Paket

library('kknn') – function kknn för k-nearest neighbor
library('glmnet') – funktion för ridge(alpha=0) och LASSO(alpha=1)
library('MASS') – för linear discriminant analysis (LDA)
library(tree) – för göra träd – även Rpart
library(e1071) – för naiveBayes
library(fastICA) – independent Component analysis (ICA) – function fastICA
library(geosphere) – disthaversine – räkna avstånd på Jorden
library(neuralnet) – neurala nätverk
library(pls) – för principal component regression (function pcr)

Bra funktioner:

```
missclass_rate = function(title ,v1, v2) {
 t_table = table(v1, v2)
 missclass = 1-sum(diag(t_table))/sum(t_table)
 print(paste0("missclassification rate for ", title, ": ", missclass))
}

conf_matrix = function(title, true, predicted) {
 t_table = table(true, predicted)
 print(title)
 print(t_table)
}
mean_square_error = function(v1, v2) {
 SE = (v1-v2)
 SE = SE^2
 SE = mean(SE)
 return(SE)
}
kernel_gauss = function(diff, h_val) {
 U = diff / h_val
 U = U^2
 return(exp(-U))
}

kernal_plot = function(diff, h_val) {
 kernal = kernel_gauss(diff, h_val)
 plot(kernal, type="l")
}
kernal_plot(seq(0,250000,1), h_distance)

mod = function(Lside, Rside) {
 returnValue = Lside - Rside * floor(Lside/Rside)
 return(returnValue)
}
```

```
 # Filter out all dates after the requested date
 sts = sts[as.Date(sts$date) < as.Date(date_in),]
 # Converts the time column from string to time, makes comparison easier later
 sts$time = strptime(sts$time, format = "%H:%M:%S")
```

```
# Remove all saved variables and plots
dev.off()
rm(list=ls())
# Setting up rng and seed
RNGversion('3.5.1')
set.seed(12345)


# Vector av nollor
v1 = rep(0, antal)

#Random
Var <- runif(50, 0, 10) # Fördelat
rand_obs = rexp(50, rate = 1.13) #??

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

Lab 1

```
# ------------------ Assignment 1 ----------------------
# Remove all saved variables and plots
dev.off()
rm(list=ls())
# Setting up rng and seed
RNGversion('3.5.1')
set.seed(12345)
```

The data file spambase.xlsx contains information about the frequency of various words, characters etc for a total of 2740 e-mails. Furthermore, these e-mails have been manually classified as spams (spam = 1) or regular e-mails (spam = 0).

1. Import the data into R and divide it into training and test sets (50%/50%) by using the following code:

```
Dataframe=read.csv2("spambase.csv")
n=dim(Dataframe)[1]
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 principle:

Y = 1 if (Y= 1|X)>0.5 otherwise Y=0

```
#Functions
# Predict and Classify the data
predict_with_classif = function(model_in, data_in, classif_val) {
  prediction = predict(model_in, newdata = data_in)
  classif = ifelse(prediction > classif_val, 1, 0)
}
# Create and print confunsion matrix
conf_matrix = function(title, true, predicted) {
  t_table = table(true, predicted)
  print(title)
  print(t_table)
}
# Calculate and print missclassification rate
missclass_rate = function(title ,v1, v2) {
  t_table = table(v1, v2)
  missclass = 1-sum(diag(t_table))/sum(t_table)
  print(paste0("missclassification rate for ", title, ": ", missclass))
}
task2n3 = function(model_in, title, data_in, classif_val) {
  prediction = predict_with_classif(model_in, data_in, classif_val)
  conf_matrix(title, data_in$Spam, prediction)
  missclass_rate(title, data_in$Spam, prediction)
}
model_glm = glm(Spam~., data = train)
task2n3(model_glm, "Training data", train, 0.5)
task2n3(model_glm, "Test data", test, 0.5)
```

3.Use logistic regression to classify the test data by the classification principle:
Y = 1 if (Y= 1|X)>0.8 otherwise Y=0
and report the confusion matrices (use table()) and the misclassification rates for training and test data. Compare the results. What effect did the new rule have?

```
task2n3(model_glm, "Training data", train, 0.8)
task2n3(model_glm, "Test data", test, 0.8)
```

Answer: Due to the new rule almost nothing gets classified as not spam. The missclassification rate is higher as result

4. Use standard classifier kknn() with K=30 from package kknn, report the the misclassification rates for the training and test data and compare the results with step 2.

```
library('kknn')
step4n5 = function(title, data_train, data_test, threshold) {
  prediction = kknn(Spam ~., train = data_train, test=data_test, k = threshold)
  prediction = ifelse(prediction$fitted.values > 0.5, 1, 0)
  conf_matrix(title, data_test$Spam, prediction)
  missclass_rate(title, data_test$Spam, prediction)
}
step4n5("Training data", train, train, 30)
step4n5("Test data", train, test, 30)
```

