Lab 1 Block 2 Report

Mohammed Bakheet, Mudith Chathuranga Silva, Sreenand.S

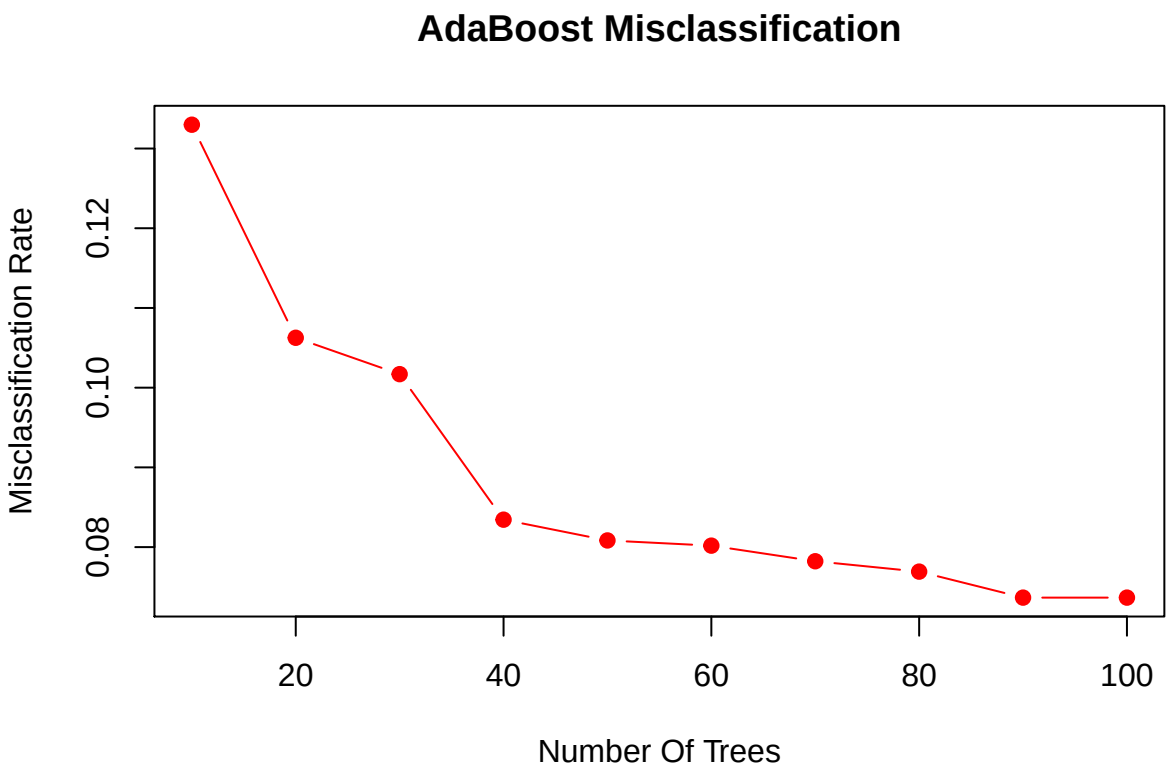
12/3/2019

Assignment 1

ADABoost

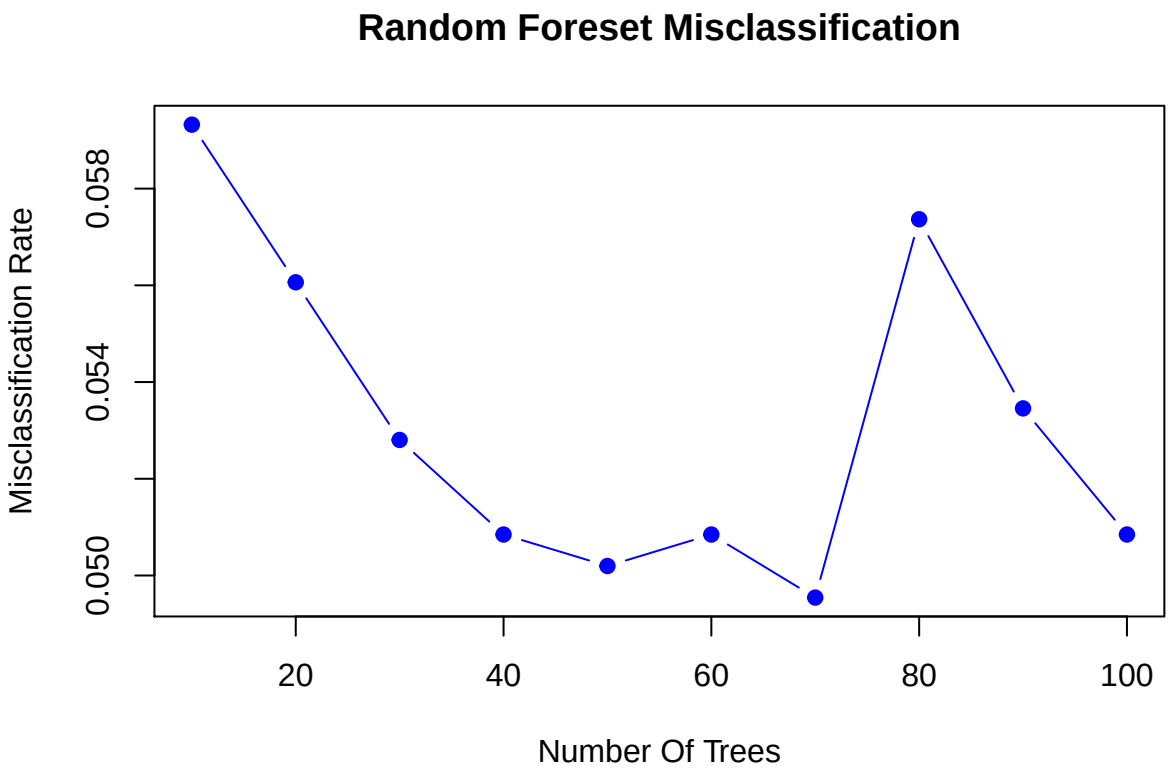
First, we need to separate train and test data. 66% of Data has been used for train and 33% has been used for the test.

