Lab1 - Block2

Fahad Hameed

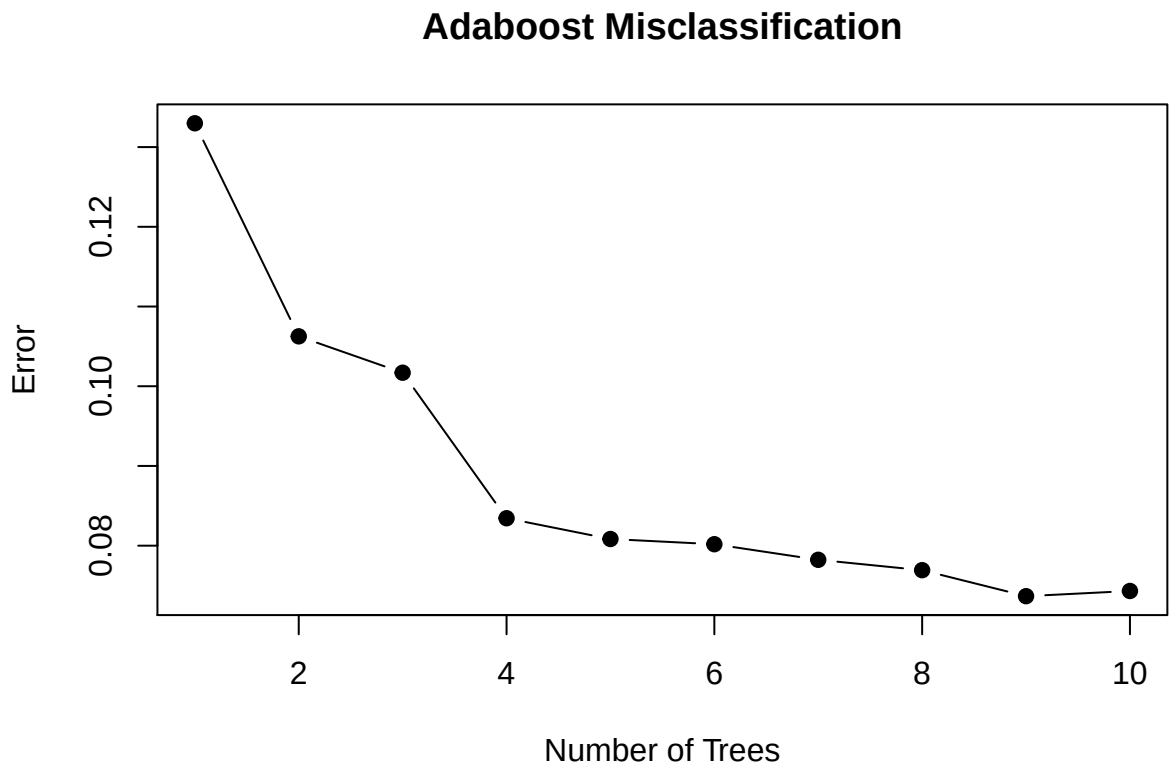
December 14, 2017

LAB 1 BLOCK 2:

Assignment 1 ENSEMBLE METHODS

In this exercise I have used Adaboost classification trees and random forests to evauluate their performance on spam data. The data set have been divided into two parts, 2/3 for training and 1/3 as test data.

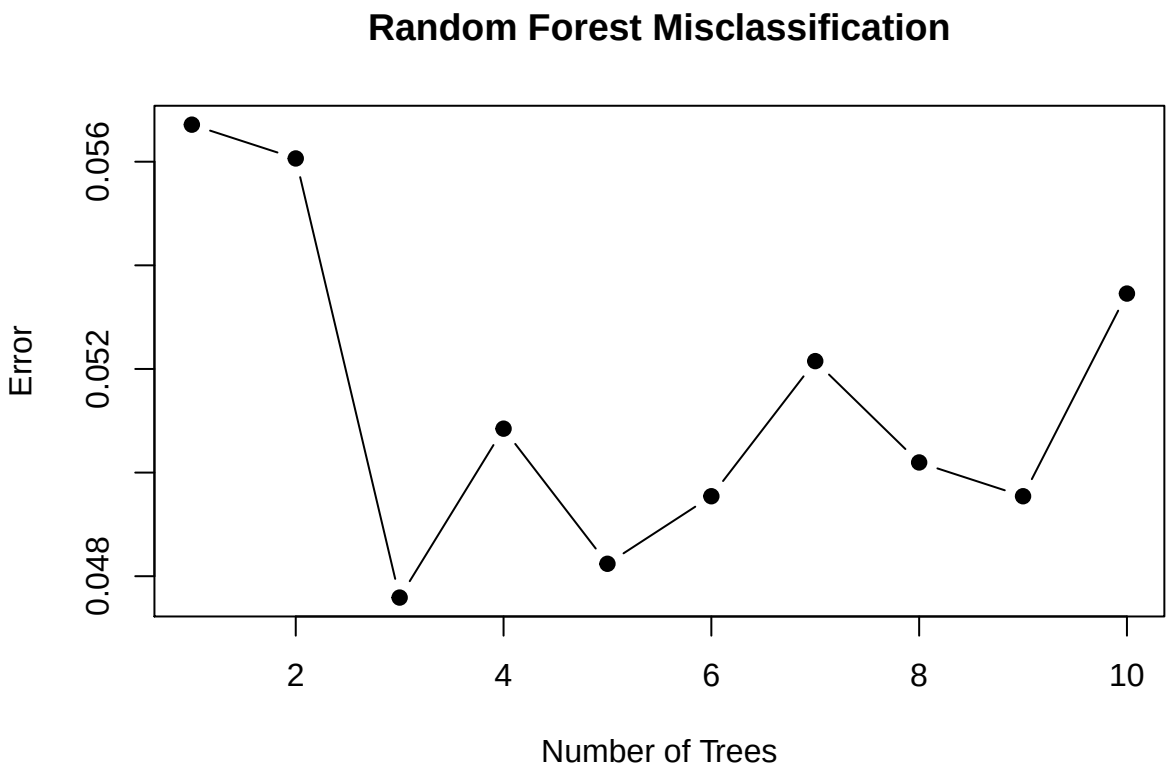
ADABOOST



Loss Function for the selected family is $LossFunction = e^{-y*f}$

The performance of the Adaboost classification trees can be seen above. We can see that the optimal would be roughly 4 trees then it barely decreases as the number of trees grows. At 8 trees the test error seems to halt so if you want to push the limit and create a substantially more complex model it may be preferable to use 8 trees. However, since the error on the test data stops monotonically decrease after 4 trees that choice can also be preffered.