5. Repeat step 4 for K=1 and compare the results with step 4. What effect does the decrease of K lead to and why?

```
step4n5("Training data", train, train, 1)
step4n5("Test data", train, test, 1)


# ------------------ Assignment 2 --------------------------------
# Remove all saved variables and plots
dev.off()
rm(list=ls())
# Setting up rng and seed
RNGversion('3.5.1')
set.seed(12345)

The data file machines.xlsx contains information about the lifetime of
certain machines, and the company is interested to know more about the
underlying process in order to determine the warranty time. The variable is following:
        Length: shows lifetime of a machine

1. Import the data to R.
Dataframe=read.csv2("machines.csv")

2. Assume the probability model p(x|theta)=theta*e^(-theta*x)
for x=Length in which observations are independent and identically distributed.
What is the distribution type of x? Write a function that computes the log-likelihood
logp(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 the plot?

loglike = function(x, theta) {
 n = length(x)
 return (n*log(theta)-theta*sum(x))
}

step2n3 = function(data_in, thetas) {
 LLH = vector("numeric", length =0)
 for(theta in thetas) {
  LLH = c(LLH, loglike(data_in, theta))
 }
 print(thetas[which.max(LLH)])
 return(LLH)
}

step2 = function(data_in, thetas) {
 LLH = step2n3(data_in$Length, thetas)
 plot(thetas, LLH, type="l",col="blue")
}
thetas = seq(0, 30, 0.01)
step2(Dataframe, thetas)
```

3. Repeat step 2 but use only 6 first observations from the data, and put the two log-likelihood curves (from step 2 and 3) in the same plot. What can you say about reliability of the maximum likelihood solution in each case?

```
step3 = function(data_in, thetas) {
  LLH = step2n3(data_in, thetas)
  points(thetas, LLH, type="l",col="red")
}
step3(Dataframe[0:6,], thetas)
```

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)=logp(x|theta)p(theta). What kind of measure is actually computed by this function? Plot the curve showing the dependence of l(theta) on theta computed using the entire data and overlay it with a plot from step 2. Find an optimal theta and compare your result with the previous findings.

```
bayes = function(x, theta) {
  lambda = 10
  log_prior = log(lambda)-lambda*theta
  return(log_prior+loglike(x,theta))
}

LLH = vector("numeric", length=0)
for(theta in thetas) {
  LLH = c(LLH, bayes(Dataframe$Length, theta))
}

points(thetas, LLH, type="l",col="green")
```

5. Use theta value found in step 2 and generate 50 new observations from p(x|theta)=theta*e^(-theta*x) (use standard random number generators). Create the histograms of the original and the new data and make conclusions.

```
rand_obs = rexp(50, rate = 1.13)
hist(rand_obs,
    col=rgb(1,0,0,0.5),
    xlim=c(0,6),
    ylim=c(0,30),
    main="Overlapping Histogram",
    xlab="Variable")

hist(Dataframe$Length,
    col=rgb(0,0,1,0.5),
    add=T)
box()
```

---------- Assignment 3 ----------

```
# Remove all saved variables and plots
```

```
dev.off()
rm(list=ls())
# Setting up rng and seed
RNGversion('3.5.1')
set.seed(12345)
```
Implement an R function that performs feature selection (best subset selection) in linear regression by using k-fold cross-validation without using any specialized function like lm() (use only basic R functions). Your function should depend on:

X: matrix containing X measurements.
Y: vector containing Y measurements
Nfolds: number of folds in the cross-validation.

You may assume in your code that matrix X has 5 columns. The function should plot the CV scores computed for various feature subsets against the number of features, and it should also return the optimal subset of features and the corresponding cross-validation (CV) score. Before splitting into folds, the data should be permuted, and the seed 12345 should be used for that purpose. Report the resulting plot and interpret it. Report the optimal subset of features and comment whether it is reasonable that these specific features have largest impact on the target.

```
mylin=function(X,Y, Xpred){
 Xpred1 = cbind(1,Xpred)
 X1 = cbind(1, X)
 beta = solve( t(X1) %*% X1) %*% t(X1) %*% Y # obtained using the "training" data matrix X
 Res = Xpred1%*%beta # y_hat for the "test" data
 return(Res)
}
myCV=function(X,Y,Nfolds){
 n=length(Y) # number of observations (rows)
 p=ncol(X) # number of covariates (variables or columns)
 set.seed(12345)
 ind=sample(n,n) # indexes are randomized
 X1=X[ind,] # randomize the order of the observations
 Y1=Y[ind]
 sF=floor(n/Nfolds) # number of observations inside each fold
 MSE=numeric(2^p-1) # vector of the length of 2^p-1 combinations
 Nfeat=numeric(2^p-1)
 Features=list() # features that will be selected
 curr=0 # current
 folds_obs <- cut(1:n,breaks=Nfolds,labels=FALSE)
 #we assume 5 features.
 for (f1 in 0:1)
  for (f2 in 0:1)
   for(f3 in 0:1)
    for(f4 in 0:1)
     for(f5 in 0:1){
      model= c(f1,f2,f3,f4,f5)
      if (sum(model)==0) next()
      SSE=0
      for (k in 1:Nfolds){
       #MISSING:compute which indices should belong to current fold
```

```
      indices <- ind[which(folds_obs==k)] #indeces of the observations in fold k
      #MISSING:implement cross-validation for model with features in "model" and iteration i.
      X_mylin <- X[-indices,which(model==1)]
      XPred_mylin <- X[indices,which(model==1)]
      Y_mylin <- Y[-indices]
      #MISSING:Get the predicted values for fold k, Ypred, and the original values for fold 'k', Yp.
      Ypred <- mylin(X_mylin, Y_mylin, XPred_mylin)
      Yp <- Y[indices]
      SSE=SSE+sum((Ypred-Yp)^2)
    }
    curr=curr+1
    MSE[curr]=SSE/n
    Nfeat[curr]=sum(model)
    Features[[curr]]=model
  }
 plot(Nfeat, MSE) #MISSING: plot MSE against number of features
 abline(h=MSE[which.min(MSE)], col="red")
 i=which.min(MSE)
 return(list(CV=MSE[i], Features=Features[[i]]))
}
```

