Lab1 Block2

Yash Pawar

03/12/2019

1. Ensemble Methods

Adaboost classification Trees

Misclassification rates for Adaboost

Misclassification Rate for adaboost Test data

```
## [1] 0.12516297 0.10691004 0.09387223 0.07953064 0.07757497 0.07431551
## [7] 0.07301173 0.07170795 0.07170795 0.07235984
```

Misclassification rate for adaboost Train data

```
## [1] 0.11998696 0.09781545 0.08933812 0.08151288 0.07662211 0.07336159
## [7] 0.06814477 0.06684056 0.06651451 0.06521030
```

Misclassification rates for random forest

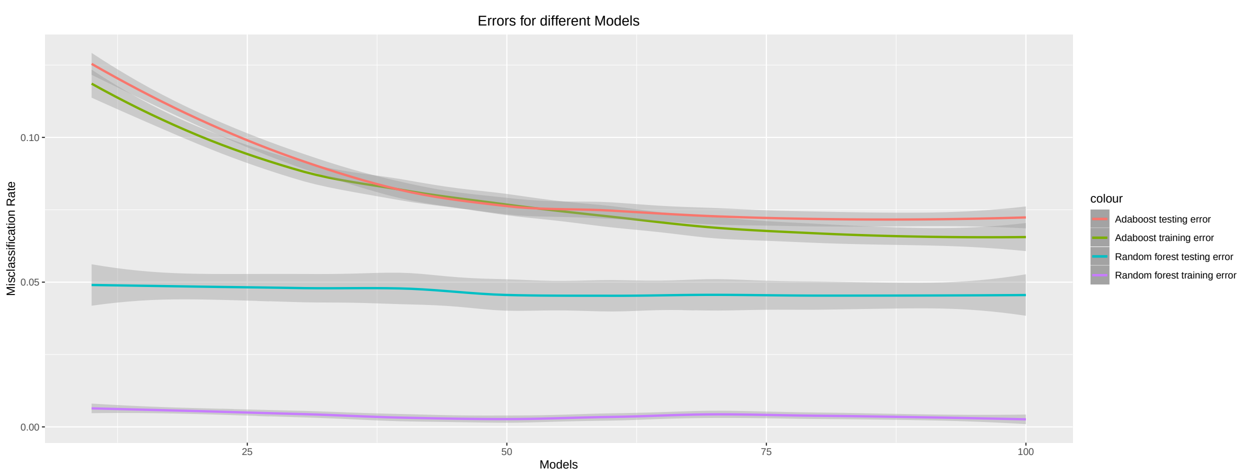
Misclassification rate for Random Forest Test data

```
## [1] 0.04954368 0.04693611 0.04954368 0.04563233 0.04823990 0.04237288
## [7] 0.04758801 0.04563233 0.04367666 0.04628422
```

Misclassification rate for Random Forest Train data

```
## [1] 0.006194979 0.005868927 0.004564721 0.003586567 0.001956309 0.003912618
## [7] 0.004238670 0.003912618 0.002608412 0.002934464
```

Plot of errors for diifernt models of Random forest and Adaboost.



2. Mixture Models

EM algorithm implemetation

The number of iterations for K=2 are:

```
## [1] 16
```

The value of pi:

```
## [1] 0.4981919 0.5018081
```

The value of mu:

```
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.4777136 0.4113065 0.5888317 0.3477062 0.6580979 0.2698303 0.7078467
## [2,] 0.5061835 0.5601552 0.4177868 0.6731171 0.3350702 0.7245255 0.2617664
##           [,8]      [,9]     [,10]
## [1,] 0.2134061 0.7941168 0.0875152
## [2,] 0.8004711 0.1681467 0.9035340
```

The number of iterations for K=3

```
## [1] 62
```

The value of pi:

```
## [1] 0.3259592 0.3044579 0.3695828
```

The value of mu:

```
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.4737193 0.3817120 0.6288021 0.3086143 0.6943731 0.1980896 0.7879447
## [2,] 0.4909874 0.4793213 0.4691560 0.4791793 0.5329895 0.4928830 0.4643990
## [3,] 0.5089571 0.5834802 0.4199272 0.7157107 0.2905703 0.7667258 0.2320784
##           [,8]      [,9]      [,10]
## [1,] 0.1349651 0.8912534 0.01937869
## [2,] 0.4902682 0.4922194 0.39798407
## [3,] 0.8516111 0.1072226 0.99981353
```

