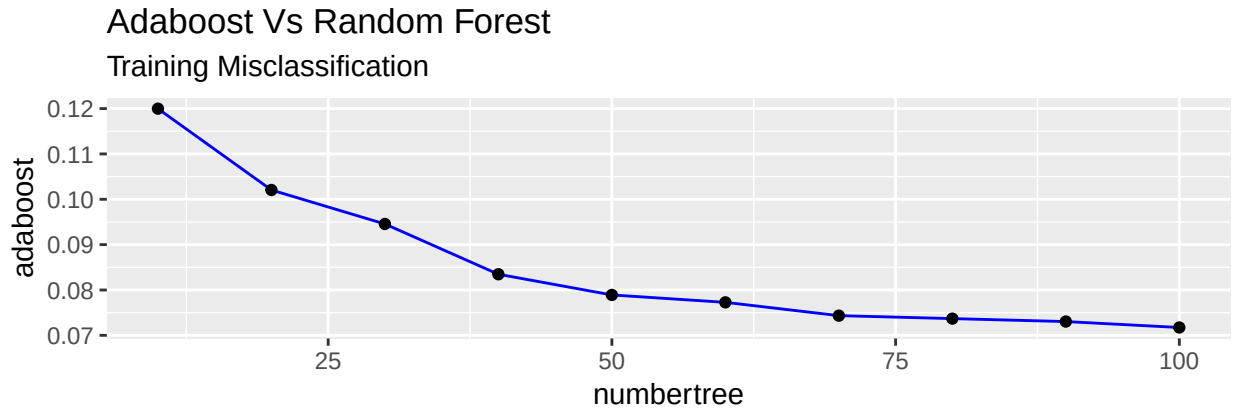
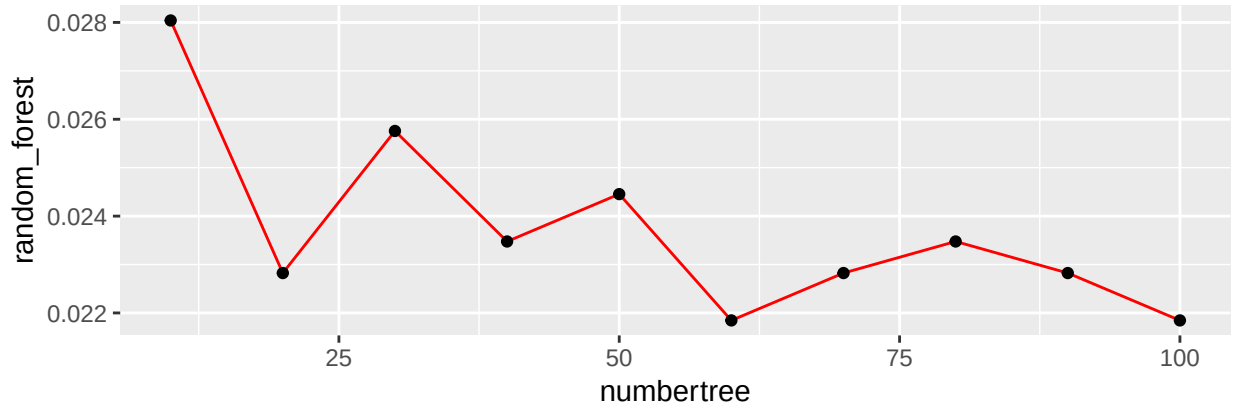
Lab1 Block2

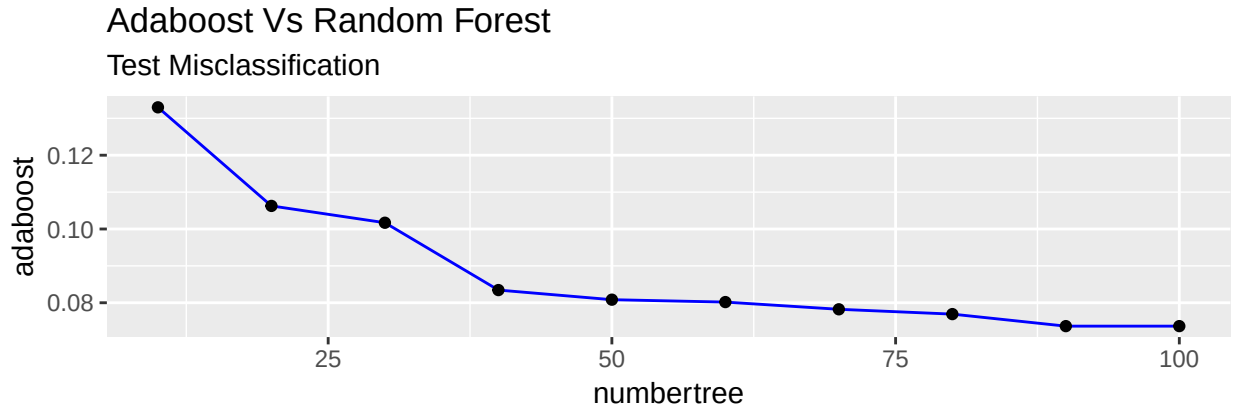
Omkar Bhutra (omkbh878)