2.Test your function on data set swiss available in the standard R repository:
  Fertility should be Y
  All other variables should be X
  Nfolds should be 5

```
data("swiss")
myCV(as.matrix(swiss[,2:6]), swiss[[1]], 5)
```

---------- Assignment 4 ----------

```
# Remove all saved variables and plots
dev.off()
rm(list=ls())
# Setting up rng and seed
RNGversion('3.5.1')
set.seed(12345)
```

The Excel file tecator.xlsx contains the results of study aimed to investigate whether a near infrared absorbance spectrum can be used to predict the fat content of samples of meat. For each meat sample the data consists of a 100 channel spectrum of absorbance records and the levels of moisture (water), fat and protein. The absorbance is -log10 of the transmittance measured by the spectrometer.
The moisture, fat and protein are determined by analytic chemistry.

1. Import data to R and create a plot of Moisture versus Protein. Do you think that these data are described well by a linear model?

```
Dataframe=read.csv2("tecator.csv")
plot(Dataframe$Moisture, Dataframe$Protein)
```

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 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?

```
# M_i = a*i * protein^i + a*(i-1) * protein^(i-1) + ... + a * protein
```

3. Divide the data into training and validation sets(50%/50%) and fit models M_i, i=1.6. For each model, record the training and the validation MSE and present a plot showing how training and validation MSE depend on i (write some R code to make this plot). Which model is best according to the plot? How do the MSE values change and why? Interpret this picture in terms of bias-variance tradeoff.

```
mean_square_error = function(v1, v2) {
 SE = (v1-v2)
 SE = SE^2
 SE = mean(SE)
 return(SE)
}

calc_MSE = function(the_model, data) {
 SE = predict(the_model, newdata = data)
 SE = data$Moisture - SE
 SE = SE^2
 MSE = mean(SE)
 return(MSE)
}

n=dim(Dataframe)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=Dataframe[id,]
test=Dataframe[-id,]
test = test[1:dim(train)[1],]
MSEs = vector("numeric", length=0)

step3 = function(data_in, data_train) {
 for(i in seq(1,6)) {
  model_i = lm(Moisture ~poly(Protein,i), data = data_train)
  prediction = predict(model_i, newdata = data_in)
  MSEs = c(MSEs, mean_square_error(prediction, data_in$Moisture))
 }
 return (MSEs)
}
```

```
MSE_train = step3(train, train)
MSE_test = step3(test, train)

plot(seq(1,6), MSE_train, type="l", col="blue", ylim=c(20,45))
lines(seq(1,6), MSE_test, type="l", col="red")
```

4. Perform variable selection of a linear model in which Fat is response and Channel1-Channel100 are predictors by using stepAIC. Comment on how many variables were selected.

```
t_data = Dataframe[,0:(dim(Dataframe)[2]-2)]
model_lm = lm(Fat ~., data=t_data)
AIC_lm = stepAIC(model_lm)
print(paste0(length(AIC_lm$coefficients), " were selected"))
```

5. Fit a Ridge regression model with the same predictor and response variables. Present a plot showing how model coefficients depend on the log of the penalty factor lambda and report how the coefficients change with lambda.

```
library('glmnet')

step5n6 = function(data_in, alpha_in) {
  t_model = glmnet(as.matrix(data_in[,2:101]), data_in[,102], alpha=alpha_in, family="gaussian")
  plot(t_model, xvar="lambda", label=TRUE)
}

step5n6(t_data, 0)
```

6. Repeat step 5 but fit LASSO instead of the Ridge regression and compare the plots from steps 5 and 6. Conclusions?

```
step5n6(t_data, 1)
```

7. Use cross-validation to find the optimal LASSO model (make sure that case lambda=0 is also considered by the procedure), report the optimal lambda and how many variables were 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.

```
lambdas = seq(0,50, 0.01)
cv_lasso = cv.glmnet(as.matrix(t_data[,2:101]), t_data[,102], alpha=1, family="gaussian",
lambda=lambdas)
plot( cv_lasso, type="b", col="Green")
```

Lab 2

---------- Assignment 1 ----------

```
# Remove all saved variables and plots
dev.off()
rm(list=ls())
# Setting up rng and seed
RNGversion('3.5.1')
set.seed(12345)
```

The data file australian-crabs.csv contains measurements of various crabs, such as Frontal lobe, Rear width and others

```
Dataframe=read.csv2("australian-crabs.csv")
```

1. Use australian-crabs.csv and make a scatterplot of carapace length (CL) versus rear width (RW) where observations are colored by Sex. Do you think that this data is easy to classify by linear discriminant analysis? Motivate your answer.

```
plot(Dataframe$CL, Dataframe$RW, col=Dataframe$sex)
```

