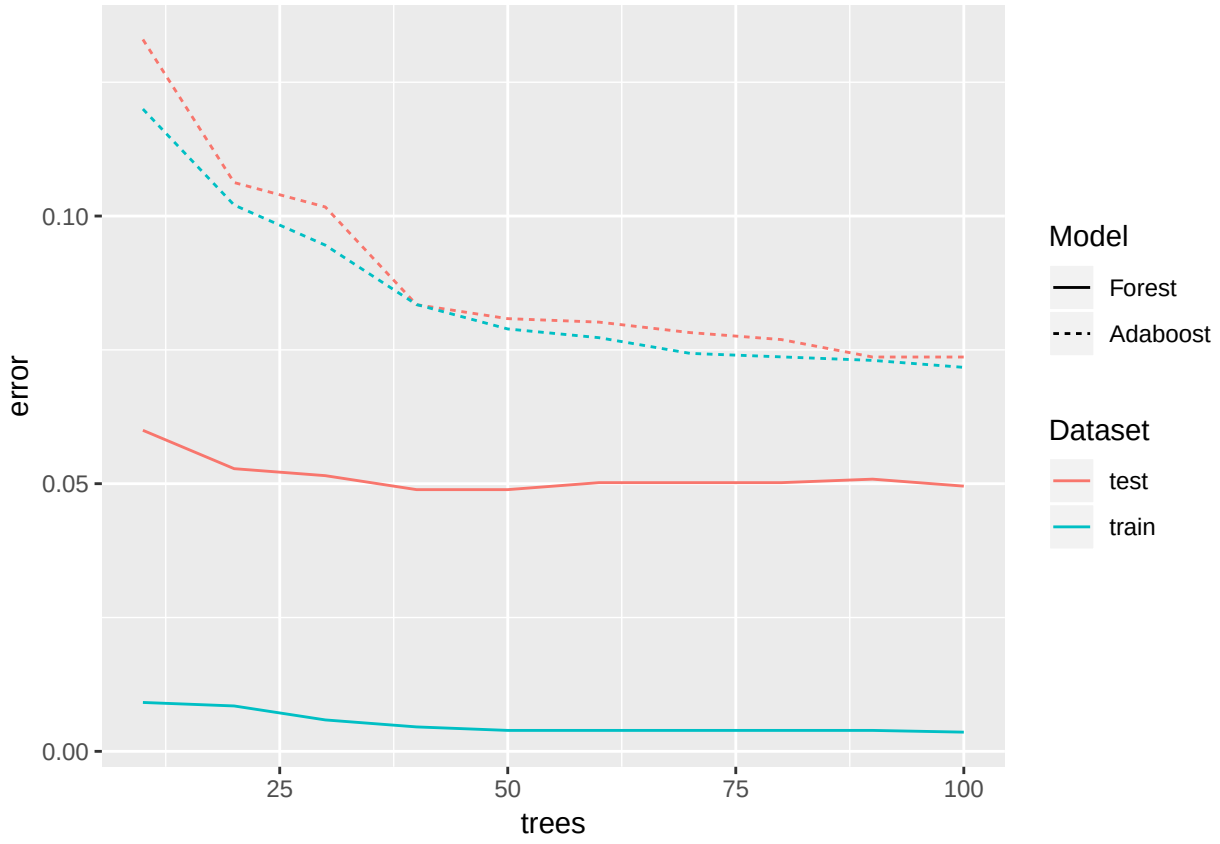
Lab1_Block2

Andreas Stasinakis

November 26, 2018

1) ENSEMBLE METHODS



Analysis

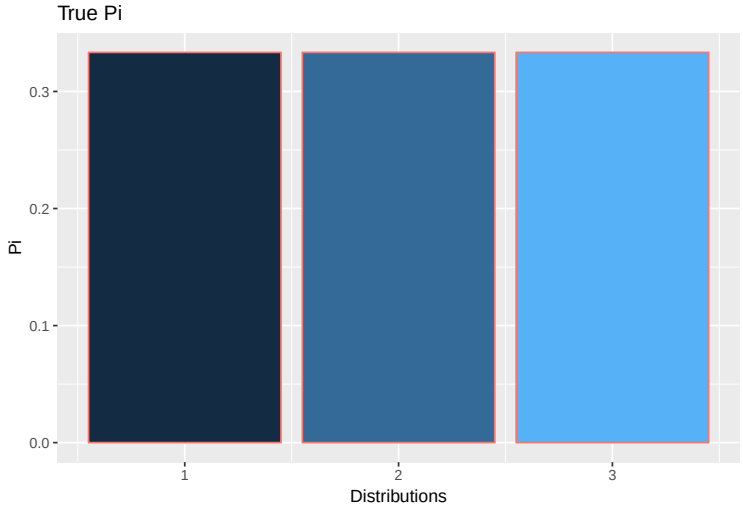
In this task, we perform two different ensemble methods for the same data, called Adaboost classification and random forest. We estimate the error rates for each method for both training and test data and finally we plot the number of trees used in each model vs the error rates for the models. Although both methods try to converge weak learners to strong learners, they target different things and this is the reason why the results bellow for each model are different. So every time that an observation misclassified, Adaboost change it weight, as a result to make predictions based on the weight. On the contrary, random forests combine many difference tries, which may not fit the data well, but with this way the predictions will be much more accurate.

