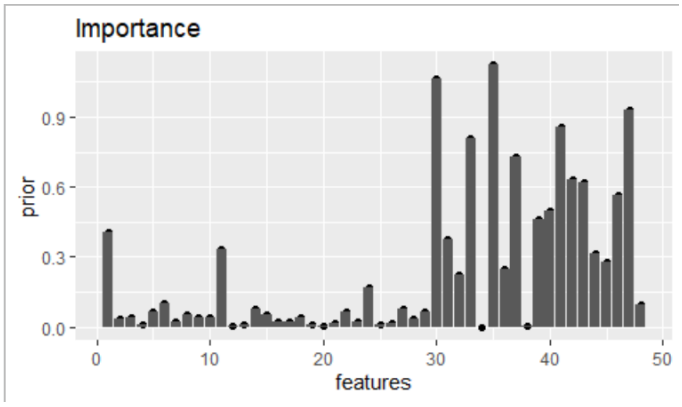
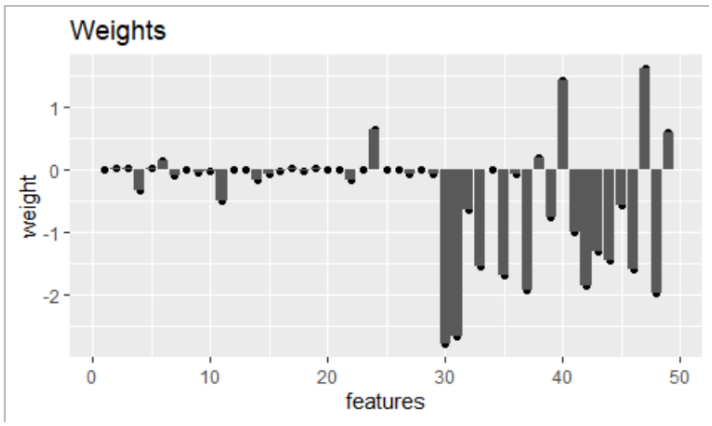
Special tasks

Task 1

The 31st word is the most important, and the 46st is the second most important word.



The final model weights reflect the importance measure.



A final misclassification of 28% is the result, with the following confusion table.

```
preds    0    1
    0 1797  532
    1   85  326
```