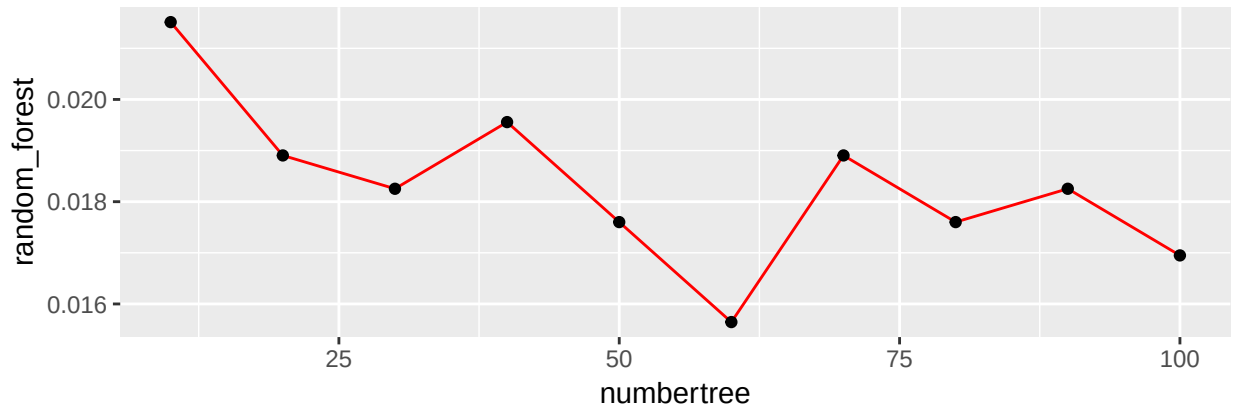
4th December 2018

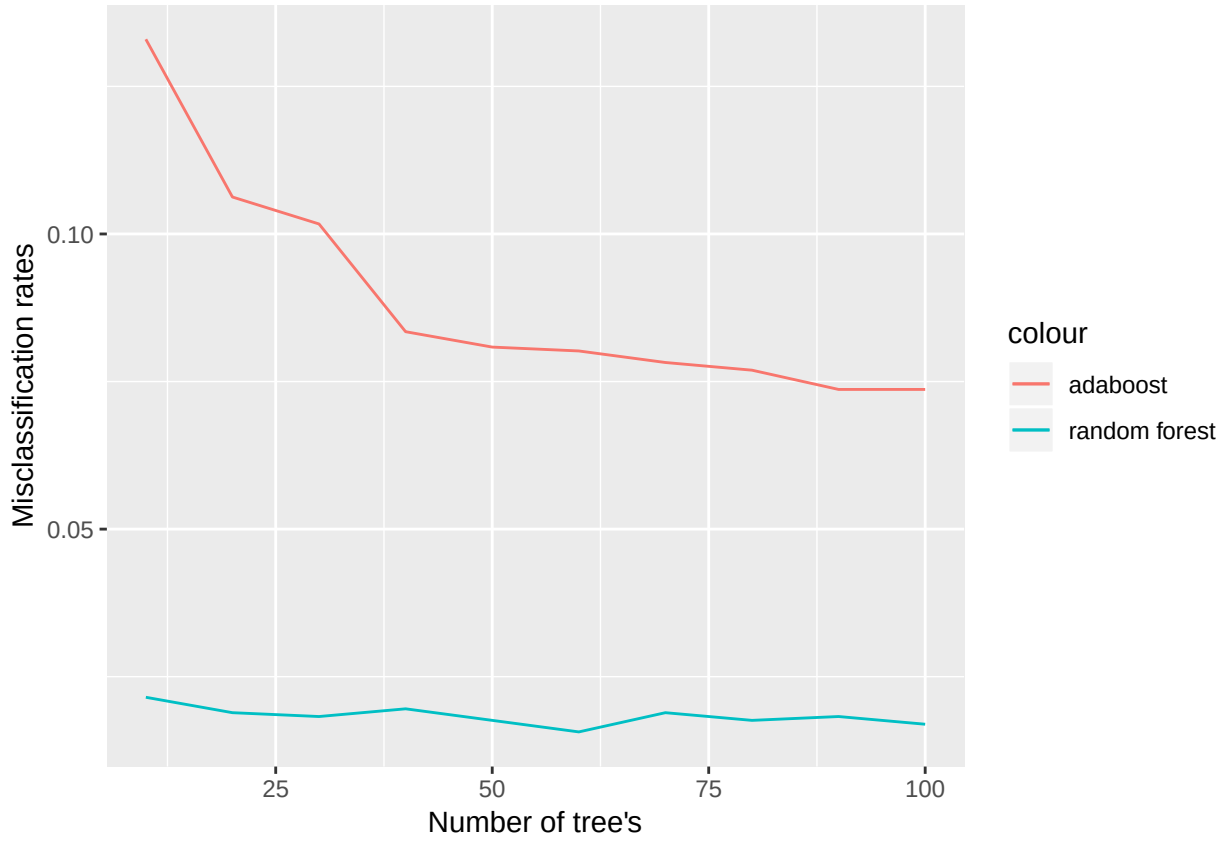
Question 1


Adaboost Vs Random Forest
Training Misclassification
adaboost
0.12
0.11
0.10
0.09
0.08
0.07
25
50
75
100
numbertree


random_forest
0.028
0.026
0.024
0.022
25
50
75
100
numbertree


Adaboost Vs Random Forest
Test Misclassification
adaboost
0.12
0.10
0.08
25
50
75
100
numbertree


