Innehåll

Regression 2
Linear regression 2
Things 2
R commands 2
Pic rows/col 2
Packages used in lab 2
Distributions 2
Chi-square (två) distribution 2
Poisson distribution 3
Exponential distribution 3
Labs 4
Lab1 4
Assignment 1 - Spam classification with nearest neighbors 4
Assignment 2 - Inference about lifetime of machines 5
Assignment 3 - Feature selection by cross-validation in a linear model - Extra 7
Assignment 4 - Linear regression and regularization 8
Lab 2 11
Assignment 1 - LDA and logistic regression 11
Assignment 2 - Analysis of credit scoring 13
Assignment 3 - Uncertainty estimation - Extra 16
Assignment 4 - Principal components 19
Lab 3 20
Assignment 1 - KERNEL METHODS 20
Assignment 2 - SUPPORT VECTOR MACHINES - Extra 22
Assignment 3 - NEURAL NETWORKS 23

Regression

Linear regression

Estimation: maximum likelihood is equal to min square: $RSS(w) = \sum_{i=1}^{n} (Y_i - \boldsymbol{w}^T \boldsymbol{X}_i)^2$. Optimal condition: $X^T (y - Xw) = 0$

X has 1s as first col: cbind(1,X), y is one col. Gives estimation as: w_hat = $(X^T X)^{-1} * X^T y$.

R: w_hat/beta = solve(t(X)%*%X, t(X)%*%Y)

Package: lm().

Things

Residuals, difference between obs value vs model prediction. Use residuals() in R. Error.

R commands

Pic rows/col

X[1,6] X[1:9,] X[-(1:9),] Entire row 4: X[4,] Pic row, col 5>: X[X[,5]>20,]

Transpose: T(X)

Inverse: solve(X) $d=X^{-1}$ b = solve(X,b)

Packages used in lab

- readr
- glmnet
- kknn
- MASS
- tree or (rpart) I choose tree.
- e1071 - Functions for latent class analysis, short time Fourier transform, fuzzy clustering, support vector machines, shortest path computation, bagged clustering, naive Bayes classifier
- fastICA
- geosphere
- neuralnet

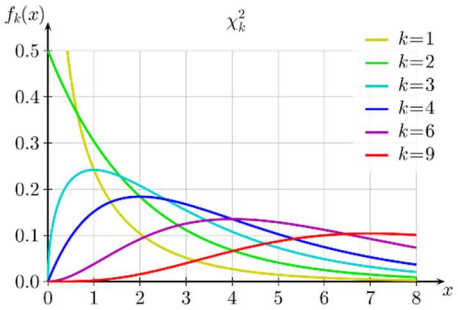
extra assign:

Distributions

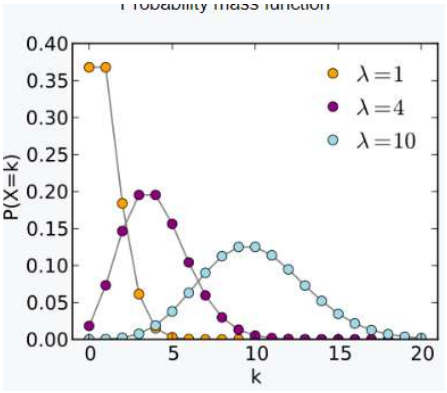
Chi-square (två) distribution

Chitvåfördelning alternativt Chikvadratfördelning, χ^2-fördelning, är inom matematisk statistik en kontinuerlig sannolikhetsfördelning med täthetsfunktionen:

$$\frac{1}{2^{k/2}\Gamma(k/2)} x^{k/2-1} e^{-x/2}$$

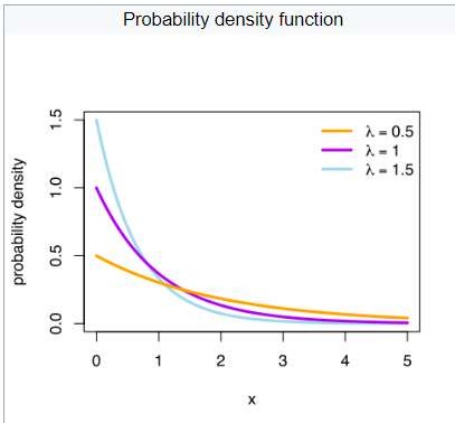


Poisson distribution



$$\frac{\lambda^k e^{-\lambda}}{k!}$$

Exponential distribution



$$\lambda e^{-\lambda x}$$

Labs

Lab1

Assignment 1 - Spam classification with nearest neighbors

```
library(readr)
library(glmnet)
data <- read_csv2("C:/Users/oskar/OneDrive/Universitet/Linköping Universitet/År4/Machine learning/Lab 1/spambase.csv")

n=dim(data)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=data[id,]
test=data[-id,]

predModel = glm(Spam~., data=train, family = binomial)

missclass=function(X, Xfit){
  n=length(X)
  print(table(X, Xfit))
  return (1-sum(diag(table(X, Xfit)))/n)
}

#Pred 1
prediction_test = predict(predModel, test, type="response")
prediction2_test = prediction_test
prediction_test[prediction_test<=0.5]=0
prediction_test[prediction_test>0.5]=1
plot(prediction_test)
missClassResult_test = missclass(test[[49]], prediction_test)
print(missClassResult_test)

prediction_train = predict(predModel, train, type="response")
prediction2_train = prediction_train
prediction_train[prediction_train<=0.5]=0
prediction_train[prediction_train>0.5]=1
plot(prediction_train)
missClassResult_train = missclass(train[[49]], prediction_train)
print(missClassResult_train)

#Pred 2
prediction2_train[prediction2_train<=0.8]=0
prediction2_train[prediction2_train>0.8]=1

prediction2_test[prediction2_test<=0.8]=0
prediction2_test[prediction2_test>0.8]=1
#plot(prediction2)

missClassResult_8_train = missclass(train[[49]], prediction2_train)
missClassResult_8_test = missclass(test[[49]], prediction2_test)
print(missClassResult_8_train)
print(missClassResult_8_test) #New rule makes more missclassifications.
```

```
#kknn
library(kknn)

knnn= kknn(as.factor(Spam)~., train, test, k=30)
missClassResultk30 = missclass(test[[49]], knnn$fitted.values)
knnn= kknn(as.factor(Spam)~., train, train, k=30)
missClassResultk30_train = missclass(train[[49]], knnn$fitted.values)

knnn1= kknn(as.factor(Spam)~., train, test, k=1)
missClassResultk1 = missclass(test[[49]], knnn1$fitted.values)
knnn1= kknn(as.factor(Spam)~., train, train, k=1)
missClassResultk1_train = missclass(train[[49]], knnn1$fitted.values)

print(missClassResultk30)
print(missClassResultk30_train)

print(missClassResultk1) #not as good
print(missClassResultk1_train)
```