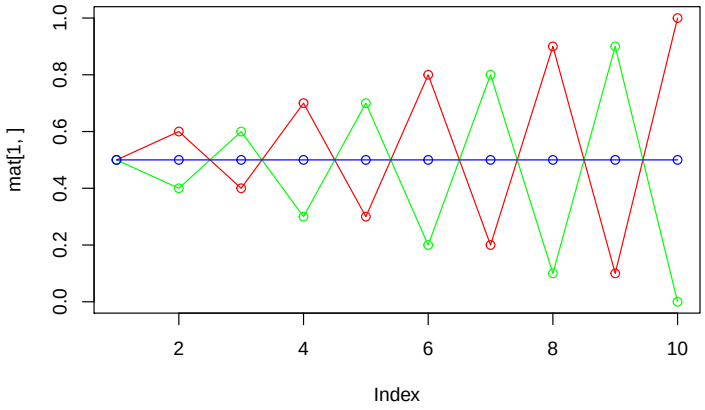
From the plot, one can easily observe that regardless of the number of trees, Adaboost methods has always higher error rate both for training and test data. For Adaboost, the difference between training and test data is not that important. More specific, Adaboost's error rates start at around 0.13 for the test and 0.12 for the train and it is reducing exponentially as the number of trees increases. It can be also said that for more than 70 trees, Adaboost's error seems to be stabilized at around 0.07. On the contrary, for the random

forest method, the test error rate starts at approximately 0.06 for 10 trees, decreases slightly until we use 30 trees and finally it fluctuates with a very low range between 0.05 and 0.055. The random forest's error rate for the training data, starts close to 0.01, slightly decreases until we use 30 trees and after that stabilized close to 0.005. In conclusion, the performance of the random forest methods is always more efficient than the Adaboost regardless of the number of trees we fit the model. Moreover , we can choose to fit the model for more than 30 trees using random forest and the difference in the error rate between the two methods would probably be huge. Therefore, the best model should be a random forest using more than 30 trees.