random_forest
0.020
0.018
0.016
25
50
75
100
numbertree



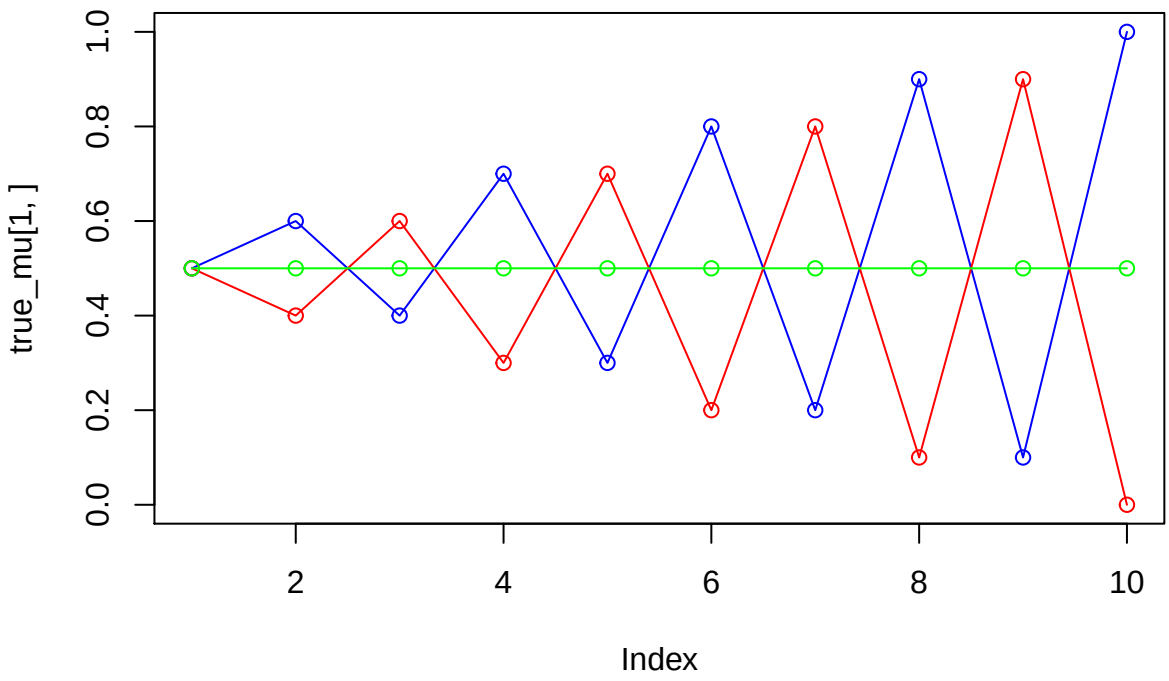

```
## $title
## [1] "Performance evaluation of Adaboost Vs Random Forest"
##
## $subtitle
## [1] "Testing Misclassification rates"
##
## attr(,"class")
## [1] "labels"
```

Two ensemble methods, Adaboost classification and Random Forest are implemented on the same data. Both methods converge weak regressions to form a committee regression (a combined strong regression). Adaboost changes the observations weight each time it is misclassified so that it can converge the solution based on recomputed weights. Random Forest combines many iterations to converge a solution which may not fit the data well but in this case is more accurate.

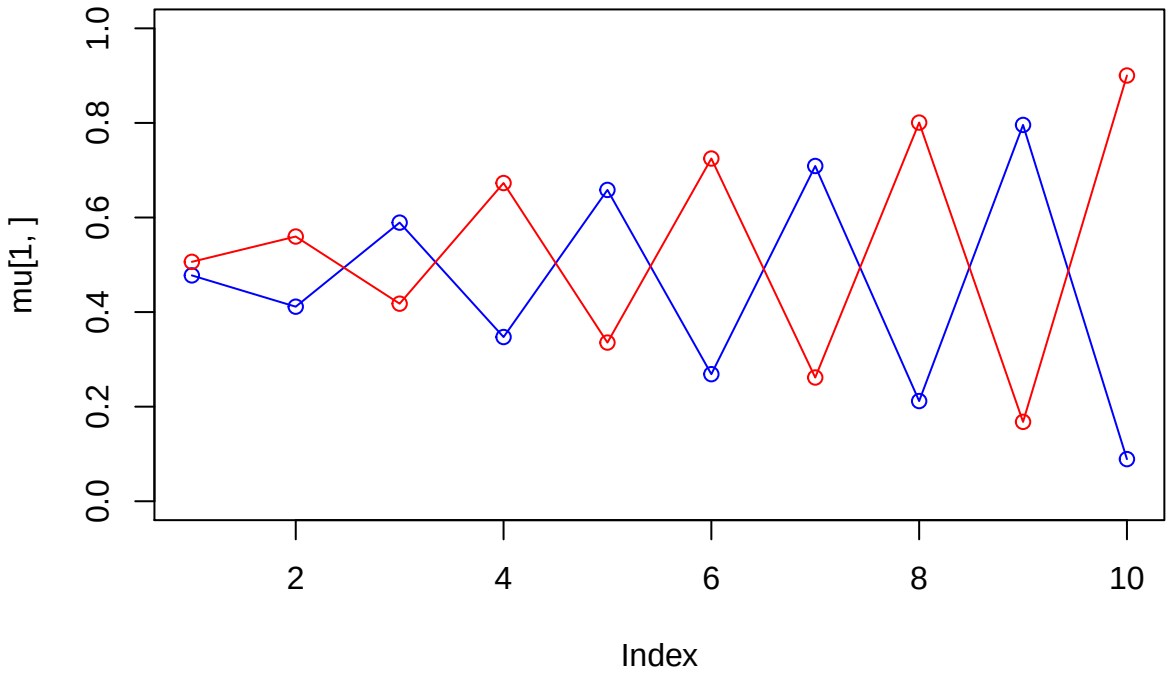
In the training data, it is observed that adaboost's misclassification rate decreases exponentially upto 65 tree's and then it slowly decreases to 0.07 upto 100 trees. The misclassification rate for random forest is choppy but generally decreases. It is seen that a model with 60 trees and 100 trees will have similar misclassification rates. In the testing data, Adaboost model with 50 trees has a low misclassification rate and can be used over a higher tree model to avoid complexity. The Random Forest model with 60 tree's has the lowest misclassification rate.