Assignment 2 - Inference about lifetime of machines

```
machines<- read_csv2("C:/Users/oskar/OneDrive/Universitet/Linköping Universitet/År4/Machine learning/Lab 1/machines.csv")

hist(machines$Length)#exponential

#log-likelihood
log_likelihood = function(theta, data){
  p=0
  data = data$Length
  for(i in data){
    p = p + log( theta*exp(-theta*i))
  }
  return (p)
}

#find my Theta
findTheta = function(data){
  p = 1:100
  i=1
  for(theta in seq(from=0, to=10, by=0.1)){
    p[i]= log_likelihood(theta, data)
    i = i+1
  }
  theta = seq(from=0, to=10, by=0.1)
  plot(theta, p)
  return(p)
}

logPTotal = findTheta(machines)
```

```
logPSix = findTheta(machines[(1:6), ])

thetaTotal = seq(from=0, to=10, by=0.1)[which.max(logPTotal)]
thetaSixFirst = seq(from=0, to=10, by=0.1)[which.max(logPSix)]

print(thetaTotal)
print(thetaSixFirst)
theta = seq(from=0, to=10, by=0.1)
plot(theta, logPTotal, col="blue", ylim=c(-100,0))
par(new=TRUE)
plot(theta, logPSix, col="red", ylim=c(-100,0))

#whti is this?
#curve(dim(machines)[1]*log(x)-x*sum(machines), from=min(machines), ylim=c(-80,0), col="blue",  to=4,
ylab="log(p(x|??))", sub="Red: 6 obs | Blue: All obs", xlab="??", add=FALSE)

#part4
#log-likelihood Bayesian
#log_likelihood_Bayesian = function(theta,lambda, data){
#  p=0
#  data = data$Length
#  for(i in data){
#    p = p +  #log( theta*exp(-theta*i)*lambda*exp(-lambda*theta))
#  }
#  return (p)
#}

findThetaBayesian = function(lambda, data){
  p = 1:100
  i=1
  for(theta in seq(from=0, to=10, by=0.1)){
    log_like_lambda = log(lambda)+(-theta*lambda)
    p[i]= log_like_lambda + log_likelihood(theta, data)
    i = i+1
  }
  theta = seq(from=0, to=10, by=0.1)
  plot(theta, p)
  return(p)
}

theta = seq(from=0, to=10, by=0.1)
LogsThetaBay = findThetaBayesian(10, machines)
thetaMaxBay = seq(from=0, to=10, by=0.1)[which.max(LogsThetaBay)]

plot(theta, logPTotal, col="blue", ylim=c(-100,-30))
par(new=TRUE)
plot(theta, LogsThetaBay, ylim=c(-100,-30))
print(thetaMaxBay)



#5
set.seed(12345)
#thetaTotal = seq(from=0, to=10, by=0.1)[which.max(logPTotal)]
```

```
new_Data = rexp(50, rate=thetaTotal)
old_Data = machines$Length

#plot hist in one diagram
p1 <- hist(old_Data)
p2 <- hist(new_Data)
plot( p1, col=rgb(0,0,1,1/4), xlim=c(0,7), ylim=c(0,35))  # first histogram
plot( p2, col=rgb(1,0,0,1/4), xlim=c(0,7),ylim=c(0,35), add=T) #second histogram

# Both behave the same way. Are distributed alike, both following the exponential distributon.
# new data followingtheta=1.1. which are generated from old_data
```

Assignment 3 - Feature selection by cross-validation in a linear model - Extra

```
#linear regression and returns predicted Y
mylin=function(X,Y, Xpred){
  Xpred1=cbind(1,Xpred)
 #MISSING: check formulas for linear regression and compute beta
  #minimizing least sqeare givew following formula: w_hat = (Xt*X)^-1 * Xt * y
  X = cbind(1, X)
  beta = solve(t(X)%*%X, t(X)%*%Y)
  Res=Xpred1%*%beta
  return(Res)
}


myCV=function(X,Y,Nfolds){
  n=length(Y)
  p=ncol(X)
  set.seed(12345)
  ind=sample(n,n)
  X1=X[ind,]
  Y1=Y[ind]
  sF=floor(n/Nfolds)
  MSE=numeric(2^p-1)
  Nfeat=numeric(2^p-1)
  Features=list()
  curr=0

  #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
```

```
        if(k!=Nfolds){
          indices = ((k-1)*sF):(k*sF)
        }else{
          indices = ((k-1)*sF):n
        }

        X_train = X1[-indices, which(model == 1)]
        Y_train = Y1[-indices]
        X_validate = X1[indices, which(model == 1)]
        Yp = Y1[indices]

                                                    #MISSING: implement cross-validation for model with features in
"model" and iteration i.
                                                    #MISSING: Get the predicted values for fold 'k', Ypred, and the
original values for folf 'k', Yp.
        Ypred = mylin(X_train, Y_train, X_validate)

         SSE=SSE+sum((Ypred-Yp)^2)
       }
       curr=curr+1
       MSE[curr]=SSE/n
       Nfeat[curr]=sum(model)
       Features[[curr]]=model

     }

 #MISSING: plot MSE against number of features
 plot(Nfeat, MSE, main = "MSE", xlab = "Number of features")

 i=which.min(MSE)
 return(list(CV=MSE[i], Features=Features[[i]]))
}

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

Assignment 4 - Linear regression and regularization