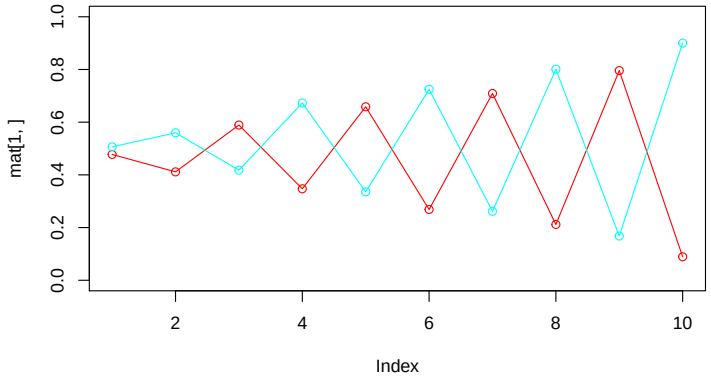
Task 2

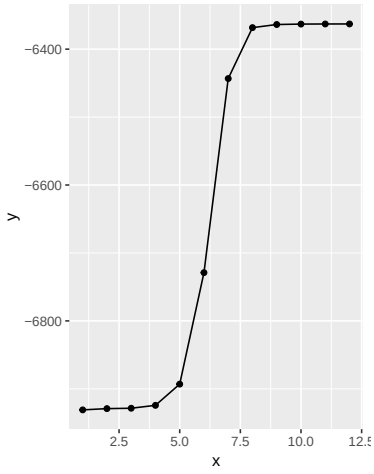
EM Algorithm

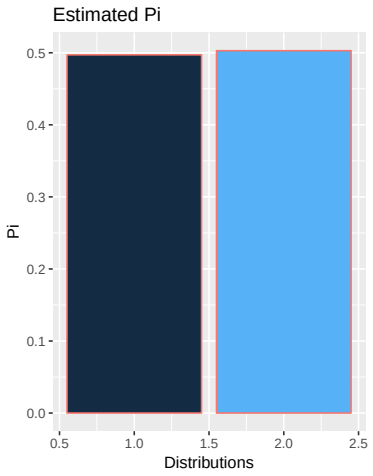




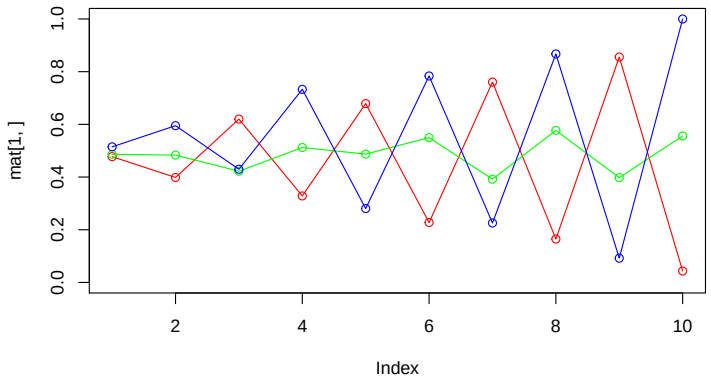
EM for K = 2

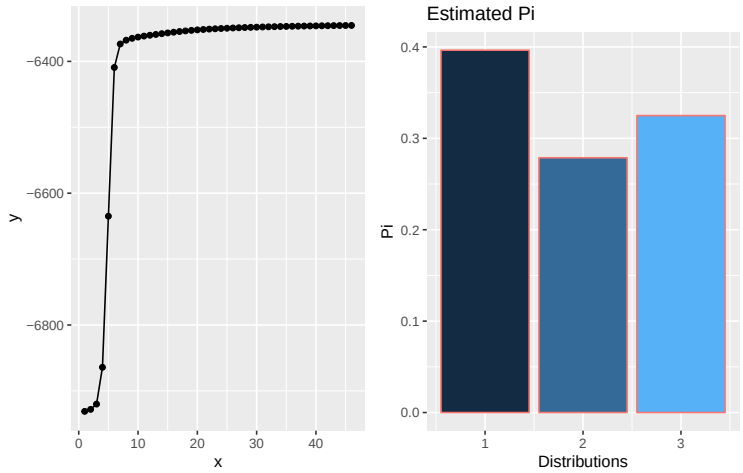




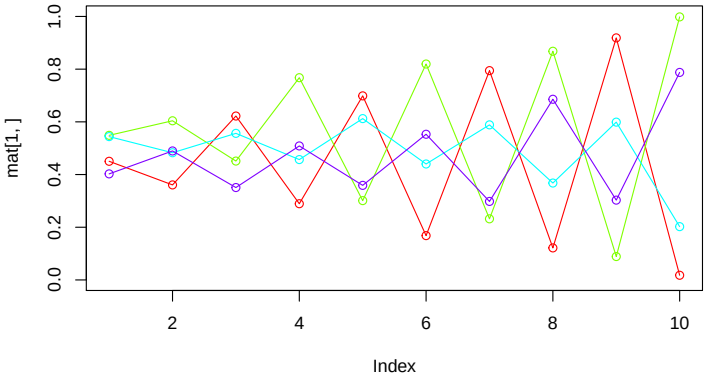


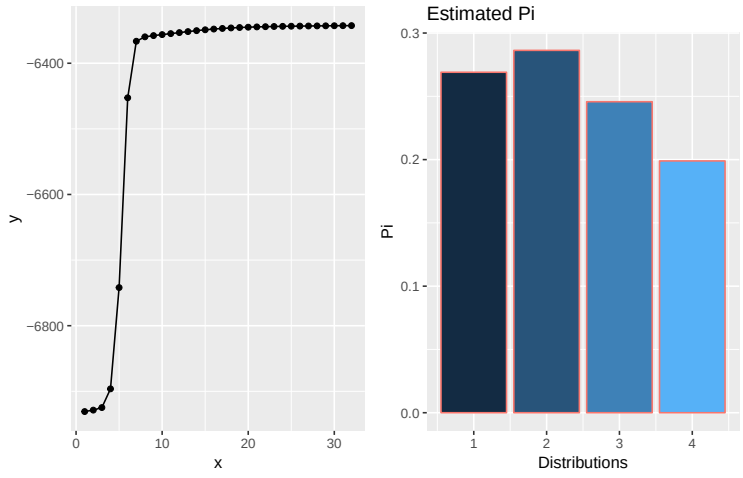
EM for K = 3




Estimated Pi
−6400
−6600
−6800
y
0
10
20
30
40
x
0.4
0.3
0.2
0.1
0.0
Pi
1
2
3
Distributions