RANDOM FOREST



The performance of the random forest can be seen above, the test error seem to stop monotonically decreasing after 4 trees and that should be the prefered choice. It can see that the test error increases as the number of trees increases after 20 trees so the model have almost fit the training data perfectly with 2 trees.

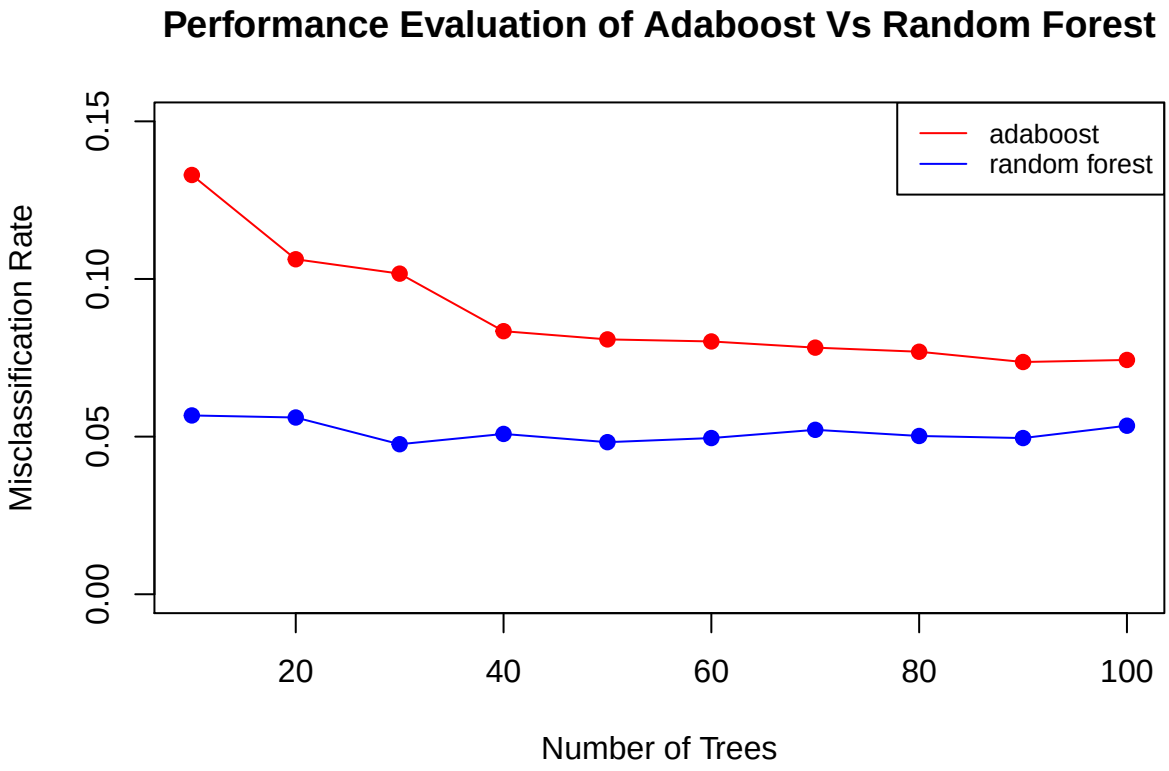
Performance Evaluation:

```
#Misclassification for Adaboost
error_rates_ada
```

```
##  [1] 0.13298566 0.10625815 0.10169492 0.08344198 0.08083442 0.08018253
##  [7] 0.07822686 0.07692308 0.07366362 0.07431551
```

```
#Misclassification for Random Forest
error_rates_random
```

```
##  [1] 0.05671447 0.05606258 0.04758801 0.05084746 0.04823990 0.04954368
##  [7] 0.05215124 0.05019557 0.04954368 0.05345502
```



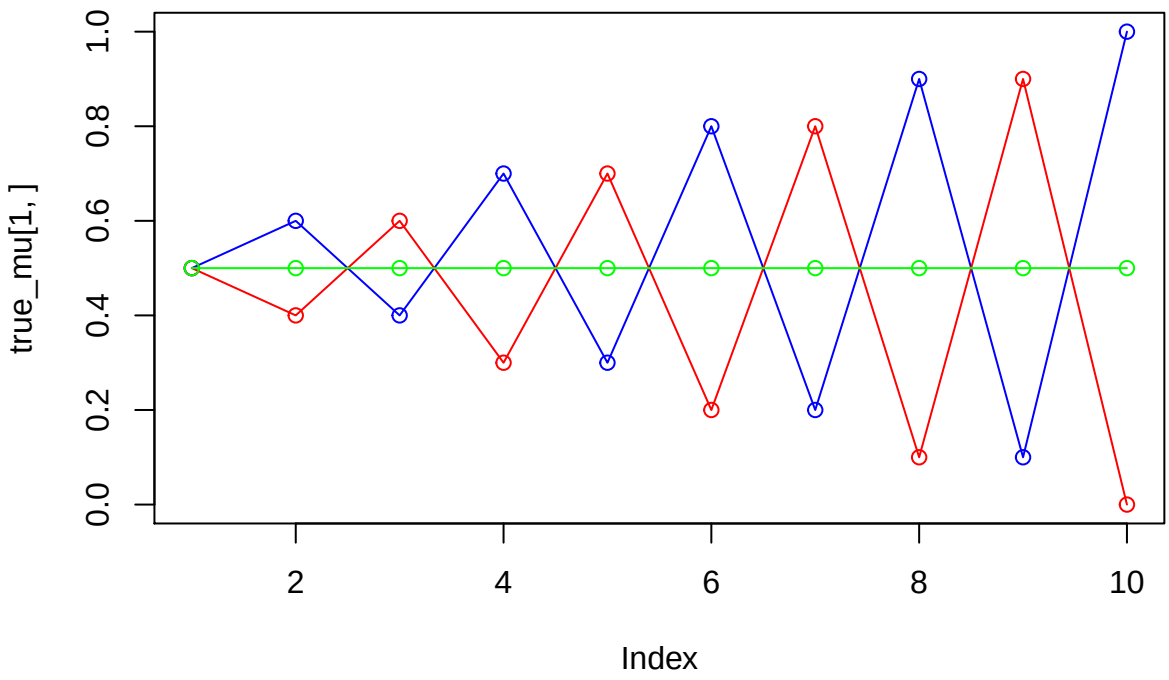
The above plot shows the comparision of random forest and adaboost performance. As shown the misclassification rate of random forest is much less than the misclassification rate of Adaboost.

