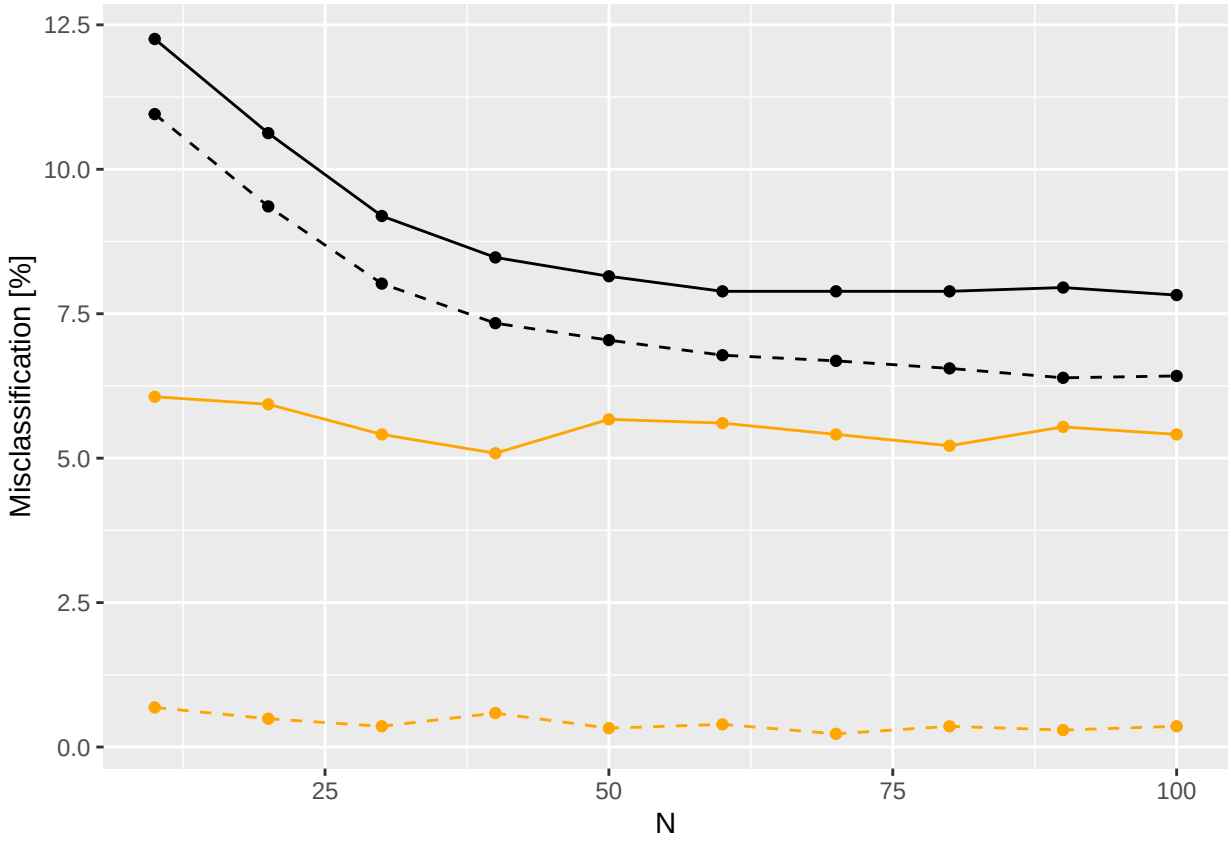
Machine Learning Assignment 02 - Group A15

Zuxiang Li (zuxli371), Marcos F. Mourao (marfr825), Agustín Valencia (aguva779)

12/04/2019

Task 01 - Ensemble Methods

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.



The misclassification rate for Adaboost decreases as the model's complexity (tree depth) grows in both train an test data. This is expected from a boosting algorithm, since the current classifier is based on how the previous classifier performed, weighting the misclassifications (with an exponential loss function) while maintaining the weight on correct classifications.

Random forest's presented a lower misclassification rate overall, but the results are somewhat indifferent regarding tree depth. The test error for training data was approximately zero, while for test data was higher (~5%). Since this model "grows" decision trees in parallel and later make the final classification by majority voting, the overall error is lower given that the error gets averaged down by all models.