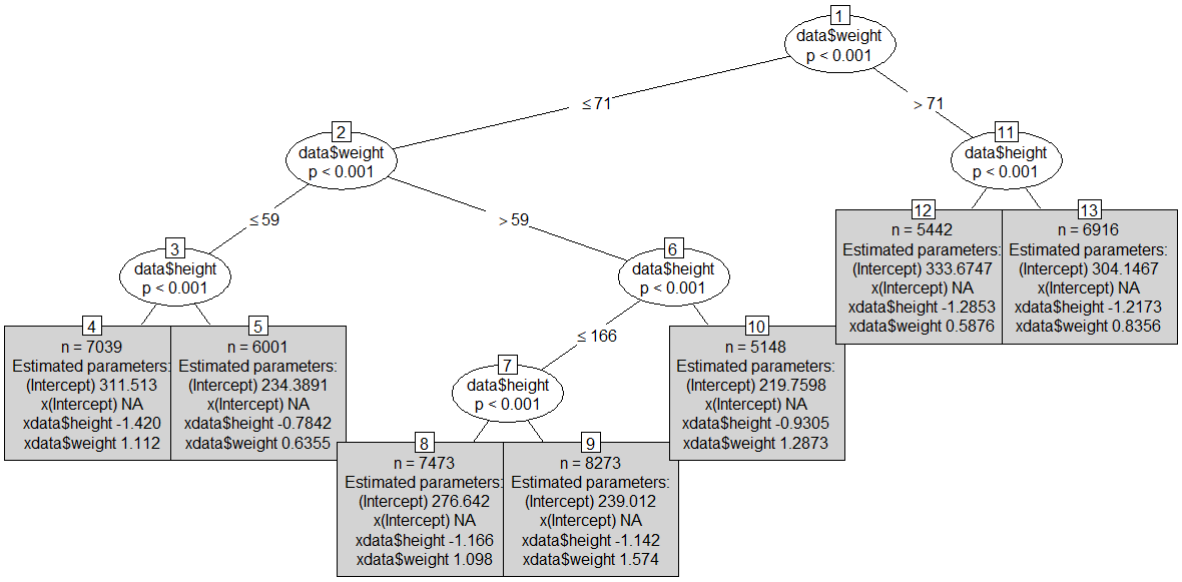
Assignment 2

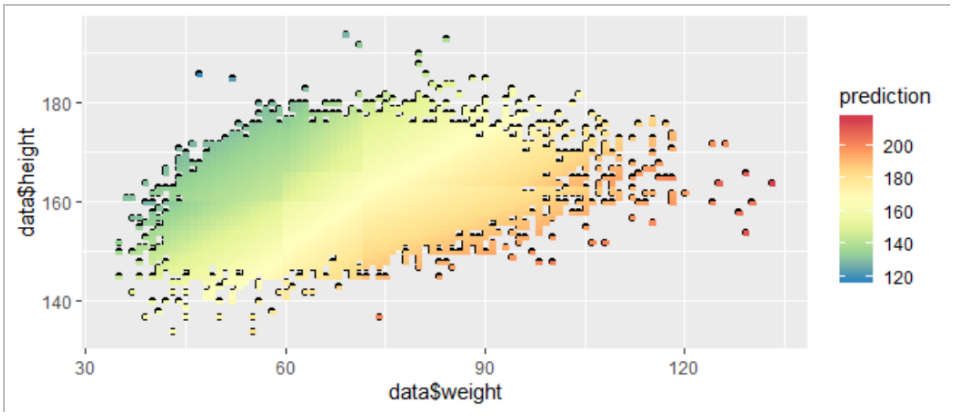
A non-linear regression with a polynomial of two is used for each terminal node in the mob model.

```
mob.fit <- function(y, x, start = NULL, weights = NULL, offset = NULL) {
   lm(y ~ x^2, start = start, offset=offset, weights=weights)
}

mob.tree <- mob(data$`Blood systolic` ~ data$height + data$weight |data$height + data$weight
                , data = data, fit = mob.fit, control = mob_control(minsize=5000))
```

The resulting tree is the following.

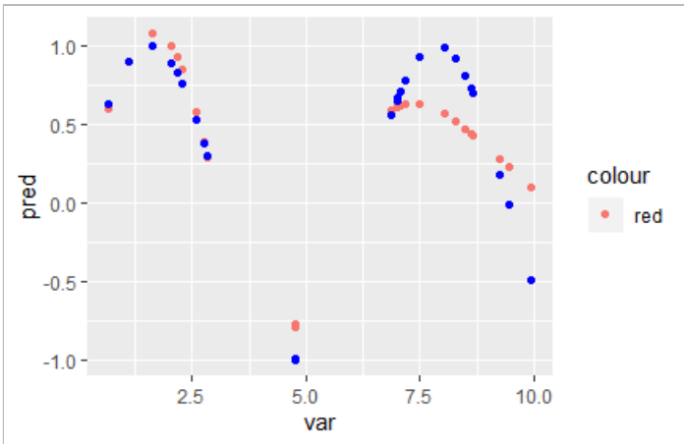




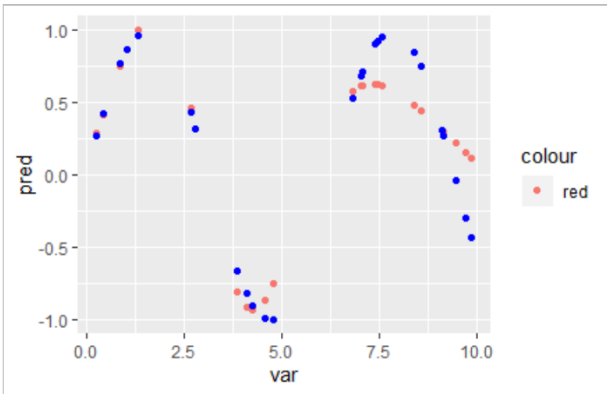
Assignment 3

The final prediction for the NN looks like this, it is pretty close but could be improved with more iterations. The code is in the appendix