2. Make 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.

```
missclass_rate = function(title ,v1, v2) {
  t_table = table(v1, v2)
  missclass = 1-sum(diag(t_table))/sum(t_table)
  print(paste0("missclassification rate for ", title, ": ", missclass))
}

step2n3 = function(t_model) {
  predict_lda = predict(model_lda, newdata = Dataframe)
  plot(Dataframe$CL, Dataframe$RW, col=predict_lda$class)
  missclass_rate("lda", predict_lda$class, Dataframe$sex)
}

library('MASS')
model_lda = lda(sex ~ RW+CL, data = Dataframe)
step2n3(model_lda)
```

3. Repeat step 2 but use priors p(Male)=0.9,p(Female)=0.1 instead. How did the classification result change and why?

```
model_lda = lda(sex ~ RW+CL, data = Dataframe, prior = c(0.1,0.9))
step2n3(model_lda)
```

4. Make a similar kind of classification by logistic regression (use function glm()), plot the classified data and compute the misclassification error. Compare these results with the LDA results. Finally, report the equation of the decision boundary and draw the decision boundary in the plot of the classified data.

```
model_glm = glm(sex ~ RW+CL, data = Dataframe, family =  "binomial")
predict_glm = predict(model_glm, newdata = Dataframe)
plot(predict_glm, type = "l")
predict_glm = ifelse(predict_glm > 0.5, "Male", "Female")
missclass_rate("GLM", predict_glm, Dataframe$sex)
plot(Dataframe$CL, Dataframe$RW, col=as.factor(predict_glm))
slope <- coef(model_glm)[2]/(-coef(model_glm)[3])
intercept <- coef(model_glm)[1]/(-coef(model_glm)[3])
print(paste0("Equation: ", intercept, " + ", slope, "x"))
abline(intercept , slope)
```

---------- Assignment 2 ----------

```
# Remove all saved variables and plots
dev.off()
rm(list=ls())
# Setting up rng and seed
RNGversion('3.5.1')
set.seed(12345)
```

The data file creditscoring.xls contains data retrieved from a database in a private enterprise. Each row contains information about one customer. The variable good/bad indicates how the customers have managed their loans. The other features are potential predictors. Your task is to derive a prediction model that can be used to predict whether or not a new customer is likely to pay back the loan.

1. Import the data to R and divide into training/validation/test as 50/25/25: use data partitioning code specified in Lecture 1e.

```
Dataframe=read.csv2("creditscoring.csv")
n=dim(Dataframe)[1]
id=sample(1:n, floor(n*0.4))
train=Dataframe[id,]
id1=setdiff(1:n, id)
id2=sample(id1, floor(n*0.3))
valid=Dataframe[id2,]
id3=setdiff(id1,id2)
test=Dataframe[id3,]
```

2. Fit a decision tree to the training data by using the following measures of impurity
   a. Deviance
   b. Gini index

and report the misclassification rates for the training and test data. Choose the measure providing the better results for the following steps.

```
library(tree)
# Calculate and print missclassification rate
missclass_rate = function(title ,v1, v2) {
 t_table = table(v1, v2)
 missclass = 1-sum(diag(t_table))/sum(t_table)
 print(paste0("missclassification rate for ", title, ": ", missclass))
}

class_and_value = function(title, t_tree, data_in, threshold) {
 classification = predict(t_tree, newdata = data_in)
 prediction = ifelse(classification[,2] > threshold, "good", "bad")
 missclass_rate(title, prediction, data_in$good_bad)
}
```

```
step2 = function(impurity) {
  t_tree = tree(good_bad ~ ., data = train, split = impurity)
  for(d in list(train, test)) {
    class_and_value(impurity, t_tree, d, 0.5)
  }
}

for(imp in c("deviance", "gini")) {
  step2(imp)
}
# Deviance is the better choice
t_tree = tree(good_bad ~ ., data = train, split = "deviance")
```

3. Use training and validation sets to choose the optimal tree depth. Present the graphs of the dependence of deviances for the training and the validation data on the number of leaves. Report the optimal tree, report it's depth and the variables used by the tree. Interpret the information provided by the tree structure. Estimate the misclassification rate for the test data.

```
calcScore = function(t_tree, data_in) {
  prediction = predict(t_tree, newdata = data_in, type="tree")
  return(deviance(prediction))
}

score_train = rep(0,20)
score_valid = rep(0,20)

for( i in 2:20) {
  pruned_tree = prune.tree(t_tree, best = i)
  score_train[i] = calcScore(pruned_tree, train)
  score_valid[i] = calcScore(pruned_tree, valid)
}

plot(score_train, type="l", col="blue", xlim = c(2,20))
plot(score_valid, type="l", col="red", xlim = c(2,20))

t_tree = prune.tree(t_tree, best = 4)
plot(t_tree)
text(t_tree, pretty=0)
# Depth is 3, variables used are: duration, history and savings
prediction = predict(t_tree, newdata = test)
prediction = ifelse(prediction[,2] > 0.5, "good", "bad")
missclass_rate("pruned tree", prediction, test$good_bad)
```

4.Use training data to perform classification using Naïve Bayes and report the confusion matrices and misclassification rates for the training and for the test data. Compare the results with those from step 3.

```
library(e1071)
navbay = naiveBayes(good_bad ~ ., data = train)
for(d in list(train, test)) {
  prediction = predict(navbay, newdata = d)
```

```
 missclass_rate("naive bayes", prediction, d$good_bad)
}
```