Assignment 2

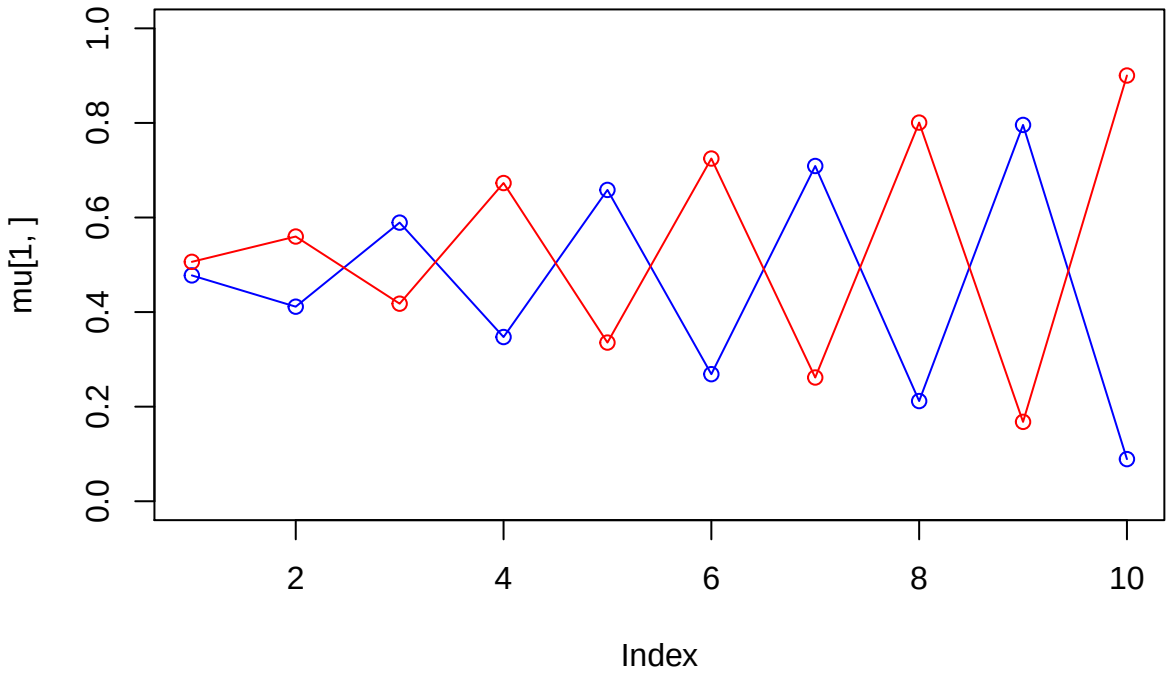
Mixture Model

The plot below shows the three true multivariate Bernoulli distributions from which the data set have been generated.

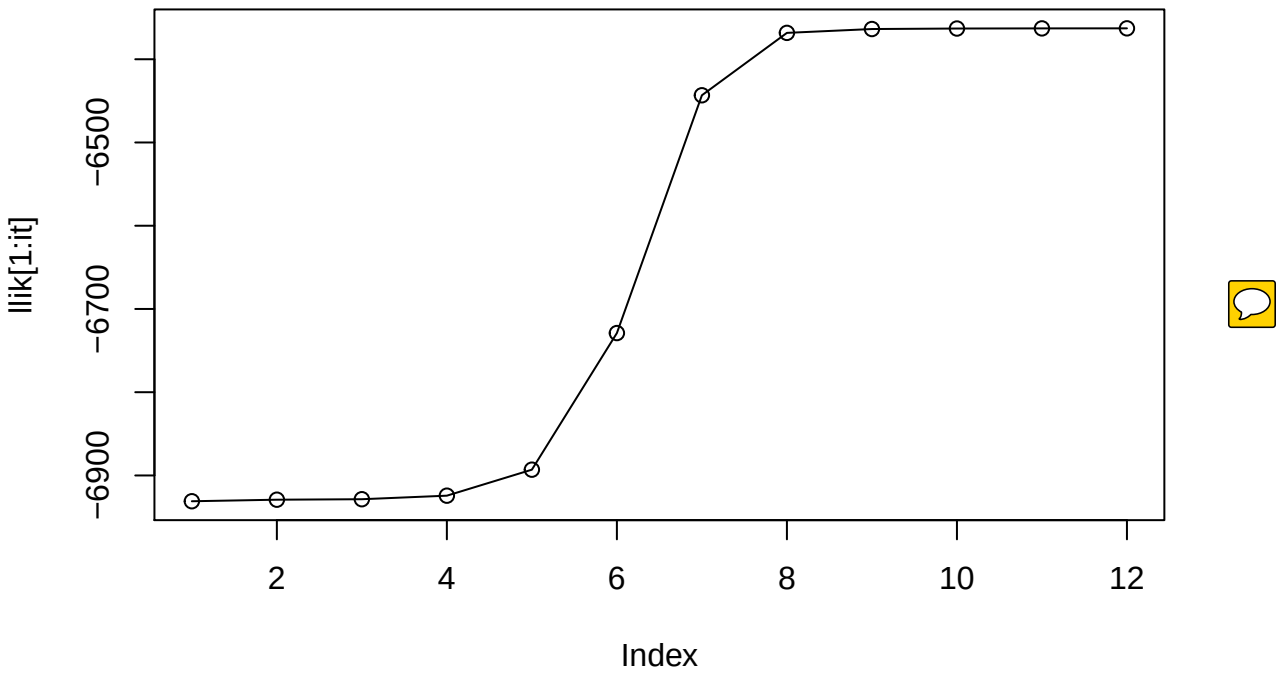
For K=2




```
## iteration:  1 log likelihood:  -6930.975
## iteration:  2 log likelihood:  -6929.125
## iteration:  3 log likelihood:  -6928.562
## iteration:  4 log likelihood:  -6924.281
## iteration:  5 log likelihood:  -6893.055
## iteration:  6 log likelihood:  -6728.948
## iteration:  7 log likelihood:  -6443.28
## iteration:  8 log likelihood:  -6368.318
## iteration:  9 log likelihood:  -6363.734
## iteration:  10 log likelihood:  -6363.109
## iteration:  11 log likelihood:  -6362.947
## iteration:  12 log likelihood:  -6362.897
```