The number of iterations for K=4 are:

```
## [1] 66
```

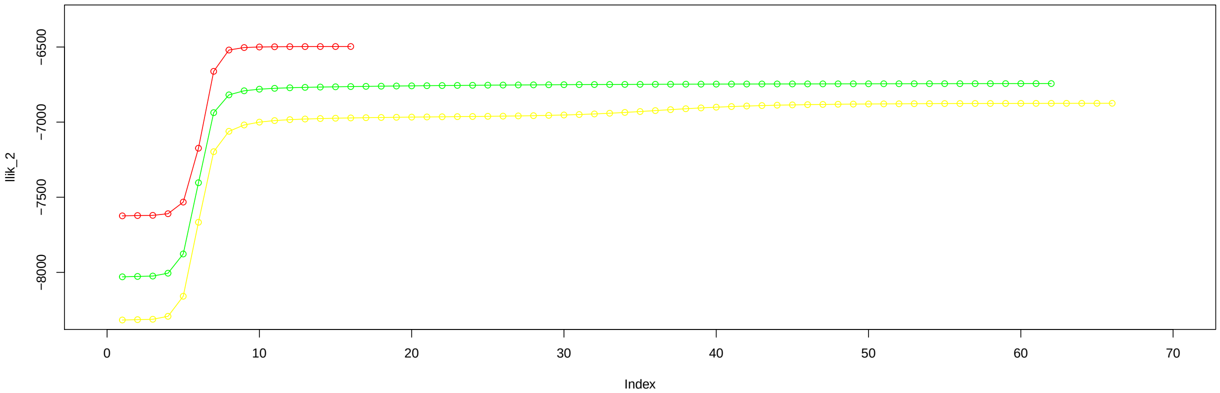
The value of pi:

```
## [1] 0.1614155 0.1383613 0.3609912 0.3392319
```

The value of mu:

```
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.4372908 0.5716691 0.6230114 0.4717152 0.4251232 0.2940734 0.4797605
## [2,] 0.5381955 0.3913346 0.2971686 0.5062848 0.6375272 0.7107583 0.4202372
## [3,] 0.5102441 0.5846281 0.4200464 0.7178717 0.2850900 0.7735833 0.2327656
## [4,] 0.4797762 0.3788928 0.6181216 0.3114748 0.6964392 0.2149967 0.7793732
##           [,8]      [,9]       [,10]
## [1,] 0.4812185 0.5945364 0.500815206
## [2,] 0.5246082 0.3534161 0.384513179
## [3,] 0.8546627 0.1022323 0.999999734
## [4,] 0.1450708 0.8791286 0.005800712
```

The plot of log likelihood iterations for different values of K



Too few components will lead to under fitting of data. However, the iterations for the classification of data in this case will be fewer (As in case of K=2) as compared to the higher number of components. Too many components will lead to higher number of iterations to determine the likelihood of disitribution of data. Also as the number of components increase there will be overfitting of data.

The higher likelihood corresponds to better classification of data as in case of K=2. Too many components will also increase the number of parameters for the mixture model resulting in a complex distribution. so idea is to find a trade-off between the higher likelihood and appropriate number of components to find the distribution of data.