It is seen from the plots that random forest method produces lower misclassification rates and hence it can be said that it performs better than adaboost. thus, a Random forest model with 60 tree's is optimal.

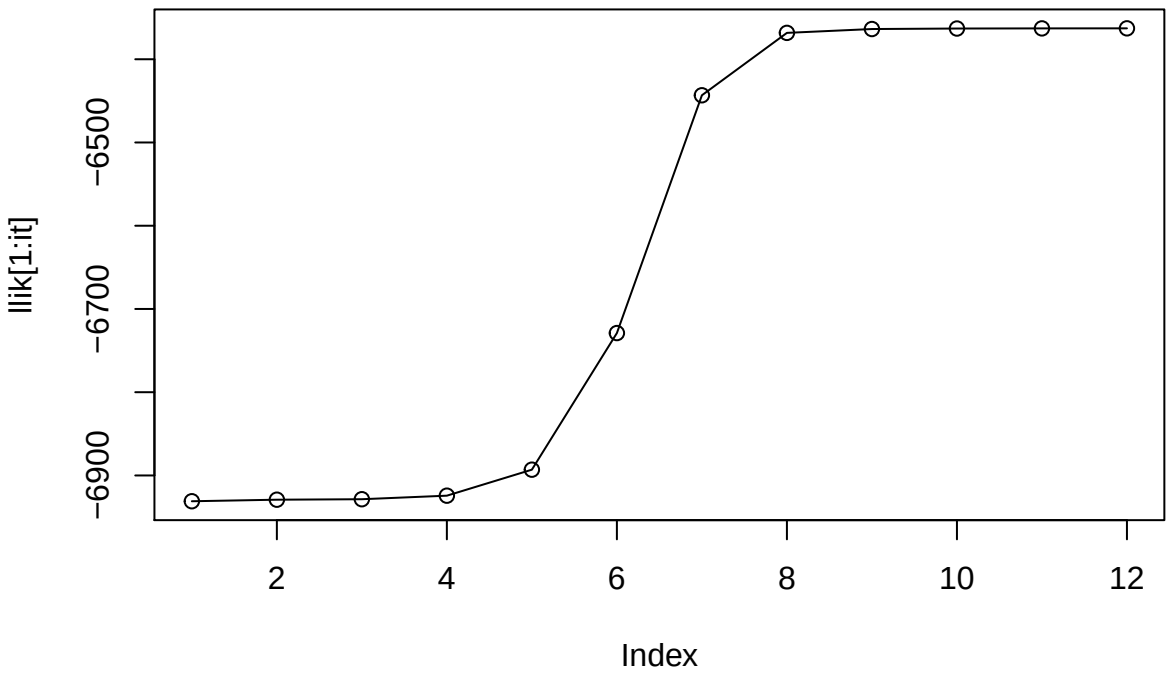
Question 2

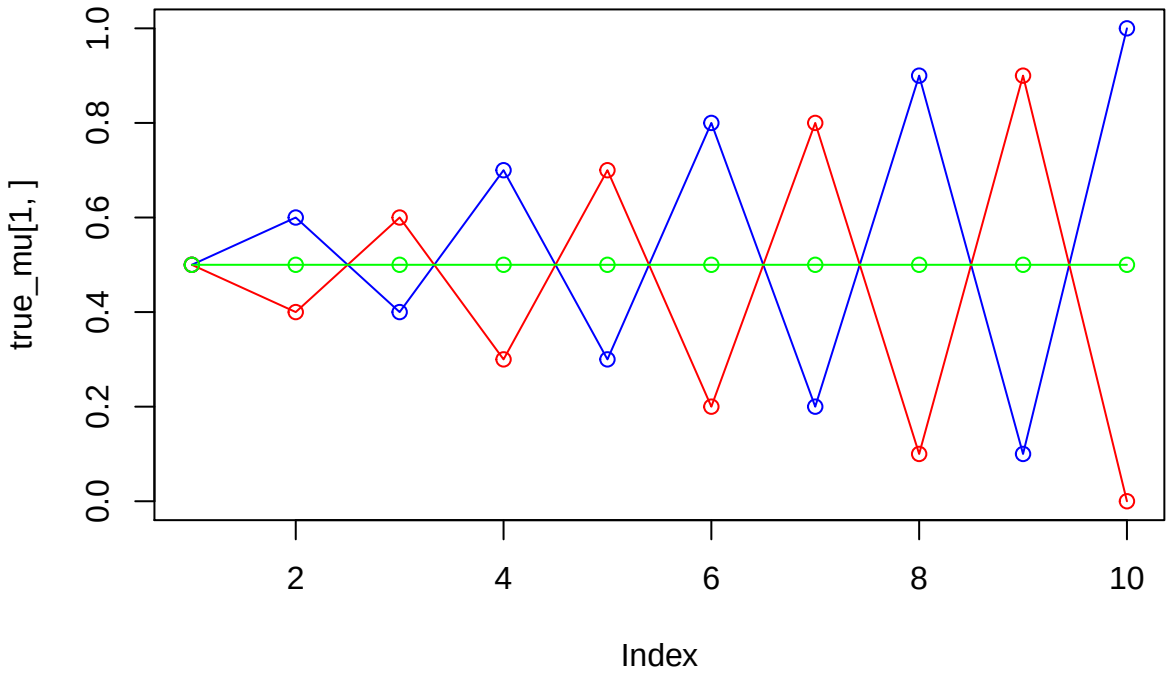



```
## iteration:  1 log likelihood:  -6930.975
## iteration:  2 log likelihood:  -6929.125
## iteration:  3 log likelihood:  -6928.562
## iteration:  4 log likelihood:  -6924.281
## iteration:  5 log