Then Blackboost was used for ADABoost. The number of trees used as control parameters. **'AdaExp()'** selected as the loss function family.



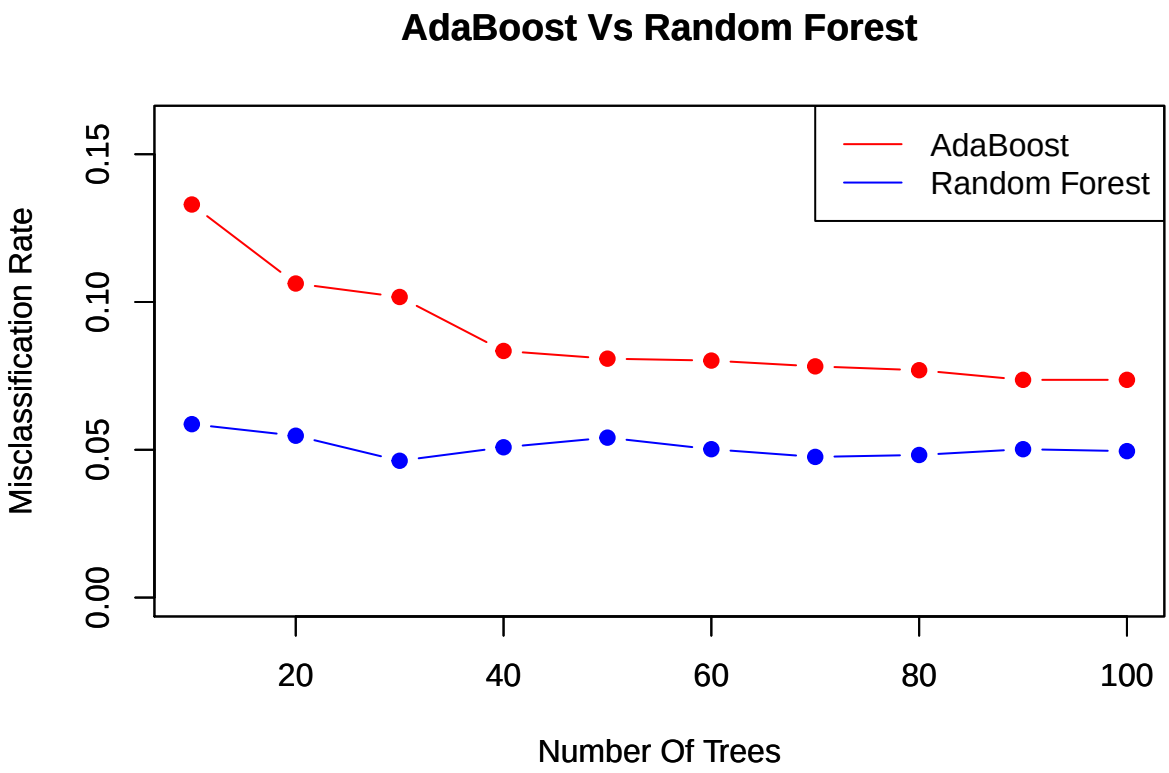
We can observe the lowest Misclassification rate at 90 Number of trees. But we can see that after 40 trees, the misclassification rate decreases slowly. Therefore 40 trees would be optimal for this case.

Random Forest



We can observe the lowest Misclassification rate at 70 Number of trees. But we can see that after 40,50 trees, the misclassification rate increases as the number of trees grows. The misclassification rate difference for 40 and 50 trees is relatively small. So the optimal option would be to select the option with the lowest misclassification rate with minimums number of trees. Therefore 40 is the optimal point.

Random Forest Vs ADABoost



By checking the above graph, it's clear that Random forest method has much lower Misclassification rates than ADABoost. Therefore Random forest has better performance than the ADABoost.

Assignment 2

From Lecture Notes :-

Mixture of multivariate Bernoulli distributions: $p(X) = \sum_k \pi_k Bernoulli(X \mid \mu_k)$

where we assume that: $Bernoulli(X \mid \mu_k) = \Pi_k \mu_{ki}^{xi}(1 - \mu_{ki})^{(1-xi)}$

Expectation Step (E Step) - Compute $p(z_{nk} \mid X_n, \mu, \pi)$ for all k and n

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

Maximum Likelihood

$\log(p(X \mid \mu, \pi)) = \sum_{n-1}^{N} log(\sum_{n-1}^{N} \pi_k p(X_n \mid \mu_n))$

Maximization Step (M Step) - Set π_k to π_k^{ML} k , and i

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

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

For Every Componenet (K) there are three graphs.

