```
#### Exam 1 - Assignment 2
# Task 1

library(mboost)
bf <- read.csv2("bodyfatregression.csv")
set.seed(1234567890)
# Gradient boosting:
# Gradient boosting is a machine learning technique for regression and
classification problems,
# which produces a prediction model in the form of an ensemble of weak
prediction models,
# typically decision trees. It builds the model in a stage-wise
fashion like other boosting
# methods do, and it generalizes them by allowing optimization of an
arbitrary
# differentiable loss function.
m <- blackboost(Bodyfat_percent~Waist_cm+Weight_kg, data=bf)
plot(Bodyfat_percent ~ Waist_cm, data=bf)
lines(bf$Waist_cm, predict(m), col="red")
mstop(m)
cvf <- cv(model.weights(m), type="kfold")
cvm <- cvrisk(m, folds=cvf, grid=1:100)
plot(cvm)
mstop(cvm)
# Optimal number of iterations, where the combined (mean) squared
error of the model was the least,
# Meaning: 35 iterations of boosting yields the best result, "proven"
using cross validation on 10 different
# sets of observations (the ones with weight 1 were included and the
ones with 0 were not)
# The black line reperestens the mean

## Task 2 and 3
library(kernlab)

# The inner cross-validation can be performed by using the argument
# cross=2 when calling the function ksvm. Hint: Recall that inner
cross-validation estimates the
# error of the different models and selects the best, which is then
evaluated by the outer cross-
# validation. So, the outer cross-validation evaluates the model
selection performed by the inner
# cross-validation
select_model = function(..data){
  n = dim(..data)[1]
  set.seed(12345)
  ids = sample(1:n, n)
  data = ..data[ids,]
  C = c(1,5)
  KERNEL = c("rbfdot", "vanilladot")
  H = c(0.01, 0.05)
  model = list()
  MCR = numeric()
  i = 0
  for(c in C)
    for(kernel in KERNEL){
      if(kernel=="rbfdot"){
        for(h in H){
          svm = ksvm(type~., data=data, kpar=list(sigma=h),
kernel=kernel, C=c, cross=2)
          i = i+1
          model[[i]] = c(c, kernel, h)
          MCR[i] = CV(svm, data)
        }
      }else if(kernel=="vanilladot"){
        svm = ksvm(type~., data=data, kernel=kernel, C=c, cross=2)
        i = i+1
        model[[i]] = c(c, kernel)
        MCR[i] = CV(svm, data)
      }
    }
  print(model)
  print(MCR)
  best = model[[which.min(MCR)]]
  c = as.numeric(best[1])
  kernel = best[2]

  if(best[2]=="rbfdot"){
    h = as.numeric(best[3])
    best.model = ksvm(type~., data=data, kpar=list(sigma=h),
kernel=kernel, C=c, cross=2)
  }else{
    best.model = ksvm(type~., data=data, kernel=kernel, C=c, cross=2)
  }
  return(best.model)
}

CV = function(model, data){
  n = dim(data)[1]
  width = floor(n/2)
  MCS = 0 # missclassified sum, number of missclassified observations
  K = 2
  for (k in 1:K){
    # Select indices
    indices = 1:width + (k-1)*width
    if(k==K) indices = ((k-1)*width+1):n

    # Make predictions
    X.k = data[indices,]
    Yfit.k = predict(model, X.k, type="response")

    # Calculate Error
    Yfit.k = ifelse(Yfit.k=="spam", 1, 0)
    Y.k = ifelse(data[indices,]$type=="spam", 1, 0)
    MCS=MCS+sum(abs(Yfit.k-Y.k)) # 1 if incorrectly predicted, 0 if
correct
  }
  MCR = MCS/n # missclassification rate, sum(missclassified)/n
  return(MCR)
```

```r
}

# import data
data(spam)

# select model
svm = select_model(spam)

## Task 4 and 5
set.seed(1234567890)
rad = runif(50, 0, 10)
sin = sin(rad)
data = data.frame(rad, sin)
train = data[1:25,] # training
valid = data[26:50,] # validation
x.train = t(train$rad)
x.valid = t(valid$rad)
y.train = train$sin
y.valid = valid$sin

# init weights
w.j = runif(10, -1, 1)
b.j = runif(10, -1, 1)
w.k = runif(10, -1, 1)
b.k = runif(1, -1, 1)

n.train = dim(train)[1]
n.valid = dim(valid)[1]

learning.rate = 1/n.train #^2 # 1/25^2 which is in [1/25, 1/25^2]
n.iterations = 5000 # number of training attempts
error.train = numeric()
error.valid = numeric()
error = numeric(n.iterations)

for(n in 1:n.iterations){
  # error train
  z.j.train = tanh(w.j%*%x.train+b.j)
  y.k.train = w.k%*%z.j.train+b.k
  error.train[n] = 1/2*sum((y.k.train-y.train)^2)

  # error train, too slow
  #for(m in 1:nrow(train)) {
  #  z_j <- tanh(w.j * train[m,]$rad + b.j)
  #  y_k <- sum(w.k * z_j) + b.k
  #  error[n] <- error[n] + (y_k - train[m,]$sin)^2
  #}
  # error[n] = error[n]/2

  # error validation
  z.j.valid = tanh(w.j%*%x.valid+b.j)
  y.k.valid = w.k%*%z.j.valid+b.k
  error.valid[n] = 1/2*sum((y.k.valid-y.valid)^2)

  cat("n: ", n, "| error.train: ", error.train[n], "| error.valid: ",
error.valid[n], "\n")
  flush.console()

  for(i in 1:n.train){
    # forward propagation
    z.j = tanh(w.j*x.train[i]+b.j)
    y.k = sum(z.j*w.k)+b.k

    # backward propagation
    d.k = y.k-y.train[i]
    d.j = (1-z.j^2)*w.k*d.k

    # update weigths and biases
    b.k = b.k - learning.rate*d.k
    w.k = w.k - learning.rate*d.k*z.j
    b.j = b.j - learning.rate*d.j
    w.j = w.j - learning.rate*d.j*x.train[i]
  }
}

plot(error.valid[-1], type="l", col=3)
lines(error.train[-1], col=1, type="l")
#lines(error[-1], type="l", col=5)

plot(x.valid, y.k.valid, col="blue")
points(x.valid, y.valid, col="red")
plot(x.train, y.k.train, col="blue")
points(x.train, y.train, col="red")
```