EM for K = 4


mat[1,]
0.0
0.2
0.4
0.6
0.8
1.0
2
4
6
8
10
Index


Estimated Pi
−6400
−6600
−6800
y
0
10
20
30
x
0.3
0.2
0.1
0.0
Pi
1
2
3
4
Distributions

Analysis

First of all we have to mention that in real life problems we will not know the distributions which the data belongs to and we will use the estimators to make predictions for this. In this special case, supposing that we do not know the number of distributions, we can be sure which K to choose from the information we have. More specific, for $K = 2$, the probabilities are almost the same and the two μ values follow the same trend as the first two true $\mu's$. However, the last true μ is always stable with value 0.5 and this is the reason why the plot for $K = 2$ does not seem wrong. So from this information is difficult to reject $K = 2$.

For $k = 3$ there is a logical split between the populations, with higher probability the first one at around 0.4 and the other two close to 0.3. All the components had estimated correctly with a high percentage and the μ plot looks like the plot for the true μ.

Finally, for $K = 4$, we can observe that while the probabilities for each component are close each other, the μ values have huge differences. More specific, we can observe to different trend, one for the first and the third μ and one for the second and fourth. This plot is the only one we can easily reject because it is clear that the $\mu's$ for the third and the fourth component follow totally different trends, trying to "translate" the third true μ value.