First Graph :- Three Multivariate Bernoulli Distributions true mean value vs index.

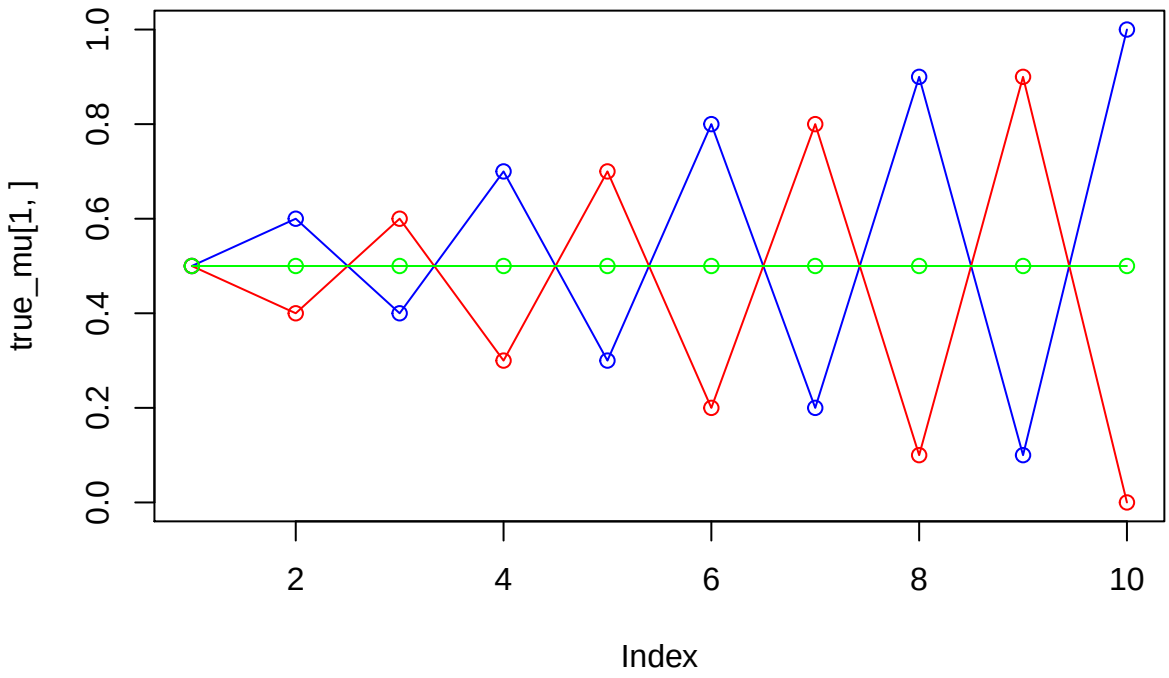
Second Graph :- Multivariate Bernoulli Distributions mean value vs index values estimated by the EM Algorithm.

Third Graph :- log value of Likelihood vs iteration.

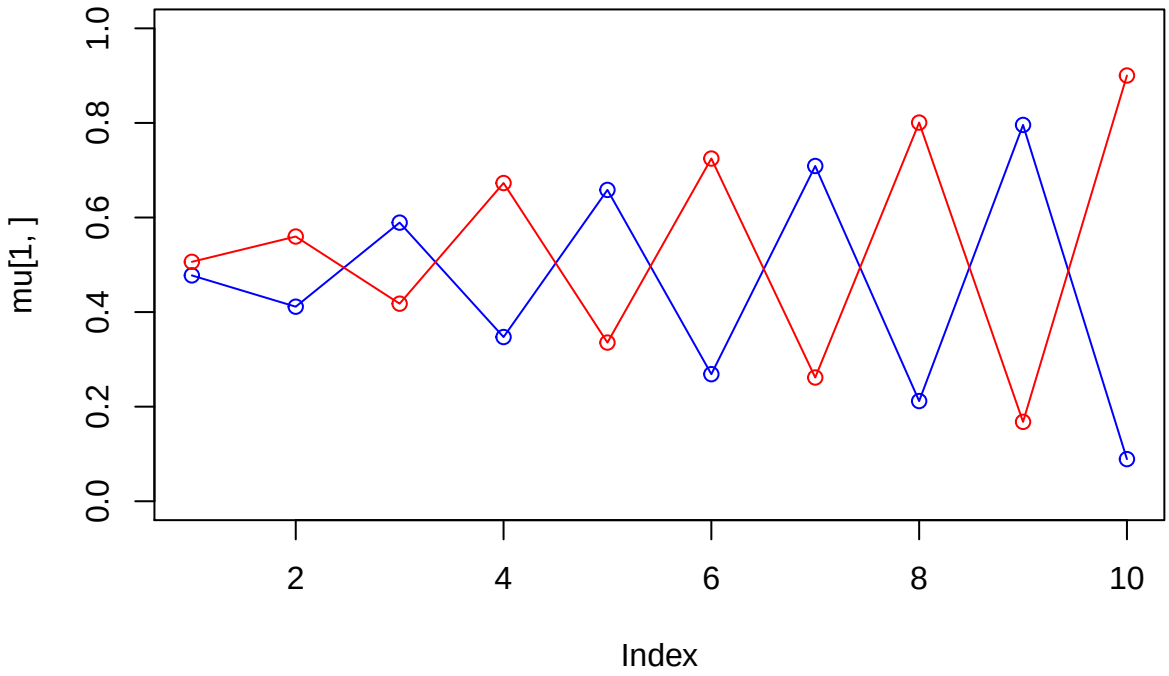
Note We got Error while we were try to generate %^% for total probability and log Likelihood.