```
tecator <- read_csv2("C:/Users/oskar/OneDrive/Universitet/Linköping Universitet/År4/Machine learning/Lab
1/tecator.csv")

#-------1---------
plot(tecator$Moisture, tecator$Protein)
# Looks lika a line, makes me think that a linear model would be good.

#-------2---------
#Consider model ???????????????? in which Moisture is normally distributed, and the expected
#Moisture is a polynomial function of Protein including the polynomial terms up to power
#???????? (i.e M1 is a linear model, M2 is a quadratic model and so on). Report a probabilistic
#model that describes ????????????????. Why is it appropriate to use MSE criterion when fitting
#this model to a training data?
```

```
#-------3---------

n=dim(tecator)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
t_train=tecator[id,]
t_test=tecator[-id,]

mse = function(x, x_pred){
  squared_error = (x-x_pred)^2
  mse = sum(squared_error)/length(x)
  return(mse)
}

#sum (i=0-n) bi*x^i

model_i_1 = glm(Moisture~Protein, data=t_train)
model_i_2 = glm(Moisture~Protein + I(Protein^2), data=t_train)
model_i_3 = glm(Moisture~Protein + I(Protein^2) + I(Protein^3), data=t_train)
model_i_4 = glm(Moisture~Protein + I(Protein^2) + I(Protein^3) + I(Protein^4), data=t_train)
model_i_5 = glm(Moisture~Protein + I(Protein^2) + I(Protein^3) + I(Protein^4) + I(Protein^5), data=t_train)
model_i_6 = glm(Moisture~Protein + I(Protein^2) + I(Protein^3) + I(Protein^4) + I(Protein^5) + I(Protein^6),
data=t_train)

#predictions for test data
pred_i_1 = predict(model_i_1, newdata = t_test)
pred_i_2 = predict(model_i_2, newdata = t_test)
pred_i_3 = predict(model_i_3, newdata = t_test)
pred_i_4 = predict(model_i_4, newdata = t_test)
pred_i_5 = predict(model_i_5, newdata = t_test)
pred_i_6 = predict(model_i_6, newdata = t_test)

#predictions for train data
pred_i_1_train = predict(model_i_1, newdata = t_train)
pred_i_2_train = predict(model_i_2, newdata = t_train)
pred_i_3_train = predict(model_i_3, newdata = t_train)
pred_i_4_train = predict(model_i_4, newdata = t_train)
pred_i_5_train = predict(model_i_5, newdata = t_train)
pred_i_6_train = predict(model_i_6, newdata = t_train)

#mse for test data
mse_test_1 = mse(t_test$Moisture, pred_i_1)
mse_test_2 = mse(t_test$Moisture, pred_i_2)
mse_test_3 = mse(t_test$Moisture, pred_i_3)
mse_test_4 = mse(t_test$Moisture, pred_i_4)
mse_test_5 = mse(t_test$Moisture, pred_i_5)
mse_test_6 = mse(t_test$Moisture, pred_i_6)

#mse for train data
mse_test_1_train = mse(t_train$Moisture, pred_i_1_train)
mse_test_2_train = mse(t_train$Moisture, pred_i_2_train)
mse_test_3_train = mse(t_train$Moisture, pred_i_3_train)
mse_test_4_train = mse(t_train$Moisture, pred_i_4_train)
```

```
mse_test_5_train = mse(t_train$Moisture, pred_i_5_train)
mse_test_6_train = mse(t_train$Moisture, pred_i_6_train)

print(mse_test_1)
print(mse_test_2)
print(mse_test_3)
print(mse_test_4)
print(mse_test_5)
print(mse_test_6)

print(mse_test_1_train)
print(mse_test_2_train)
print(mse_test_3_train)
print(mse_test_4_train)
print(mse_test_5_train)
print(mse_test_6_train)

mses_test = c(mse_test_1, mse_test_2, mse_test_3, mse_test_4, mse_test_5, mse_test_6)
mses_train = c(mse_test_1_train, mse_test_2_train, mse_test_3_train, mse_test_4_train, mse_test_5_train,
mse_test_6_train)
which.min(mses_test)
which.min(mses_train)

plot((1:6), mses_test, col="red")

plot((1:6), mses_train, col="red")

# Both
plot((1:6), mses_test, col="blue", ylim=c(23,45) )
par(new=TRUE)
plot((1:6), mses_train, col="red", ylim=c(23,45)  )



#4
sub_of_tecator = subset(tecator, select = - c(Protein, Moisture, Sample))

n=dim(sub_of_tecator)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
t_train_sub=sub_of_tecator[id,]
t_test_sub=sub_of_tecator[-id,]

library(MASS)
linear_model_fat = glm(Fat~., data=sub_of_tecator)
step = stepAIC(linear_model_fat, direction = "both")

step$anova
summary(step)

covariates = scale(subset(sub_of_tecator, select = -Fat))
response = scale(subset(sub_of_tecator, select = Fat))

#-------5---------ridge
```

```
ridge = glmnet(as.matrix(covariates), response, alpha=0, family= "gaussian" )
plot(ridge, xvar="lambda", label=TRUE)

#-------6---------
lasso = glmnet(as.matrix(covariates), response, alpha=1, family= "gaussian")
plot(lasso, xvar = "lambda", label=TRUE) #higher lambda, lower vairance, higher variance.

