Innehåll

[Regression 1](#_Toc29730224)

[Linear regression 1](#_Toc29730225)

[Things 1](#_Toc29730226)

[R commands 1](#_Toc29730227)

[Pic rows/col 1](#_Toc29730228)

[Packages used in lab 1](#_Toc29730229)

[Distributions 1](#_Toc29730230)

[Chi-square (två) distribution 1](#_Toc29730231)

[Poisson distribution 2](#_Toc29730232)

[Exponential distribution 2](#_Toc29730233)

[Labs 3](#_Toc29730234)

[Lab1 3](#_Toc29730235)

[Assignment 1 - Spam classification with nearest neighbors 3](#_Toc29730236)

[Assignment 2 - Inference about lifetime of machines 4](#_Toc29730237)

[Assignment 3 - Feature selection by cross-validation in a linear model - Extra 6](#_Toc29730238)

[Assignment 4 - Linear regression and regularization 7](#_Toc29730239)

[Lab 2 10](#_Toc29730240)

[Assignment 1 - LDA and logistic regression 10](#_Toc29730241)

[Assignment 2 - Analysis of credit scoring 12](#_Toc29730242)

[Assignment 3 - Uncertainty estimation - Extra 15](#_Toc29730243)

[Assignment 4 - Principal components 18](#_Toc29730244)

[Lab 3 19](#_Toc29730245)

[Assignment 1 - KERNEL METHODS 19](#_Toc29730246)

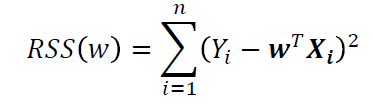
[Assignment 2 - SUPPORT VECTOR MACHINES - Extra 21](#_Toc29730247)

[Assignment 3 - NEURAL NETWORKS 22](#_Toc29730248)



# Regression

## Linear regression

Estimation: maximum likelihood is equal to min square: . Optimal condition: XT (y – Xw) = 0  
X has 1s as first col: cbind(1,X), y is one col. Gives estimation as: w\_hat = ( XT X )-1 \* XT 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

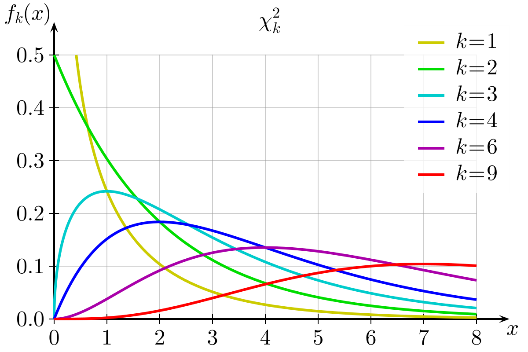
* readr
* glmnet
* kknn
* MASS
* tree or (rpart) I choose tree.
* e1071 - Functions for latent class analysis, short time Fourier transform, fuzzy clustering, support vector machines, shortest path computation, bagged clustering, naive Bayes classifier
* fastICA
* geosphere
* neuralnet

extra assign:

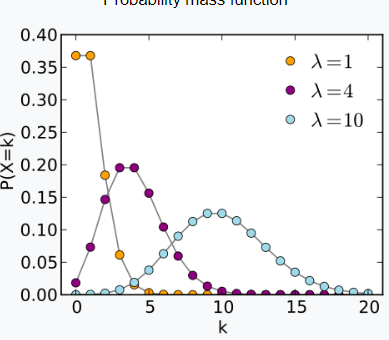
# Distributions

## Chi-square (två) distribution

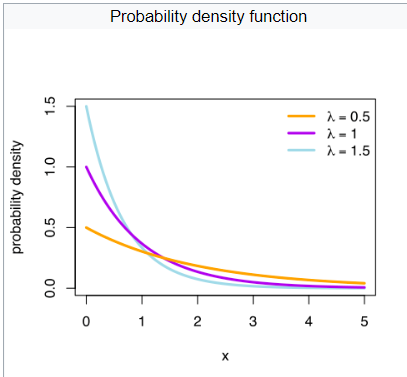
Chitvåfördelning alternativt Chikvadratfördelning, χ²-fördelning, är inom matematisk statistik en kontinuerlig sannolikhetsfördelning med täthetsfunktionen:

## Poisson distribution

## Exponential distribution

# 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")