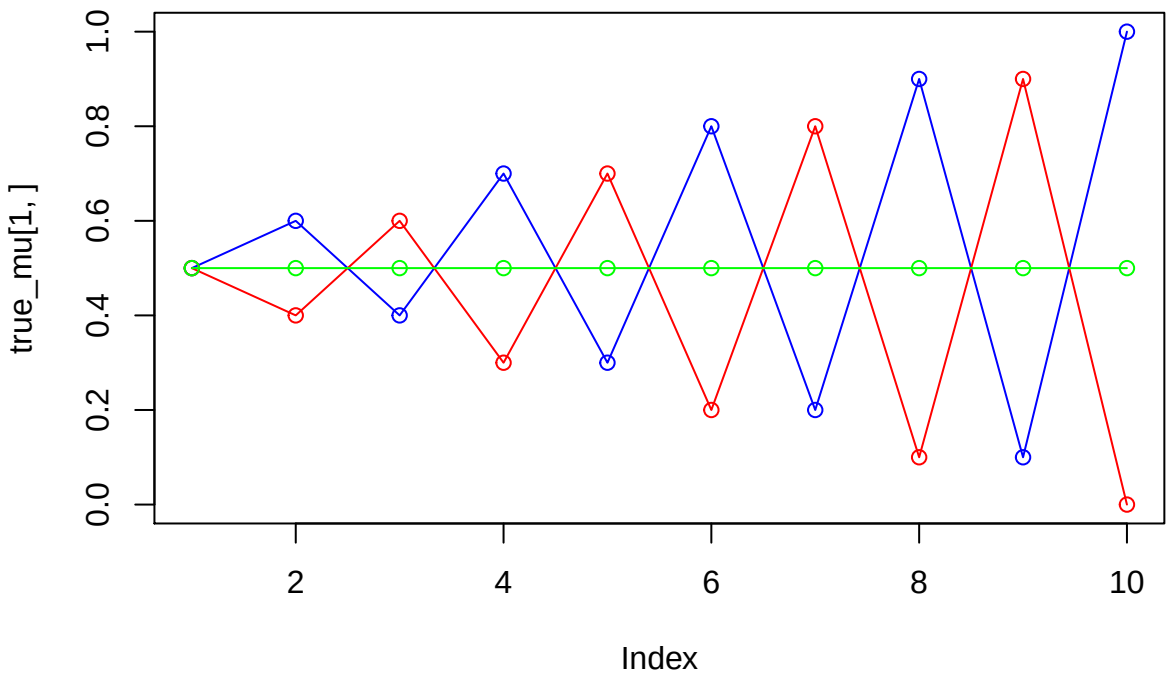



```
## value of updated pi is  0.497125 0.502875
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.4775488 0.4113939 0.5892308 0.3472420 0.6583712 0.2686589 0.7089490
## [2,] 0.5062860 0.5597531 0.4177551 0.6728856 0.3354854 0.7247188 0.2616231
##           [,8]      [,9]      [,10]
## [1,] 0.2118629 0.7957549 0.08905747
## [2,] 0.8007511 0.1678555 0.90027808
```



