lambda)  
#Proportion of variation  
propVar=lambda/sum(lambda)\*100  
print(propVar)  
#Function for calculating the number of components needed for explaining at least 95% of the variation.  
calcNoVars = function(data){  
 noOfVars=1  
 sumOfVariation=data[noOfVars]  
 while(sumOfVariation<90){  
 noOfVars=noOfVars+1  
 sumOfVariation=sumOfVariation+data[noOfVars]  
 }  
 return(noOfVars)  
}  
print(calcNoVars(propVar))  
summary(pcaAnalysis)  
new\_base=pcaAnalysis$x  
set.seed(12345)  
lasso\_PCA=cv.glmnet(new\_base[,1:5], response, alpha=1, family="poisson", lambda=seq(0,50,0.1))  
plot(lasso\_PCA)  
opt\_lambda\_PCA=lasso\_PCA$lambda.min  
coef(lasso\_PCA, s="lambda.min")  
  
##Answer: 5 components are needed to describe more than 90 % of the variation in the data. The complexity of the model decreases as lambda increases since a higher lambda penalizes the coefiicients more. 3 features are selected by the LASSO regression. The probabilistic model is: Yi~P(exp(7.484554465-0.035756922\*PC1-0.009395205\*PC2-0.004745676\*PC3+0.011627449\*PC4+0.003882809\*PC5))

##Assignment 2

##You are asked to use the function neuralnet of the R package of the same name to train a neural network (NN) to  
##mimic the trigonometric sine function. You should run the following code to obtain the training and test data.  
  
##Produce the code to train the NN on the training data tr and test it on the data te. Use a single hidden layer with three units. Initialize the weights at random in the interval [-1,1]. Use the default values for the rest of parameters in the function neuralnet. You may need to use the function compute. Confirm that you get results similar to the following figure. The black dots are the training data. The blue dots are the test data. The red dots are the NN predictions for the test data.  
  
library(neuralnet)  
set.seed(1234567890)  
Var <- runif(50, 0, 3)  
tr <- data.frame(Var, Sin=sin(Var))  
Var <- runif(50, 3, 9)  
te <- data.frame(Var, Sin=sin(Var))  
n = dim(tr)[1]  
# Random initialization of the weights in the interval [-1, 1]   
winit <- runif(10, -1, 1)  
nn=neuralnet(Sin~Var, data=tr, hidden=3, startweights=winit)  
pred=predict(nn, newdata=te)  
plot(tr$Var, tr$Sin, xlim=c(0,9), ylim=c(-2,2), xlab="Var", ylab="Sin")  
points(te$Var, te$Sin, col="blue")  
points(te$Var, pred, col="red")  
  
##Answer: The plot resembles the one given so it is confirmed.  
  
##In the previous figure, it is not surprising the poor performance on the range [3,9] because no training point falls in that interval. However, it seems that the predictions converge to -2 as the value of Var increases. Why do they converge to that particular value ? To answer this question, you may want to look into the weights of the NN learned.  
  
sigmoid=function(input){  
 return(1/(1+exp(-input)))  
}  
  
z1=sigmoid(0.61705\*te$Var-1.51988)  
z2=sigmoid(1.995\*te$Var-1.27708)  
z3=sigmoid(-1.61733\*te$Var+4.89639)  
y=-3.92871\*z1+2.67522\*z2+0.84607\*z3-0.62953  
print(y)  
  
##Answer: When Var goes towards infinity the first and second node in the hidden layer is activated and the third  
##node in the hidden layer is not. This yields an approximate response value of -3.92871+2.67522-0.62953 (bias)=-1,88302 which can be seen in the graph.   
  
##You are asked to use the function ksvm from the R package kernlab to learn a support vector machine (SVM) to classify the spam dataset that is included with the package. You should use the radial basis function kernel (also known as Gaussian) with a width of 0.05.  
# You should select the most appropriate value for the C parameter, i.e. you should perform model selection. For this task, you can use any method that you deem appropriate.  
# In the previous question, you may have obtained an error message “no support vectors found” for C = 0. Can you give a plausible explanation for this error ?  
# Estimate the generalization error of the SVM with the C value selected above. Use any method of your choice.  
# Once a SVM has been fitted, a new point is essentially classified according to the sign of a linear combination of support vectors. You are asked to produce the pseudocode (no implementation is required) for computing this linear combination. Your pseudocode should make use of the functions alphaindex, coef and b. See the help of ksvm for information about these functions.  
  
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],]  
  
C=seq(0.2,10.2,0.5)  
trainScore=numeric(21)  
validScore=numeric(21)  
for(i in 1:length(C)){  
 svmmodel=ksvm(type~., data=train, kernel="rbfdot", kpar=list(sigma=0.05), C=C[i])  
 pred=predict(svmmodel, newdata=valid)  
 confusion=table(valid$type, pred)  
 validScore[i]=missclass(confusion, valid)  
}  
plot(1:21, validScore, col="red", type="b")  
bestModel=which(min(validScore) == validScore)  
bestParam=C[bestModel]  
  
finalModel=ksvm(type~., data=spam[index[1:3501],], kernel="rbfdot", kpar=list(sigma=0.05), C=bestParam)  
finalPred=predict(svmmodel, newdata=test)  
finalConfusion=table(test$type, finalPred)  
finalMisclass=missclass(finalConfusion, test)  
  
##Answer: The optimal C-value is 1.2. When C=0 and no support vector was found you still are under the assumption that the data is linearly separable. When no support vector is found I assume that the data is not linearly separable. The generalization error for the optimal model is approximately 0.08.