Contents

[Labs 3](#_Toc29737716)

[Lab 1 3](#_Toc29737717)

[Assignment 1 – Logistic regression and KKNN-method 3](#_Toc29737718)

[Assignment 2 – Probabilistic model and Bayesian model (max.likelihood computations) 5](#_Toc29737719)

[Assignment 4 – Linear regression, Ridge, LASSO, stepAIC 7](#_Toc29737720)

[Lab 2 9](#_Toc29737721)

[Assignment 1 - LDA and logistic regression. Draw decision boundary 9](#_Toc29737722)

[Assignment 2 – Split method, Tree modeling (gini and deviance), Naïve Bayes, loss matrix 11](#_Toc29737723)

[Assignment 3 – Bootstrap with tree models 15](#_Toc29737724)

[Assignment 4 – Principal components, PCA analysis, ICA analysis, Trace plots 17](#_Toc29737725)

[Lab 3 18](#_Toc29737726)

[Assignment 1 – Kernel methods, Smoothing coefficients 18](#_Toc29737727)

[Assignment 2 – Support vector machines 21](#_Toc29737728)

[Assignment 3 – Neural networks, weight initialization 23](#_Toc29737729)

[Exam 24](#_Toc29737730)

[2016-01-09 24](#_Toc29737731)

[Assignment 1 – Tree, LASSO, Modified error function 24](#_Toc29737732)

[Assignment 2 - SVM with Kernel, nested cross-validation 26](#_Toc29737733)

[2017-04-18 27](#_Toc29737734)

[Assignment 1 – Naïve bayes, logistic regression, PCA 27](#_Toc29737735)

[Assignment 2 – Poisson distributed linear model, bootstrap prediction band 29](#_Toc29737736)

[2018-01-11 30](#_Toc29737737)

[Assignment 1 – PCA, PCA regression, LDA, tree 30](#_Toc29737738)

[Assignment 2 – Neural networks, difference between layers 32](#_Toc29737739)

[2019-01-16 34](#_Toc29737740)

[Assignment 1 – Poisson distributed maxlikelihood, Poisson LASSO regression, tree, PCA 34](#_Toc29737741)

[Assignment 2 – Neural networks, converge against value, analyze weights, SVMs 37](#_Toc29737742)

[R 38](#_Toc29737743)

[Examples 38](#_Toc29737744)

[Linear regression 38](#_Toc29737745)

[Linear regression – confidence intervals 39](#_Toc29737746)

[Ridge regression 39](#_Toc29737747)

[Lasso regression 40](#_Toc29737748)

[Stepwise selection (stepAIC, specify which way under *direction*) 40](#_Toc29737749)

[Holdout method (dividation into train, valid, test) 40](#_Toc29737750)

[Cross-validation – K folds and different predictor sets 40](#_Toc29737751)

[Logistic regression 40](#_Toc29737752)

[Linear Discriminative Analysis (LDA) and Quadratic Discriminative Analysis (QDA) 41](#_Toc29737753)

[Naïve bayes 41](#_Toc29737754)

[Decision trees 41](#_Toc29737755)

[Least Absolute Deviation (LAD) regression 42](#_Toc29737756)

[Bootstrap 42](#_Toc29737757)

[Principle Component Analysis (PCA) 43](#_Toc29737758)

[Probabilistic PCA 44](#_Toc29737759)

[Independent Component Analysis (ICA) 44](#_Toc29737760)

[Distance between geographical points of interest 44](#_Toc29737761)

[Neural networks 44](#_Toc29737762)

[Additional functions 45](#_Toc29737763)

[Distributions 45](#_Toc29737764)

[Useful functions 45](#_Toc29737765)

[Standard home-made functions 45](#_Toc29737766)

[Theory 47](#_Toc29737767)

[Degrees of freedom 47](#_Toc29737768)

[Ridge regression 47](#_Toc29737769)

[LASSO regression 47](#_Toc29737770)

[Bias-variance trade-off 47](#_Toc29737771)

[Logistic regression 48](#_Toc29737772)

[Linear discriminative analysis 48](#_Toc29737773)

[Naïve bayes 48](#_Toc29737774)

[Trees 48](#_Toc29737775)

[Models 48](#_Toc29737776)

[Link functions 48](#_Toc29737777)

[Args against deterministic models 49](#_Toc29737778)

[Bootstrap 49](#_Toc29737779)

[PCA 49](#_Toc29737780)

[ICA 49](#_Toc29737781)

[Kernel 49](#_Toc29737782)

[Support vector machines 49](#_Toc29737783)

[Neural networks 50](#_Toc29737784)

[Lecture notes 50](#_Toc29737785)

# Labs

## Lab 1

### Assignment 1 – Logistic regression and KKNN-method

#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 – Probabilistic model and Bayesian model (max.likelihood computations)

#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 – Linear regression, Ridge, LASSO, stepAIC

#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.