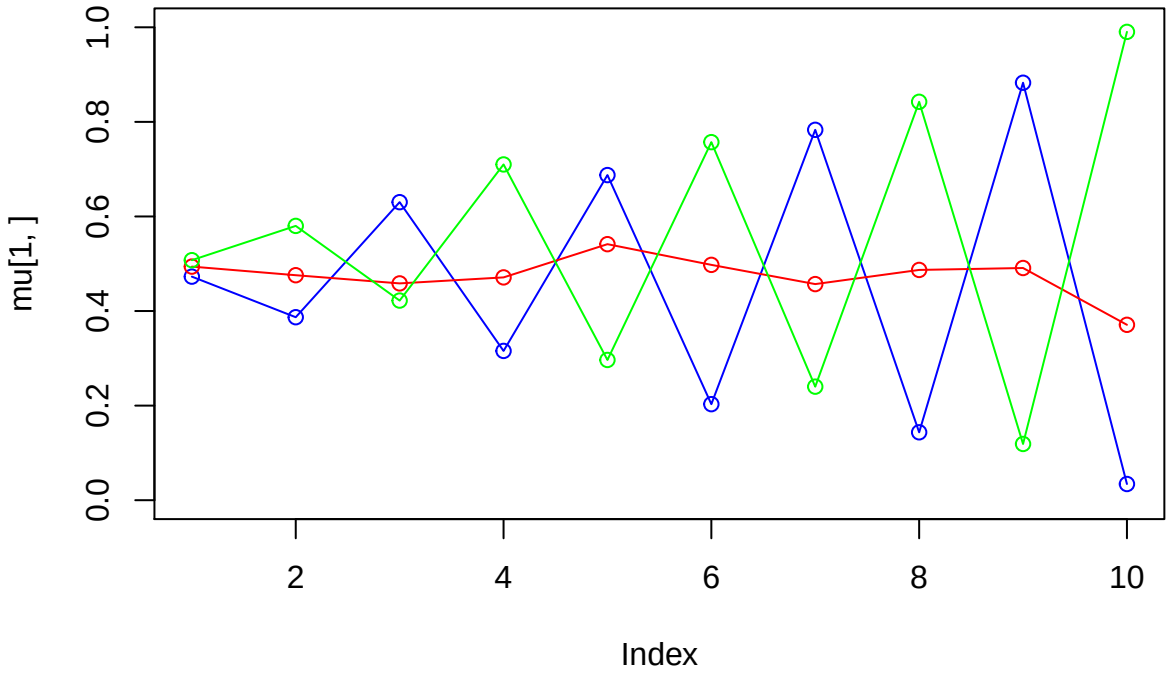
The plot shows two multivariate Bernoulli distributions estimated by the expectation-maximization (EM) algorithm. We can see that the multivariate Bernoulli with equal probabilities for each class has not affected EM particular much in order to find the other two distributions. This is probably because equal probabilities even out in the long run, i.e. the noise from that distribution is approximately equal for both sides of the coin.

For K=3

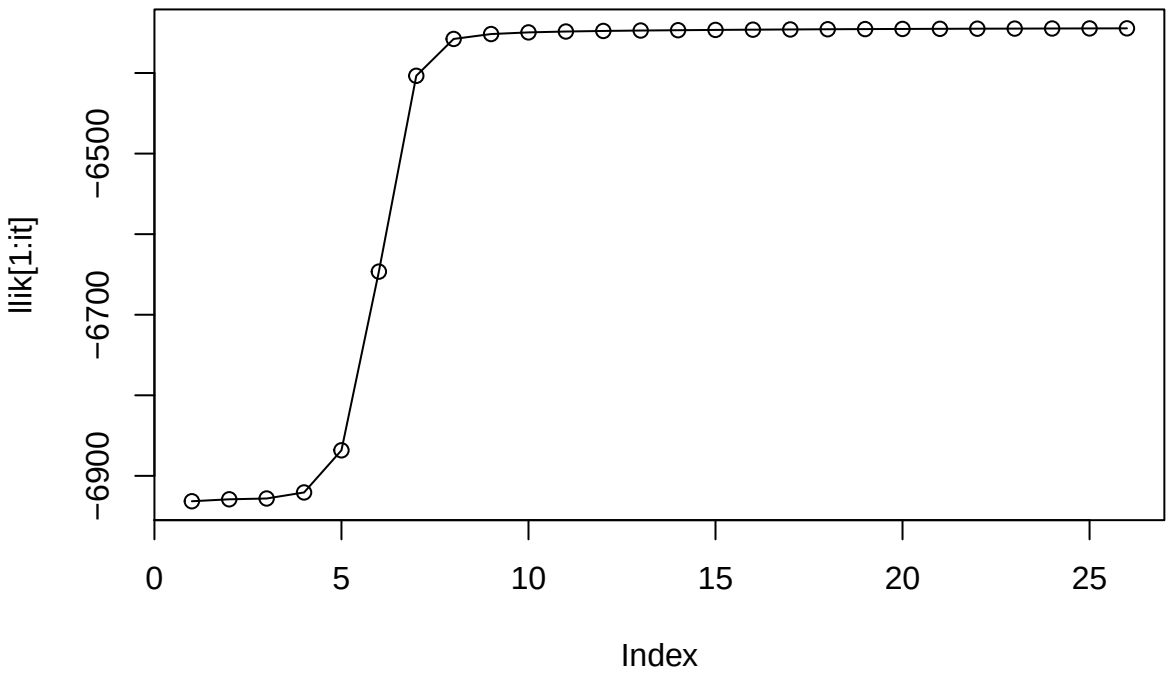



```
## iteration:  1 log likelihood:  -6931.482
## iteration:  2 log likelihood:  -6929.074
## iteration:  3 log likelihood:  -6928.081
## iteration:  4 log likelihood:  -6920.57
## iteration:  5 log likelihood:  -6868.29
## iteration:  6 log likelihood:  -6646.505
## iteration:  7 log likelihood:  -6403.476
## iteration:  8 log likelihood:  -6357.743
## iteration:  9 log likelihood:  -6351.637
## iteration:  10 log likelihood:  -6349.59
## iteration:  11 log likelihood:  -6348.513
## iteration:  12 log likelihood:  -6347.809
## iteration:  13 log likelihood:  -6347.284
## iteration:  14 log likelihood:  -6346.861
## iteration:  15 log likelihood:  -6346.506
## iteration:  16 log likelihood:  -6346.2
## iteration:  17 log likelihood:  -6345.934
## iteration:  18 log likelihood:  -6345.699
## iteration:  19 log likelihood:  -6345.492
## iteration:  20 log likelihood:  -6345.309
## iteration:  21 log likelihood:  -6345.147
## iteration:  22 log likelihood:  -6345.003
## iteration:  23 log likelihood:  -6344.875
## iteration:  24 log likelihood:  -6344.762
```

```
## iteration:  25 log likelihood:  -6344.66
## iteration:  26 log likelihood:  -6344.57
```