Appendix

Appendix

```
knitr::opts_chunk$set(echo = FALSE, message = FALSE, warning = FALSE, error = FALSE, fig.pos = "h")
library("mboost")
library("randomForest")
library("ggplot2")
library("gridExtra")

#load the data
sp <- read.csv2("../dataset/spambase.csv")
sp$Spam <- as.factor(sp$Spam)

# Spliting the data to training and test
n = dim(sp)[1]
set.seed(12345)
id = sample(1:n, floor(n*(2/3)))
train = sp[id, ]
test = sp[-id, ]
# Fit the model for different number of trees(10,20,...,100)
misclass_rf_test = c()
misclass_rf_train = c()

for (i in seq(from = 10, to = 100,by = 10)) {

  set.seed(1627)

  # fit the model
  forest = randomForest(formula = Spam ~., data = train, ntree = i)

  #predict for training data
  predict_train = predict(forest,newdata = train)

  #confusion mattrix for training data
   conf_table_train = table(predict_train , train$Spam)

  #Misclassification rate for training data
  misclass_rf_train = c(misclass_rf_train, (conf_table_train[1,2] + conf_table_train[2,1])/sum(conf_tabl

  # Predict for test data
  predict_test = predict(forest,newdata = test)

  # Confusion matrix
   conf_table_test = table(predict_test , test$Spam)

  # Misclassification rate for test data
  misclass_rf_test= c(misclass_rf_test, (conf_table_test[1,2] + conf_table_test[2,1])/sum(conf_table_tes
}

#Data frame with all misclassification rates for dif. number of trees
rf_errors_test = data.frame("trees" = seq(from = 10,to = 100,by = 10),"error" = misclass_rf_test,"Datase
rf_errors_train = data.frame("trees" = seq(from = 10,to = 100,by = 10),"error" = misclass_rf_train,"Data
```

```
rf_errors = rbind(rf_errors_test,rf_errors_train)


#Adaboosting Model
# Loop for all different trees
misclass_boost_test = c()
misclass_boost_train = c()

for (i in seq(from = 10, to = 100,by = 10)){

  #fit the model with i number of trees
  boost = blackboost(formula = Spam ~., data = train, family = AdaExp(), control = boost_control(mstop =

  #Misclassification rate for training data
  pred_train = predict(object = boost, newdata = train, type = "class")
  conf_boost_table_train = table( pred_train , train$Spam)
  misclass_boost_train = c(misclass_boost_train, (conf_boost_table_train[1,2] +
                                                    conf_boost_table_train[2,1])/sum(conf_boost_table_trai

  #prediction for Adaboost
  predict_f = predict(boost, newdata = test,type = "class")

  #confusion matrix for test data
  conf_boost_table = table( predict_f , test$Spam)

  #Misclassification rates for test data
  misclass_boost_test = c(misclass_boost_test, (conf_boost_table[1,2] + conf_boost_table[2,1])/sum(conf_
}

#Data frame with all misclassification rates for dif. number of trees
boost_errors_test = data.frame("trees" = seq(from = 10,to = 100,by = 10), "error" = misclass_boost_test
boost_errors_train = data.frame("trees" = seq(from = 10,to = 100,by = 10), "error" = misclass_boost_trai
boost_errors = rbind(boost_errors_test,boost_errors_train)


#All the data we need for the plot (n_trees,errors,dataset,method)
errors = rbind(rf_errors,boost_errors)

# Plot the errors for random Forest and Adaboost
ggplot(errors) + geom_line(aes_string(x="trees",
                                      y = "error",
                                      colour="Dataset",
                                      linetype="Model"))


#Function for the plot of estimated pi
pi_plot = function(df){
  ggplot(df) +
  geom_col(mapping = aes(x = df$x, y = df$y, fill = x, color = "white")) +
  labs(title = "Estimated Pi",x = "Distributions", y = "Pi" ) +
  theme(legend.position = "none")

}
```

```r
#function for the plot of estimated mu
mu_plot = function(mat){
  my_rainbow = rainbow(n = nrow(mat))
  mu_plot = plot(mat[1,], type="o", col= my_rainbow[1], ylim=c(0,1))
    for (i in 2:nrow(mat)) {
      points(mat[i,], type="o",col  = my_rainbow[i])
    }
}

#Generate the data
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 for true pi and true mu
df = data.frame(x = c(1:3), y = true_pi)
ggplot(df) +
  geom_col(mapping = aes(x = df$x, y = df$y, fill = x, color = "white")) +
  labs(title = "True Pi",x = "Distributions", y = "Pi" ) +
  theme(legend.position = "none")
mu_plot(true_mu)

# Producing the real 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])
  }
}

# function for EM algorithm
# Input K =  number of components
# output plots for best estimated pi,mu and maxlikelihood
EM_algorithm = function(K){
  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 parameters
  pi <- runif(K,0.49,0.51)
  pi <- pi / sum(pi)
  for(k in 1:K) {
```

```
  mu[k,] <- runif(D,0.49,0.51)
}

# Run the EM for max_it iterations
for(it in 1:max_it) {
  Sys.sleep(0.5)

  # E-step: Computation of the fractional component assignments
  # Loop for all the observations
  for (i in 1:N) {
    temp_z = c()

    # Loop for all the components
    for (k in 1:K) {

    # Calculate Bernulli prob.
    temp_mu = mu[k,]^x[i,]
    temp_mu2 = (1- mu[k,])^(1-x[i,])
    temp_z = c(temp_z, prod(temp_mu,temp_mu2))

    }
    z[i,] = (pi*temp_z) / sum(pi*temp_z)
  }

  #maximum likelihood
  # we can to the calculations and we will use matrix multiplucation
  # Probability  for each bernulli
  p_mu = exp(x %*% t(log(mu)) + (1-x) %*% t(log(1-mu)))

  # convert pi vector to a matrix
  ml_pi = matrix(pi,byrow = TRUE,nrow = 1000, ncol = K)

  # calculate the max likelihood
  llik[it] = sum(log(rowSums(p_mu * ml_pi)))



  # print the log likelihood for every iteration
  #cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
  #flush.console()

  # Stop if the lok likelihood has not changed significantly
  if (llik[it] - llik[it-1] < min_change && it > 2 ){
    break()
  }

  #M-step: ML parameter estimation from the data and fractional component assignments
  prob = c(colSums(z))
  pi = prob/N
  mu = (t(z) %*% x)/prob

}
```

```
  #Plot for the estimated mu
  mu_plot(mu)


  #Plot for the maximum likelihood
  likelihood = data.frame(x = 1:it,y = llik[1:it])
  #likelihod_plot = plot(llik[1:it], type="o")
  likelihood_plot = ggplot(likelihood,aes(x = x, y = y)) + geom_line() + geom_point()


  #Bar plot for estimated pi
  df_pi = data.frame(x = c(1:K), y = pi)
  plot_pi = pi_plot(df = df_pi)

  grid.arrange(likelihood_plot, plot_pi, ncol = 2)



}
EM_algorithm(2)

EM_algorithm(3)

EM_algorithm(4)
```