## Lab 2

### Assignment 1 - LDA and logistic regression. Draw decision boundary

#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 – Split method, Tree modeling (gini and deviance), Naïve Bayes, loss matrix

#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 – Bootstrap with tree models

#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 – Principal components, PCA analysis, ICA analysis, Trace plots

#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.

## Lab 3

### Assignment 1 – Kernel methods, Smoothing coefficients

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 – Support vector machines

##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 – Neural networks, weight initialization

## 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.

# Exam

## 2016-01-09

### Assignment 1 – Tree, LASSO, Modified error function

##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 - SVM with Kernel, nested cross-validation

##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 – Naïve bayes, logistic regression, PCA

##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 – Poisson distributed linear model, bootstrap prediction band

##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 – PCA, PCA regression, LDA, tree

#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 – Neural networks, difference between layers

##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 – Poisson distributed maxlikelihood, Poisson LASSO regression, tree, PCA

##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 – Neural networks, converge against value, analyze weights, SVMs

##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.

# R

## Examples

### Linear regression

fit=lm(formula, data, subset, weights,…)

* data is the data frame containing the predictors and response values
* formula is expression for the model
* subset which observations to use (training data)?
* weights should weights be used?

fit is object of class lm containing various regression results.

• Useful functions (many are generic, used in many other models)

– Get details about the particular function by ”.”, for ex. predict.lm

summary(fit)

predict(fit, newdata, se.fit, interval)

coefficients(fit) # model coefficients

confint(fit, level=0.95) # CIs for model parameters

fitted(fit) # predicted values

residuals(fit) # residuals

**Example of ordinary least squares regression**

mydata=read.csv2("Bilexempel.csv")

fit1=lm(Price~Year, data=mydata)

summary(fit1)

fit2=lm(Price~Year+Mileage+Equipment, data=mydata)

summary(fit2)

### Linear regression – confidence intervals

fitted <- predict(fit1, interval = "confidence")

# plot the data and the fitted line

attach(mydata)

plot(Year, Price)

lines(Year, fitted[, "fit"])

# plot the confidence bands

lines(Year, fitted[, "lwr"], lty = "dotted", col="blue")

lines(Year, fitted[, "upr"], lty = "dotted", col="blue")

detach(mydata)

### Ridge regression

Use package glmnet with alpha = 0 (Ridge)

data=read.csv("machine.csv", header=F)

covariates=scale(data[,3:8])

response=scale(data[, 9])

model0=glmnet(as.matrix(covariates), response, alpha=0,family="gaussian")

plot(model0, xvar="lambda", label=TRUE)

**Choosing best model using cross-validation**

model=cv.glmnet(as.matrix(covariates), response, alpha=0,family="gaussian")

model$lambda.min

plot(model)

coef(model, s="lambda.min")

**How good is this model in prediction?**

ind=sample(209, floor(209\*0.5))

data1=scale(data[,3:9])

train=data1[ind,]

test=data1[-ind,]

covariates=train[,1:6]

response=train[, 7]

model=cv.glmnet(as.matrix(covariates), response, alpha=1,family="gaussian", lambda=seq(0,1,0.001))

y=test[,7]

ynew=predict(model, newx=as.matrix(test[, 1:6]), type="response")

#Coefficient of determination

sum((ynew-mean(y))^2)/sum((y-mean(y))^2)

sum((ynew-y)^2)

### Lasso regression

Use package glmnet with alpha = 1 (LASSO)

### Stepwise selection (stepAIC, specify which way under *direction*)

library(MASS)

fit <- lm(V9~.,data=data.frame(data1))

step <- stepAIC(fit, direction="both")

step$anova

summary(step)

### Holdout method (dividation into train, valid, test)

**How to partition into train/test**  
  
n=dim(data)[1]

set.seed(12345)

id=sample(1:n, floor(n\*0.7))

train=data[id,]

test=data[-id,]