#-------7---------
#gamma = 1, gives me lasso(0 is ridge), lambda = is which lambda to include.
lasso_mse = cv.glmnet(covariates, response, gamma=1, lambda = seq(from = 0, to=1, by=0.01))
plot(lasso_mse)


#library #i=3 best.(readr)
#spambase <- read_delim("C:/Users/oskar/OneDrive/Universitet/Linköping Universitet/År4/Machine learning/Lab
1/spambase.csv",
#                       ";", escape_double = FALSE, trim_ws = TRUE)
```

Lab 2

Assignment 1 - LDA and logistic regression

```
RNGversion('3.5.1')
library(readr)
set.seed(12345)

#Assignment1
australian_crabs = read.csv("C:/Users/oskar/OneDrive/Universitet/Linköping Universitet/År4/Machine learning/Lab
2/australian-crabs.csv")

#--------step1----------
australian_crabs_males = subset(australian_crabs, sex=="Male")
australian_crabs_females = subset(australian_crabs, sex=="Female")

plot(australian_crabs_males[['CL']], australian_crabs_males[['RW']], ylim=c(6,20), xlim=c(15,45), col="red",
ylab="RW", xlab="CL")
par(new=TRUE)
plot(australian_crabs_females[['CL']], australian_crabs_females[['RW']], ylim=c(6,20), xlim=c(15,45), col="blue",
ylab="RW", xlab="CL")


#---------Step2----------
library(MASS)

lda_pred = lda(sex~CL + RW, data=australian_crabs)

print(lda_pred)
pred = predict(lda_pred, australian_crabs)
table(australian_crabs[['sex']], pred$class)

predicted_dataset = data.frame(pred$class, australian_crabs[['CL']], australian_crabs[['RW']])
names(predicted_dataset) = c('sex', 'CL', 'RW')
```

```
plot(subset(predicted_dataset, sex=="Male")[['CL']], subset(predicted_dataset, sex=="Male")[['RW']], ylim=c(6,20),
xlim=c(15,45), col="red", ylab="RW", xlab="CL")
par(new=TRUE)
plot(subset(predicted_dataset, sex=="Female")[['CL']], subset(predicted_dataset, sex=="Female")[['RW']],
ylim=c(6,20), xlim=c(15,45), col="blue", ylab="RW", xlab="CL")

# Misclassification function
misclass=function(X, Xfit){
  n=length(X)
  return (1-sum(diag(table(X, Xfit)))/n)
}
lda_pred_misclassification = misclass(australian_crabs[['sex']], pred$class)
print(lda_pred_misclassification)

#-------step3-------
lda_pred_wprior = lda(sex~CL + RW, data=australian_crabs, prior = c(0.1, 0.9))
print(lda_pred_wprior)

pred_wprior = predict(lda_pred_wprior, australian_crabs)
table(australian_crabs[['sex']], pred_wprior$class)
predicted_dataset_wprior = data.frame(pred_wprior$class, australian_crabs[['CL']], australian_crabs[['RW']])
names(predicted_dataset_wprior) = c('sex', 'CL', 'RW')

plot(subset(predicted_dataset_wprior, sex=="Male")[['CL']], subset(predicted_dataset_wprior, sex=="Male")[['RW']],
ylim=c(6,20), xlim=c(15,45), col="red", ylab="RW", xlab="CL")
par(new=TRUE)
plot(subset(predicted_dataset_wprior, sex=="Female")[['CL']], subset(predicted_dataset_wprior,
sex=="Female")[['RW']], ylim=c(6,20), xlim=c(15,45), col="blue", ylab="RW", xlab="CL")

lda_pred_misclassification_wprior = misclass(australian_crabs[['sex']], pred_wprior$class)
print(lda_pred_misclassification_wprior)

#--------step 4----------
#check if sex is as factor
str(australian_crabs)

logistic_regression = glm(as.factor(sex) ~ CL + RW, data=australian_crabs, family = binomial)
print(logistic_regression)

#dessission boundary
intercept = coef(logistic_regression)[1]/(-coef(logistic_regression)[3])
slope = coef(logistic_regression)[2]/(-coef(logistic_regression)[3])
x = seq(15,45, by=1)
y = slope*x + intercept


prediction_LR = predict(logistic_regression, australian_crabs, type="response")
predicted_sex = prediction_LR
predicted_sex[prediction_LR<0.5] = 'Female' #prediction_LR will return all rows that are under threashold!
predicted_sex[prediction_LR>=0.5] = 'Male'

predicted_sex
australian_crabs[["sex"]]
```

```
misclass(australian_crabs[["sex"]], predicted_sex)

table(australian_crabs[["sex"]], predicted_sex)

predicted_dataset_LR = data.frame(predicted_sex, australian_crabs[['CL']], australian_crabs[['RW']])
names(predicted_dataset_LR) = c('sex', 'CL', 'RW')


plot(x, y, type="l", ylim=c(6,20),  xlim=c(15,45), ylab="RW", xlab="CL")
par(new=TRUE)
plot(subset(predicted_dataset_LR, sex=="Male")[['CL']], subset(predicted_dataset_LR, sex=="Male")[['RW']],
ylim=c(6,20), xlim=c(15,45), col="red", ylab="RW", xlab="CL")
par(new=TRUE)
plot(subset(predicted_dataset_LR, sex=="Female")[['CL']], subset(predicted_dataset_LR, sex=="Female")[['RW']],
ylim=c(6,20), xlim=c(15,45), col="blue", ylab="RW", xlab="CL")
```

Assignment 2 - Analysis of credit scoring

