lab1block2

Prudhvi Peddmallu

23 April 2019

1. ENSEMBLE METHODS

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. 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: sp <- read.csv2(“spambase.csv”) sp$Spam < -as.factor(sp$Spam)

Adaboost classification trees & randomForest.

```r
library(mboost)
library(randomForest)
library(ggplot2)
sp <- read.csv2("spambase.csv")
sp$Spam <- as.factor(sp$Spam)
n=dim(sp)[1]
set.seed(12345)
id=sample(1:n, floor(n*0.7))
sp_train=sp[id,]
sp_test=sp[-id,]
n_trees <- seq(10, 100, by = 10)
```

```r
residualerror_ab <- sapply(n_trees, function(x){
model = blackboost(Spam~., sp_train,
family = AdaExp(),control = boost_control(mstop = x))
yhat <- predict(model, sp_test, type = "class")
c_matrix <- table(yhat,sp_test$Spam)
1-(sum(diag(c_matrix))/sum(c_matrix))
})
ab_errors <- data.frame("ntrees" = n_trees, "error" = residualerror_ab)
ggplot(data = ab_errors, aes(x = n_trees, y = error))+geom_point()+geom_line() +
xlab("Number of Trees") + ylab("Error") + ggtitle("Number of Trees vs Error")
```

```r
residualerror_rf <- sapply(n_trees, function(x){
model <- randomForest(Spam~., sp_train, importance=TRUE,
proximity=TRUE,ntree = x)
yhat <- predict(model, sp_test, type = "class")
c_matrix <- table(yhat,sp_test$Spam)
1-(sum(diag(c_matrix))/sum(c_matrix))
})
```

```r
rf_errors <- data.frame("ntrees" = n_trees, "error" = residualerror_rf)
ggplot(data = rf_errors, aes(x = n_trees, y = error))+geom_point()+geom_line() +
xlab("Number of Trees") + ylab("Error") + ggtitle("Number of Trees vs Error")
```

2-method

Ada boost classification

```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")}
```

```r
library(randomForest)
```

```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_RF=spambase[id,]
```

```
test_RF=spambase[-id,]
```

Random forest

```
#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")  }
```

2-method-Adaboost classification trees & randomForest.

Loading Input files

```
spam_data <- read.csv(file = "spambase.data", header = FALSE)
colnames(spam_data)[58] <- "Spam"
spam_data$Spam <- factor(spam_data$Spam, levels = c(0,1), labels = c("0", "1"))
```

Splitting into Train and Test with 66% and 33% ratio.

```
set.seed(12345)
n =  NROW(spam_data)
id = sample(1:n, floor(n*(2/3)))
train = spam_data[id,]
test = spam_data[-id,]
```

Trainning the Model

Adaboost with varying depth

```
final_result <- NULL
for(i in seq(from = 10, to = 100, by = 10)){
```

```r
ada_model <- mboost::blackboost(Spam~.,
                                 data = train,
                                 family = AdaExp(),
                               control=boost_control(mstop=i))
forest_model <- randomForest(Spam~., data = train, ntree = i)
prediction_function <- function(model, data){
  predicted <- predict(model, newdata = data, type = c("class"))
  predict_correct <- ifelse(data$Spam == predicted, 1, 0)
  score <- sum(predict_correct)/NROW(data)
  return(score)
}
train_ada_model_predict <- predict(ada_model, newdata = train, type = c("class"))
test_ada_model_predict <- predict(ada_model, newdata = test, type = c("class"))
train_forest_model_predict <- predict(forest_model, newdata = train, type = c("class"))
test_forest_model_predict <- predict(forest_model, newdata = test, type = c("class"))
test_predict_correct <- ifelse(test$Spam == test_forest_model_predict, 1, 0)
train_predict_correct <- ifelse(train$Spam == train_forest_model_predict, 1, 0)
train_ada_score <-  prediction_function(ada_model, train)
test_ada_score <-  prediction_function(ada_model, test)
train_forest_score <-  prediction_function(forest_model, train)
test_forest_score <-  prediction_function(forest_model, test)
iteration_result <- data.frame(number_of_trees = i,
                               accuracy = c(train_ada_score,
                                            test_ada_score,
                                            train_forest_score,
                                            test_forest_score),
                               type  = c("train", "test", "train", "test"),
                               model = c("ADA", "ADA",  "Forest", "Forest"))
final_result <- rbind(iteration_result, final_result)
}
final_result$error_rate_percentage <- 100*(1 - final_result$accuracy)
ggplot(data = final_result, aes(x = number_of_trees,
                                y = error_rate_percentage,
                                group = type, color = type)) +
  geom_point() +
  geom_line() +
  ggtitle("Error Rate vs. increase in trees") + facet_grid(rows = vars(model))
```