**How to partition into train/valid/test**

n=dim(data)[1]

set.seed(12345)

id=sample(1:n, floor(n\*0.4))

train=data[id,]

id1=setdiff(1:n, id)

set.seed(12345)

id2=sample(id1, floor(n\*0.3))

valid=data[id2,]

id3=setdiff(id1,id2)

test=data[id3,]

### Cross-validation – K folds and different predictor sets

**Try models with different predictor sets**

data=read.csv("machine.csv", header=F)

library(cvTools)

fit1=lm(V9~V3+V4+V5+V6+V7+V8, data=data)

fit2=lm(V9~V3+V4+V5+V6+V7, data=data)

fit3=lm(V9~V3+V4+V5+V6, data=data)

f1=cvFit(fit1, y=data$V9, data=data,K=10, foldType=" consecutive")

f2=cvFit(fit2, y=data$V9, data=data,K=10, foldType=" consecutive")

f3=cvFit(fit3, y=data$V9, data=data,K=10, foldType=" consecutive")

res=cvSelect(f1,f2,f3)

plot(res)

### Logistic regression

Use glm() with family=”binomial”

* Predicted probabilities: predict(fit, newdata, type=”response”)

### Linear Discriminative Analysis (LDA) and Quadratic Discriminative Analysis (QDA)

Syntax in R: Library MASS

lda(formula, data, ..., subset, na.action)  
•Prior – classprobabiliies  
•Subset– indices, if training data should be used   
  
qda(formula, data, ..., subset, na.action)  
predict(..)

### Naïve bayes

Use package e1071

library(MASS)

library(e1071)

n=dim(housing)[1]

ind=rep(1:n, housing[,5])

housing1=housing[ind,-5]

fit=naiveBayes(Sat~., data=housing1)

fitYfit=predict(fit, newdata=housing1)

table(Yfit,housing1$Sat)

### Decision trees

Use package *tree*, alternative *rpart*

tree(formula, data, weights, control, split = c("deviance", "gini"), ...)

print(), summary(), plot(), text()

**Example**

library(tree)

n=dim(biopsy)[1]

fit=tree(class~., data=biopsy)

plot(fit)

text(fit, pretty=0)

fit

summary(fit)

**Misclassification results**

Yfit=predict(fit, newdata=biopsy, type="class")

table(biopsy$class,Yfit)

**Selecting optimal tree by penalizing**

* Use Cv.Tree()

set.seed(12345)

ind=sample(1:n, floor(0.5\*n))

train=biopsy[ind,]

valid=biopsy[-ind,]

fit=tree(class~.,data=train)

set.seed(12345)

cv.res=cv.tree(fit)

plot(cv.res$size, cv.res$dev, type="b", col="red")

plot(log(cv.res$k), cv.res$dev, type="b", col="red")

**Selecting optimal tree by train/validation**

fit=tree(class~., data=train)

trainScore=rep(0,9)

testScore=rep(0,9)

for(i in 2:9) {

prunedTree=prune.tree(fit,best=i)

pred=predict(prunedTree, newdata=valid, type="tree")

trainScore[i]=deviance(prunedTree)

testScore[i]=deviance(pred)

}

plot(2:9, trainScore[2:9], type="b", col="red", ylim=c(0,250))

points(2:9, testScore[2:9], type="b", col="blue")

**Use final tree decided by methods above**

finalTree=prune.tree(fit, best=5)

Yfit=predict(finalTree, newdata=valid, type="class")

table(valid$class,Yfit)

### Least Absolute Deviation (LAD) regression

Use package L1pack

### Bootstrap

Use package boot

Functions:

* Boot()
* Boot.ci() – 1 parameter
* Envelope() – many parameters

Example: boot(data, statistic, R, sim=”ordinary”, ran.gen = function(d, p) d, mle=NULL, …)

Random random generation for parametric bootstrap:

* Rnorm()
* Runif()
* …

**Nonparametric bootstrap:**

* Write a function statistic that depends on dataframe and index and returns the estimator

library(boot)

data2=data[order(dat a$Area),]#reorderingdata accordingto Area

# computingbootstrapsamples

f=function(data, ind){

data1=data[ind,]# extractbootstrapsample

res=lm(Price~Area, data=data1) #fit linearmodel

#predictvaluesfor all Area valuesfrom the original data

priceP=predict(res,newdata=data2)

return(priceP)