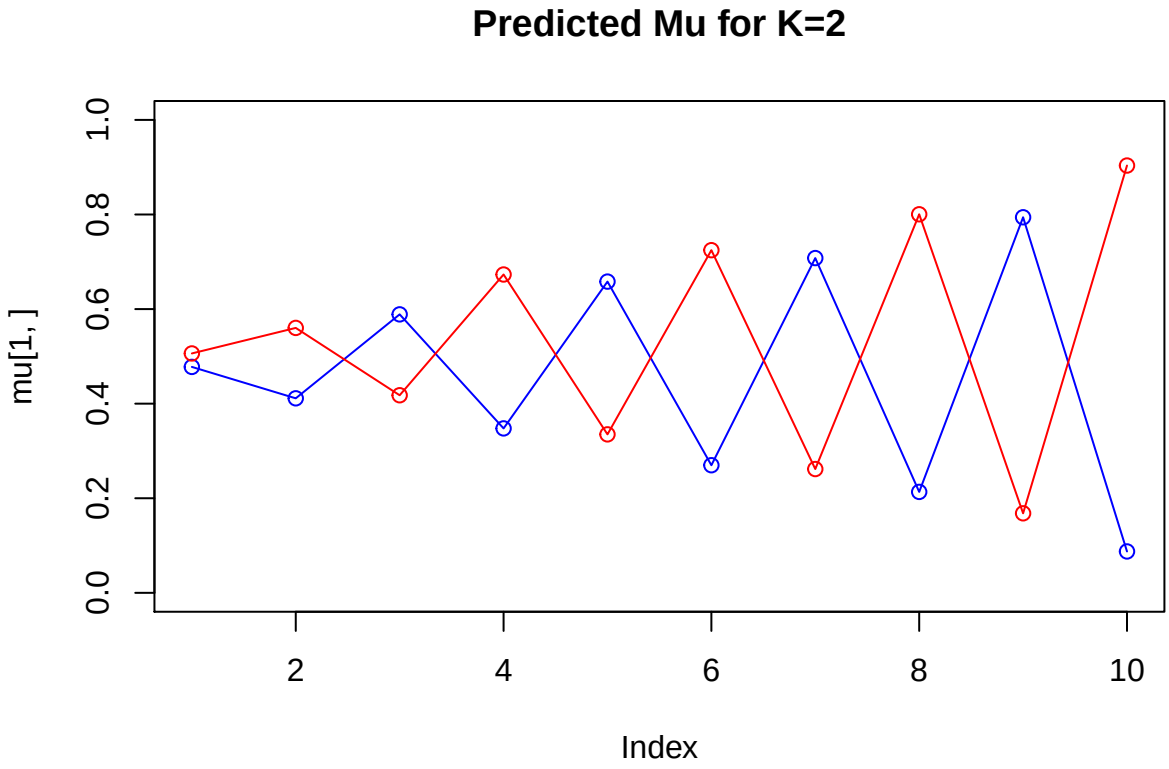
For both test and training data, random forest performed better than AdaBoost (i.e. lower misclassification rates).

Note - We are varying the number of trees and not the depth of each of these trees.
Good explanation otherwise.
You have it correct in your code though.