2. MIXTURE MODELS-EM

Your task is to implement the EM algorithm for mixtures of multivariate Benoulli distributions. Please use the template in the next page to solve the assignment. Then, use your implementation to show what happens when your mixture models has too few and too many components, i.e. set K = 2; 3; 4 and compare results. Please provide a short explanation as well. ## Description of the EM algorithm EM is an iterative expectation maximumation technique. The way this works is for a given mixed distribution we guess the components of the data. This is done by first guessing the number of components and then randomly initializing the parameters of the said distribution (Mean, Varience).

Sometimes the data do not follow any known probability distribution but a mixture of known distributions such as:

$$p(x) = \sum_{k=1}^{K} p(k).p(x|k)$$

where p(x|k) are called mixture components and p(k) are called mixing coefficients: where p(k) is denoted by

$$\pi_k$$

With the following conditions

$$0 \leq \pi_k \leq 1$$

and

$$\sum_k \pi_k = 1$$

We are also given that the mixture model follows a Bernoulli distribution, for bernoulli we know that

$$Bern(x|\mu_k) = \prod_i \mu_{ki}^{x_i}.(1-\mu_{ki})^{(1-x_i)}$$

The EM algorithm for an Bernoulli mixed model is:

Set pi and mu to some initial values Repeat until pi and mu do not change E-step: Compute p(z|x) for all k and n M-step: Set pi^k to pi^k(ML) from likehood estimate, do the same to mu

M step:

$$p(z_{nk}|x_n, \mu, \pi) = Z = \frac{\pi_k p(x_n|\mu_k)}{\sum_k p(x_n|\mu_k)}$$

E step:

$$\pi_k^{ML} = \frac{\sum_N p(z_{nk}|x_n, \mu, \pi)}{N}$$

$$\mu_{ki}^{ML} = \frac{\sum_n x_{ni} p(z_{nk}|x_n, \mu, \pi)}{\sum_n p(z_{nk}|x_n, \mu, \pi)}$$

The maximum likehood of E step is:

$$\log_e p(X|\mu, \pi) = \sum_{n=1}^{N} \log_e \sum_{k=1}^{K} .\pi_k.p(x_n|\mu_k)$$

Given question 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)
```

```r
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")
```

Code

To compare the results for K = 2,3,4, the em_loop-function provides a graphical analysis for every iteration. The function includes comments which explain what I did at which step to create the EM algorithm. The function will be finally run with K = 2,3,4.