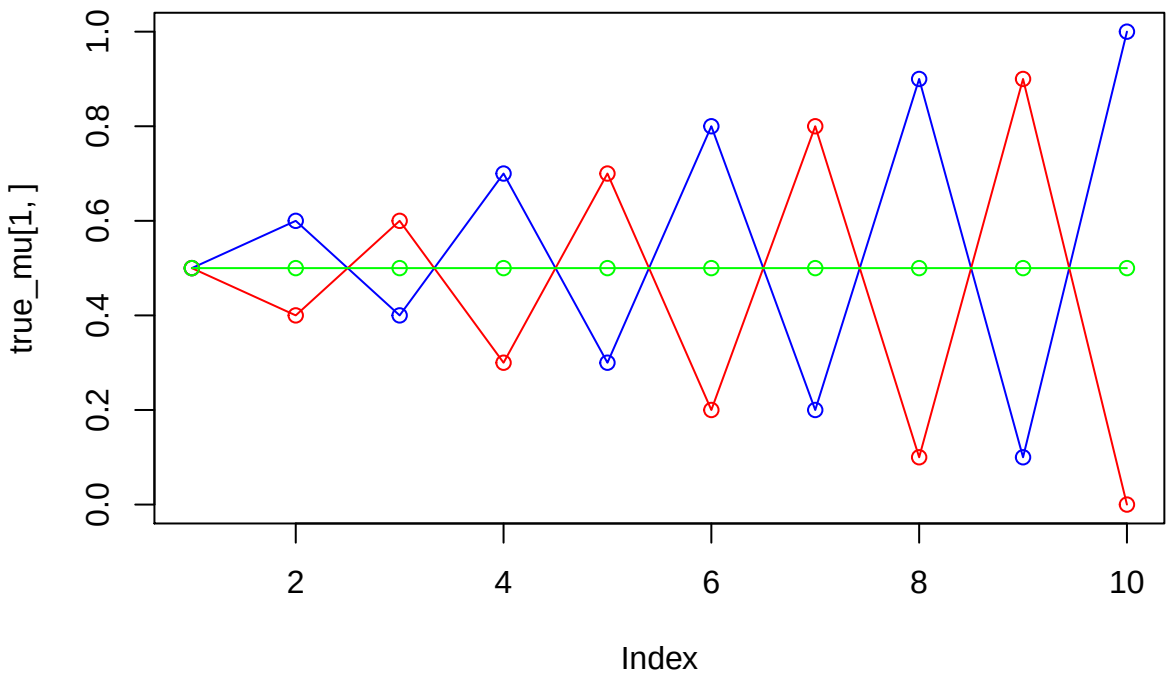



```
## value of updated pi is  0.3416794 0.2690298 0.3892909
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.4727544 0.3869396 0.6302224 0.3156325 0.6875038 0.2030173 0.7832090
## [2,] 0.4939501 0.4757687 0.4584644 0.4711358 0.5413928 0.4976325 0.4569664
## [3,] 0.5075441 0.5800156 0.4221148 0.7100227 0.2965478 0.7571593 0.2400675
##           [,8]      [,9]      [,10]
## [1,] 0.1435650 0.8827796 0.03422816
## [2,] 0.4869015 0.4909904 0.37087402
## [3,] 0.8424441 0.1188864 0.99033611
```



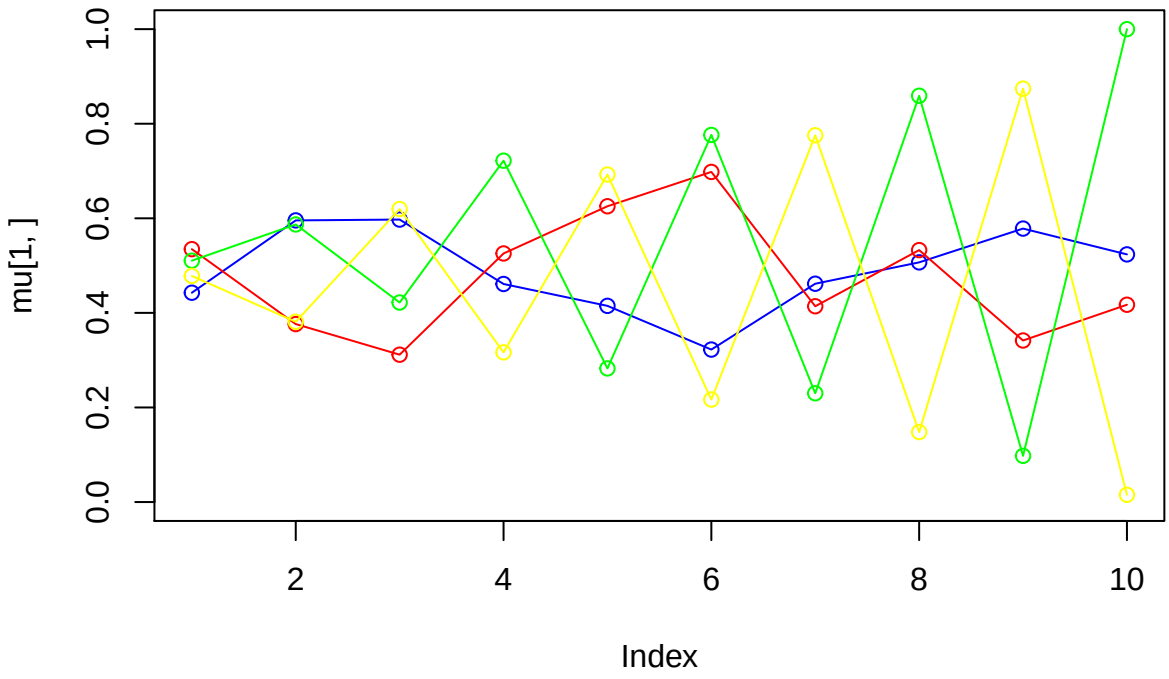
The plot above shows three multivariate Bernoulli distributions estimated by the EM algorithm. The distributions found are pretty similar to the true ones with exceptation of the uniform one which have been influenced by the other two distributions and/or bad luck on the coin flips that have resulted in seemingly unfair coins. One plot shows the log-likelihood versus the number of iterations.