}

res=boot(data2, f, R=1000) #make bootstrap

**Parametric bootstrap**

* Compute value mle that estimates model parameters from the data
* Write function ran.gen that depends on data and mle and which generates new data
* Write function statistics that depend on data which will be generated by ran.gen and should return the estimator

mle=lm(Price~Area, data=data2)

summaryMLE = summary(mle)

rng=function(data, mle ) {

data1=data.frame(Price=data$Price, Area=data$Area)

n=length(data$Price)

#generatenew Price

data1$Price=rnorm(n,predict(mle, newdata=data1),sd(summaryMLE$residuals))

return(data1)

}

f1=function(data1){

res=lm(Price~Area, data=data1) #fit linearmodel

#predictvaluesfor all Area valuesfrom the original data

priceP=predict(res,newdata=data2)

return(priceP)

}

res=boot(data2, statistic=f1, R=1000, mle =mle,ran.gen=rng , sim="parametric")

**Bootstrap confidence bands for linear model**

e=envelope(res) #computeconfidencebands

fit=lm(Price~Area, data=data2)

priceP=predict(fit)

plot(Area, Price, pch=21, bg="orange")

points(data2$Area,priceP,type="l") #plotfittedline

#plotcofidencebands

points(data2$Area,e$point[2,], type="l", col="blue")

points(data2$Area,e$point[1,], type="l", col="blue")

**Bootstrap prediction bands for linear model**

mle=lm(Price~Area, data=data2)

f1=function(data1){

res=lm(Price~Area, data=data1) #fit linearmodel

#predictvaluesfor all Area valuesfrom the original data

priceP=predict(res,newdata=data2)

n=length(data2$Price)

predictedP=rnorm(n,priceP, sd(mle$residuals))

return(predictedP)

}

res=boot(data2, statistic=f1, R=10000, mle=mle, ran.gen=rng, sim="parametric")

### Principle Component Analysis (PCA)

* Formulas: Prcomp(), biplot(), screeplot()
* Use package pls for making Principle component regression
* $rotation – the coefficients

mydata=read.csv2("tecator.csv")

data1=mydata

data1$Fat=c()

res=prcomp(data1)

lambda=res$sdev^2

#eigenvalues

lambda

#proportion of variation

sprintf("%2.3f",lambda/sum(lambda)\*100)

screeplot(res)

**Principal component loadings (U)**

U=res$rotation

head(U)

**Data in (PC1, PC2) – scores (Z)**

plot(res$x[,1], res$x[,2], ylim=c(-5,15))

**Trace plots**

U= res$rotation

plot(U[,1], main="Traceplot, PC1")

plot(U[,2],main="Traceplot, PC2")

### Probabilistic PCA

* Use pcaMethods from Bioconductor

### Independent Component Analysis (ICA)

R package: fastICA

S <- cbind(sin((1:1000)/20), rep((((1:200)-100)/100), 5))

A < - matrix(c(0.291, 0.6557, -0.5439, 0.5572), 2, 2)

X < - S %\*% A #mixing signals

a < - fastICA(X, 2) #now separate them

### Distance between geographical points of interest

Use package geosphere

Use function distHaversine(x, y) which returns the distance between the two points whereas x and y are 2 dimensional vectors/data frames (i.e. pairs of coordinates of latitude and longitude)

### Neural networks

Use package neuralnet

Use function neuralnet(formula, data, hidden, threshold, startweights)

* Formula – X~Y
* Data – the data used
* Hidden – the number of nodes in each hidden layers (can be vector of c(5, 3) which corresponds to first hidden layer with 5 nodes and second hidden layer with 3 nodes)
* Threshold – the threshold of the partial derivatives of the error function as stopping criteria
* Startweights – a vector containing starting values of the weights (needs to be initialized), if NULL random initialization