```r
em_loop = function(K) {
# Initializing 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 = K) # true mixing coefficients
true_mu = matrix(nrow = K, ncol = D) # true conditional distributions
true_pi = c(rep(1/K, K))
if (K == 2) {
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)
plot(true_mu[1,], type = "o", xlab = "dimension", col = "blue",
ylim = c(0,1), main = "True")
points(true_mu[2,], type="o", xlab = "dimension", col = "red",
main = "True")
} else if (K == 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", xlab = "dimension", col = "blue", ylim=c(0,1),
main = "True")
points(true_mu[2,], type = "o", xlab = "dimension", col = "red",
main = "True")
points(true_mu[3,], type = "o", xlab = "dimension", col = "green",
main = "True")
} else {
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)
true_mu[4,] = c(0.3,0.5,0.5,0.7,0.5,0.5,0.5,0.5,0.4,0.5)
plot(true_mu[1,], type = "o", xlab = "dimension", col = "blue",
ylim = c(0,1), main = "True")
points(true_mu[2,], type = "o", xlab = "dimension", col = "red",
main = "True")
points(true_mu[3,], type = "o", xlab = "dimension", col = "green",
main = "True")
points(true_mu[4,], type = "o", xlab = "dimension", col = "yellow",
main = "True")
}
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
# Producing the training data
for(n in 1:N) {
k = sample(1:K, 1, prob=true_pi)
for(d in 1:D) {
x[n,d] = rbinom(1, 1, true_mu[k,d])
}
}
# 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)
}
#EM algorithm
for(it in 1:max_it) {
# Plotting mu
# Defining plot title
title = paste0("Iteration", it)
if (K == 2) {
plot(mu[1,], type = "o", xlab = "dimension", col = "blue", ylim = c(0,1), main = title)
points(mu[2,], type = "o", xlab = "dimension", col = "red", main = title)
} else if (K == 3) {
plot(mu[1,], type = "o", xlab = "dimension", col = "blue", ylim = c(0,1), main = title)
points(mu[2,], type = "o", xlab = "dimension", col = "red", main = title)
points(mu[3,], type = "o", xlab = "dimension", col = "green", main = title)
} else {
plot(mu[1,], type = "o", xlab = "dimension", col = "blue", ylim = c(0,1), main = title)
points(mu[2,], type = "o", xlab = "dimension", col = "red", main = title)
points(mu[3,], type = "o", xlab = "dimension", col = "green", main = title)
points(mu[4,], type = "o", xlab = "dimension", col = "yellow", main = title)
}
Sys.sleep(0.5)
# E-step: Computation of the fractional component assignments
for (n in 1:N) {
# Creating empty matrix (column 1:K = p_x_given_k; column K+1 = p(x|all k)
p_x = matrix(data = c(rep(1,K), 0), nrow = 1, ncol = K+1)
Calculating p(x|k) and p(x|all k)
for (k in 1:K) {
# Calculating p(x|k)
for (d in 1:D) {
p_x[1,k] = p_x[1,k] * (mu[k,d]^x[n,d]) * (1-mu[k,d])^(1-x[n,d])
}
p_x[1,k] = p_x[1,k] * pi[k] # weighting with pi[k]
# Calculating p(x|all k) (denominator)
p_x[1,K+1] = p_x[1,K+1] + p_x[1,k]
}
# Calculating z for n and all k
for (k in 1:K) {
z[n,k] = p_x[1,k] / p_x[1,K+1]
}
}
# Log likelihood computation
for (n in 1:N) {
for (k in 1:K) {
log_term = 0
for (d in 1:D) {
log_term = log_term + x[n,d] * log(mu[k,d]) + (1-x[n,d]) * log(1-mu[k,d])
}
llik[it] = llik[it] + z[n,k] * (log(pi[k]) + log_term)
}
}
cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
```

```r
flush.console()
# Stop if the log likelihood has not changed significantly
if (it != 1) {
if (abs(llik[it] - llik[it-1]) < min_change) {
break
}
}
# M-step: ML parameter estimation from the data and fractional component assignments
# Updating pi
for (k in 1:K) {
pi[k] = sum(z[,k])/N
}
# Updating mu
for (k in 1:K) {
mu[k,] = 0
for (n in 1:N) {
    mu[k,] = mu[k,] + x[n,] * z[n,k]
}
mu[k,] = mu[k,] / sum(z[,k])
}
}
# Printing pi, mu and development of log likelihood at the end
return(list(
pi = pi,
mu = mu,
logLikelihoodDevelopment = plot(llik[1:it],
type = "o",
main = "Development of the log likelihood",
xlab = "iteration",
ylab = "log likelihood")
))
}
```

K=2

```r
em_loop(2)
```

K=3

```r
## K=3
em_loop(3)
```

K=4

```r
em_loop(4)
```

Analysis

Comparing the final plots for each of the cases, it becomes clear that when the mixture model has more components (K = 4), the EM algorithm does not perform as accurate as for fewer components (K = 2 or K = 3). The segregation between each component gets diluted as the components get higher.

EM algo with matrix