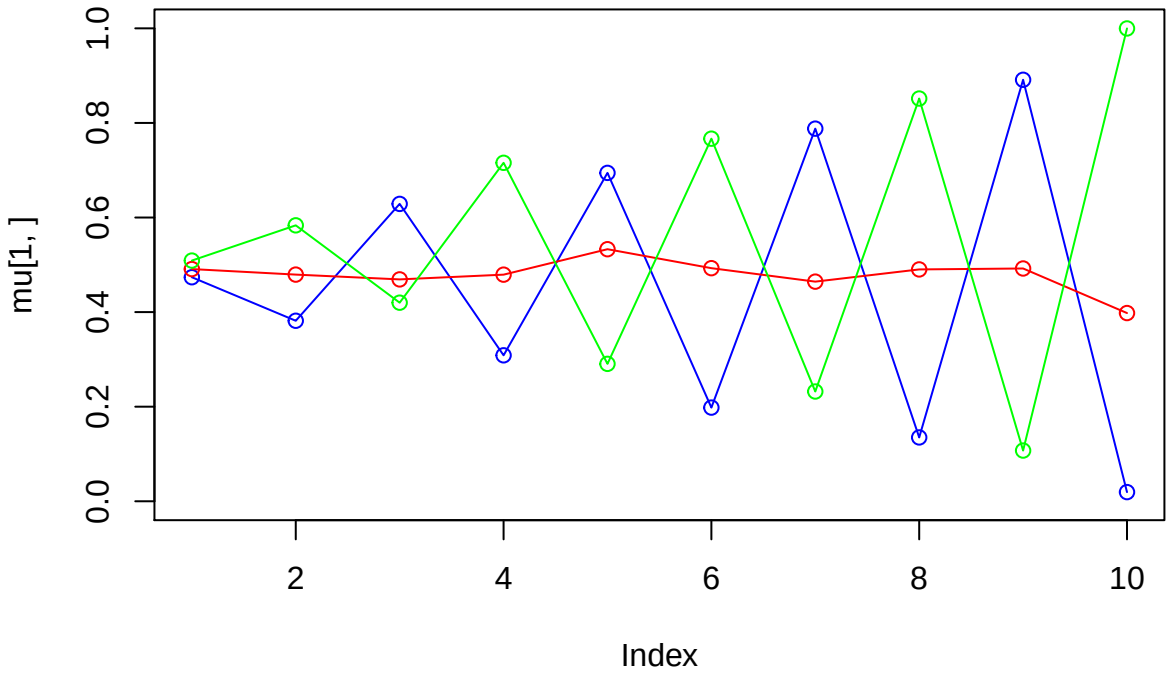
Task 02 - Mixture Models

Your task is to implement the EM algorithm for mixtures of multivariate Benoulli distributions. 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.