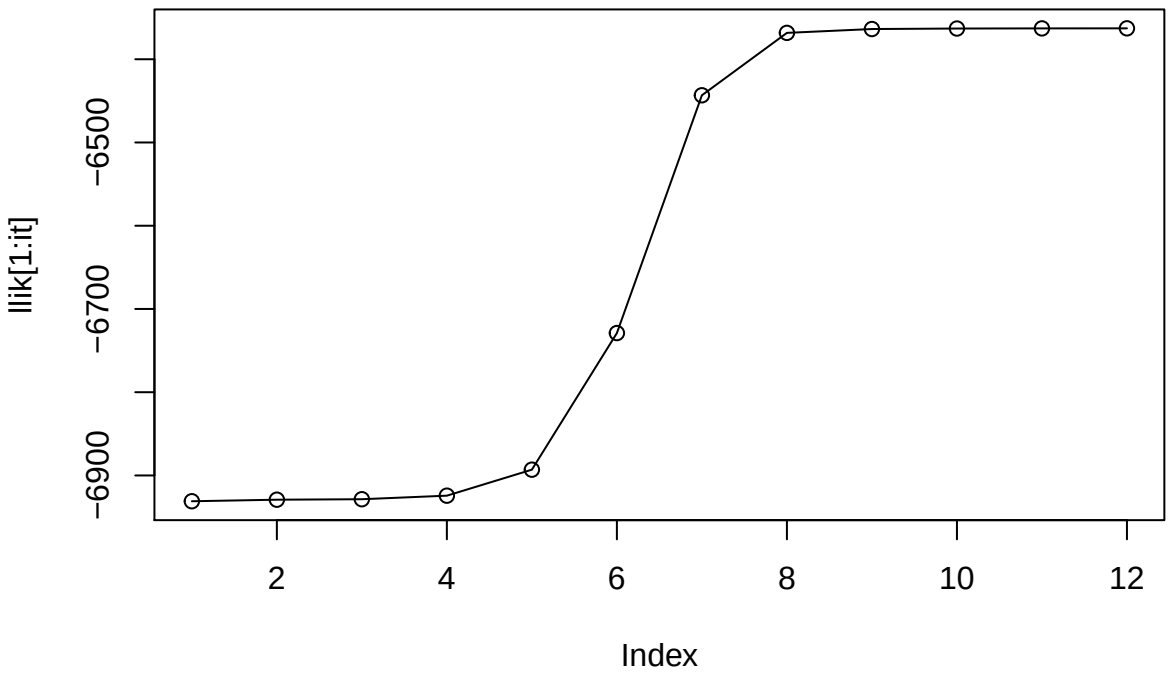
Error in mu[component,] %^% x[tpoint,] : could not find function "%^%" That's why we use prod()

For K = 2

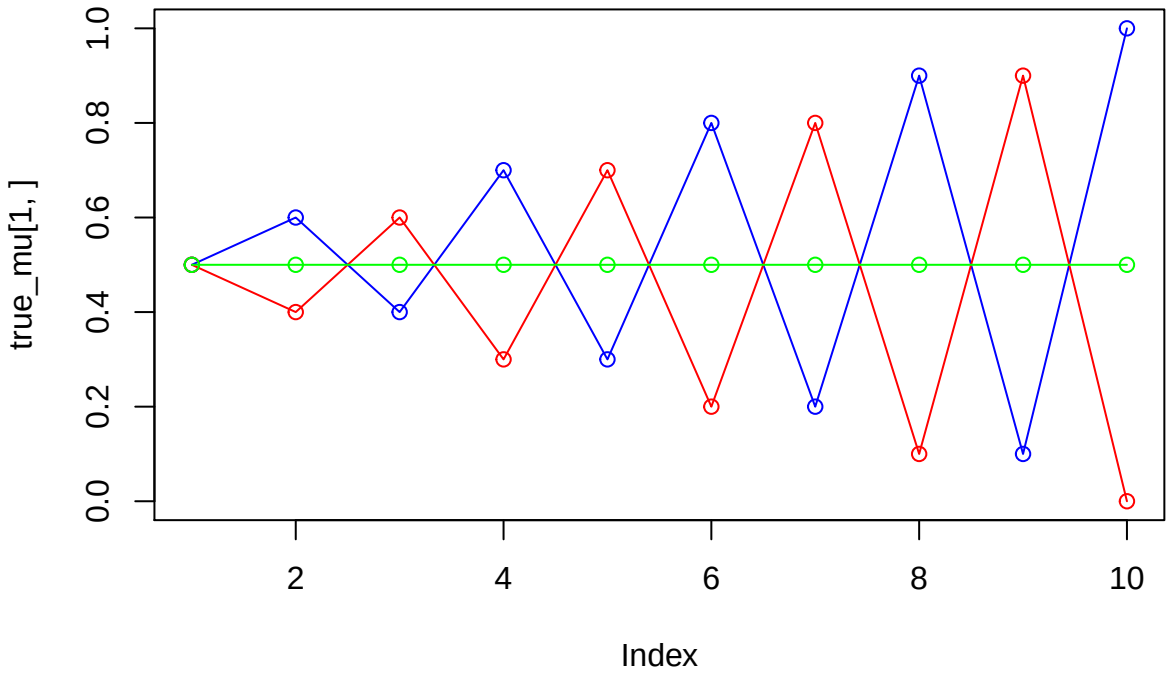



```
## iteration:  1 log likelihood:  -6930.975
## iteration:  2 log likelihood:  -6929.125
## iteration:  3 log likelihood:  -6928.562
## iteration:  4 log likelihood:  -6924.281
## iteration:  5 log likelihood:  -6893.055
## iteration:  6 log likelihood:  -6728.948
## iteration:  7 log likelihood:  -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
```