likelihood:  -6893.055
## iteration:  6 log likelihood:  -6728.948
## iteration:  7 log likelihood: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

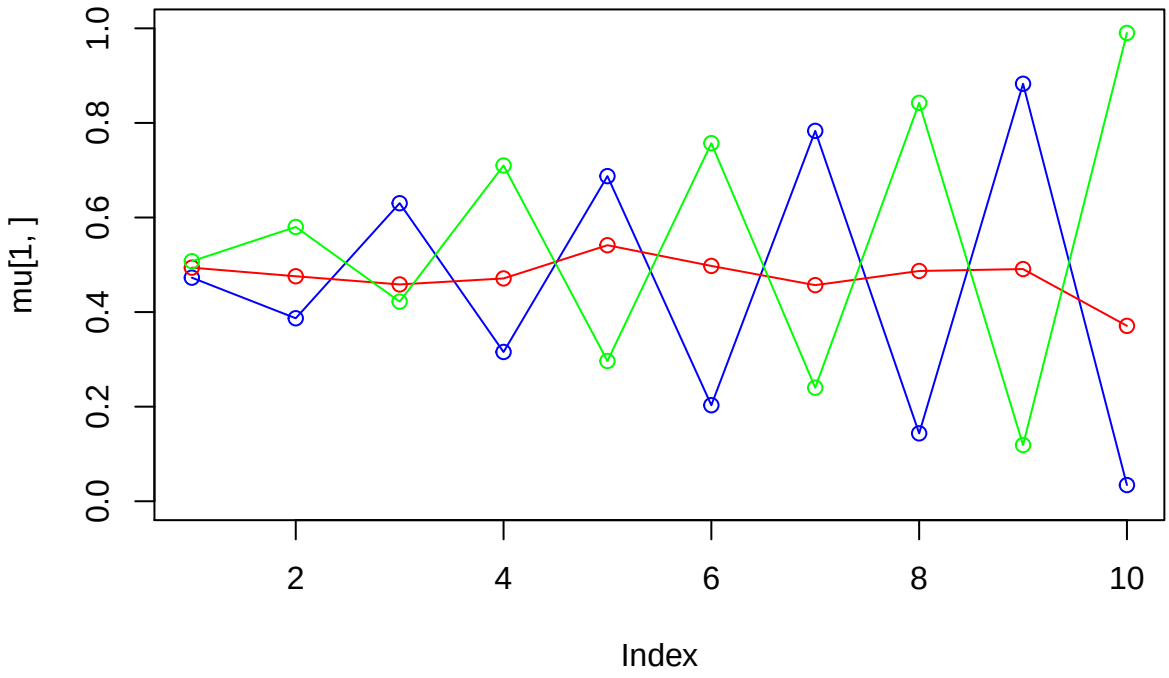



```
## value of updated pi is  0.497125 0.502875
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.4775488 0.4113939 0.5892308 0.3472420 0.6583712 0.2686589 0.7089490
## [2,] 0.5062860 0.5597531 0.4177551 0.6728856 0.3354854 0.7247188 0.2616231
##           [,8]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