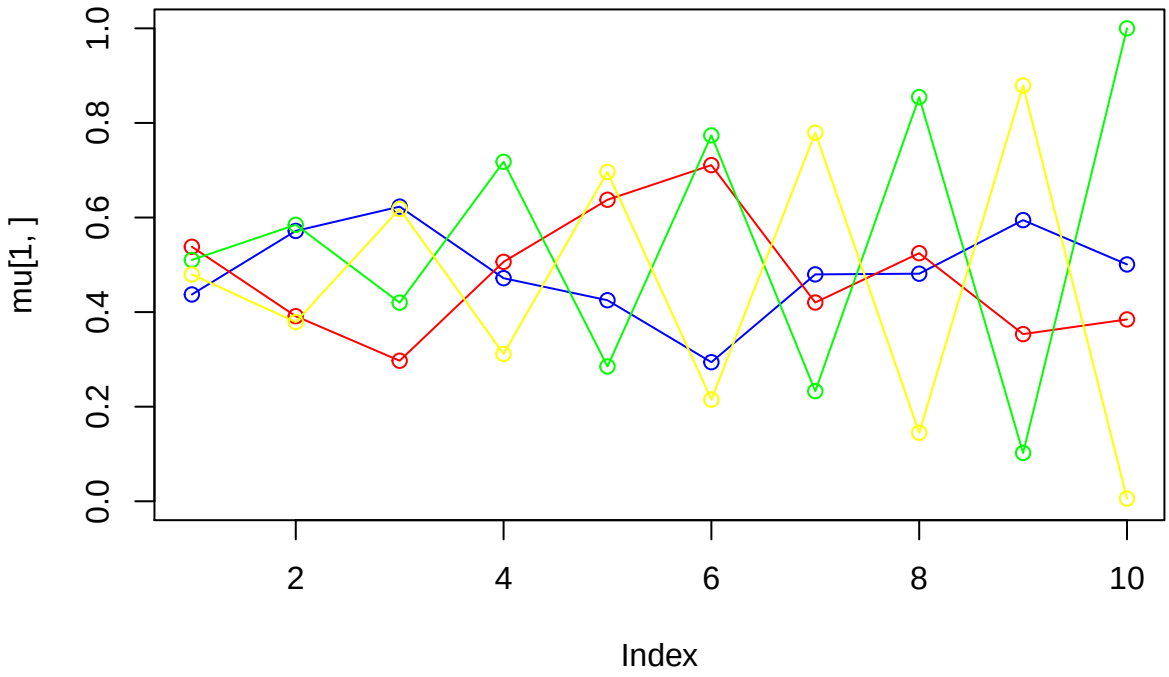
Solution




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

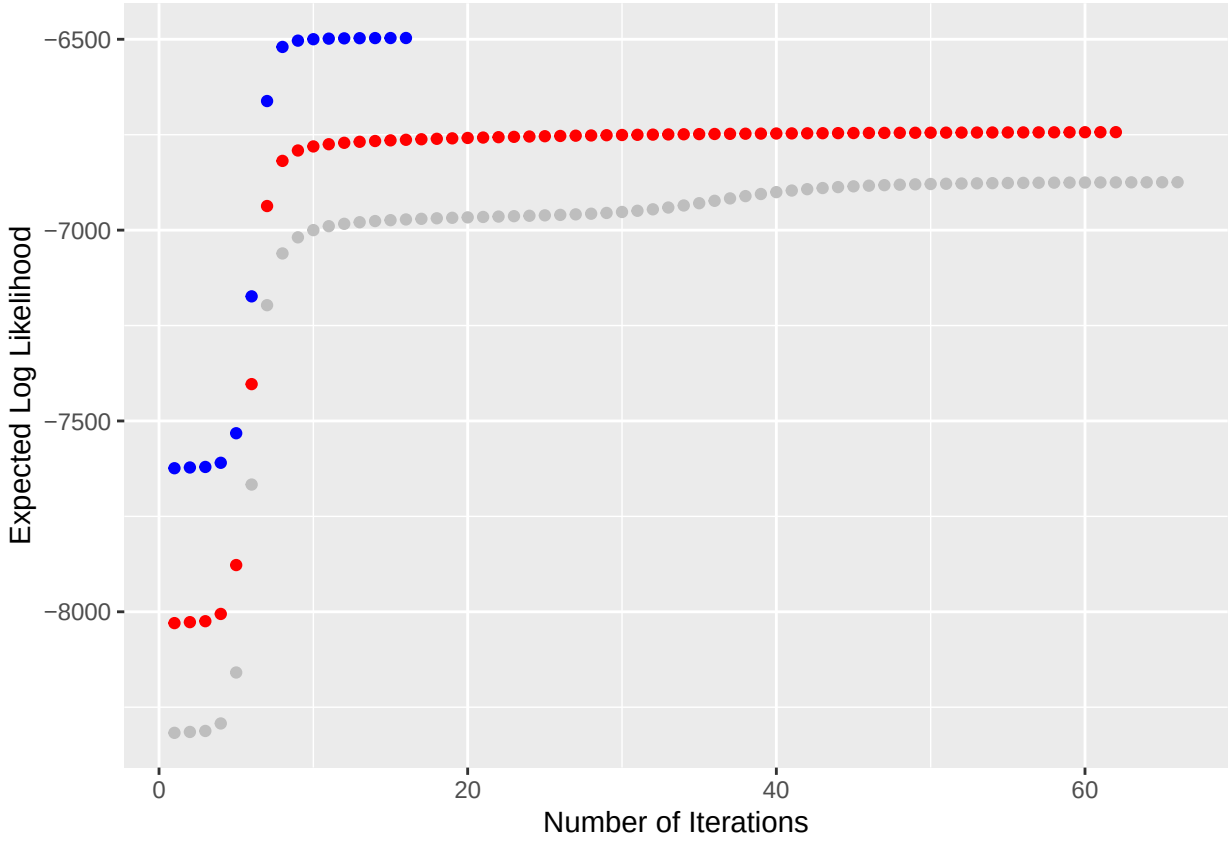



```
## [1] 0.3259592 0.3044579 0.3695828
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.4737193 0.3817120 0.6288021 0.3086143 0.6943731 0.1980896 0.7879447
## [2,] 0.4909874 0.4793213 0.4691560 0.4791793 0.5329895 0.4928830 0.4643990
## [3,] 0.5089571 0.5834802 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
```




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

Your log-likelihood computation is not correct. You are calculating the joint log-likelihood of observed data and latent variables. You are supposed to calculate the log-likelihood of the observed data only.