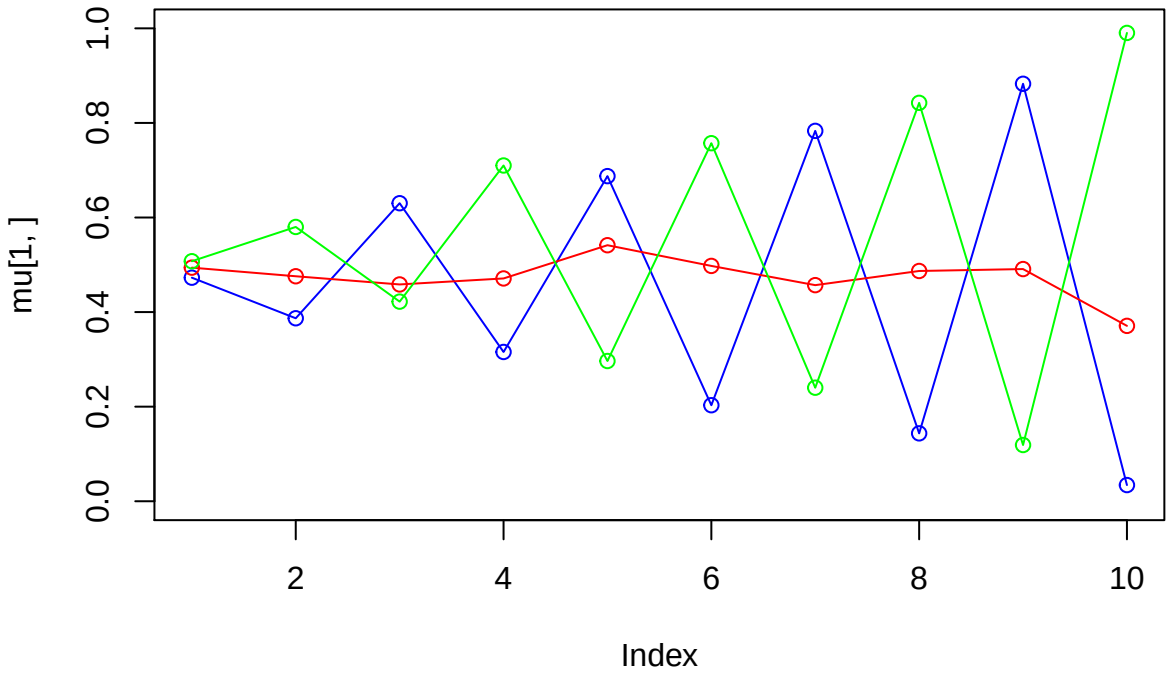

mu[1,]
0.0
0.2
0.4
0.6
0.8
1.0
2
4
6
8
10
Index

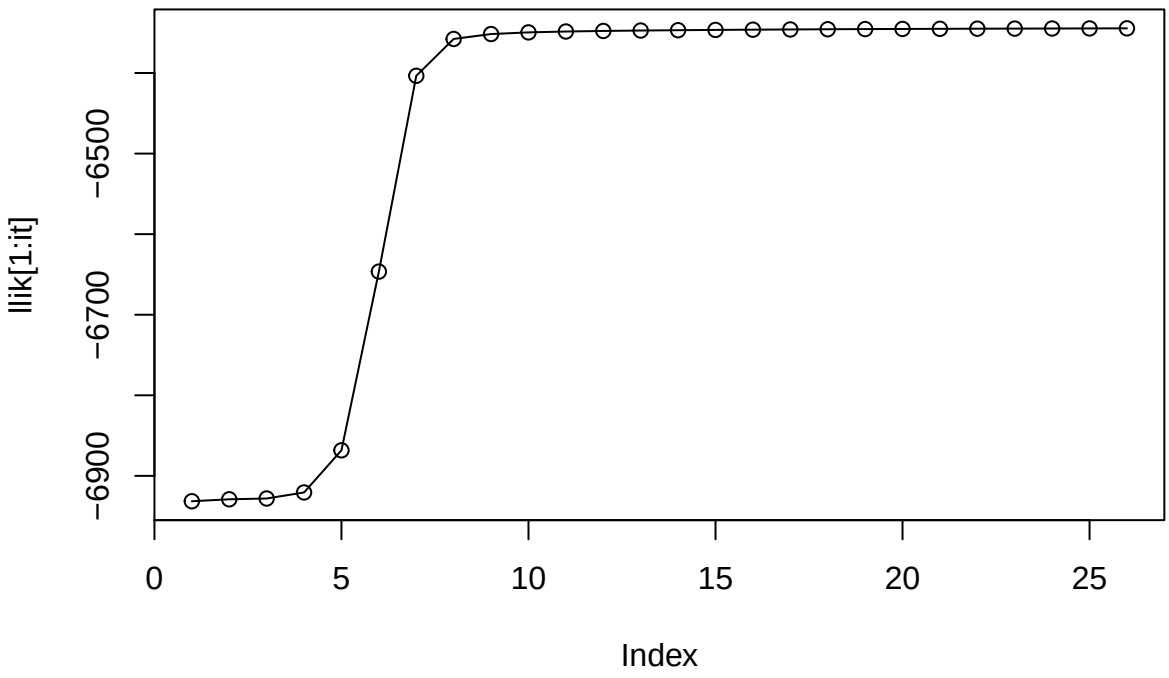


For K = 3

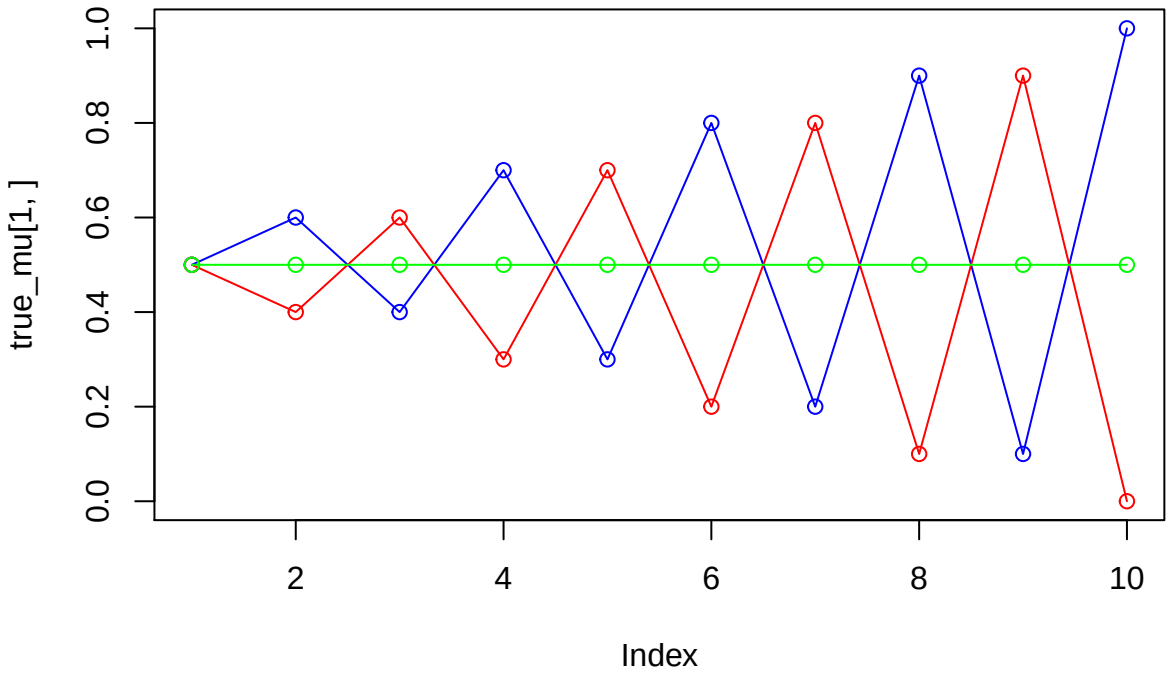



```
## iteration:  1 log likelihood:  -6931.482
## iteration:  2 log likelihood:  -6929.074
## iteration:  3 log likelihood:  -6928.081
## iteration:  4 log likelihood:  -6920.57
## iteration:  5 log likelihood:  -6868.29
## iteration:  6 log likelihood:  -6646.505
## iteration:  7 log likelihood:  -6403.476
## 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
```


