

## Contents

Labs .....	2
Lab 1 .....	2
Assignment 1 – Logistic regression and KNN-method .....	2
Assignment 2 – Probabilistic model and Bayesian model (max.likelihood computations) .....	4
Assignment 4 – Linear regression, Ridge, LASSO, stepAIC .....	6
Lab 2 .....	9
Assignment 1 - LDA and logistic regression. Draw decision boundary .....	9
Assignment 2 – Split method, Tree modeling (gini and deviance), Naïve Bayes, loss matrix.....	11
Assignment 3 – Bootstrap with tree models .....	14
Assignment 4 – Principal components, PCA analysis, ICA analysis, Trace plots .....	16
Lab 3 .....	18
Assignment 1 – Kernel methods, Smoothing coefficients .....	18
Assignment 2 – Support vector machines .....	21
Assignment 3 – Neural networks, weight initialization .....	22
Exam.....	23
2016-01-09.....	23
Assignment 1 – Tree, LASSO, Modified error function .....	23
Assignment 2 - SVM with Kernel, nested cross-validation.....	25
2017-04-18.....	26
Assignment 1 – Naïve bayes, logistic regression, PCA .....	26
Assignment 2 – Poisson distributed linear model, bootstrap prediction band .....	28
2018-01-11.....	29
Assignment 1 – PCA, PCA regression, LDA, tree .....	29
Assignment 2 – Neural networks, difference between layers .....	32
2019-01-16.....	33
Assignment 1 – Poisson distributed maxlikelihood, Poisson LASSO regression, tree, PCA .....	33
Assignment 2 – Neural networks, converge against value, analyze weights, SVMs .....	36
R .....	38
Examples .....	38
Linear regression .....	38
Linear regression – confidence intervals .....	38
Ridge regression .....	38
Lasso regression .....	39
Stepwise selection (stepAIC, specify which way under <i>direction</i> ) .....	39

Holdout method (dividation into train, valid, test).....	39
Cross-validation – K folds and different predictor sets.....	40
Logistic regression .....	40
Linear Discriminative Analysis (LDA) and Quadratic Discriminative Analysis (QDA) .....	40
Naïve bayes .....	40
Decision trees .....	40
Least Absolute Deviation (LAD) regression.....	41
Bootstrap.....	41
Principle Component Analysis (PCA) .....	43
Probabilistic PCA.....	43
Independent Component Analysis (ICA).....	43
Distance between geographical points of interest .....	43
Neural networks .....	44

## Labs

### Lab 1

#### Assignment 1 – Logistic regression and KNN-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  $\hat{Y} = 1$  if  $p(Y=1 | X) > 0.5$ , otherwise  $\hat{Y} = 0$  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 classification 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 kkn() 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 a lot. 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 KNN model were significantly worse on the test data. This implies that KNN classification with  $K=30$  is worse than logistic regression in this case.*

*#5: Repeat step 4 for  $K=1$ . Classify according to kkn with  $k=1$  for test and train data sets. What does the decrease of  $K$  lead to and why?*

```
kkn_1_train = knn(formula = Spam~., train, train, k=1)
kkn_1_test = knn(formula = Spam~., train, test, k=1)
confusion_matrix_kkn1_train = table(train$Spam, kkn_1_train$fitted.values>0.5)
missclass_kkn1_train = missclass(confusion_matrix_kkn1_train, train)
confusion_matrix_kkn1_test = table(test$Spam, kkn_1_test$fitted.values>0.5)
missclass_kkn1_test = missclass(confusion_matrix_kkn1_test, test)
print(confusion_matrix_kkn1_train)
print(missclass_kkn1_train)
print(confusion_matrix_kkn1_test)
print(missclass_kkn1_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  $x = \text{Length}$  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 the*

```

ype 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 m
aximum 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="r
ed", ylim=c(0,100))
curve(-loglikelihood(x, y), xlab="Theta", from=0, to=20, add=TRUE, col="blue", yli
m=c(0,100))

#Conclusion: The graph is increasing at a much slower pace when only using the fir
st 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 wher
eas the one with only six values, the theta value could be anything from the minim
um value and forward.

#4: Assume now a Bayesian model with  $p(x|\theta)=\theta e^{(-\theta x)}$  and a prior  $p(\theta)=\lambda e^{(-\lambda \theta)}$ ,  $\lambda=10$ 
#Write a function computing  $l(\theta)=\log(p(x|\theta)*p(\theta))$ . What kind of measur
e is actually computed by this
#function? Plot the curve showing the dependence of  $l(\theta)$  on  $\theta$  computed usi
ng the entire data and overlay it with a plot from step 2. Find an optimal  $\theta$  a
nd 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, t
o=10, add=TRUE, col="blue",
  ylim=c(20,300))