```
#----------Step 1----------
creditscoring = read.csv2("C:/Users/oskar/OneDrive/Universitet/Linköping Universitet/År4/Machine learning/Lab
2/creditscoring.csv")

RNGversion('3.5.1')

n=dim(creditscoring)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.5))
train=creditscoring[id,]

id1=setdiff(1:n, id)
set.seed(12345)
id2=sample(id1, floor(n*0.25))
valid=creditscoring[id2,]

id3=setdiff(id1,id2)
test=creditscoring[id3,]

library(tree)
#or:
library(rpart)

#Training data to fit model
fit_deviance = tree(good_bad~. , split = "deviance", data = train)
fit_gini = tree(good_bad~. , split = "gini", data = train)

summary(fit_deviance)
summary(fit_gini)

#Predict using test data.
predict_deviance = predict(fit_deviance, newdata = test, type = "class")

#table(test[["good_bad"]], predict_deviance)
```

```
misclass_deviance = misclass(test[["good_bad"]], predict_deviance)
print(misclass_deviance)

predict_gini = predict(fit_gini, newdata = test, type = "class")

#table(test[["good_bad"]], predict_gini)
misclass_gini = misclass(test[["good_bad"]], predict_gini)
print(misclass_gini)

#----------Step 3----------
#Deviance is chosen due to lower misclassification rate for test data.
summary(fit_deviance)

train_score = rep(0,15)
test_score = rep(0,15)

for(i in 2:15) {
  pruned_tree = prune.tree(fit_deviance, best = i)
  pred = predict(pruned_tree, newdata=valid, type="tree")

  train_score[i] = deviance(pruned_tree)
  test_score[i] = deviance(pred)
}

plot(2:15, train_score[2:15], type="b", col="red", ylim=c(200,550), ylab="Deviance", xlab="No. of leaves")
points(2:15, test_score[2:15], type="b", col="blue")

test_score[1] = 5000
which.min(test_score)

## Min when best=4
test_score[4]
pruned_tree = prune.tree(fit_deviance, best = 4)
summary(pruned_tree)
plot(pruned_tree)
text(pruned_tree, pretty = 0)

#Misclass for test
prediction_test = predict(pruned_tree, newdata = test, type = "class")
table(test[["good_bad"]], prediction_test)
misclass(test[["good_bad"]], prediction_test)

#----------Step 4 ----------
library(MASS)
library(e1071)

fit_naive_bayes =naiveBayes(good_bad~., data=train)
summary(fit_naive_bayes)
#train data
predict_naive_bayes_train = predict(fit_naive_bayes, newdata = train)
table(train[["good_bad"]], predict_naive_bayes_train)
misclass(train[["good_bad"]], predict_naive_bayes_train)
#test data
```

```
predict_naive_bayes_test = predict(fit_naive_bayes, newdata = test)
table(test[["good_bad"]], predict_naive_bayes_test)
misclass(test[["good_bad"]], predict_naive_bayes_test)
# remember: 1-(sum(diag(table))/sum(table))

#----------Step 5-----------
# TPR = true positive rate(y-axis)
# FPR = false positive reate(x-axis)
predict_naive_bayes_test = predict(fit_naive_bayes, newdata = test, type= "raw")
predict_naive_bayes_test

pi = seq(from = 0.05, to = 0.95, by = 0.05 )
n = length(pi)

#Naive Bayes
TPR = rep(0,n)
FPR = rep(0,n)
for( i in 1:n){
  predict = predict_naive_bayes_test[,2]
  predict = ifelse(predict>pi[i], "good", "bad")
  table = table(test[["good_bad"]], predict)
  print(table)
  TPR[i] = (table[2, 2])/sum(table[2, ])
  FPR[i] = (table[1, 2])/sum(table[1, ])
}

# tree ROC
#str(test)
prediction_test = predict(pruned_tree, newdata = test, type = "vector")

n = length(pi)
TPR_tree = rep(0,n)
FPR_tree = rep(0,n)
for( i in 1:n){
  pred = as.vector(prediction_test[,2])
  pred = ifelse(pred>pi[i], "good", "bad")
  if ( sum(pred=="bad")==0) {
    FPR_tree[i] = 1
    TPR_tree[i] = 1
  } else if ( sum(pred=="good")==0) {
    TPR_tree[i] = 0
    FPR_tree[i] = 0
  } else {
    table = table(test[["good_bad"]], pred)
    print(table)
    TPR_tree[i] = (table[2, 2])/sum(table[2, ])
    FPR_tree[i] = (table[1, 2])/sum(table[1, ])
  }
}

plot(FPR_tree, TPR_tree, xlim = (0:1), ylim= (0:1), type="b", col="red", xlab="FPR", ylab="TPR", main="ROC")
par(new=TRUE)
plot(FPR, TPR, xlim = (0:1), ylim= (0:1), type="b", col="blue", xlab="FPR", ylab="TPR")
```

```
#-------Step 6-------------
fit_naive_bayes =naiveBayes(good_bad~., data=train)
summary(fit_naive_bayes)
#train data
naive_bayes_train = predict(fit_naive_bayes, newdata = train, type="raw")
predict_train = ifelse(naive_bayes_train[,2]/naive_bayes_train[,1]>10, "good", "bad")

table(train[["good_bad"]], predict_train)
misclass(train[["good_bad"]], predict_train)