5. Use the optimal tree and the Naïve Bayes model to classify the test data by using the following principle: Y=1 if p(Y="good"|x)>pi otherwise Y=0, where pi = 0.05, 0.1, ..., 0.9, 0.95. Compute the TPR and FPR values for the two models and plot the corresponding ROC curves. Conclusion?

```
conf_matrix = function(true, predicted) {
 t_table = table(true, predicted)
 return(t_table)
}

step5 = function(pi,model_in, d, pred_type) {
 TPR = rep(1:length(pi))
 FPR = rep(1:length(pi))
 for(i in 1:length(pi)) {
  prediction = predict(model_in, newdata = d, type=pred_type)
  prediction = ifelse(prediction[,2] > pi[i], "good", "bad")
  confusion = conf_matrix(d$good_bad, prediction)
  TPR[i] = confusion[1,1]/sum(confusion[1,])
  FPR[i] = confusion[2,1]/sum(confusion[2,])
 }
 return(c(TPR, FPR))
}

pi = seq(0.05,0.95,0.05)
vals_navbay = step5(pi, navbay, train, "raw")
vals_tree = step5(pi, t_tree, train, "vector")
plot(vals_navbay[1:19], vals_navbay[20:38], type = "l", col="blue")
plot(vals_tree[1:19], vals_tree[20:38], type = "l", col="blue")
```

6.Repeat Naïve Bayes classification as it was in step 4 but use the following loss matrix: L = (0,1; 10,0) and report the confusion matrix for the training and test data. Compare the results with the results from step 4 and discuss how the rates has changed and why.

```
for(d in list(train, test)) {
 prediction = predict(navbay, newdata = d, type = "raw")
 classified = rep(1:dim(prediction)[1])
 for(i in 1:dim(prediction)[1]) {
  classified[i] = ifelse(prediction[i,1]*10 < prediction[i,2]*1, "good", "bad")
 }
 print(conf_matrix(d$good_bad, classified))
}
```

---------- Assignment 4 ----------

```
# Remove all saved variables and plots
dev.off()
rm(list=ls())
# Setting up rng and seed
RNGversion('3.5.1')
set.seed(12345)
```

The data file NIRspectra.csv contains near-infrared spectra and viscosity levels for a collection of diesel fuels. Your task is to investigate how the measured spectra can be used to predict the viscosity.

```
Dataframe=read.csv2("NIRSpectra.csv")
```

1. Conduct a standard PCA by using the feature space and provide a plot explaining how much variation is explained by each feature. Does the plot show how many PC should be extracted? Select the minimal number of components explaining at least 99% of the total variance. Provide also a plot of the scores in the coordinates (PC1, PC2). Are there unusual diesel fuels according to this plot?

```
#PCA is unsupervised learning
Dataframe$Viscosity = c()
pca = prcomp(Dataframe)
lambda = pca$sdev^2
plot(pca)
sprintf("%2.3f",lambda/sum(lambda)*100)
# We see from the line above that the first 2 components cover 99%
plot(pca$x[,1], pca$x[,2], ylim=c(-0.15,0.15))
```

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 = pca$rotation
plot(U[,1], main="PC1")
plot(U[,2], main="PC2")
```

3. Perform Independent Component Analysis with the number of components selected in step 1 (set seed 12345). Check the documentation for the fastICA method in R and do the following:

a. Compute W' = K*W and present the columns of W' in form of the trace plots. Compare with the trace plots in step 2 and make conclusions. What kind of measure is represented by the matrix W'?

b. Make a plot of the scores of the first two latent features and compare it with the score plot from step 1.

```
library(fastICA)
fica = fastICA(Dataframe, 2)
# Use %*% when you want to use matrix operation
Wtick = fica$K %*% fica$W
plot(Wtick[,1], main="PC1")
plot(Wtick[,2], main="PC2")
```

Lab 3

```
------------------ Assignment 1 ---------------------------------
# Remove all saved variables and plots
dev.off()
rm(list=ls())
# Setting up rng and seed
RNGversion('3.5.1')
set.seed(12345)
```

Implement a kernel method to predict the hourly temperatures for a date and place in Sweden. To do so, you are provided with the files stations.csv and temps50k.csv. These files contain information about weather stations and temperature measurements in the stations at different days and times.

The data have been kindly provided by the Swedish Meteorological and Hydrological Institute (SMHI).You are asked to provide a temperature forecast for a date and place in Sweden. The forecast should consist of the predicted temperatures from 4 am to 24 pm in an interval of 2 hours. Use a kernel that is the sum of three Gaussian kernels:

The first to account for the distance from a station to the point of interest.

The second to account for the distance between the day a temperature measurement was made and the day of interest.

The third to account for the distance between the hour of the day a temperature measurement was made and the hour of interest.

Choose an appropriate smoothing coefficient or width for each of the three kernels above. Answer to the following questions:

Show that your choice for the kernels' width is sensible, i.e. that it gives more weight to closer points. Discuss why your of definition of closeness is reasonable.

Instead of combining the three kernels into one by summing them up, multiply them. Compare the results obtained in both cases and elaborate on why they may differ. Note that the file temps50k.csv may contain temperature measurements that are posterior to the day and hour of your forecast. You must filter such measurements out, i.e. they cannot be used to compute the forecast. Feel free to use the template below to solve the assignment.