Validation. Red is prediction and blue is real values



Training. Red is prediction, blue is real values



Appendix code

Assignment 1

```
library("readxl")

library("ggplot2")

library("MASS")

library("glmnet")



#B=batch size, q=samplesize for each batch, prior= importance vector for weights

randomLasso = function(B,q,prior){

 feat.weights = matrix(0L,nrow=B,ncol=p+1)

 #step 1a/2a Draw B bootstrap samples with size n

 #by sampling with replacement from the original training data set.


 for(i in seq(1:B)){

  obsSample =sample(1:n, sampleSize)


  if(missing(prior)){

   #1b

   # For the b1th bootstrap sample, b1 ??? {1, ., B},

   # randomly select q1 candidate variables,

   # and apply lasso to obtain estimators ??^(b1)j for ??j,

   # j = 1, ., p. Estimators are zero for coefficients

   # of those unselected variables, either outside the subset of

   # q1 variables, or excluded by lasso.

    featSample = sample(1:p,q)

  }else {
```

```
  # 2b
  # For the b2th bootstrap sample,
  # b2 ??? {1, ., B}, randomly select q2 candidate
  # variables with selection probability of xj
  # proportional to its importance Ij obtained in
  # step 1c, and apply lasso (or adaptive lasso)
  # to obtain estimators ??^(b2)j for ??j, j = 1, ., p.
  # Estimators are zero for coefficients of those
  # unselected variables, either outside the subset of q2 variables,
  # or excluded by lasso.
  featSize = min(q,length(which(prior!=0)))
  featSample = sample(1:p,featSize,prob=prior)
 }



 cv.X = as.matrix((data.feat[obsSample,featSample]))
 cv.Y = sapply(data.y[obsSample,],function(x) as.double(x))
 cv.glmnet.fit = cv.glmnet(x=cv.X,y=cv.Y,alpha=1,family="binomial",nfolds=4)
 tmp.coffs = coef(cv.glmnet.fit, s = "lambda.min")

 #save variable into feat.weights, could be optimized
 for(p in 1:p){
  for(q in 1:length(tmp.coffs[,1])){
   #If statement to place weihghts in right index in tmp.coffs
   if(names(tmp.coffs[,1][q]) == names(data.feat)[p]){
    feat.weights[i,p] =  tmp.coffs[,1][q]
   }
  }
 }
 feat.weights[i,p+1] = tmp.coffs[1]
}
```

```
  return (feat.weights)
}

data <- read_excel("spambase.xlsx")

#observation-size
n=dim(data)[1]
#feature-size
p = dim(data)[2]-1
#ammount of bootstrap steps
B = 10
#size of observation sample for each bootstrap step !! Not specified in task
sampleSize = floor(n/B)
#size of feature sample for each bootstrap step
q = round(p/2)

data.feat = data[,-49]
data.y = data[,49]
colnames(data.y)[1] = names(data[,49])

prob = rep(1, length=p)

#feat.weights store weights for each bootstrap step in rows
feat.weights = randomLasso(B,q)[,-p+1]
importance = vector(mode="numeric",length=p)
names(importance) = names(feat.weights)

# 1c
# Compute the importance measure of xj by Ij=???B???1???Bb1=1??^(b1)j?????????.
for(i in 1:p){
  importance[i] = abs((sum(feat.weights[,i])/B))
```

```
}

# plot importance per word

which.max(importance[-31])

qplot(y=importance,xlab="features", ylab="prior") + labs(title="Importance")
+geom_bar(stat="identity")

#

feat.weights.final.summed = randomLasso(B,q,importance)

feat.weights.final = vector(mode="double",length=p+1)

# 2c

# Compute the final estimator ??^j of ??j by ??^j=B???1???Bb2=1??^(b2)j.

for(i in 1:p+1){

 feat.weights.final[i] = sum(feat.weights.final.summed[,i])/B

}

qplot(y=feat.weights.final,xlab="features", ylab="weight") + labs(title="Weights")
+geom_bar(stat="identity")


cv.X = as.matrix((data.feat))


preds = cv.X %*% feat.weights.final[-(p+1)] + feat.weights.final[p+1]

preds = as.numeric(preds>0.5)


conftable = table(preds,as.matrix(data.y))

mcr = 1 - sum(diag(conftable))/sum(conftable)

conftable

mcr
```