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

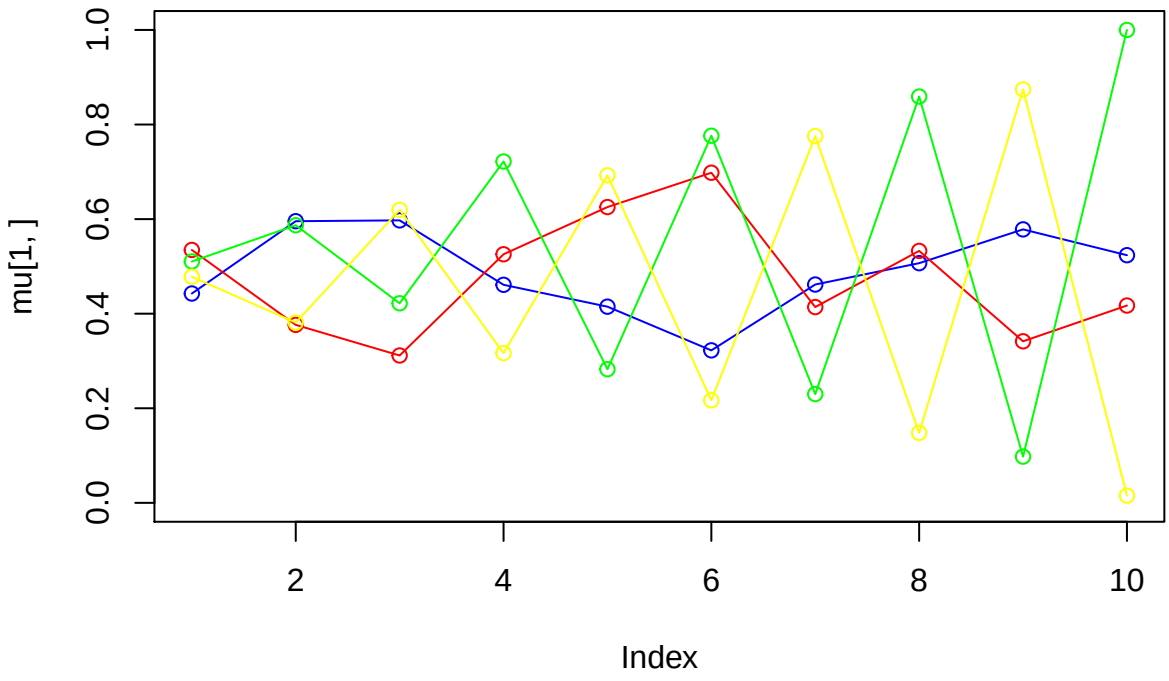


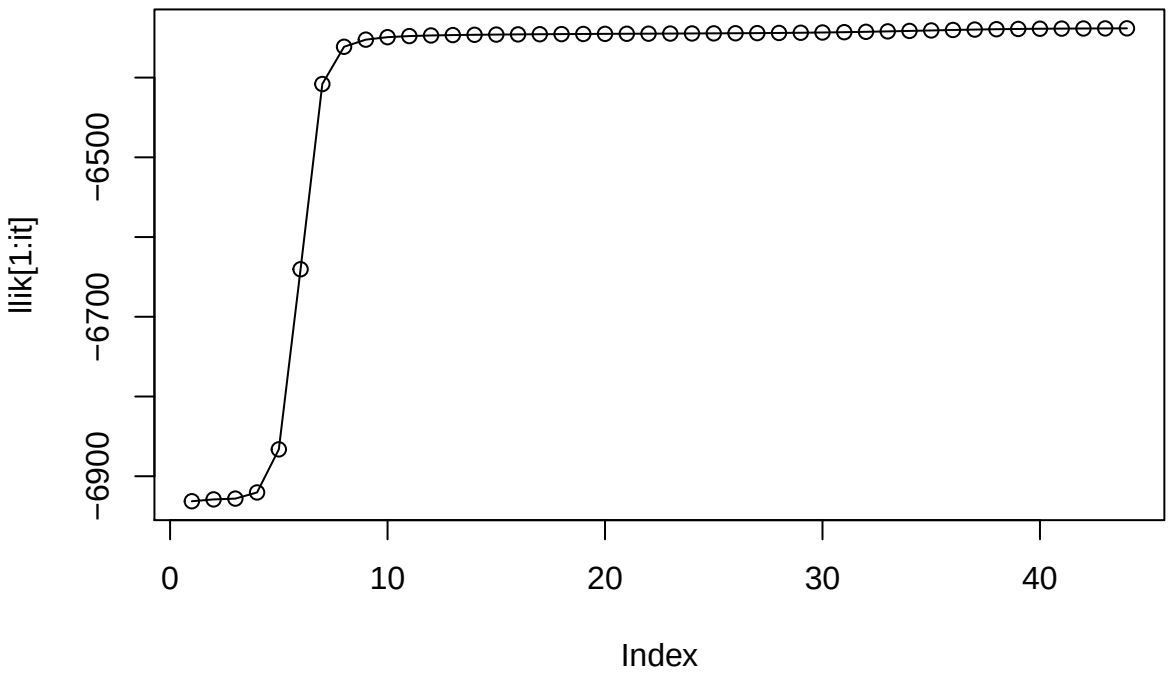
For K = 4




```
## 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
```





Too many components will result in overfitting and also few components will result in underfitting. For the case K = 2 (Few components) 12 iterations need to run in order to generate μ value near to the true μ value. For the case K = 4 (Too many components) 44 iterations need to run in order to generate μ value near to the true μ value which results in overfitting.

For K = 3, distributions estimated by EM algorithm is kind of similar to the true μ values except for the uniform distribution. That is because of the other two Bernoulli distributions.