```
library(geosphere)
stations <- read.csv("stations.csv")
temps <- read.csv("temps50k.csv")
st <- merge(stations,temps,by="station_number")

# Function to calculate the kernel value
kernel_gauss = function(diff, h_val) {
  U = diff / h_val
  U = U^2
  return(exp(-U))
}

kernal_plot = function(diff, h_val) {
  kernal = kernel_gauss(diff, h_val)
  plot(kernal, type="l")
}
# These values are decided by looking at the plots
h_distance <- 75000
h_date <-14
h_time <-4
kernal_plot(distance_diff <- seq(0,250000,1), h_distance)
kernal_plot(seq(0,30,1), h_date)
kernal_plot(seq(0,24,0.5), h_time)

# Function to calculate mod, required for days
mod = function(Lside, Rside) {
  returnValue = Lside - Rside * floor(Lside/Rside)
  return(returnValue)
}

# Functions to calculate the differences
calc_distance = function(input, sts) {
```

```
  dist = distHaversine(data.frame(sts$longitude,sts$latitude), input)
  return(dist)
}

# Take into consideration the difference in days  between years
# Two days ahead five years ago should give differnce 2 days
# Also that it allows days backwards from 25/02 to 24/02 is 1 day, not 364
calc_day = function(input, sts) {
  days = difftime(sts$date, input)
  days = mod(days, 365)
  days = ifelse(days > 365/2, (365 - days), days)
  return(days)
}
calc_time = function(input, sts) {
  time = as.numeric(difftime(input, sts$time, units = "secs"))
  time = time /3600
  time = ifelse(abs(time) > 12, (24 -abs(time)), abs(time))
  return(time)
}

# Main function
predict_temp = function(lat_in, long_in, date_in, times_in,sts) {
  # Filter out all dates after the requested date
  sts = sts[as.Date(sts$date) < as.Date(date_in),]
  # Converts the time column from string to time, makes comparison easier later
  sts$time = strptime(sts$time, format = "%H:%M:%S")
  times_in = strptime(times_in, format = "%H:%M:%S")
  date_in = as.Date(date_in)
  # Calculating the difference and kernel values for distance and date
  distance = calc_distance(c(long_in, lat_in),sts)
  day = calc_day(date_in, sts)
  kernel_distance = kernel_gauss(distance, h_distance)
  kernel_day = kernel_gauss(day, h_date)
  temps_sum = rep(0,length(times_in))
  temps_mul = rep(0,length(times_in))

   for(i in 1:length(times_in)) {
     # Calculating the difference and kernel values for time
     time = calc_time(times_in[i], sts)
     kernel_time = kernel_gauss(time, h_time)
     # Summing/ multiplying all kernel value and calculate the predicted temperature
     k_sum = kernel_distance+kernel_day+kernel_time
     k_mul = kernel_distance*kernel_day*kernel_time
     temps_sum[i] = sts$air_temperature%*% k_sum / sum(k_sum)
     temps_mul[i] = sts$air_temperature%*% k_mul /sum(k_mul)
   }
  #Plot and print result
  plot(temps_sum, type="o")
  print("With summerizing:")
```

```
  print(temps_sum)
  plot(temps_mul, type="o")
  print("With multiplying:")
  print(temps_mul)
}

# Inputs from user
lat <- 58.4274
long <- 14.826
date <- "2013-11-04"
times <- 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")
# Give predictions
predict_temp(lat, long, date, times, st)

------------------ Assignment 2 --------------------------------
# Remove all saved variables and plots
dev.off()
rm(list=ls())
# Setting up rng and seed
RNGversion('3.5.1')
set.seed(12345)
```

Use the function ksvm from the R package kernlab to learn a SVM for classifying the spam dataset that is included with the package. Consider the radial basis function kernel (also known as Gaussian) with a width of 0.05. For the C parameter, consider values 0.5, 1 and 5. This implies that you have to consider three models.

- Perform model selection, i.e. select the most promising of the three models (use any method of your choice except cross-validation or nested cross-validation).
- Estimate the generalization error of the SVM selected above (use any method of your choice except cross-validation or nested cross-validation).
- Produce the SVM that will be returned to the user, i.e. show the code.
- What is the purpose of the parameter C ?

```
data(spam)
data <- spam
n=dim(data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.70))
train=data[id,]
test=data[-id,]
ksvm.model <- function(x) {
  model <- ksvm(type~., data = train, kernel = "rbfdot", kpar=list(sigma=0.05), C = x)
  pred.model <- predict(model, test)
  confmat <- table(test$type, pred.model)
  misclass <- 1 - sum(diag(confmat)/sum(confmat))
  cat("\nModel misclassification rate for C = ", x, "is: ", misclass)
  confmat
}
ksvm.model(0.5)
```

```
ksvm.model(1)
ksvm.model(5)
model.new <- ksvm(type~., data = train, kernel = "rbfdot", kpar=list(sigma=0.05), C = 1)
model.new
```

---------- Assignment 3 ----------

```
# Remove all saved variables and plots
dev.off()
rm(list=ls())
# Setting up rng and seed
RNGversion('3.5.1')
set.seed(12345)
```

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, i = 1, ..., 10. Initialize the weights of the neural network to random values in the interval [-1:1]. Use a neural network with a single hidden layer of 10 units. Use the default values for the arguments not mentioned here. Choose the most appropriate value for the threshold. Motivate your choice. Provide the final neural network learned with the chosen threshold. Feel free to use the following template.

