Untitled

Prudhvi Peddmallu

28 November 2018

1. ENSEMBLE METHODS

Adaboost classification trees

The file spambase.csv contains information about the frequency of various words, characters, etc. for a total of 4601 e-mails. Furthermore, these e-mails have been classified as spams (spam = 1) or regular e-mails (spam = 0). You can find more information about these data at https://archive.ics.uci.edu/ml/datasets/Spambase Your task is to evaluate the performance of Adaboost classification trees and random forests on the spam data. Specifically, provide a plot showing the error rates when the number of trees considered are 10; 20; : : : ; 100. To estimate the error rates, use 2/3 of the data for training and 1/3 as hold-out test data. To learn Adaboost classification trees, use the function blackboost() of the R package mboost. Specify the loss function corresponding to Adaboost with the parameter family. To learn random forests, use the function randomForest of the R package randomForest. To load the data, you may want to use the following code:

```r
#reading trhe data file
spambase <- read.csv2("spambase.csv")
spambase$Spam <- as.factor(spambase$Spam)
```

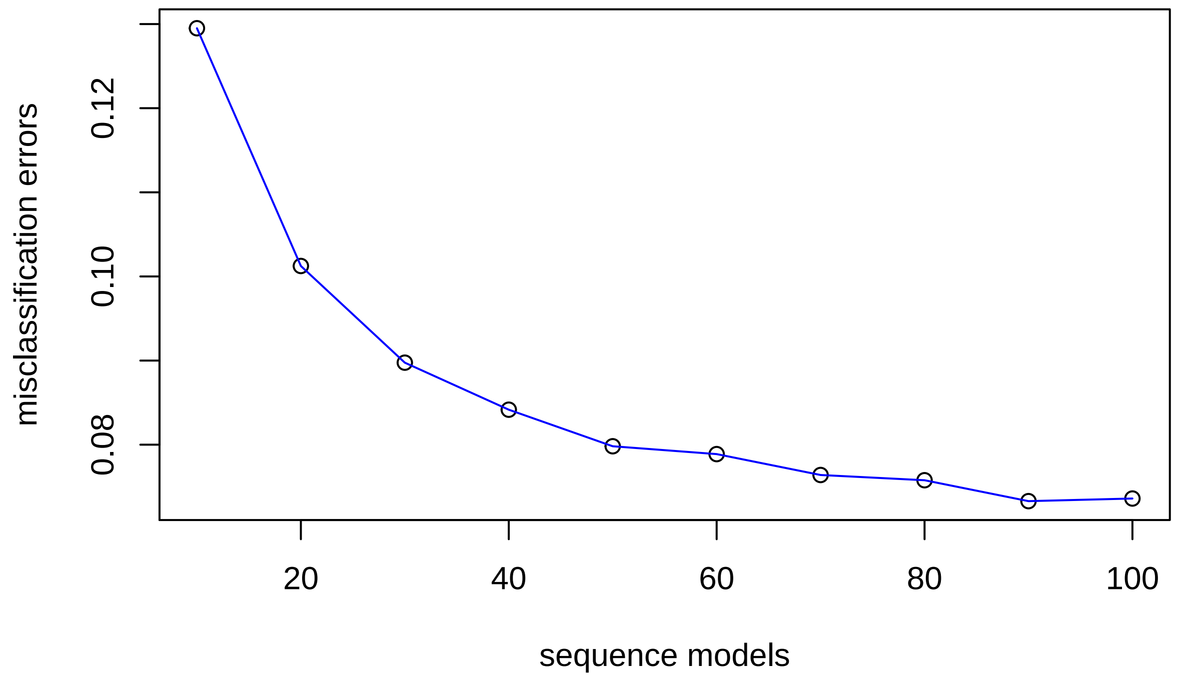
```r
#libraries
library(mboost)
```

```r
#Divideing the data into train(2/3=70%) and test(1/3=30%)
n=dim(spambase)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.7))
train=spambase[id,]
test=spambase[-id,]
```

```r
#Ada boost classification
#loop for sequence to get order 10, 20, 30...100
storeing<- c()
sequences <- c()
count<-0
  for(i in seq(from=10, to=100 ,by =10))
  {
     sequences<-c(i,sequences)
#fitting the model adaboost
spambase_EM_fit <- blackboost(Spam~., data = train, family = AdaExp(),control = boost_control(mstop =i))
#prediction we are giving type = class because the adaboost gives the tree contain the classes
predectionEM_train <- predict(spambase_EM_fit,newdata=train,type='class')
#confusion matrix
confusionmatrix_trainEM = table(train$Spam,predectionEM_train)
# misclassification errors according to i(10,20,30...)
diagonal_trainEM<-diag(confusionmatrix_trainEM)
summ_trainEM<-sum(confusionmatrix_trainEM)
misclassificationrate_trainEM<-1-(sum(diagonal_trainEM)/sum(summ_trainEM))
storeing<-c(misclassificationrate_trainEM,storeing)

}
```

```
#plot the misclassification vs i
{plot(x=sequences,y=storeing,
xlab="sequence models",
ylab="misclassification errors"
)
lines(sequences,storeing,col="blue")}
```