For K=4



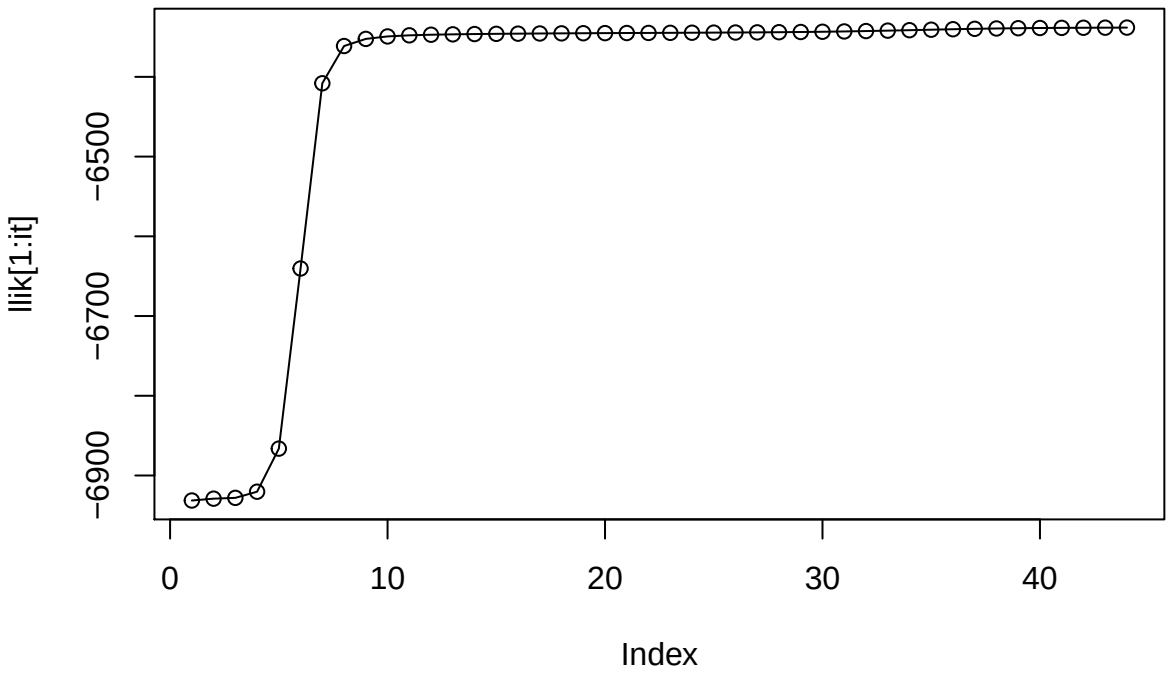

```
## iteration:  1 log likelihood:  -6931.372
## iteration:  2 log likelihood:  -6929.087
## iteration:  3 log likelihood:  -6928.057
## iteration:  4 log likelihood:  -6920.335
## iteration:  5 log likelihood:  -6866.277
## iteration:  6 log likelihood:  -6640.396
## iteration:  7 log likelihood:  -6408.058
## iteration:  8 log likelihood:  -6361.322
## iteration:  9 log likelihood:  -6352.413
## iteration:  10 log likelihood:  -6349.293
## iteration:  11 log likelihood:  -6347.902
## iteration:  12 log likelihood:  -6347.148
## iteration:  13 log likelihood:  -6346.663
## iteration:  14 log likelihood:  -6346.308
## iteration:  15 log likelihood:  -6346.028
## iteration:  16 log likelihood:  -6345.797
## iteration:  17 log likelihood:  -6345.601
## iteration:  18 log likelihood:  -6345.43
## iteration:  19 log likelihood:  -6345.279
## iteration:  20 log likelihood:  -6345.142
## iteration:  21 log likelihood:  -6345.015
## iteration:  22 log likelihood:  -6344.894
## iteration:  23 log likelihood:  -6344.775
## iteration:  24 log likelihood:  -6344.652
```

```
## iteration:  25 log likelihood:  -6344.52
## iteration:  26 log likelihood:  -6344.373
## iteration:  27 log likelihood:  -6344.2
## iteration:  28 log likelihood:  -6343.992
## iteration:  29 log likelihood:  -6343.737
## iteration:  30 log likelihood:  -6343.421
## iteration:  31 log likelihood:  -6343.033
## iteration:  32 log likelihood:  -6342.57
## iteration:  33 log likelihood:  -6342.036
## iteration:  34 log likelihood:  -6341.451
## iteration:  35 log likelihood:  -6340.849
## iteration:  36 log likelihood:  -6340.272
## iteration:  37 log likelihood:  -6339.757
## iteration:  38 log likelihood:  -6339.327
## iteration:  39 log likelihood:  -6338.988
## iteration:  40 log likelihood:  -6338.732
## iteration:  41 log likelihood:  -6338.544
## iteration:  42 log likelihood:  -6338.406
## iteration:  43 log likelihood:  -6338.304
## iteration:  44 log likelihood:  -6338.228
```




```
## value of updated pi is  0.1547196 0.1418652 0.3514089 0.3520062
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.4426228 0.5955990 0.5973038 0.4611075 0.4148259 0.3224465 0.4616759
## [2,] 0.5347882 0.3763616 0.3116137 0.5256451 0.6254569 0.6980795 0.4139865
## [3,] 0.5103748 0.5869840 0.4219499 0.7218615 0.2825337 0.7763136 0.2299954
```

```
## [4,] 0.4781150 0.3812010 0.6195949 0.3165236 0.6926095 0.2166850 0.7756026
##           [,8]       [,9]      [,10]
## [1,] 0.5068223 0.57827821 0.52366273
## [2,] 0.5327794 0.34159869 0.41722943
## [3,] 0.8591562 0.09774851 0.99998228
## [4,] 0.1479707 0.87418437 0.01530099
```



The plot above shows four multivariate Bernoulli distributions estimated by the EM algorithm. The blue and red curves are quite chaotic that do not resemble any of the true ones but taking the average would approximate the multivariate Bernoulli distribution with uniform parameters pretty well. So the EM algorithm have basically modelled two distributions based on the noise from the uniform one which is not surprising given that there are only three true distributions and that one is the most unpredictable.