```
mean_square_error = function(v1, v2) {
 SE = (v1-v2)
 SE = SE^2
 SE = mean(SE)
 return(SE)
}

# Function to predict and return the MSE
# For NN use compute
pred_MSE = function(model_in, data_in) {
 prediction = compute(model_in, covariate=data_in)$net.result
 return(mean_square_error(prediction, data_in$Sin))
}
library(neuralnet)
Var <- runif(50, 0, 10)
trva <- data.frame(Var, Sin=sin(Var))
tr <- trva[1:25,] # Training
va <- trva[26:50,] # Validation
#Hidden = 10 from assignment
hidden_val = 10
i_vals = seq(1,10)
# Random initialization of the weights in the interval [-1, 1]
# Number of starting weights is number of connections in the plot of the nn
winit = runif(31, -1, 1)
MSE_tr = rep(0, length(i_vals))
```

```
MSE_va = rep(0, length(i_vals))
 for(i in i_vals) {
   nn <- neuralnet(Sin ~ Var, data = tr, hidden = hidden_val, startweights = winit, threshold = i/1000)
   MSE_tr[i] = MSE_tr[i] + pred_MSE(nn, tr)
   MSE_va[i] = MSE_va[i] + pred_MSE(nn, va)
 }
plot(MSE_tr, type= "l")
plot(MSE_va, type= "l")
# Lowest MSE for i = 5
nn <- neuralnet(Sin ~ Var, data = tr, hidden = hidden_val, startweights = winit, threshold = 5/1000)
plot(nn)
# Plot of the predictions (black dots) and the data (red dots)
plot(prediction(nn)$rep1)
points(trva, col = "red")
```

Tentauppgifter

The data file video.csv contains characteristics of a sample of Youtube videos. Import data to R and divide it randomly (50/50) into training and test sets.

1. Perform principal component analysis using the numeric variables in the training data except of "utime" variable. Do this analysis with and without scaling of the features. How many components are necessary to explain more than 95% variation of the data in both cases? Explain why so few components are needed when scaling is not done.

```
data0=read.csv("video.csv")
data1=data0
data1$codec=c()
n=dim(data1)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=data1[id,]
test=data1[-id,]
data11=data1
data11$utime=c()
res=prcomp(data11)
lambda=res$sdev^2
sprintf("%2.3f",cumsum(lambda)/sum(lambda)*100)
res=prcomp(scale(data11))
lambda=res$sdev^2
sprintf("%2.3f",cumsum(lambda)/sum(lambda)*100)
```

2. Write a code that fits a principle component regression ("utime" as response and all scaled numerical variables as features) with MM components to the training data and estimates the training and test errors, do this for all feasible MM values. Plot dependence of the training and test errors on MM and explain this plot in terms of bias-variance tradeoff. (Hint: prediction function for principal component regression has some peculiarities, see predict.mvr)

```
library(pls)
```

```
trE=numeric(17)
testE=numeric(17)
for (i in 1:17){
 pcrN=pcr(utime~., 17, data=train,  scale=T)
 Yf=predict(pcrN, ncomp=i)
 Yt=predict(pcrN, newdata=test, ncomp=i)
 trE[i]=mean((train$utime-Yf)^2)
 testE[i]=mean((test$utime-Yt)^2)
}
plot(testE, type="l", col="red", ylim=c(100,300), ylab="Error")
points(trE, type="l", col="blue")
```

3.Use PCR model with MM=8 and report a fitted probabilistic model that shows the connection between the target and the principal components.

```
pcrF=pcr(utime~., 8, data=train, validation="none", scale=T)
mean(residuals(pcrF)^2)
Yloadings(pcrF)
```

4. Use original data to create variable “class” that shows “mpeg” if variable “codec” is equal to “mpeg4”, and “other” for all other values of “codec”. Create a plot of “duration” versus “frames” where cases are colored by “class”. Do you think that the classes are easily separable by a linear decision boundary?

```
data2=data0
data2$class=ifelse(data2$codec=="mpeg4", "mpeg4", "other")
data2$codec=c()
plot(data2$frames,data2$duration, col=as.factor(data2$class), cex=0.5, xlab="frames",
ylab="duration")
data2$frames=scale(data2$frames)
data2$duration=scale(data2$duration)
```

5. 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.

```
library(MASS)
m3=lda(as.factor(class)~frames+duration, data=data2)
plot(data2$frames,data2$duration, col=predict(m3)$class,  cex=0.5, xlab="frames", ylab="duration")
missclass=function(X,X1){
 n=length(X)
 return(1-sum(diag(table(X,X1)))/n)
}
missclass(data2$class, predict(m3, type="class")$class)
```

6. Fit a decision tree model with “class” as target and “frames” and “duration” as features to the entire dataset, choose an appropriate tree size by cross-validation. Report the training error. How many leaves are there in the final tree? Explain why such a complicated tree is needed to describe such a simple decision boundary.