## Additional functions

### Distributions

* Normal distribution (gaussian, link=”identity”)
  + dnorm(x, mean = 0, sd = 1, log = FALSE)
  + pnorm(q, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
  + qnorm(p, mean = 0, sd = 1, lower.tail = TRUE, log.p = FALSE)
  + rnorm(n, mean = 0, sd = 1)
* Exponential distribution
  + dexp(x, rate = 1, log = FALSE)
  + pexp(q, rate = 1, lower.tail = TRUE, log.p = FALSE)
  + qexp(p, rate = 1, lower.tail = TRUE, log.p = FALSE)
  + rexp(n, rate = 1)
* Poisson distribution (link = log)
  + dpois(x, lambda, log = FALSE)
  + ppois(q, lambda, lower.tail = TRUE, log.p = FALSE)
  + qpois(p, lambda, lower.tail = TRUE, log.p = FALSE)
  + rpois(n, lambda)
* Binomial distribution (link = logit)
  + dbinom(x, size, prob, log = FALSE)
  + pbinom(q, size, prob, lower.tail = TRUE, log.p = FALSE)
  + qbinom(p, size, prob, lower.tail = TRUE, log.p = FALSE)
  + rbinom(n, size, prob)

### Useful functions

* sapply(x, y) – input x in specified function y, returns vector of response values.
* Strptime(x, format=”%Y-%m-%d %H:%M:%S”) – convert string to datetime object with specified format
* Format(date, format=”%H”) – returns the specified part of the format string of a datetime object
* Cbind(x, y) – adds culumns y to x
* Rbind(x, y) – adds rows y to x
* Ifelse(eval, true, false) – returns true if eval is TRUE and false otherwise
* Which(x) – returns the TRUE indices of the eval x
* Optimize(function, interval) – optimizes the function specified over the interval specified
* Lty in plot – specifies sign of symbols (21 for confidence intervals)
* Nrow(Dataframe) – number of rows in dataframe
* Ncols(Dataframe) – number of columns in dataframe
* Paste(x, y, sep=””) – join multiple vectors together
* Data(x) – load a built-in dataset into the environment
* X[c(1, 5)] – selects elements 1 and 5
* Is-na(a) – is missing
* Is.null(a) – is null
* L[[2]] – second element of list
* L[1] – new list with only first element
* L$x – element named x
* L[‘y’] – new list with only element named y

**Ggplot**

Ggplot(data, aes(x=x,y=y, colour=class)) +

Geom\_point()

### Standard home-made functions

**Misclassification rate**

missclass=function(conf\_matrix, fit\_matrix){  
 n=length(fit\_matrix[,1])  
 return(1-sum(diag(conf\_matrix))/n)  
}

**Boundary line for logistic regression**boundaryline = function(length, coefficientvector, prior) {  
 return(-coefficientvector[1]/coefficientvector[3]-(coefficientvector[2]/coefficientvector[3])\*length+  
 log(prior/(1-prior))/coefficientvector[3])  
}

**Compare test and train scores for pruned tree**

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)  
}

**Classify data in to class according to prior**

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)  
}

**Calc TPR and FPR to use in ROC-curve**

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)  
}

**Estimate error for different neural network models**

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)  
}

# Theory

* General error estimation – trained data tested on not before seen test data. Often train model on train data, optimize parameters on validation data. Train chosen model on train and validation data and test generalization error on test data
* Model returned to user – the model which is trained on all the data available. Estimated data is the generalization error
* Naïve bayes – assumption: strong independence between variables. When applying PCA system with naïve bayes you receive very good classifiers since all the features in PCA are independent of each other.

## Degrees of freedom

* Larger covariance => larger connection => better approximation => model more complex

## Ridge regression

* High degree of polynomial leads to overfitting
* Theorem MAP estimate to the Bayesian ridge is equal to solution in frequentist ridge
* Particularly useful if the explanatory variables are strongly correlated to each other
* Degrees of freedom decrease when lambda increases

## LASSO regression

* Yields sparse solution due to square compared to ridge which is circle

## Bias-variance trade-off

* If bias(y(hat)(x0))=0, the estimator is unbiased
* ML estimators are asymptotically unbiased if the model is enough complex
* However, unbiasedness does not mean a good choice
* Risk = actual variance + bias ^2 + var(estimator)
* When model complexity increases, bias goes down and variance goes up and vice versa

## Logistic regression

* Proof

## Linear discriminative analysis

* Use log(ratio) for the boundary line to be linear
* Important: covariance matrices are the same
* Difference between logistic regression:
  + Coefficients estimated differently
* Not robust to outliers
* LDA is often a good classifier even if the assumption of normality and common covariance matrix are not satisfied

## Naïve bayes

* Special case of LDA with diagonal covariance matrix – assumes mutual independence between features
* MLE are means and variances (per class)