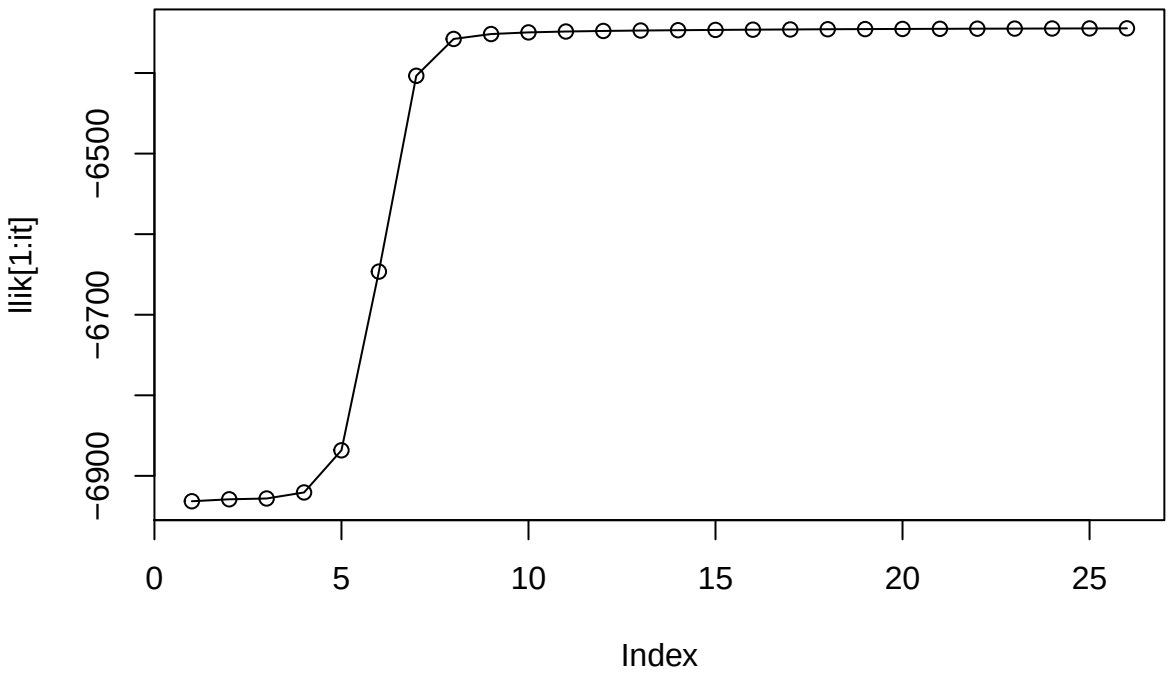

llik[1:it]
−6900
−6700
−6500
2
4
6
8
10
12
Index

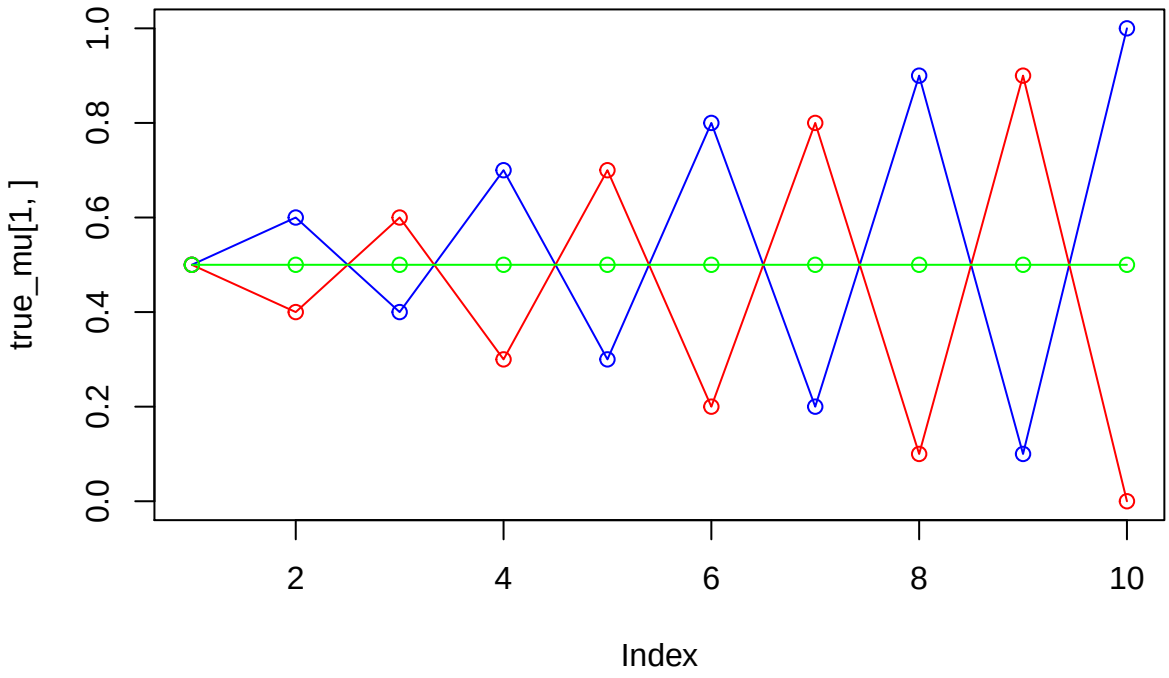



```
## iteration:  1 log likelihood:  -6931.482
## iteration:  2 log likelihood:  -6929.074
## iteration:  3 log likelihood:  -6928.081
## iteration:  4 log likelihood:  -6920.57
## iteration:  5 log likelihood:  -6868.29
## iteration:  6 log likelihood:  -6646.505
## iteration:  7 log likelihood:  -6403.476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



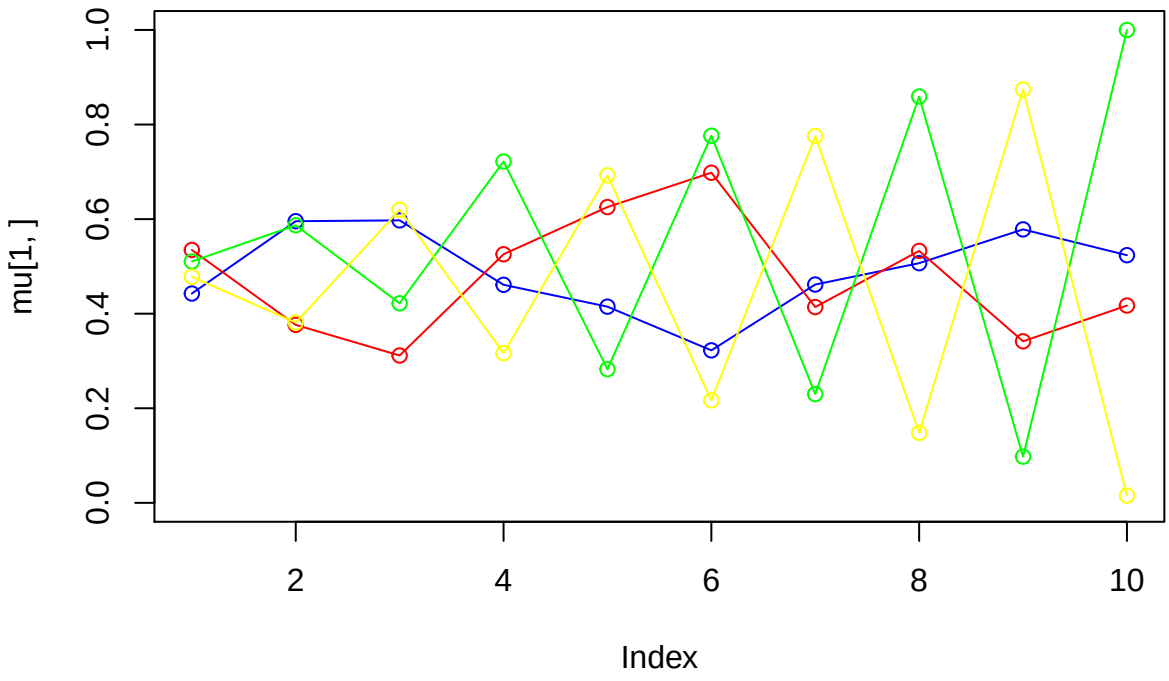

