Lab2 block2 Ensemble methods 732A95 ML

Anton Persson antpe404

9 december 2016

Lab2a block 2

In this lab I chose to make all my code in the chunks visible in their specific assignments, why I don't attach all the code in the appendix. I started off by looking at the data to get a feel for it, and how a tree would possibly look.

Assignement 2.1

I used 2/3 of the data as training set and 1/3 as testing set, as instructed. Since the bagging method guarantees that the bagged error is at most the same error as the average individual errors, I did compute the average errors of the bagged models. The code for the computations is shown below.

```r
#Assignment 2.
bodyfat<-read.csv2("data/bodyfatregression.csv", sep=";", header=T)
library(tree)

#2.1
set.seed(1234567890)
#sampling<-sample(1:nrow(bodyfat))
bodyfat_samplad<-bodyfat[sample(1:nrow(bodyfat)),]
bodyfat_tr<-bodyfat_samplad[1:73,] #Traningsset
bodyfat_te<-bodyfat_samplad[74:110,] #testset

#The upper bound
felen_upper<-integer(0)
set.seed(1234567890)
for ( i in 1:100){
  saf<-sample(1:nrow(bodyfat_tr), replace=T)
  bodyfat_bag<-bodyfat_tr[saf,]
  fat_tree<-tree(formula=Bodyfat_percent~., data=bodyfat_bag, split="deviance")
  fitsen<-predict(fat_tree, newdata=bodyfat_te)
  felen_upper[i]<-mean((fitsen-bodyfat_te$Bodyfat_percent)**2)
}

upperbound<-mean(felen_upper)
```

My result is that the upper bound of the squared error of the bagging regression tree is 37.103.

Assignment 2.2

The code to repeat the same task but with cross validation (3 folds) instead of a hold out test data set is shown below.

```r
folds<-3
baggingar<-100
set.seed(1234567890)
folds_data<-suppressWarnings(split(bodyfat, 1:folds))

alla_fel<-matrix(0, nrow=folds, ncol=baggingar)

for (i in 1:folds){
  training<-folds_data[-i]
  del1_train<-data.frame(training[1])
  colnames(del1_train)<-colnames(bodyfat)
  del2_train<-data.frame(training[2])
  colnames(del2_train)<-colnames(bodyfat)
  training<-rbind(del1_train, del2_train)
  testing<-data.frame(folds_data[i])
  colnames(testing)<-colnames(bodyfat)

  for (j in 1:baggingar){

  urval<-sample(1:nrow(training), replace=T)
  bodyfat_bag<-training[urval,]
  fat_tree<-tree(formula=Bodyfat_percent~., data=bodyfat_bag, split="deviance")
  fitsen<-predict(fat_tree, newdata=testing)
  alla_fel[i, j]<-mean((fitsen-testing$Bodyfat_percent)**2)
}
}

upperbound_2<-mean(alla_fel)
```

The results I receive from the code above says that the upper bound when using three folds CV is 40.53.

Assignment 2.3

I assume that it's supposed to be *trees* instead of *tree* in the instructions, i.e. plural. However, I would return a list of all trees created by the bagging regression tree, but with all data used as training data. The code for that is presented below.

```
trees_fulldataset<-list() #empty list to place the trees in.
set.seed(1234567890)

for ( i in 1:100){
  saf<-sample(1:nrow(bodyfat), replace=T)
  bodyfat_bag<-bodyfat[saf,]
  fat_tree<-tree(formula=Bodyfat_percent~., data=bodyfat_bag, split="deviance")
  trees_fulldataset[[i]]<-fat_tree
}
```