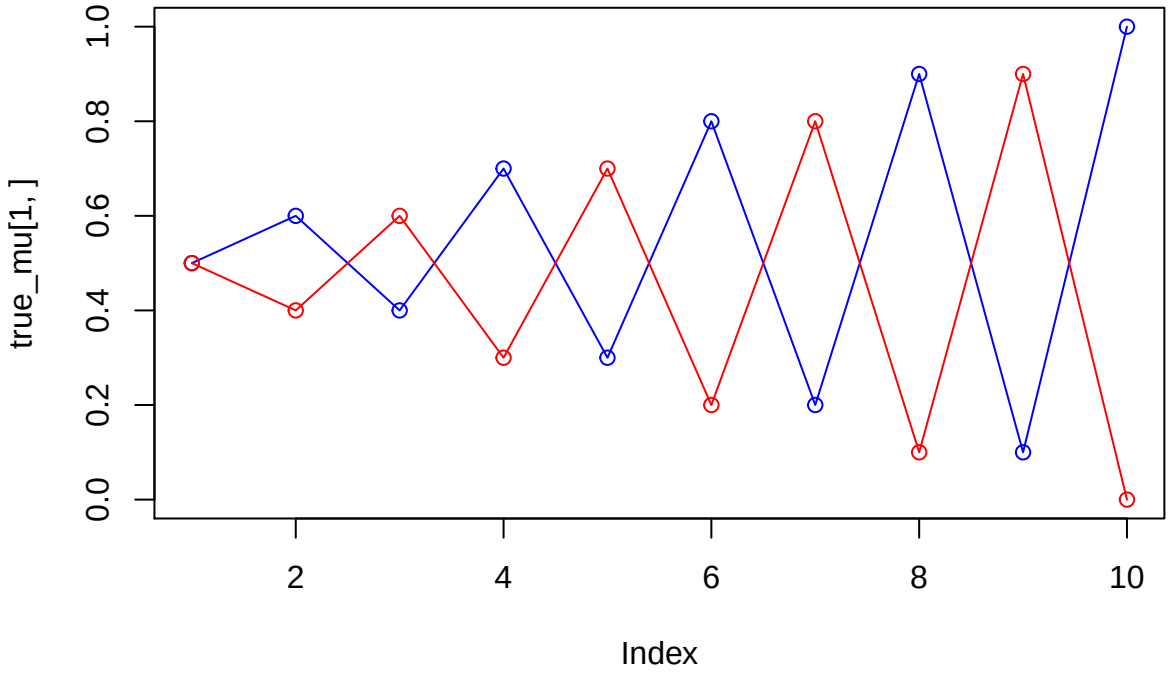
In this case, we generate data using true_mu and true_pi. For K=2, the blue dots, which means too few components, the blue points in the plot, only 16 iterations were made, this model is underfitted. For K=4, the grey dots, too many components, grey dots in the plot, 66 iterations were made, then this model is overfitted. Only when K=3, the red dots in plot, we got similar mu and pi compare to the true values.

The values of `true_mu` and predicted `mu` are shown below:

K=2

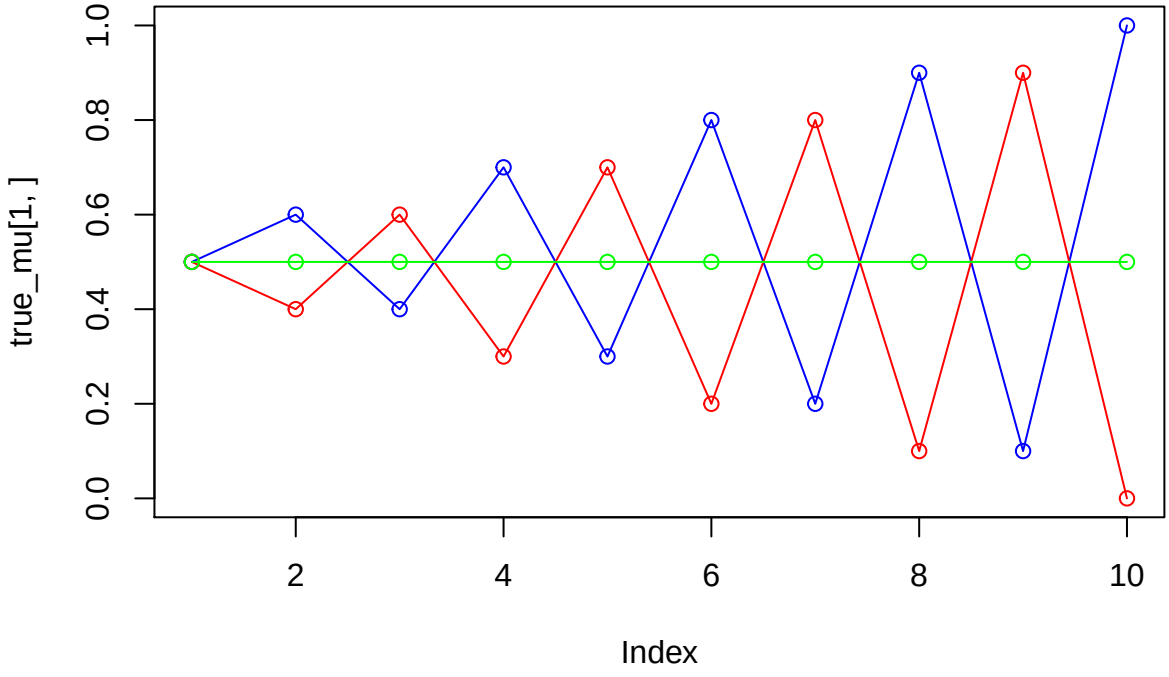
What is your motivation behind saying that the model if underfitting/overfitting? We cannot conclude this based on the plots of mu alone.