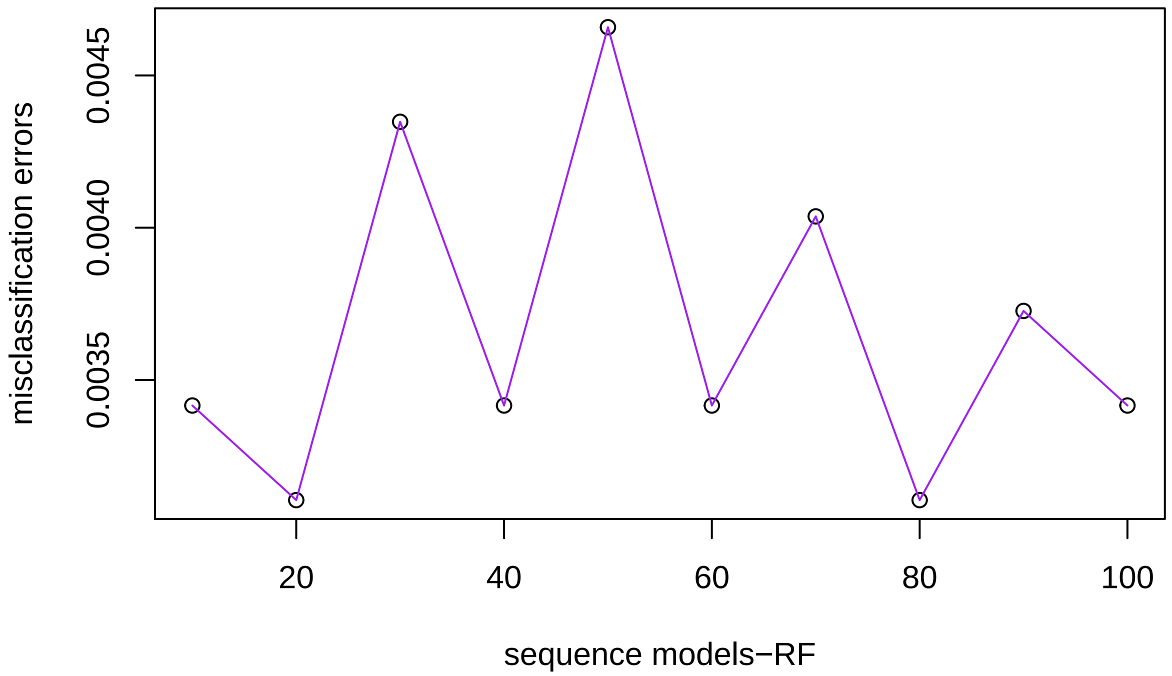
Analysis:-In the ada boost classification we can see in the plot the no of trees decreases and the error is decreases.in the plot the error is stablized from 80 to 100.

Random forests

```
library(randomForest)
```

```
#Divideing the data into train(2/3=70%) and test(1/3=30%)
n=dim(spambase)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.7))
train_RF=spambase[id,]
test_RF=spambase[-id,]
```

```
#Random forest
storeing_RF<-c()
sequences_RF<-c()
count<-0
for(i in seq(from=10,to=100,by=10)){
 sequences_RF<-c(i,sequences_RF)
#fitting the model adaboost
spambase_RF_fit <- randomForest(Spam~., data = train_RF,control = boost_control(mstop =i))
#prediction we are giving type = class because the adaboost gives the tree contain the classes
predectionRF_train <- predict(spambase_RF_fit,newdata=train_RF,type='class')
#confusion matrix
confusionmatrix_trainRF = table(train_RF$Spam,predectionRF_train)
# misclassification errors according to i(10,20,30...)
diagonal_trainRF<-diag(confusionmatrix_trainRF)
summ_trainRF<-sum(confusionmatrix_trainRF)
misclassificationrate_trainRF<-1-(sum(diagonal_trainRF)/sum(summ_trainRF))
storeing_RF<-c(misclassificationrate_trainRF,storeing_RF)
}
#plot the misclassification vs i
{plot(x=sequences_RF,y=storeing_RF,
xlab="sequence models-RF",
ylab="misclassification errors"
)
lines(sequences_RF,storeing_RF,col="purple")  }
```



Analysis:- In the random forest plot we can see the error is continiously increasing when we have less number of trees the error is high compare when we have more number of trees.