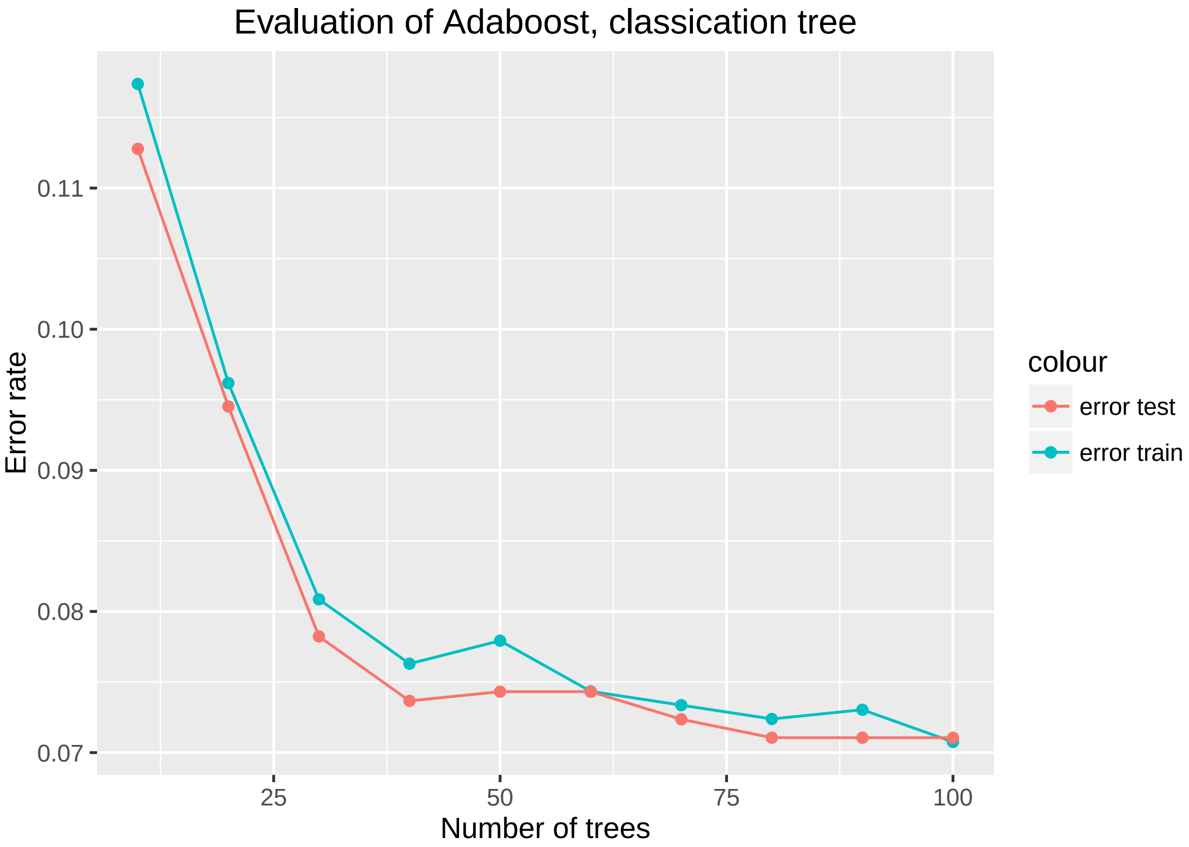
As seen in the code above, I put every single tree in different elements of a list. The list trees_fulldataset thus contains 100 trees. I present one of them, the third tree, just to show how what it looks like.

```
## node), split, n, deviance, yval
##       * denotes terminal node
##
##  1) root 110 10880.00 22.90
##    2) Waist_cm < 95.45 63  2378.00 16.21
##      4) Waist_cm < 84.05 23   465.70 11.57 *
##      5) Waist_cm > 84.05 40  1132.00 18.88
##       10) Weight_kg < 86.65 26   425.50 20.69 *
##       11) Weight_kg > 86.65 14   461.50 15.50
##         22) Waist_cm < 90.5 6    58.83 10.83 *
##         23) Waist_cm > 90.5 8   174.00 19.00 *
##    3) Waist_cm > 95.45 47  1893.00 31.87
##      6) Waist_cm < 104.8 25   627.80 28.08
##       12) Weight_kg < 83.5 5   107.20 32.60 *
##       13) Weight_kg > 83.5 20   393.00 26.95 *
##      7) Waist_cm > 104.8 22   497.30 36.18
##       14) Waist_cm < 109.2 17   323.10 34.76 *
##       15) Waist_cm > 109.2 5    24.00 41.00 *
```