Appendix

```
knitr::opts_chunk$set(echo = TRUE,fig.width=16, fig.height=6)
##Load Packages
library(mboost)
library(randomForest)
library(ggplot2)
sp <- read.csv2("spambase.csv")
sp$Spam <- as.factor(sp$Spam)
sp$Spam
n = dim(sp)[1]
set.seed(12345)
id = sample(1:n, floor(n*(2/3)))
sp_train = sp[id,]
sp_test = sp[-id,]
models = seq(10,100,10)
misclassification_rate_adaboost = numeric()
misclassification_rate_randomforest = numeric()
misc_train_adaboost = numeric()
misc_train_randomforest = numeric()

for (i in 1:10) {
  train_adaboost = blackboost(Spam ~., data = sp_train, family = AdaExp(), control = boost_control(msto
  test_adaboost = predict.mboost(train_adaboost, sp_test, type = "class")
  pred_ada_train = predict.mboost(train_adaboost, sp_train, type = "class")
  #condition_adaboost = ifelse(test_adaboost>0.5,1,0)
  #plot(sp_test, test_adaboost)
  confusion_matrix_adaboost = table(sp_test$Spam, test_adaboost)
  confusion_train_adaboost = table(sp_train$Spam, pred_ada_train)
```

```r
  #misclassification rates for adaboost training and testing data
  misclassification_rate_adaboost[i] = c(1 - sum(diag(confusion_matrix_adaboost))/sum(confusion_matrix_a
  misc_train_adaboost[i] = c(1 - sum(diag(confusion_train_adaboost))/sum(confusion_train_adaboost))

  #Training Random forest
  train_randomforest = randomForest(Spam ~., data = sp_train, ntree = i*10)
  test_randomforest = predict(train_randomforest, sp_test)
  pred_random_train = predict(train_randomforest, sp_train)
  confusion_matrix_randomforest = table(sp_test$Spam, test_randomforest)
  confusion_train_random = table(sp_train$Spam, pred_random_train)

  #Misclassification rates for training and testing for Random forest
  misclassification_rate_randomforest[i] = c(1 - sum(diag(confusion_matrix_randomforest))/sum(confusion_
  misc_train_randomforest[i] = c(1 - sum(diag(confusion_train_random))/sum(confusion_train_random))
  }



plot_data = cbind.data.frame("models" = seq(10,100,10), misclassification_rate_adaboost, misc_train_adab
                             misclassification_rate_randomforest, misc_train_randomforest)

misclassification_rate_adaboost
  misc_train_adaboost

misclassification_rate_randomforest
misc_train_randomforest
ggplot(data = plot_data) +
  geom_smooth(aes(x = models, y = misc_train_adaboost, color = "Adaboost training error")) +
  geom_smooth(aes(x = models, y = misclassification_rate_adaboost, color = "Adaboost testing error")) +
  geom_smooth(aes(x = models, y = misc_train_randomforest, color = "Random forest training error")) +
  geom_smooth(aes(x = models, y = misclassification_rate_randomforest, color = "Random forest testing e
  ylab("Misclassification Rate") + xlab("Models") + ggtitle("Errors for different Models") +
  theme(plot.title = element_text(hjust = 0.5))
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])
```

```r
  }
}
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
# 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
change = 0

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


  for (n in 1:N) {
    phi = c()
    for (k in 1:K) {
      bern_prob = prod((mu[k,]^x[n,]),(1-mu[k,])^(1-x[n,]))


      phi = c(phi, pi[k]*bern_prob)


    }
    z[n,]= phi/sum(phi)

  }


  #
  # ff <- matrix(0, N,K)
  # for (k in 1:K) {
  #   for (n in 1:N) {
  #     #for (d in 1:D) {
  #     # a1 = (mu[k,]^x[n,])
  #     # a2 = (1-mu[k,])^(1-x[n,])
  #     bern_prob[n,] = (mu[k,]^x[n,])*(1-mu[k,])^(1-x[n,])
  #     product_bern[n] = prod(bern_prob[n,])
  #     ff[n,] <- pi * product_bern[n]
  #     z[n,] <- ff[n,k] / sum(ff[n,])
```

```r
  #
  #   }
  # }
  #



  log_prob_z = log((pi[k]*(bern_prob))/sum(pi[k]*(bern_prob)))
  prob_z = exp(log_prob_z)

  #Log likelihood computation.
  # Your code here
  log_lik = 0
  for (n in 1:N) {
    for (k in 1:K) {
      log_lik = log_lik + z[n,k]*(log(pi[k]) + sum(x[n,]*log(mu[k,]) + (1-x[n,])*log(1-mu[k,])))
    }
  }