For too few parameters that is for K=2, the logliklihood function runs for less iterations giving μ near to the true values of μ while for too many parameters the convergence steps increases. For K=3, the logliklihood value converges in neither too many nor too few steps, that is expected to provide result that is neither underfitted nor overfitted. For K=4 the convergence steps increases and the updated pi values for pi1 and pi2 differs greatly from the true value.

APPENDIX

```
## Question 1

spambase <- read.csv2("spambase.csv", header = TRUE, sep = ";", quote = "\"",
                      dec = ",", fill = TRUE)
spambase <- as.data.frame(spambase)
```

```
### Adaboost

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


number_of_trees <- seq(from = 10,to = 100, by = 10)

adaboost <- function(ntrees)
{
fit <- blackboost(as.factor(Spam) ~., data = train,
           control = boost_control(mstop = ntrees, nu=0.1),
           family = AdaExp())

# misclassification training:

ypredict <- predict(fit, newdata = train, type= "class")

conf_mat <- table(ypredict,train$Spam)

error_train <- 1-sum(diag(conf_mat))/sum(conf_mat)

# misclassification test

ypredict <- predict(fit, newdata = test, type= "class")

conf_mat <- table(ypredict,test$Spam)

error_test <- 1-sum(diag(conf_mat))/sum(conf_mat)

return(list(error_train = error_train,error_test = error_test))
}

error_rates_ada <- as.data.frame(t(sapply(number_of_trees, adaboost)))

training_error <- as.vector(unlist(error_rates_ada$error_train))

test_error <- as.vector(unlist(error_rates_ada$error_test))

plot(training_error,type = "b",main="Training Misclassification", xlab= "Number of Trees", ylab= "Error
      col="red", pch=19, cex=1)

plot(test_error,type = "b",main="Test Misclassification", xlab= "Number of Trees", ylab= "Error",
     col="blue", pch=19, cex=1)

# Loss Function = exp(-y * f)

training = sample(1:n,floor(n*2/3))

random_forest <- function(ntrees)
```

```
{
  fit <- randomForest(as.factor(Spam) ~ ., data=spambase, subset = training, importance=TRUE,
                      ntree = ntrees)

  # train misclassification

  ypredict <- predict(fit, newdata = train, type= "class")

  conf_mat <- table(ypredict,train$Spam)

  error_train <- 1-sum(diag(conf_mat))/sum(conf_mat)


  # test misclassification
  ypredict <- predict(fit, test,type ="class")

  conf_mat <- table(ypredict,test$Spam)

  error_test <- 1-sum(diag(conf_mat))/sum(conf_mat)
  return(list(error_train = error_train,error_test = error_test))
}

error_rates_random <- as.data.frame(t(sapply(number_of_trees, random_forest)))

training_error_rf <- as.vector(unlist(error_rates_random$error_train))

test_error_rf <- as.vector(unlist(error_rates_random$error_test))

plot(training_error_rf,type = "b",main="Training Misclassification", xlab= "Number of Trees", ylab= "Er
     col="red", pch=19, cex=1)


plot(test_error_rf,type = "b",main="Test Misclassification", xlab= "Number of Trees", ylab= "Error",
     col="blue", pch=19, cex=1)


df <- data.frame(adaboost = training_error, random_forest = training_error_rf,
                 no_of_trees = number_of_trees)

p1 <- ggplot(df, aes(no_of_trees, y = adaboost)) +
  geom_line(colour="blue")+geom_point()+
  ggtitle("Adaboost Vs Random Forest","Training Misclassification")

p2 <- ggplot(df, aes(no_of_trees, y = random_forest)) +
  geom_line(colour="red")+geom_point()

grid.arrange(p1,p2,ncol=1,nrow=2)


df1 <- data.frame(adaboost = test_error, random_forest = test_error_rf,
                 no_of_trees = number_of_trees)

p1 <- ggplot(df1, aes(no_of_trees, y = adaboost)) +
```

```
  geom_line(colour="blue")+geom_point()+
  ggtitle("Adaboost Vs Random Forest", "Test Misclassification")

p2 <- ggplot(df1, aes(no_of_trees, y = random_forest)) +
  geom_line(colour="red")+geom_point()

grid.arrange(p1,p2,ncol=1,nrow=2)

## Question 2
mixture_model <- function(my_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=my_k # 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(j in 1:my_k) {
   mu[j,] <- runif(D,0.49,0.51)
}
pi
mu

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 assignment

# Bernoulli distribution
for (n in 1:N)
{
  prob_x=0

  for (k in 1:K)
  {
    prob_x=prob_x+prod( ((mu[k,]^x[n,])*((1-mu[k,])^(1-x[n,]))) )*pi[k] #
  }

  for (k in 1:K)
  {

    z[n,k]=pi[k]*prod( ((mu[k,]^x[n,])*((1-mu[k,])^(1-x[n,]))) ) / prob_x

  }

}

#Log likelihood computation.

likelihood <-matrix(0,nrow =1000,ncol = K)

llik[it] <-0
for(n in 1:N)
{

  for (k in 1:K)
  {
    likelihood[n,k] <- pi[k]*prod( ((mu[k,]^x[n,])*((1-mu[k,])^(1-x[n,]))))
  }
```

```
    llik[it]<- sum(log(rowSums(likelihood)))

  }


  cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
  flush.console()
  # Stop if the lok likelihood has not changed significantly
  if (it > 1)
  {
    if (llik[it]-llik[it-1] < min_change)
    {
      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")
      }

      break
    }
  }

  #M-step: ML parameter estimation from the data and fractional component assignments

  mu<- (t(z) %*% x) /colSums(z)

  # N - Total no. of observations
  pi <- colSums(z)/N
}

cat("value of updated pi is " , pi )
cat("\n")
sprintf("value of updated mu is")
print(mu)

plot(llik[1:it], type="o")

}

mixture_model(2)
```

```
mixture_model(3)
mixture_model(4)
```