Assignment 4 Adaboost

The first task is the evualute the Adaboost algorithm and it's performance in classification trees. As in assignment 2, I use 2/3 as training data and 1/3 as test data. The required plot is shown below the code that produces it.

```r
library(mboost)
library(randomForest)
library(ggplot2)

spam<-read.csv2("data/spambaselab2b2.csv", sep=";", header=T)
spam$Spam<-as.factor(spam$Spam)
set.seed(1234567890)
spam_samplad<-spam[sample(1:nrow(spam)), ]
spam_tr<-spam_samplad[1:round((2/3)*nrow(spam)), ]
spam_te<-spam_samplad[-(1:round((2/3)*nrow(spam))), ]

sekvens<-seq(10,100, 10)
training_errors<-integer()
test_errors<-integer()
index<-1
for (i in sekvens){
  modellen_ct<-blackboost(Spam~., data=spam_tr, family=AdaExp(),  control=boost_control(mstop=i))

  tejbell_train<-table(pred=predict(modellen_ct, newdata= spam_tr, type="class"), truth=spam_tr$Spam)
  training_errors[index]<-1-sum(diag(tejbell_train))/sum(tejbell_train)

  tejbell_test<-table(pred=predict(modellen_ct, newdata= spam_te, type="class"), truth=spam_te$Spam)
  test_errors[index]<-1-sum(diag(tejbell_test))/sum(tejbell_test)
  index<-index+1
}

plotredo_ct<-data.frame(cbind(sekvens,training_errors, test_errors))

number_of_trees_plot<-ggplot(data=plotredo_ct)+geom_point(aes(x=sekvens, y=training_errors, col="error t
  geom_line(aes(x=sekvens, y=training_errors, col="error train"))+
  geom_point(aes(x=sekvens, y=test_errors, col="error test"))+
  geom_line(aes(x=sekvens, y=test_errors, col="error test"))+xlab("Number of trees")+
  ylab("Error rate")+ggtitle("Evaluation of Adaboost, classication tree")

number_of_trees_plot
```

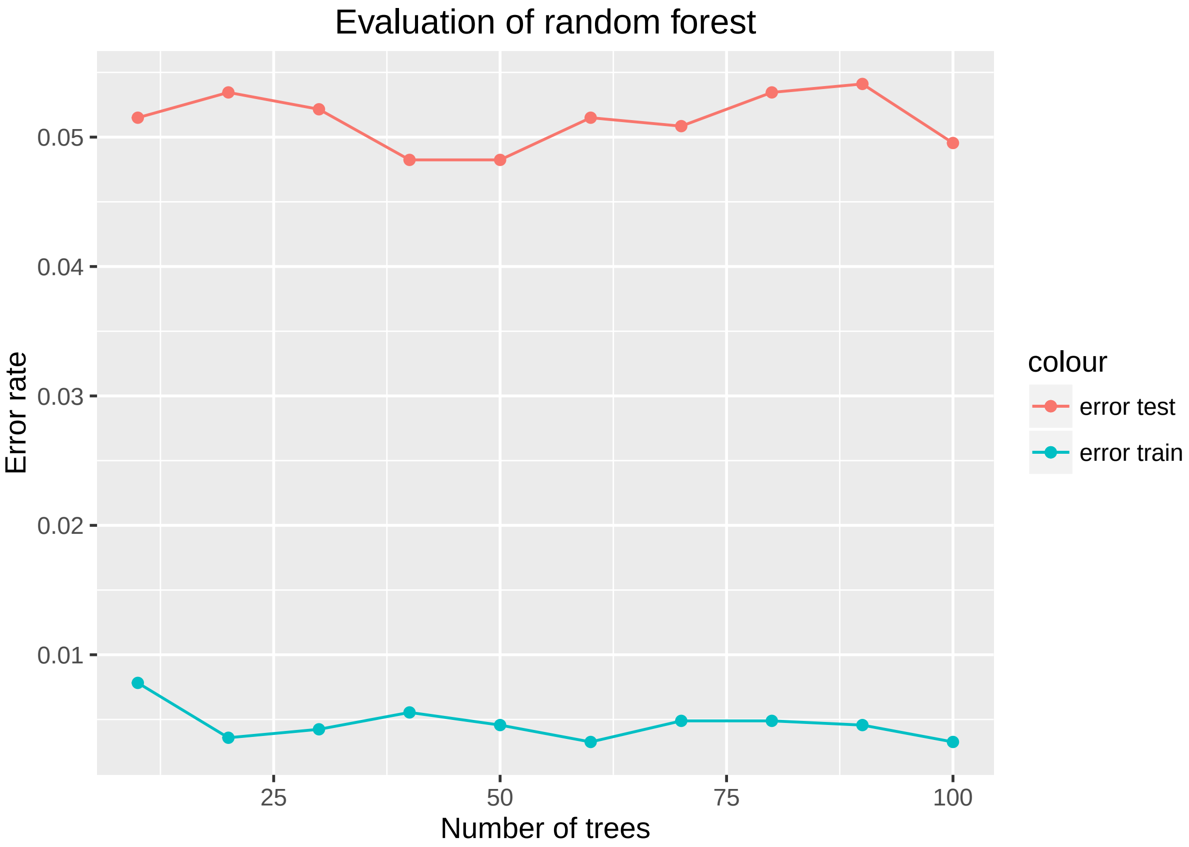


From the plot above I conclude that around 40 trees/iterations seems to be optimal. More iterations than that only improves the performance barely. According to the *elbow* I would say 40 is a good number of iterations in this case. Note that iterations is represented by “Number of trees” in the plot, since every iteration generates a tree.