```
library(tree)
m4=tree(as.factor(class)~frames+duration, data=data2)
set.seed(12345)
```

```
cv.res=cv.tree(m4)
plot(cv.res$size, cv.res$dev, type="b", col="red")
print(m4)
plot(m4)
missclass(data2$class, predict(m4, type="class"))
```

#----------

#You are asked to use the function ksvm from the R package kernlab to learn a support vector machine (SVM) for classifying the spam dataset that is included with the package. Consider the radial basis function kernel (also known as Gaussian) with a width of 0.05. For the C parameter, consider values 0.5, 1 and 5. This implies that you have to consider three models.

(2p) 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).

(1p) Estimate the generalization error of the SVM selected above (use any method of your choice except cross-validation or nested cross-validation).

(1p) Produce the SVM that will be returned to the user, i.e. show the code. (1p) What is the purpose of the parameter C ?

```
library(kernlab)
set.seed(1234567890)
data(spam)
# Model selection
index <- sample(1:4601)
tr <- spam[index[1:2500], ]
va <- spam[index[2501:3501], ]
te <- spam[index[3502:4601], ]
filter <- ksvm(type~.,data=tr,kernel="rbfdot",kpar=list(sigma=0.05),C=0.5)
mailtype <- predict(filter,va[,-58])
t <- table(mailtype,va[,58])
(t[1,2]+t[2,1])/sum(t)
filter <- ksvm(type~.,data=tr,kernel="rbfdot",kpar=list(sigma=0.05),C=1)
mailtype <- predict(filter,va[,-58])
t <- table(mailtype,va[,58])
(t[1,2]+t[2,1])/sum(t)
filter <- ksvm(type~.,data=tr,kernel="rbfdot",kpar=list(sigma=0.05),C=5)
mailtype <- predict(filter,va[,-58])
t <- table(mailtype,va[,58])
(t[1,2]+t[2,1])/sum(t)
# Error estimation
filter <- ksvm(type~.,data=spam[index[1:3501], ],kernel="rbfdot",kpar=list(sigma=0.05),C=1)
mailtype <- predict(filter,te[,-58])
t <- table(mailtype,te[,58])
(t[1,2]+t[2,1])/sum(t)
# Final model
filter <- ksvm(type~.,data=spam,kernel="rbfdot",kpar=list(sigma=0.05),C=1)
```

#----------

Train a neural network (NN) to learn the trigonometric sine function. To do so, sample 50 points uniformly at random in the interval [0, 10]. Apply the sine function to each point. The resulting pairs

are the data available to you. Use 25 of the 50 points for training and the rest for validation. The validation set is used for early stop of the gradient descent. Consider threshold values i/1000 with i = 1, . . . , 10. Initialize the weights of the neural network to random values in the interval [−1, 1]. Consider two NN architectures: A single hidden layer of 10 units, and two hidden layers with 3 units each. Choose the most appropriate NN architecture and threshold value. Motivate your choice. Feel free to reuse the code of the corresponding lab.
(1p) Estimate the generalization error of the NN selected above (use any method of your choice).
(1p) In the light of the results above, would you say that the more layers the better ? Motivate your answer.

```
library(neuralnet)
# two layers
set.seed(1234567890)
Var <- runif(50, 0, 10)
trva <- data.frame(Var, Sin=sin(Var))
tr <- trva[1:25,] # Training
va <- trva[26:50,] # Validation
# plot(trva)
# plot(tr)
# plot(va)
restr <- vector(length = 10)
resva <- vector(length = 10)
winit <- runif(22, -1, 1) # Random initializaiton of the weights in the interval [-1, 1]
for(i in 1:10) {
  nn <- neuralnet(formula = Sin ~ Var, data = tr, hidden = c(3,3), startweights = winit, threshold =
i/1000, lifesign = "full")
# nn$result.matrix
aux <- compute(nn, tr[,1])$net.result # Compute predictions for the trainig set and their squared
error
  restr[i] <- sum((tr[,2] - aux)**2)/2
  aux <- compute(nn, va[,1])$net.result # The same for the validation set
  resva[i] <- sum((va[,2] - aux)**2)/2
}
plot(restr, type = "o")
plot(resva, type = "o")
restr
resva

# one layer
set.seed(1234567890)
Var <- runif(50, 0, 10)
trva <- data.frame(Var, Sin=sin(Var))
tr <- trva[1:25,] # Training
va <- trva[26:50,] # Validation
# plot(trva)
# plot(tr)
# plot(va)
restr <- vector(length = 10)
resva <- vector(length = 10)
```

```
winit <- runif(41, -1, 1) # Random initializaiton of the weights in the interval [-1, 1]
for(i in 1:10) {
  nn <- neuralnet(formula = Sin ~ Var, data = tr, hidden = c(10), startweights = winit, threshold =
i/1000, lifesign = "full")
# nn$result.matrix
  aux <- compute(nn, tr[,1])$net.result # Compute predictions for the trainig set and their squared
error
  restr[i] <- sum((tr[,2] - aux)**2)/2
  aux <- compute(nn, va[,1])$net.result # The same for the validation set
  resva[i] <- sum((va[,2] - aux)**2)/2
}
plot(restr, type = "o")
plot(resva, type = "o")
restr
resva
# estimate generalization error for the best run above (one layer with threshold 4/1000)
Var <- runif(50, 0, 10)
te <- data.frame(Var, Sin=sin(Var))
winit <- runif(31, -1, 1)
nn <- neuralnet(formula = Sin ~ Var, data = trva, hidden = 10, startweights = winit,  threshold =
4/1000, lifesign = "full")
sum((te[,2] - compute(nn, te[,1])$net.result)**2)/2
```