#test data
naive_bayes_test = predict(fit_naive_bayes, newdata = test, type="raw")
predict_test = ifelse(naive_bayes_test[,2]/naive_bayes_test[,1]>10, "good", "bad")
misclass(test[["good_bad"]], predict_test)
table = table(test[["good_bad"]], predict_test)
print(table)
```

Assignment 3 - Uncertainty estimation - Extra

```
RNGversion('3.5.1')
library(readr)

state = read.csv2("C:/Users/oskar/OneDrive/Universitet/Linköping Universitet/År4/Machine learning/Lab
2/state.csv")
state = data.frame(state)
#---Step 1---
state_ordered = state[order(state$MET, decreasing = FALSE),]
plot(state_ordered$MET, state_ordered$EX, main ="MET vs EX", xlab = "MET", ylab = "EX")
# qudratic or polinomial?

#---Step 2---
library(tree)
tree_model = tree(EX ~ MET, state_ordered, control = tree.control(nobs = nrow(state_ordered), minsize = 8))
plot(tree_model)
text(tree_model, pretty = 0)
cv_tree = cv.tree(tree_model)
best_size = cv_tree$size[which.min(cv_tree$dev)]

#following plot showes that best = 4 is the best tree.
plot(x=cv_tree$size, y=cv_tree$dev, type = "b", col= "blue")

pruned_tree = prune.tree(tree_model, best = best_size)
pred = predict(pruned_tree, newdata = state_ordered)

hist(residuals(pruned_tree), breaks = 20)

plot(x=state_ordered$MET, y=state_ordered$EX)
points(x=state_ordered$MET , y=pred, col="red", type = "l")
```

```
#---step 3---
set.seed(12345)
B=1000
sample_size = nrow(state_ordered)

boot_predictions = matrix(nrow=sample_size, ncol=B)
for(i in 1:B){
  samples = sample(seq(1,nrow(state_ordered),1), size = sample_size, replace=TRUE)
  data = state_ordered[samples, ]

  tree_model = tree(EX ~ MET, data = data, control = tree.control(nobs= nrow(state_ordered), minsize = 8))
  pruned_tree = prune.tree(tree_model, best=best_size)
  boot_predictions[, i] = predict(pruned_tree, newdata = state_ordered)
}
#calcultate std deviations and confidence bands using quantiles function
conf_band = apply(boot_predictions, 1, function(x){
  band = quantile(x, probs = c(0.025, 0.975))
  return(band)
})

#Plot conf. intervalls:
plot(x=state_ordered$MET, y=state_ordered$EX, col = "red", xlab = "MET", ylab = "EX", main = "95% conf. bands")
points(x=state_ordered$MET , y=pred, col="red", type = "l")
lines(x=state_ordered$MET, y=conf_band[1, ], col = "blue")
lines(x=state_ordered$MET, y=conf_band[2, ], col = "blue")


#use bootstrap ----------TEST
library(boot)
#function to generate datapoints for non-parametic bootstrap
f=function(data, ind){
  data1=data[ind,]# extract bootstrap sample
  tree_model=tree(EX ~ MET, data1, control = tree.control(nobs = nrow(data1), minsize = 8)) #fit linear model
  pruned_tree = prune.tree(tree_model, best = best_size)

  #predict values for all Area values from the original data
  priceP=predict(pruned_tree, newdata=state_ordered)
  return(priceP)
}
res=boot(state_ordered, f, R=1000) #make bootstrap
res
boot_data = data.frame(res$t)
conf_band = apply(boot_data, 2, function(col){
  band = quantile(col, probs = c(0.025, 0.975))
  return(band)})
conf_band

plot(x=state_ordered$MET, y=state_ordered$EX, col = "red", xlab = "MET", ylab = "EX", main = "95% conf. bands")
points(x=state_ordered$MET , y=pred, col="red", type = "l")
lines(x=state_ordered$MET, y=conf_band[1, ], col = "blue")
lines(x=state_ordered$MET, y=conf_band[2, ], col = "blue")

# end ----------TEST
```

```
#-----Step 4-------
#parametric bootstrap for confidence intervalls:
#values for distr_gen. sigmna = std_dev
tree_model = tree(EX ~ MET, state_ordered, control = tree.control(nobs = nrow(state_ordered), minsize = 8))
pruned_tree = prune.tree(tree_model, best = best_size)
pred = predict(pruned_tree, newdata = state_ordered)
residuals = state_ordered$EX - pred
std_dev = sd(residuals)

distr_gen = function(data, tree_model){
  pred = predict(tree_model, newdata = data)
  res = rnorm(nrow(data), mean = pred, sd= std_dev )
  data$EX = res
  return(data)
}

tree_model = pruned_tree

stat = function(data){
 tree_model = tree(EX ~ MET, data=data,
             control = tree.control(nobs = nrow(data),
                          minsize = 8))
 pruned_tree = prune.tree(tree_model, best = best_size)
 pred = predict(pruned_tree, newdata = state_ordered)
 return(pred)
}

res_para = boot(state_ordered, statistic = stat ,
   mle = tree_model,
   R = 1000,
   sim = "parametric",
   ran.gen = distr_gen )

boot_data = data.frame(res_para$t)
conf_band = apply(boot_data, 2, function(col){
 band = quantile(col, probs = c(0.025, 0.975))
 return(band)})
conf_band

plot(x=state_ordered$MET, y=state_ordered$EX, col = "red", xlab = "MET", ylab = "EX", main = "95% conf. bands")
points(x=state_ordered$MET , y=pred, col="red", type = "l")
lines(x=state_ordered$MET, y=conf_band[1, ], col = "blue")
lines(x=state_ordered$MET, y=conf_band[2, ], col = "blue")

test = envelope(res_para)
lines(x=state_ordered$MET, y=test$point[1,], col = "green")
lines(x=state_ordered$MET, y=test$point[2,], col = "green")