APPENDIX

```
RNGversion('3.5.1')
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 * 2/3))
train = sp[id,] # train data 2/3
test = sp[-id,] # test data 1/3
```

```r
adaBoost = function() {
  number_of_trees <- seq(from = 10,to = 100, by = 10) # number of trees considered are 10; 20; : : : ;
  misclassification_rate = c()

  for (tree in number_of_trees) {
    classifier = blackboost(formula = Spam ~.,
                            data = train,
                            control = boost_control(mstop = tree),
                            family = AdaExp())

    y_pred = predict(classifier, newdata = test, type = 'class') # Type Class for classification trees
    confu_mat = table(y_pred, test$Spam)

    error_rate = 1 - (sum(diag(confu_mat)) / sum(confu_mat))

    misclassification_rate = c(misclassification_rate ,error_rate)
  }

  adaBoost = data.frame(trees = number_of_trees,
                        error = misclassification_rate)
  return(adaBoost)
}


plotAdaBoost = function() {
  adaBoostDataSet = adaBoost()
  # ggplot(adaBoostDataSet, aes(x = trees, y = error)) +
  #   geom_point(colour = 'red') +
  #   geom_line(colour = 'red') +
  #   ggtitle('AdaBoost Misclassification') +
  #   xlab('Number Of Trees') +
  #   ylab('Misclassification Rate')

  plot(x = adaBoostDataSet$trees,
       y = adaBoostDataSet$error,
       type = 'b',
       main = 'AdaBoost Misclassification',
       xlab = 'Number Of Trees',
       ylab = 'Misclassification Rate',
       col = 'red',
       pch = 19,
       cex = 1)
}

plotAdaBoost()


# ----------------------------

# Random Forest

rForest = function() {
  number_of_trees <- seq(from = 10,to = 100, by = 10) # number of trees considered are 10; 20; : : : ;
```

```
  misclassification_rate = c()

  for (tree in number_of_trees) {
    classifier = randomForest(formula = Spam ~.,
                              data = train,
                              importance = TRUE,
                              ntree = tree)

    y_pred = predict(classifier, newdata = test, type = 'class') # Type Class for classification trees
    confu_mat = table(y_pred, test$Spam)

    error_rate = 1 - (sum(diag(confu_mat)) / sum(confu_mat))

    misclassification_rate = c(misclassification_rate ,error_rate)
  }

  rfData = data.frame(trees = number_of_trees,
                      error = misclassification_rate)
  return(rfData)
}

plotRandomForest = function() {
  randomForestDataSet = rForest()

  plot(x = randomForestDataSet$trees,
       y = randomForestDataSet$error,
       type = 'b',
       main = 'Random Foreset Misclassification',
       xlab = 'Number Of Trees',
       ylab = 'Misclassification Rate',
       col = 'blue',
       pch = 19,
       cex = 1)
}

plotRandomForest()

# ----------------------------

# Random Forest Vs Ada Boost Comparison

compareAdaBoostAndRF = function() {
  adaBoostDataSet = adaBoost()
  randomForestDataSet = rForest()

  plot(x = adaBoostDataSet$trees,
       y = adaBoostDataSet$error,
       type = 'b',
       main = 'AdaBoost Vs Random Forest',
       xlab = 'Number Of Trees',
       ylab = 'Misclassification Rate',
       col = 'red',
       pch = 19,
```

```
      cex = 1,
      ylim = c(0,0.16)
      )

  par(new=TRUE)  # Combine two Plots

  plot(x = randomForestDataSet$trees,
      y = randomForestDataSet$error,
      type = 'b',
      main = 'AdaBoost Vs Random Forest',
      xlab = 'Number Of Trees',
      ylab = 'Misclassification Rate',
      col = 'blue',
      pch = 19,
      cex = 1,
      ylim = c(0,0.16)
      )

  legend("topright",
        legend = c("AdaBoost","Random Forest"),
        col = c("red","blue"),
        lty = 1,
        cex = 1
        )

}

compareAdaBoostAndRF()
```

```
mixture_model_data = function(kValue) {
  RNGversion('3.5.1')
  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 = kValue # 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) {

  # We need to Print only the last diagram.
  # We can itroduce this code before break the statement

  #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 (tpoint in 1:N) {
    total_probability = 0

    # Calculate Total Probability (For All K components)
    for (component in 1:K) {
      total_probability = total_probability + (prod((mu[component,] ^ x[tpoint,]) * ((1 - mu[componen
    }

    # Calculate Probability For K component
    for (component in 1:K) {
      z[tpoint,component] = (pi[component] * prod((mu[component,] ^ x[tpoint,]) * ((1-mu[component,])
    }
  }

  #Log likelihood computation.
  # Your code here

  likelihood = matrix(nrow = N,ncol = K)

  llik[it] = 0

  for(tpoint in 1:N) {
    for (component in 1:K) {
```

```
        likelihood[tpoint,component] = pi[component] * prod(((mu[component,] ^ x[tpoint,]) * ((1 - mu[c
      }
    }
    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
    # Your code here

    if ((llik[it] - llik[it-1] < min_change) && (it > 1)) {
      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
    # Your code here

    mu = (t(z) %*% x) / colSums(z)
    pi = colSums(z) / N
  }
  pi
  mu
  plot(llik[1:it], type="o")
}
```

References

https://stats.stackexchange.com/questions/55132/em-algorithm-manually-im

https://cedar.buffalo.edu/~srihari/CSE574/Chap9/Ch9.4-MixturesofBernou pdf