Assignment 2

```
library("partykit")

library("sandwich")

library("ggplot2")


data <- read_csv2("women.csv")

# vignette("mob", package = "partykit")


#1

mob.fit <- function(y, x, start = NULL, weights = NULL, offset = NULL) {

  lm(y ~ x^2, start = start, offset=offset, weights=weights)

}


mob.tree <- mob(`Blood systolic` ~ height + weight |height + weight

            , data = data, fit = mob.fit, control = mob_control(minsize=5000))

plot(mob.tree)

prediction = predict(mob.tree,type="response")


ggplot(aes(x=weight,y=height),data=data) + geom_point() + geom_raster(aes(fill=prediction))
+scale_fill_distiller(palette = "Spectral")
```

Assignment 3

```
RNGversion('3.5.1')

library(readr)

library(ggplot2)



#calculate prediction for input x

frwProp = function(x){

 z_j = tanh(w_j * x + b_j)

 y = sum(w_k * z_j) + b_k

 return(y)

}


#add error to errorArr from data by

# calculating sum-of-squares

calcError = function(errArr,dat){

 for (j in 1:nrow(dat)) {

  y = frwProp(dat[j,1])

  errArr[i] = errArr[i] + (y - dat[j,2])^2

 }

 return(errArr)

}


#Predict and plot

predAndPlot = function(dat){


 pred = vector(length=nrow(dat))

 var = vector(length=nrow(dat))

 for(n in 1:nrow(dat)) {

  z_j <-tanh(w_j * dat[n,1] + b_j)

  y_k <-sum(w_k * z_j) + b_k
```

```
   pred[n] <- y_k
   var[n] = dat[n,]$Var
  }

  qplot(x=var,y=pred,col="red") + geom_point(aes(x=Var,y=Sin),data=dat,col="blue")
}



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)
w_j <- runif(10, -1, 1)
b_j <- runif(10, -1, 1)
w_k <- runif(10, -1, 1)
b_k <- runif(1, -1, 1)

l_rate <- 1/nrow(tr)^2
n_ite = 5000 #more n_ite will make better predictions, ex 20000.
error <- rep(0, n_ite)
error_va <- rep(0, n_ite)
for(i in 1:n_ite) {
  # error computation: Your code here

  #calculate errors for current iteration
  error_va = calcError(error_va,va)
  error = calcError(error,tr)
```

```
cat("i: ", i, ", error train: ", error[i]/2, ", error_va: ", error_va[i]/2, "\n")
flush.console()



for(n in 1:nrow(tr)) {
  # 1. forward propagation: Your code here
  z_j = tanh(w_j * tr[n,1] + b_j)
  y_k = b_k + sum(w_k * z_j)
  # backward propagation: Your code here

  #error from forward propagation
     err = y_k - tr[n,2]

  #3. Brackprop, compute:

  # Output zj from hidden layer depending by activation function zj = tan(A_j)
  dZj_dAj = (1-z_j^2)
  # hidden layer term in activation function by weight from x to node j
  dAj_dWj = tr[n,1]
  #prediction y by output from hidden layer
  dY_dZj = w_k
  #sum-square-error function by prediction
  dE_dY = err

  #prediction by weight from last layer
  dY_dWk = z_j
  #prediction on intercept for last node layer
  dY_dBk = 1
  #acitivation from hidden layer on hidden layer node intercept
  dAj_dBj = 1
```

```
    #calculate error function by weight and intercept with chain-rule
    dE_dWj = dE_dY * dY_dZj * dZj_dAj * dAj_dWj
    dE_dWk = dE_dY * dY_dWk
    dE_dBk = dE_dY * dY_dBk
    dE_dBj = dE_dY * dY_dZj * dZj_dAj * dAj_dBj

    #Perform gradient descent on the weights and intercepts
    w_k = w_k - dE_dWk*l_rate
    w_j = w_j - dE_dWj*l_rate

    b_k = b_k - dE_dBk*l_rate
    b_j = b_j - dE_dBj*l_rate
  }
}

# print final weights and errors
w_j
b_j
w_k
b_k
plot(error/2, ylim=c(0, 5))
points(error_va/2, col = "red")

# plot prediction on training data

predAndPlot(va)
predAndPlot(tr)
```