  llik[it] = log_lik
  cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
  flush.console()
  # Stop if the log likelihood has not changed significantly
  # Your code here

  # if(it>1){
    if(abs(change - llik[it]) < min_change){
      break()
    } else{
      change = llik[it]
    }

  #}
  #M-step: ML parameter estimation from the data and fractional component assignments
  # Your code here
  pi = colMeans(z)
  for (k in 1:K) {
    for (i in 1:D) {
      mu[k,i] = sum(z[,k]*x[,i])/sum(z[,k])
    }
  }
}
llik_2 = llik[1:it]
length(llik[1:it])
pi
mu_2 = mu
mu_2
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
```

```r
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
change = 0

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


  for (n in 1:N) {
    phi = c()
    for (k in 1:K) {
      bern_prob = prod((mu[k,]^x[n,]),(1-mu[k,])^(1-x[n,]))


      phi = c(phi, pi[k]*bern_prob)


    }
    z[n,]= phi/sum(phi)

  }
```

```r
  #
  # ff <- matrix(0, N,K)
  # for (k in 1:K) {
  #   for (n in 1:N) {
  #     #for (d in 1:D) {
  #     # a1 = (mu[k,]^x[n,])
  #     # a2 = (1-mu[k,])^(1-x[n,])
  #     bern_prob[n,] = (mu[k,]^x[n,])*(1-mu[k,])^(1-x[n,])
  #     product_bern[n] = prod(bern_prob[n,])
  #     ff[n,] <- pi * product_bern[n]
  #     z[n,] <- ff[n,k] / sum(ff[n,])
  #
  #   }
  # }
  #



  log_prob_z = log((pi[k]*(bern_prob))/sum(pi[k]*(bern_prob)))
  prob_z = exp(log_prob_z)

  #Log likelihood computation.
  # Your code here
  log_lik = 0
  for (n in 1:N) {
    for (k in 1:K) {
      log_lik = log_lik + z[n,k]*(log(pi[k]) + sum(x[n,]*log(mu[k,]) + (1-x[n,])*log(1-mu[k,])))
    }
  }
  llik[it] = log_lik
  cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
  flush.console()
  # Stop if the log likelihood has not changed significantly
  # Your code here

  # if(it>1){
    if(abs(change - llik[it]) < min_change){
      break()
    } else{
      change = llik[it]
    }

  #}
  #M-step: ML parameter estimation from the data and fractional component assignments
  # Your code here
  pi = colMeans(z)
  for (k in 1:K) {
    for (i in 1:D) {
      mu[k,i] = sum(z[,k]*x[,i])/sum(z[,k])
    }
  }
}
```

```r
llik_3 = llik[1:it]
length(llik[1:it])
pi
mu_3 = mu
mu_3
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
# 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
change = 0

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


  for (n in 1:N) {
```

```
  phi = c()
  for (k in 1:K) {
    bern_prob = prod((mu[k,]^x[n,]),(1-mu[k,])^(1-x[n,]))


    phi = c(phi, pi[k]*bern_prob)


  }
  z[n,]= phi/sum(phi)

}



#
# ff <- matrix(0, N,K)
# for (k in 1:K) {
#   for (n in 1:N) {
#     #for (d in 1:D) {
#     # a1 = (mu[k,]^x[n,])
#     # a2 = (1-mu[k,])^(1-x[n,])
#     bern_prob[n,] = (mu[k,]^x[n,])*(1-mu[k,])^(1-x[n,])
#     product_bern[n] = prod(bern_prob[n,])
#     ff[n,] <- pi * product_bern[n]
#     z[n,] <- ff[n,k] / sum(ff[n,])
#
#   }
# }
#




#log_prob_z = log((pi[k]*(bern_prob))/sum(pi[k]*(bern_prob)))
#prob_z = exp(log_prob_z)

#Log likelihood computation.
# Your code here
log_lik = 0
for (n in 1:N) {
  for (k in 1:K) {
    log_lik = log_lik + z[n,k]*(log(pi[k]) + sum(x[n,]*log(mu[k,]) + (1-x[n,])*log(1-mu[k,])))
  }
}
llik[it] = log_lik
cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
flush.console()
# Stop if the log likelihood has not changed significantly
# Your code here

# if(it>1){
  if(abs(change - llik[it]) < min_change){
    break()
  } else{
```

```r
      change = llik[it]
    }

  #}
  #M-step: ML parameter estimation from the data and fractional component assignments
  # Your code here
  pi = colMeans(z)
  for (k in 1:K) {
    for (i in 1:D) {
      mu[k,i] = sum(z[,k]*x[,i])/sum(z[,k])
    }
  }
}
llik_4 = llik[1:it]
length(llik[1:it])
pi
mu_4 = mu
mu_4
plot(llik_2, type="o", col="red", xlim = c(0,70), ylim = c(-8300,-6300))
points(llik_3,type="o", col="green")
points(llik_4,type="o", col="yellow")
```