```
## value of updated pi is  0.3416794 0.2690298 0.3892909
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.4727544 0.3869396 0.6302224 0.3156325 0.6875038 0.2030173 0.7832090
## [2,] 0.4939501 0.4757687 0.4584644 0.4711358 0.5413928 0.4976325 0.4569664
## [3,] 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
```


llik[1:it]
−6900
−6700
−6500
0
5
10
15
20
25
Index



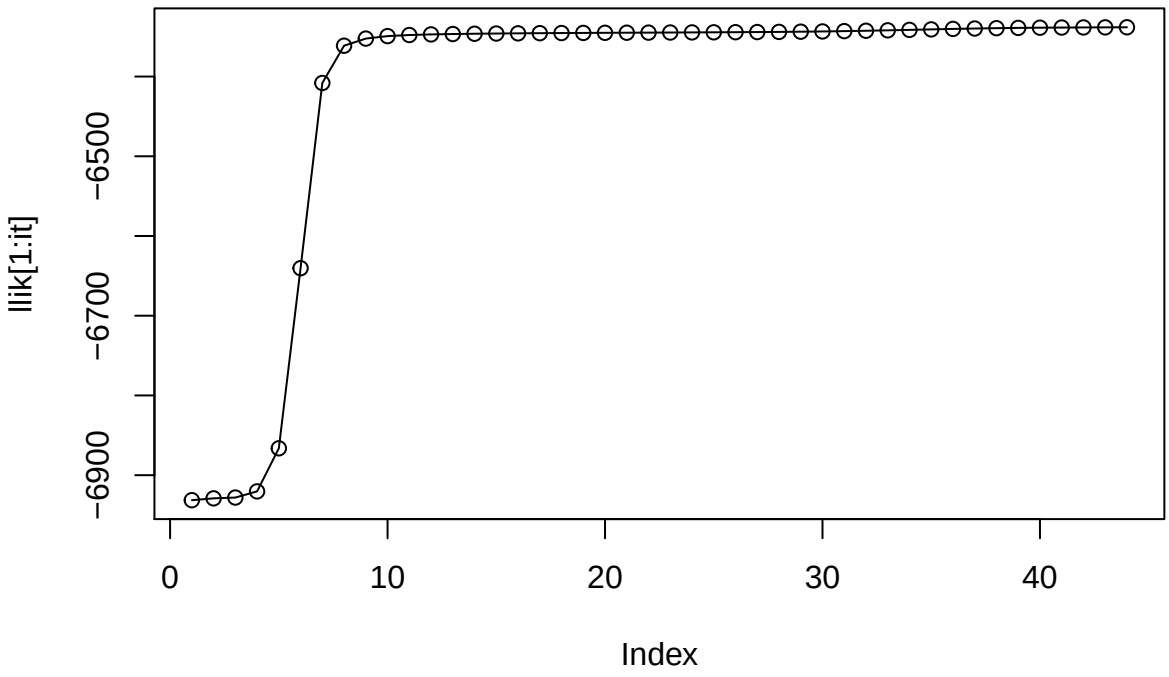

```
## iteration:  1 log likelihood:  -6931.372
## iteration:  2 log likelihood:  -6929.087
## iteration:  3 log likelihood:  -6928.057
## iteration:  4 log likelihood:  -6920.335
## iteration:  5 log likelihood:  -6866.277
## iteration:  6 log likelihood:  -6640.396
## iteration:  7 log likelihood:  -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
```



For k=2 (Two bernoulli distribution componenets) It is observed that two Bernoulli components with equal probabilities for each class has not affected the classification much. This is due to equivalant probabilities cancel out each other effects. The likelihood maximization is completed within 12 iterations. It also shows how in the log likelihood, starting out at ???6950 , goes exponential beyond the 4th iteration and increases upto 8th iteration and stablises around ???6350. It is observed that the final $\hat{\mu}_{\cdot,2}$ values are following the same trend as the true μ_1 and μ_2. Component μ_3 is distributed equally between $\hat{\mu}_{1,2}$ and $\hat{\mu}_{2,2}$ since it does not have distinct features: they are all equally likely, resulting in the maximum variance possible for a Bernoulli distribution ($Var(x) = \mu \cdot (1 - \mu) = 0.5 \cdot 0.5 = 0.25$).

For k=3 (Three bernoulli distribution componenets) When $K = 3$, $\hat{\pi}_{\cdot,3}$ values fluctuate around 0.33. The log-likelihood follows the same trend as the previous iteration. Increasing exponentially, after the 8th iteration. In the final $\hat{\mu}_{\cdot,3}$ values plot, the result resamble closely the hidden phenomena behind the data..

When $K = 4$, if we examine final $\hat{\mu}_{\cdot,4}$ plot, we can see the pairs $\{\hat{\mu}_{1,4}, \hat{\mu}_{2,4}\}$, and $\{\hat{\mu}_{3,4}, \hat{\mu}_{4,4}\}$ following the same trends. However it is still clear to see $\hat{\mu}_{1,4}$ and $\hat{\mu}_{2,4}$ behave as μ_1 and μ_2 in the real latent parameters. $\hat{\mu}_{3,4}$ and $\hat{\mu}_{4,4}$ have lowest probability in $\hat{\pi}_k$ plot.

In conclusion we can say that, if we increase the number of components, with the increase in complexity, some redundant components with similar behaviours can be found, as in the case of $K = 4$. Without the true distributions, it is difficult choice between $K = 2$ and $K = 3$: the extra component may or may not bepresent in the real data. It would be necessary to run further analysis.

For k=4 (Four bernoulli distribution componenets) The two curves are very noisy and they do not resemble the true curves,taking the average would approximate the multivariate Bernoulli distribution with uniform parameters well. So the EM algorithm has modelled two distributions based on the noise from the uniform one due to the prescence of only three true distributions. For too few parameters i.e for K=2, the log-likelihood function runs for fewer iterations giving μ near to the true values of μ while for too many parameters the convergence iterations increases. For K=3, the log-likelihood value converges in optimal iterations, that is expected to provide correct. For K=4 the convergence steps increases and the updated pi values for pi1 and pi differs greatly from the true value.

Appendix