#Conclusion: When using an bayesian model we have a prior that gives the model inf
ormation beforehand which helps

```

*#fitting the model. The optimal theta is now 0.91 which is close to the datasets mean value, 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  $M_i$  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.  $M_1$  is a linear model,  $M_2$  is a quadratic model and so on). Report a probabilistic model that describes  $M_i$ . Why is it appropriate to use MSE criterion when fitting this model to a training data?*

*#Conclusion: A probabilistic model describing  $M(i)$  is:  $M(i) = w_0 + w_1 * X + w_2 * X^2 + \dots + w_i * X^i$  (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  $M_i$ ,  $i=1, \dots, 6$ . 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 were 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)
missclass = missclass(confusion_matrix, Dataframe)
print(confusion_matrix)
print(missclass)
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 noticeable 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 left 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 = misclass(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 notable in the confusion matrix that no males classified correctly. This is also due to the high prior which basically says that it 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 = misclass(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 coincidence and no real conclusion can be drawn regarding the exact same misclassification rate except from that the models see*

m 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:  $RW(\hat{w}) = -(beta_0 + beta_1 * CL) / beta_2$

## 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 training data to perform classification using Naives bayes and report the confusion matrices and misclassification rates for the training 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 matrix 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 corresponding 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 can 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)
# computing bootstrapsamples
f=function(data, ind){
  data1=data[ind,]# extract bootstrapsample
  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 accomodate 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 linear model
  prunedtree=prune.tree(treemodel, best=3)
  n=length(Dataframe$EX)
  #predict values for all EX values from the original data
  predData=predict(prunedtree,newdata=Dataframe)
  predictedEX=rnorm(n, predData, sd(summaryMLE$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 should 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 preferred => 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 concluded 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 explained 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 noticeable that for the first component the dependency on the features increases as the index increases. It is also noticeable 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 which 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 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 similarly 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 func-
tion kernel (also known as Gaussian) with a width of 0.05. For the parameter C, co-
nsider 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 y-
our 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)
missclass1=missclass(confusion1, valid)
print(confusion1)
print(missclass1)

svmmodel2=ksvm(type~., data=train, kernel="rbfdot", kpar=list(sigma=0.05), C=1)
pred2=predict(svmmodel2, newdata=valid)
confusion2=table(valid$type, pred2)
missclass2=missclass(confusion2, valid)
print(confusion2)
print(missclass2)
```

```

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  $i = 1, \dots, 10$ . 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 number 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*

cted probabilities of  $Y_i=1$ . 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 see 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 linear model
  #predict values for all Visitor values from 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 bank 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 randomly 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 better and 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 PCs.*

*##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 "code c" 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 th at 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$codec == 'mpeg4', 'mpeg', 'ot
her'))
plot(Dataframe2$duration, Dataframe2$frames, col=c("red", "blue")[Dataframe2$clas
s], 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)
LDADData=Dataframe2
LDADData$duration=scale(LDADData$duration)
LDADData$frames=scale(LDADData$frames)
ldamodel=lda(class~duration+frames, data=LDADData)
predicted_lda=predict(ldamodel, data=LDADData)
confusion_matrix=table(LDADData$class, predicted_lda$class)
missclass=missclass(confusion_matrix, LDADData)
print(confusion_matrix)
print(missclass)
plot(Dataframe2$duration, Dataframe$frames, col=c("red", "blue")[predicted_lda$cla
ss], 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 w hich can also be seen in the plot. The linear patterns are different for the two c lasses 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 i s 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  $i=1, \dots, 10$ . 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, star
tweights=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="MS
E")
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(validSco
re2[7:10] == min2)))
print(finalModel)
print(optimal_i)

```

*##Answer: The most appropriate model is using a one layer architecture with 10 uni  
ts and using a threshold index of 4/1000. This is because this model yields the lo  
west MSE when applied to the validation data. No, it is not always best to use mul  
tiple 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, startweigh
ts=winit)
results=as.data.frame(finalModel$net.result)
pred = predict(finalModel, newdata = test)
genererror = 1/n*sum((pred-test$Sin)^2)
print(genererror)
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 mi  
nus-loglikelihood of Mortality values for a given  $\lambda$  (use only basic R functions,  
do not use implemented Poisson distribution in R). Compute the minus log-likelihoo  
d values for  $\lambda = 10, 110, 210, \dots, 2910$  and produce a plot showing the dependence  
of the minus log-likelihood on the value of  $\lambda$ . Define the optimal value of  $\lambda$  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 mortalit
y 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 t
raining and test sets and fit a LASSO regression with Mortality as a Poisson distr
ibuted target and all other variables as features. Select the optimal parameters i
n the LASSO regression by the crossvalidation and report the optimal LASSO penaliz
ation parameter and also the test MSE. Is the MSE actually the best way of measuri
ng the error rate in this case? Report also the optimal LASSO coefficients and rep
ort which variables seem to have the biggest impact on the target. Check the value
of intercept  $\alpha$ , compute  $\exp(\alpha)$  and compare it with the optimal  $\lambda$  in step 1. Ar
e 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.mi
n")
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. Th

```

e 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  $\lambda = 0, 0.1, 0.2, \dots, 50$ . Provide a plot that shows the dependence of the cross-validation error on  $\log(\lambda)$ . Does complexity of the model increase when  $\lambda$  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)
screplot(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 coefficients more. 3 features are selected by the LASSO regression. The probabilistic model is:  $Y_i \sim P(\exp(7.484554465 - 0.035756922 \cdot PC1 - 0.009395205 \cdot PC2 - 0.004745676 \cdot PC3 + 0.011627449 \cdot PC4 + 0.003882809 \cdot PC5))$*

## 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(sigm

```

```
a=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 – class probabilities
- 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(data$a$Area),]#reordering data according to Area
# computing bootstrap samples
f=function(data, ind){
  data1=data[ind,]# extract bootstrap sample
  res=lm(Price~Area, data=data1) #fit linear model
  #predict values for all Area values from 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$Price)
  #generate new Price
  data1$Price=rnorm(n,predict(mle, newdata=data1),sd(summaryMLE$residuals))
  return(data1)
}
```

```
f1=function(data1){
  res=lm(Price~Area, data=data1) #fit linear model
  #predict values for all Area values from 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) #compute confidence bands
fit=lm(Price~Area, data=data2)
priceP=predict(fit)
plot(Area, Price, pch=21, bg="orange")
points(data2$Area,priceP,type="l") #plot fitted line
#plot confidence bands
points(data2$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

```
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 –  $X \sim Y$
- 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