## Trees

* Regression trees
  + Target is a continuous variable
* Classification trees
  + Target is a class (qualitative) variable
* Number of areas in graph represent the number of leaves and terminal nodes of tree
* Normal model leads to regression trees
  + Objective: MSE to evaluate
* Mulitnoulli model leads to classification trees
  + Objective: cross-entropy (deviance)
* Robust to outliers
* High variance: small change in response => totally different tree
* Lack of smoothness
* Large trees may be needed to model an easy system

## Models

* Normally distributed targets => linear regression
* Bernoulli and multinomial targets => logistic regression

### Link functions

* F is link function (in principle arbitrary)
* F^-1 is activation function

### Args against deterministic models

* The model does not really describe actual data (error is not explained)
  + No difference in modelling data A (Poisson) and B (Normal)
  + Estimation strategy for A is not good for B
* The model typically gives a deterministic answer, no information about uncertainty

## Bootstrap

* Nonparametric bootstrap can be applied to any deterministic estimator, distribution-free
* Parametric bootstrap is however more exact if the distribution form is correct
* Bootstrap works for all distribution types
* Can be bad for small datasets, for example when n<40
* Parametric bootstrap works even for small samples

## PCA

* Data become more approximate (but less data to store)
* PC1, … , PCM are eigenvectors of sample covariance
* Another view: minimize the distance between the original and projected data
* Do we need to scale features?
  + If scaling, less likely to have same direction as original base
  + You need to scale data
  + Don’t want to scale data if all components is in same scale (e.g. in centimeters or meters)
  + Otherwise, always scale
* Equation: new base Z=X\*Um where X is original data and Um is the coefficients (e.g. $rotation)
* Approx X values: X(approx.) = Z\*Um(transpose) where columns have been removed to only use most important PCA components in Z

## ICA

* Probabilistic PCA does not capture latent factors
  + Rotation invariance
* ICA assumes latent features are independent
  + Assuming noise-free x
* Convert X into PCA coordinate system: X’=XU
* Z=X’V
* Full transformation matrix: Uica=U\*V

## Kernel

Gaussian kernel: k(u) = exp(−∣∣u∣∣2 ) where ∣∣ ⋅ ∣∣ is the Euclidean norm.

▸ Cauchy kernel: k(u) = 1/(1 + ∣∣u∣∣D+1 )

▸ Epanechnikov kernel: k(u) = (1 − ∣∣u∣∣2 )1{∣∣u∣∣≤1}

▸ Moving window kernel: k(u) = 1{u∈S(0,1)}

* Small width implies considering few points. So, the variance will be large (similar to variance of a single point). The bias will be small since the points considered are close to x.
* Large width implies considering many points. So, the variance will be small and the bias will be large
* Cross-validation (K-fold): Not always best to choose large K (big train set) since this implies large variance on the test set (since it is so small), K=5, 10 works well

## Support vector machines

* With no penalty, assumes that training set is linearly separable in the feature space (but not necessarily in input space)
* Aim for the separating hyperplane that maximizes the margin (i.e. the smallest perpendicular distance from any point to the hyperplane) so as to minimize the generalization error
* When adding penalty for penalizing misclassified points we drop assumption of linear separability in feature space
* A higher C implies penalizing violation of constraints more which yields a more non-smooth support vector. Creation of “islands” around points for example to avoid misclassification

## Neural networks

* SVMs imply data-selected user-defined basis functions
* NNs imply a user-defined number of data-selected basis functions
* Neural networks is the same as ordinary linear classifiers without its activation functions
* For a large variety of activation functions, the two-layer NN can uniformly approximate any continuous function to arbitrary accuracy provided enough hidden units
* Initialize weights at random, but
  + Too small magnitude values may cause losing signal in the forward or backward passes, and
  + Too big magnitude values may cause the activation function to saturate and lose gradient
* Initialize the weights according to prior knowledge: Almost-zero for hidden units that are unlikely to interact, and bigger magnitude for the rest
* Initialize the weights to almost-zero values so that the initial model is almost-linear, i.e. sigmoid functions is almost linear around the zero. Let the algorithm to introduce non-linearities when needed
  + Note however that this initialization makes the sigmoid function take a value around half its saturation level. That is why hyperbolic tangent function (tanh(x)) is sometimes preferred in practise

# Lecture notes