#prediction bands
statistic = function(data){
 tree_model = tree(EX ~ MET, data=data,
             control = tree.control(nobs = nrow(data),
```

```
                                minsize = 8))
  pruned_tree = prune.tree(tree_model, best = best_size)
  pred = predict(pruned_tree, newdata = state_ordered)
  res = rnorm(nrow(data), mean = pred, sd= std_dev )
  return(res)
}
res_pred_band = boot(state_ordered, statistic = statistic ,
            mle = tree_model,
            R = 1000,
            sim = "parametric",
            ran.gen = distr_gen )
pred_band = envelope(res_pred_band)
lines(x=state_ordered$MET, y=pred_band$point[1,])
lines(x=state_ordered$MET, y=pred_band$point[2,])

#step 5
hist(residuals(pruned_tree),
    breaks = 20, main ="Histogram of the residuals",
    xlab = "Residual")
#chi-square model would be preffered.
```

Assignment 4 - Principal components

```
NIR_spectra = read.csv2("C:/Users/oskar/OneDrive/Universitet/Linköping Universitet/År4/Machine learning/Lab
2/NIRSpectra.csv")

#--------Step 1------------
data1 = NIR_spectra
data1$Viscosity = c()
res = prcomp(data1)

#squaring sdev to get values that are (proportional to) eigenvalues
lambda = res$sdev^2
X = res$x

#hom much variance is explained in each component
sprintf("%2.3f",lambda/sum(lambda)*100)

#histogram of explained variance
screeplot(res)

# extract 2 components to get 99 explenation of total variance. PC1, PC2.
plot(res$x[,1], res$x[,2], xlab ="PC1", ylab="PC2")

#----------Step 2---------------
plot(res$rotation[,1], main="Traceplot of PC1")
plot(res$rotation[,2], main="Traceplot of PC2")

#----------Step 3-------------
library(fastICA)
set.seed(12345)
ica = fastICA(data1, 2)
```

```
W_fnutt = ica$K %*% ica$W
plot(W_fnutt[,1], main="Traceplot of W'1")
plot(W_fnutt[,2], main="Traceplot of W'2")

#Plot of scores for the two latent features
plot(ica$S, main="ICA Score", xlab="Latent Feature 1", ylab="Latent Feature 2")

#TESTing
plot(ica$X, main = "Pre-processed data")
plot(ica$X %*% ica$K, main = "PCA components")
plot(ica$S, main = "ICA components")
plot(ica$K)
```

Lab 3

Assignment 1 - KERNEL METHODS

```
RNGversion('3.5.1')
library(readr)
library(geosphere)

#---Assignment 1 ----
stations = read.csv("C:/Users/oskar/OneDrive/Universitet/Linköping Universitet/År4/Machine learning/Lab 3/stations.csv")
temps = read.csv("C:/Users/oskar/OneDrive/Universitet/Linköping Universitet/År4/Machine learning/Lab 3/temps50k.csv")

set.seed(1234567890)
st <- merge(stations,temps,by="station_number")


h_distance <- 80000 # These three values are up to the students
h_date <- 10
h_time <- 4
a <- 58.4274 # The point to predict (up to the students)
b <- 14.826
date <- "2013-11-04" # The date to predict (up to the students)
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")
temp <- vector(length=length(times))
temp_mult <- vector(length=length(times))

# Students' code here
#test h values
max(distHaversine(data.frame(st$latitude, st$longitude), c(a,b)))
dist = seq(0,200000, 1)
y = exp(-(dist/h_distance)^2)
plot(y, type="l", main = "Distance kernel")
#satisfied with h_distance = 80000 beacause then we stop to care if distance>than200km
dat = seq(0,30, 1)
y = exp(-(dat/h_date)^2)
plot(y, type="l", main = "Date kernel")
#satisfied with h_date = 10 because then we stop to care if the date is older than 25 days.
tim = seq(0, 24, 1)
```

```
y = exp(-(tim/h_time)^2)
plot(y, type="l", main = "Time kernel")
#satisfied with h_time = 4 because then only times within 7 hours is used for estimate.

# remove all posterior dates
str(st)
st$date = as.Date(st$date, format = "%Y-%m-%d")
#remove earlier times than 04:00:00
filtered_data = st[st$date <= date, ]
filtered_data = filtered_data[!(filtered_data$date==date && substr(filtered_data$time, 1, 2)<substr(times[1], 1, 2))]

#Gussian kernel is used: k(u) = exp(-||u||^2),
#||.|| is the Euclidean norm.
euclidean = function(X){
  return (sqrt(sum(X^2)))
}

#gussian kernel
kernel_dist = function(X, X_n, h){
  distance = distHaversine(X, X_n)
  u = (distance)/h  # calculate u = X-Xn/h
  return(exp(-euclidean(u))) #calculate k
}

kernel_date = function(X, X_n, h){
  distance = as.numeric((X - X_n))%%365.25
  if (distance > 365/2) {
    distance = 365 - distance
  }
  u = distance / h
  return(exp(-euclidean(as.numeric(u))))
}

kernel_time = function(X, X_n, h){
  distance = as.numeric(X) - as.numeric(X_n)
  if (distance > 12){
    distance = 24-distance
  }
  u = distance/h
  return(exp(-euclidean(u)))
}



#calculate y with dist
#filtered_data = filtered_data[order(filtered_data$latitude, filtered_data$longitude),]