Assignment 4 random forest

The second task is the evualute the performance of random forests. I still use 2/3 as training data and 1/3 as test data. The required plot is shown below the code that produces it.

```r
sekvens<-seq(10,100, 10)
training_errors_rf<-integer()
test_errors_rf<-integer()
index<-1
for (i in sekvens){
  modellen_rf<-randomForest(Spam ~ ., data=spam_tr, ntree=i, norm.votes=FALSE)

  tr_tab<-table(predict(modellen_rf, newdata= spam_tr, type="class"), spam_tr$Spam)
  training_errors_rf[index]<-1-sum(diag(tr_tab))/sum(tr_tab)

  test_tab<-table(predict(modellen_rf, newdata= spam_te, type="class"), spam_te$Spam)
  test_errors_rf[index]<-1-sum(diag(test_tab))/sum(test_tab)
  index<-index+1
}
#training_errors_rf
#test_errors_rf

plotredo_rf<-data.frame(cbind(sekvens,training_errors_rf, test_errors_rf))

ggplot(data=plotredo_rf)+geom_point(aes(x=sekvens, y=training_errors_rf, col="error train"))+
  geom_line(aes(x=sekvens, y=training_errors_rf, col="error train"))+
  geom_point(aes(x=sekvens, y=test_errors_rf, col="error test"))+
  geom_line(aes(x=sekvens, y=test_errors_rf, col="error test"))+xlab("Number of trees")+
  ylab("Error rate")+ggtitle("Evaluation of random forest")
```



The test error seems to be minimized at 40 iterations in this case, the random forest, as well. After 40 trees, the test error actually increases in this case. Looking at the training error, I conclude that the improvement almost stops after 20 trees. I suppose that means that the model manages to model the training data as good as it gets after 20 trees already.

To compare the two I conclude that the random forest seems to be more stable at low number of iterations. The error rate for the random forest are quite low already at 10 iterations, meanwhile the Adaboost classification tree improves a lot from 10 to 40 iterations.

Lab2b block 2

The second part of the lab, lab2b, is about the EM-algorithm.

Assignment 1

I used the attached template to implement the EM algorithm, in this case with a mixture of multivariate Bernoulli distributions. The code and the plot for how the log likelihood changes over number of iterations are shown below.