2. MIXTURE MODELS-EM ALGORITHM

K=3

```r
#PROFESSOR CODE
# 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)
# # E-step: Computation of the fractional component assignments
# # Your code here
# #Log likelihood computation.
# # Your code here
# cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
# flush.console()
# # Stop if the lok likelihood has not changed significantly
# # Your code here
# #M-step: ML parameter estimation from the data and fractional component assignments
# # Your code here
# }
# pi
# mu
# plot(llik[1:it], type="o")
```

```
#mycode
# 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
# eachValue <- matrix(nrow=N, ncol=K)
# total <- vector(length = N)
# # 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)
#
# # E-step: Computation of the fractional component assignments
# for (i in 1:N) {
```

```
# for (j in 1:k) {
# p <- prod(dbinom(x[i, ], 1, mu[j, ]))
# z[i,j] <- pi[j] * p
# eachValue[i,j] <- z[i,j]
# }
#
# #Using Baye's Theorem
# z[i, ] <- z[i, ] /sum(z[i, ])
# # for calculating log likelihood
# total[i] <- log(sum(eachValue[i,]))
# }
# #Log likelihood computation.
# llik[it] <- sum(total)
# cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
# flush.console()
# # Stop if the lok likelihood has not changed significantly
# if(it > 1 && (llik[it]-llik[it-1]<=min_change)){
# break
# }
# #M-step: ML parameter estimation from the data and fractional component assignments
# for(j in 1:K)
# {
# NK <- sum(z[,j])
# mu[j,] <- (t(z[,j])%*%x)/sum(z[,j])
# pi[j] <- NK/N
# }
# }
# pi
# mu
# plot(llik[1:it], type="o")
# ###
# 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=2 # 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
# eachValue <- matrix(nrow=N, ncol=K)
# total <- vector(length = N)
# # 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)
# # E-step: Computation of the fractional component assignments
# for (i in 1:N) {
# for (j in 1:k) {
# p <- prod(dbinom(x[i, ], 1, mu[j, ]))
# z[i,j] <- pi[j] * p
# eachValue[i,j] <- z[i,j]
# }
#
# #Using Baye's Theorem
# z[i, ] <- z[i, ] /sum(z[i, ])
# # for calculating log likelihood
# total[i] <- log(sum(eachValue[i,]))
# }
# llik[it] <- sum(total)
# cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
# flush.console()
# # Stop if the lok likelihood has not changed significantly
# if(it > 1 && (llik[it]-llik[it-1]<=min_change)){
# break
# }
# #M-step: ML parameter estimation from the data and fractional component assignments
# for(j in 1:K)
# {
# NK <- sum(z[,j])
# mu[j,] <- (t(z[,j])%*%x)/sum(z[,j])
# pi[j] <- NK/N
# }
# }
# pi
# mu
# plot(llik[1:it], type="o")
# 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=4 # 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
# eachValue <- matrix(nrow=N, ncol=K)
# total <- vector(length = N)
# # 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)
# # E-step: Computation of the fractional component assignments
# for (i in 1:N) {
# for (j in 1:k) {
# p <- prod(dbinom(x[i, ], 1, mu[j, ]))
# z[i,j] <- pi[j] * p
# eachValue[i,j] <- z[i,j]
# }
# #Using Baye's Theorem
# z[i, ] <- z[i, ] /sum(z[i, ])
# # for calculating log likelihood
# total[i] <- log(sum(eachValue[i,]))
# }
# #Log likelihood computation.
# llik[it] <- sum(total)
# cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
# flush.console()
# # Stop if the lok likelihood has not changed significantly
# if(it > 1 && (llik[it]-llik[it-1]<=min_change)){
# break
# }
# #M-step: ML parameter estimation from the data and fractional component assignments
# for(j in 1:K)
# {
# NK <- sum(z[,j])
# mu[j,] <- (t(z[,j])%*%x)/sum(z[,j])
# pi[j] <- NK/N
# }
# }
# pi
# mu
# plot(llik[1:it], type="o")
```