#kernel_weight = 0
n = nrow(filtered_data)
k=vector("numeric", length = n)
k_mult = vector("numeric", length = n)
k_loop=vector("numeric", length = n)
k_mult_loop = vector("numeric", length = n)
```

```
for(i in 1:n){
  k_dist = kernel_dist(c(filtered_data$longitude[i], filtered_data$latitude[i]), c(a,b), h_distance )
  k_date = kernel_date(filtered_data$date[i], as.Date(date), h_date)

  k[i]=k_date + k_dist
  k_mult[i] = k_date * k_dist
}
for(j in 1:(length(times))){
  for (i in 1:n) {
    k_time = kernel_time(substr(filtered_data$time[i], 1, 2), substr(times[j], 1, 2), h_time)
    k_loop[i] = k[i] + k_time
    k_mult_loop[i] = k_mult[i] * k_time

    temp[j] = temp[j] + k_loop[i]*filtered_data$air_temperature[i]
    temp_mult[j] = temp_mult[j] + k_mult_loop[i] * filtered_data$air_temperature[i]
  }
  temp[j] = temp[j] / sum(k_loop)
  temp_mult[j] = temp_mult[j] /sum(k_mult_loop)
}

#test_temp = test_temp/kernel_weight #kernel weighted temp.
#test_temp
plot(temp, xaxt = "n", type = "b", main = "Kernel Addition")
axis(1, at=1:length(df.times), labels=df.times)

plot(temp_mult, xaxt = "n", type = "b", main ="Kernel multiplication")
axis(1, at=1:length(df.times), labels=df.times)
```

Assignment 2 - SUPPORT VECTOR MACHINES - Extra

```
##Use the function ksvm from the R package kernlab to learn a SVM for classifying the spam dataset that is included
#with the package. Consider the radial basis function kernel (also known as Gaussian) with a width of 0.05. For the
#parameter C, consider values 0.5, 1 and 5. This implies that you have to consider three models.
# Perform model selection, i.e. select the most promising of the three models (use any method of your choice except
#cross-validation or nested-cross-validation)
# Estimate the generalization error of the SVM selected above (use any method of your choice except cross-
validation
#or nested cross validation)
# Produce the SVM that will be returned to the user, i.e. show the code
# What is the purpose of the parameter C?

library(kernlab)
set.seed(1234567890)
data(spam)

#Create function for misclassification rate
missclass=function(conf_matrix, fit_matrix){
  n=length(fit_matrix[,1])
  return(1-sum(diag(conf_matrix))/n)
}


index=sample(1:4601)
```

```
train=spam[index[1:2500],]
valid=spam[index[2501:3501],]
test=spam[index[3502:4601],]

svmmodel1=ksvm(type~., data=train, kernel="rbfdot", kpar=list(sigma=0.05), C=0.5)
pred1=predict(svmmodel1, newdata=valid)
confusion1=table(valid$type, pred1)
misclass1=missclass(confusion1, valid)
print(confusion1)
print(misclass1)

svmmodel2=ksvm(type~., data=train, kernel="rbfdot", kpar=list(sigma=0.05), C=1)
pred2=predict(svmmodel2, newdata=valid)
confusion2=table(valid$type, pred2)
misclass2=missclass(confusion2, valid)
print(confusion2)
print(misclass2)

svmmodel3=ksvm(type~., data=train, kernel="rbfdot", kpar=list(sigma=0.05), C=5)
pred2=predict(svmmodel3, newdata=valid)
confusion3=table(valid$type, pred2)
misclass3=missclass(confusion3, valid)
print(confusion3)
print(misclass3)

##Conclusion: The model with the C value of 1 is the best since it has the lowest misclassification rate. However,
##since the application is classification of spam emails, the value of C=0.5 is the best since it classified the least
##nonspam emails as spam.

finalmodel=ksvm(type~., data=spam[index[1:3501],], kernel="rbfdot", kpar=list(sigma=0.05), C=1)
finalpred=predict(finalmodel, newdata=test)
finalconfusion=table(test$type, finalpred)
finalmisclass=missclass(finalconfusion, test)
print(finalconfusion)
print(finalmisclass)

##Answer: The purpose of the parameter C is to put a weight to the cost function. The higher C the more cost will a
##constraint violation yield.

#Final model

finalmodel=ksvm(type~., data=spam, kernel="rbfdot", kpar=list(sigma=0.05), C=1)
```

Assignment 3 - NEURAL NETWORKS

```
library(neuralnet)
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
# Random initialization of the weights in the interval [-1, 1]
#set.seed(12345). Did not use this becaus it was not used in the code skeleton.
```

```
winit <- runif(31, -1, 1)
n = 10
SE_tr = vector("numeric", length = n)
SE_va = vector("numeric", length = n)
for(i in 1:n) {
  nn <- neuralnet(Sin ~ Var, data=tr, hidden = c(10), startweights = winit, threshold = i/1000 )

  p_tr = predict(nn, newdata = tr)
  SE_tr[i] = sum((tr$Sin - p_tr)^2)
  p_va = predict(nn, newdata = va)
  SE_va[i] = sum((va$Sin - p_va)^2)
}

which.min(SE_va) # 4/1000 has the lowest error.

plot(SE_tr, col = "red", ylim = c(0.001, 0.035), ylab = "Sum of Squared Error")
par(new=TRUE)
plot(SE_va, col = "blue", ylim = c(0.001, 0.035), ylab = "Sum of Squared Error")

#4/1000has the lowest Squared Error. Therefore 4/1000 is shosen as threshold.
plot(nn <- neuralnet(Sin ~ Var, data=tr, hidden = c(10), startweights = winit, threshold = 4/1000))

# Plot of the predictions (black dots) and the data (red dots)
plot(prediction(nn)$rep1)
points(trva, col = "red")
```