```
knitr::opts_chunk$set(echo = TRUE)
library(mboost)
library(randomForest)
library(dplyr)
library(ggplot2)
library(plotly)
library(knitr)
library(manipulate)
library(gridExtra)
spambase <- read.csv2("spambase.csv", header = TRUE, sep = ";", quote = "\"",
dec = ",", fill = TRUE)
spambase$Spam<-as.factor(spambase$Spam)

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

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

adaboost <- function(notree)
{
spambase.gb <- blackboost(Spam ~., data = train,control = boost_control(mstop = notree),family = AdaExp

predicty <- predict(spambase.gb, newdata = train, type= "class")
confusionmatrix <- table(predicty,train$Spam)
errortrain <- 1-sum(diag(confusionmatrix))/sum(confusionmatrix)


predicty <- predict(spambase.gb, newdata = test, type= "class")
confusionmatrix <- table(predicty,test$Spam)
errortest <- 1-sum(diag(confusionmatrix))/sum(confusionmatrix)

return(list(errortrain= errortrain, errortest=errortest))
}

errorrates <- as.data.frame(t(sapply(numbertree, adaboost)))
trainingerror <- as.vector(unlist(errorrates$errortrain))
testerror <- as.vector(unlist(errorrates$errortest))
```

```
training = sample(1:n,floor(n*2/3))
random_forest <- function(notree)
{
spambase.rf <- randomForest(Spam ~ ., data=spambase, subset = training, importance=TRUE,ntree = notree)

ypredict <- predict(spambase.rf, newdata = train, type= "class")
confusionmatrix <- table(ypredict,train$Spam)
error_train <- 1-sum(diag(confusionmatrix))/sum(confusionmatrix)

ypredict <- predict(spambase.rf, newdata=test,btype ="class")
confusionmatrix <- table(ypredict,test$Spam)
error_test <- 1-sum(diag(confusionmatrix))/sum(confusionmatrix)
return(list(error_train = error_train,error_test = error_test))
}

errorrates_random <- as.data.frame(t(sapply(numbertree, random_forest)))
trainingerror_rf <- as.vector(unlist(errorrates_random$error_train))
testerror_rf <- as.vector(unlist(errorrates_random$error_test))

df <- data.frame(adaboost = trainingerror, random_forest = trainingerror_rf, numberoftrees = numbertree)

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

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

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


df1 <- data.frame(adaboost = testerror, random_forest = testerror_rf,nooftrees = numbertree)

p3 <- ggplot(df1, aes(x=numbertree, y = adaboost)) +geom_line(colour="blue")+geom_point()+
ggtitle("Adaboost Vs Random Forest", "Test Misclassification")

p4 <- ggplot(df1, aes(x=numbertree, y = random_forest)) +geom_line(colour="red")+geom_point()

grid.arrange(p3,p4,ncol=1,nrow=2)


ggplot(df1, aes(numbertree)) +
  geom_line(aes(y = adaboost, colour = "adaboost")) +
  geom_line(aes(y = random_forest, colour = "random forest"))+ labs(x="Number of tree's",y="Misclassifi
ggtitle("Performance evaluation of Adaboost Vs Random Forest","Testing Misclassification rates")

mixture_model <- function(newk)
{
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=newk # 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(m in 1:newk) {
mu[m,] <- runif(D,0.49,0.51)
}
pi
mu
for(it in 1:max_it)
{
Sys.sleep(0.5)
# E-step: Computation of the fractional component assignment
# Distributions
for (o in 1:N)
{
p=0
for (l in 1:K)
{
p=p+prod( ((mu[l,]^x[o,])*((1-mu[l,])^(1-x[o,]))) )*pi[l] #
}
for (l in 1:K)
{
z[o,l]=pi[l]*prod( ((mu[l,]^x[o,])*((1-mu[l,])^(1-x[o,]))) ) / p
}
}
#Log likelihood
ll <-matrix(0,nrow =1000,ncol = K)
llik[it] <-0
for(o in 1:N)
{
for (l in 1:K)
{
ll[o,l] <- pi[l]*prod( ((mu[l,]^x[o,])*((1-mu[l,])^(1-x[o,]))))
```

```
}
llik[it]<- sum(log(rowSums(ll)))
}
cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
flush.console()
# Stop if the log 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)
```