```r
em_mat <- function(k){
set.seed(1234567890)
# max number of EM iterations
max_it <- 100
# min change in log likelihood between two consecutive EM iterations
min_change <- 0.1
#------------- Producing Training data and Initialization ----------------#
# number of training points
N <- 1000
# number of dimensions
D <- 10
# training data
x <- matrix(nrow=N, ncol=D)
# true mixing coefficients
true_pi <- vector(length = k)
true_pi <- rep(1/k, k)
# true conditional distributions
true_mu <- matrix(nrow = k, ncol = D)
if(k == 2){
plot(true_mu[1,], type="o", col="blue", ylim=c(0,1))
points(true_mu[2,], type="o", col="red")
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)
}else if(k == 3){
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")
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)
}else {
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")
points(true_mu[4,], type="o", col="yellow")
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)
true_mu[4,]=c(0.3,0.5,0.5,0.7,0.5,0.5,0.5,0.5,0.4,0.5)}
# Producing the training data
for(n in 1:N) {
l <- sample(1:k,1,prob=true_pi)
for(d in 1:D) {
x[n,d] <- rbinom(1,1,true_mu[l,d])
}
}
# fractional component assignments
z <- matrix(nrow = N, ncol = k)
# mixing coefficients
pi <- vector(length = k)
# conditional distributions
```

```r
mu <- matrix(nrow = k, ncol = D)
# log likelihood of the EM iterations
llik <- vector(length = max_it)
# Random initialization of the paramters
pi <- runif(k,0.49,0.51)
pi <- pi / sum(pi)
for(i in 1:k) {
mu[i,] <- runif(D,0.49,0.51)
}
#----------------------- Iteration stage -----------------------#
for(it in 1:max_it) {
if(k == 2){
plot(mu[1,], type="o", col="blue", ylim=c(0,1))
points(mu[2,], type="o", col="red")
}else if(k == 3){
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")
}else{
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
# Updating z matrix
p_Xn_MUn <- exp(x %*% log(t(mu)) + (1 - x) %*% log(1 - t(mu)))
numerator <- matrix(rep(pi,N), ncol = k, byrow = TRUE) * p_Xn_MUn
denominator <- rowSums(numerator)
Z_nk <- numerator/denominator
# Updating pi
pi <- colSums(Z_nk)/N
# Updating mu
mu <- (t(Z_nk) %*% x)/colSums(Z_nk)
#Log likelihood computation.
llik[it] <- sum(Z_nk * ((x %*% log(t(mu)) + (1 - x) %*% log(1 - t(mu))
) + matrix(rep(pi,N), ncol = k, byrow = TRUE)))
cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
flush.console()
# Stop if the log likelihood has not changed significantly
if(it >= 2){
if((llik[it] - llik[it-1]) < min_change){break()}
}
#M-step: ML parameter estimation from the data and fractional component assignments
# pi_ML
pi_ML <- pi
#mu_ML
mu_ML <- mu
}
#----------------------- output stage -----------------------#
df <- data.frame(Iteration = 1:length(llik[which(llik != 0.000)])
, log_likelihood = llik[which(llik != 0.000)])
plot <- ggplot(data = df) +
geom_point(mapping = aes(x = Iteration, y = log_likelihood),
color = 'black') +
geom_line(mapping = aes(x = Iteration, y = log_likelihood),
color = 'black', size = 1) +
ggtitle('Maximum likelihood vs Number of iterations') +
theme(plot.title = element_text(hjust = 0.5)) +
theme_light()
output <- list(pi_ML = pi_ML,
mu_ML = mu_ML,
plot = plot
)
output
}
EM_2 <- em_mat(2)
EM_2$plot
EM_2$pi_ML
EM_2$mu_ML
EM_3 <- em_mat(3)
EM_3$plot
EM_3$pi_ML
EM_3$mu_ML
EM_4 <- em_mat(4)
EM_4$plot
EM_4$pi_ML
EM_4$mu_ML
```

EM-2 method

```r
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
```

```r
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,]))
}
```

```
#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=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")
```