`true_mus` :



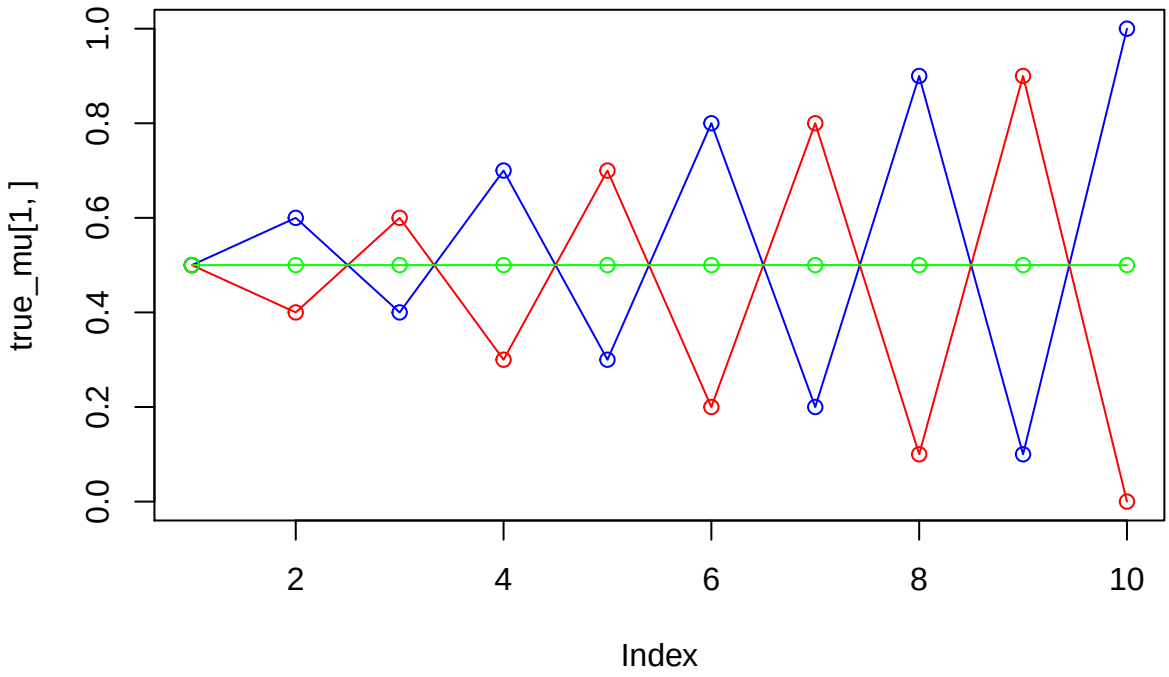
K=3

`true_mus` :



K=4

`true_mus`:


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

Code Appendix

```
knitr::opts_chunk$set(fig.pos = "!h")
library(mboost)
library(randomForest)
library(ggplot2)
library(partykit)
library(GLDEX)
set.seed(1234567890)
sp <- read.csv2("data/spambase.csv")
sp$Spam <- as.factor(sp$Spam)
## Splitting data
n <- dim(sp)[1]
idxs <- sample(1:n, floor(2*n/3))
train <- sp[idxs,]
test <- sp[-idxs,]
get_missed <- function (true, predicted) {
    confussion <- table(true, predicted)
    fn <- confussion[1,2]
    fp <- confussion[2,1]
    total <- sum(confussion)
    miss <- (fp + fn) / total * 100
    return(miss)
}
# Training
nums <- seq(10,100,10)
formula <- Spam ~ .
error_rates_train_bb <- c()
error_rates_test_bb <- c()
error_rates_train_rf <- c()
error_rates_test_rf <- c()
depths <- c()
for (i in nums) {
    bb <- blackboost (
            Spam ~ .,
            data = train,
            family = AdaExp(),
            control = boost_control(mstop = i)
     )

    rf <- randomForest(
            Spam ~ .,
            data = train,
            ntree = i
    )

    predicted <- predict(bb, train, type = "class")
    miss <- get_missed(train$Spam, predicted)
    error_rates_train_bb <- append(error_rates_train_bb, miss)

    predicted <- predict(bb, test, type = "class")
    miss <- get_missed(test$Spam, predicted)
    error_rates_test_bb <- append(error_rates_test_bb, miss)
```

```
    predicted <- predict(rf, train, type = "class")
    miss <- get_missed(train$Spam, predicted)
    error_rates_train_rf <- append(error_rates_train_rf, miss)

    predicted <- predict(rf, test, type = "class")
    miss <- get_missed(test$Spam, predicted)
    error_rates_test_rf <- append(error_rates_test_rf, miss)

    depths <- append(depths, i)
}
df <- data.frame(
    nums = nums,
    error_rates_train_bb = error_rates_train_bb,
    error_rates_test_bb = error_rates_test_bb,
    error_rates_train_rf = error_rates_train_rf,
    error_rates_test_rf = error_rates_test_rf
)
p <- ggplot() +
    geom_line(aes(x = depths, y = error_rates_train_bb), color = "black", linetype = "dashed") +
    geom_line(aes(x = depths, y = error_rates_test_bb), color = "black") +
    geom_line(aes(x = depths, y = error_rates_train_rf), color = "orange", linetype = "dashed") +
    geom_line(aes(x = depths, y = error_rates_test_rf), color = "orange") +
    geom_point(aes(x = depths, y = error_rates_train_bb), color = "black") +
    geom_point(aes(x = depths, y = error_rates_test_bb), color = "black") +
    geom_point(aes(x = depths, y = error_rates_train_rf), color = "orange") +
    geom_point(aes(x = depths, y = error_rates_test_rf), color = "orange") +
    xlab("N") + ylab("Misclassification [%]")
p
RNGversion('3.5.1')
EM_algorithm<-function(K){
  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
```

```r
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)
}

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){
    denominator<-c()
    numerator<-c()
    for(k in 1:K){
      new_numerator<-pi[k]*prod(mu[k,]^x[n,]*(1-mu[k,])^(1-x[n,]))
      numerator<-c(numerator,new_numerator)
      denominator<-sum(numerator)
    }
    z[n,]<-numerator/denominator
  }
  #Log likelihood computation.
  sum=0
  for(n in 1:N){
    for(k in 1:K){
      sum=sum+z[n,k]*(log(pi[k])+sum(x[n,]*log(mu[k,])+(1-x[n,])*log(1-mu[k,])))
    }
  }
  llik[it]<-sum
  # Your code here
  #cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
  flush.console()
  # Stop if the lok likelihood has not changed significantly
  if(it>1){
    if(abs(llik[it]-llik[it-1])<min_change){
      titStr <- paste("Predicted Mu for K=", K, sep = "")
      plot(mu[1,], type="o", col="blue", ylim=c(0,1),main = titStr)
      if(K>=2)
        points(mu[2,], type="o", col="red")
      if(K>=3)
        points(mu[3,], type="o", col="green")
      if(K>=4)
        points(mu[4,], type="o", col="yellow")
```

```
        print(pi)
        print(mu)
        break
      }

    }
    # Your code here
    #M-step: ML parameter estimation from the data and fractional component assignments
    # Your code here
    pi=apply(z,2,mean)
    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])
    }
  }
  pi
  mu
  #plot(llik[1:it], type="o")
  return(llik[1:it])
  #return(list(pi=pi,mu=mu,res=,llik=llik[1:it])
}

em1<-EM_algorithm(K=2)
em2<-EM_algorithm(K=3)
em3<-EM_algorithm(K=4)

ggplot()+geom_point(aes(x=c(1:length(em1)),y=em1),color="blue")+
  geom_point(aes(x=c(1:length(em2)),y=em2),color="red")+
  geom_point(aes(x=c(1:length(em3)),y=em3),color="grey")+
  xlab("Number of Iterations")+ylab("Expected Log Likelihood")
true_mu <- matrix(nrow=3, ncol=10) # true conditional distributions
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")
true_mu <- matrix(nrow=3, ncol=10) # true conditional distributions
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")
   true_mu <- matrix(nrow=3, ncol=10) # true conditional distributions
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")
```