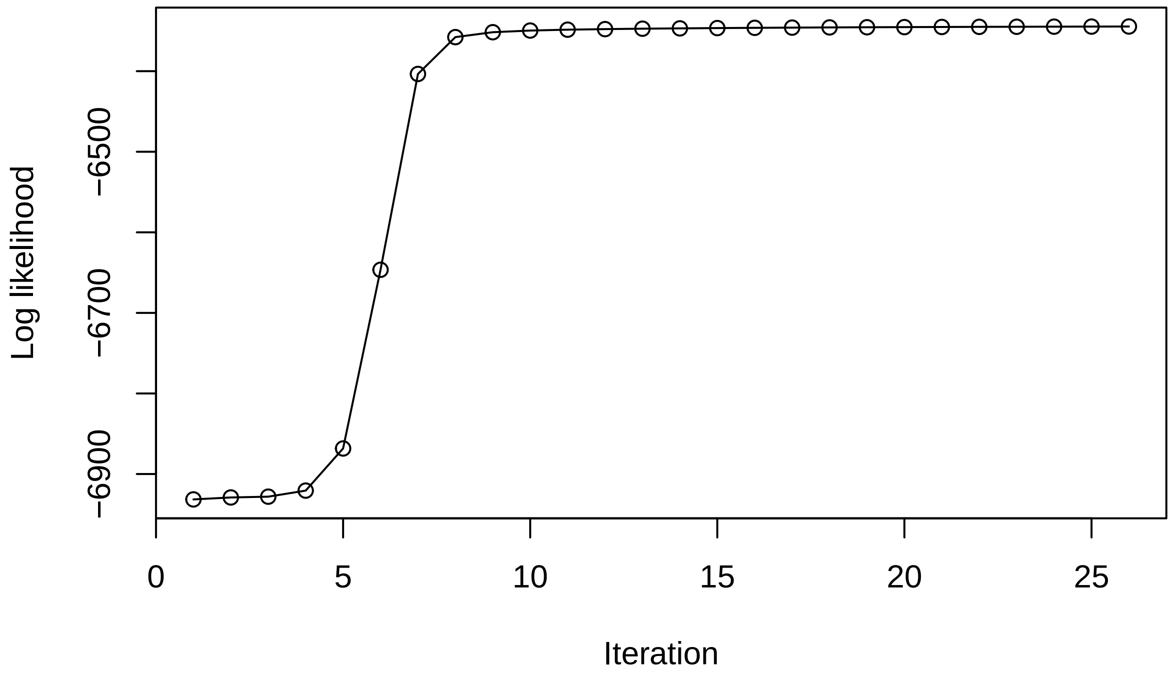
```
set.seed(1234567890)
max_it <- 100 # max number of EM iterations
min_change <- 0.1 # min change in log likelihood between two consecutive EM iterations
N=1000 # number of training points
D=10 # number of dimensions
x <- matrix(nrow=N, ncol=D) # training data
true_pi <- vector(length = 3) # true mixing coefficients
true_mu <- matrix(nrow=3, ncol=D) # true conditional distributions
true_pi=c(1/3, 1/3, 1/3)
true_mu[1,]=c(0.5,0.6,0.4,0.7,0.3,0.8,0.2,0.9,0.1,1)
true_mu[2,]=c(0.5,0.4,0.6,0.3,0.7,0.2,0.8,0.1,0.9,0)
true_mu[3,]=c(0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5,0.5)
#plot(true_mu[1,], type="o", col="blue", ylim=c(0,1))
#points(true_mu[2,], type="o", col="red")
#points(true_mu[3,], type="o", col="green")

# Producing the training data
for(n in 1:N) {
  k <- sample(1:3,1,prob=true_pi)
  for(d in 1:D) {
    x[n,d] <- rbinom(1,1,true_mu[k,d])
  }
}
K=3 # number of guessed components
z <- matrix(nrow=N, ncol=K) # fractional component assignments
pi <- vector(length = K) # mixing coefficients
mu <- matrix(nrow=K, ncol=D) # conditional distributions
llik <- vector(length = max_it) # log likelihood of the EM iterations
# Random initialization of the paramters
pi <- runif(K,0.49,0.51)
pi <- pi / sum(pi)
for(k in 1:K) {
  mu[k,] <- runif(D,0.49,0.51)
}
#pi
#mu



for(it in 1:max_it) {

  #plot(mu[1,], type="o", col="blue", ylim=c(0,1))
  #points(mu[2,], type="o", col="red")
```

```
#points(mu[3,], type="o", col="green")
#points(mu[4,], type="o", col="yellow")
#Sys.sleep(0.5)

bern<-matrix(1,nrow=N, ncol=K)
for (k in 1:K){
  for (n in 1:N){
    for (d in 1:D){
      #slide 6
      bern[n, k]<-bern[n,k]*(mu[k,d]**(x[n,d])) * ((1-mu[k,d])**(1-x[n,d]))
    }
  }
}

# E-step: Computation of the fractional component assignments
#Your code here
#Compute p(znk given x n,mu,pi) for all n
#slide 9
for (k in 1:K){
  for (n in 1:N){
    z[n, k]<-bern[n,k] * pi[k] / sum(bern[n, ]*pi)
  }
}
#Här är divisionen för p(znk given x n,mu,pi) enl slide 9.

#slide8
secondpart<-integer(0)
vector<-integer(0)
firstpart<-integer(0)
for (n in 1:N){
  for (k in 1:K){
    firstpart[k]<-pi[k]*(bern[n,k])

  }
  secondpart[n]<-sum((firstpart))

}

llik[it]<-sum(log(secondpart))


#cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
flush.console()
# Stop if the lok likelihood has not changed significantly
# Your code here
if (it > 1){
  if ((abs(llik[it]-llik[it-1]))<min_change){
    cat("The likelihood doesn't change enough to continue iterating")
    break
  }
}
#M-step: ML parameter estimation from the data and fractional component assignments
# Your code here
```

```
  pi <- colSums(z) / nrow(z)
  #slide 10
  for (k in 1:K){
    for (dim in 1:D){
      mu[k,dim] <- sum( z[,k]*x[,dim] )/sum( z[,k] )
    }
  }
}
```

```
## The likelihood doesn't change enough to continue iterating
```

```
#pi
#mu
plot(llik[1:it], type="o", ylab="Log likelihood", xlab="Iteration")
```


Log likelihood
Iteration