732A99 Machine Learning - Block 2 - Lab1

Min-Chun Shih

Contents

1 ENSEMBLE METHODS

Read data

```
sp <- read.csv("spambase.csv")
sp$Spam <- as.factor(sp$Spam)
```

Split data - Split whole data to 2/3 training data and 1/3 test data.

```
set.seed(12345)
n <- nrow(sp)
ids <- sample(1:n, n*2/3)
train <- sp[ids, ]
test <- sp[-ids, ]
```

Adaboost classification trees

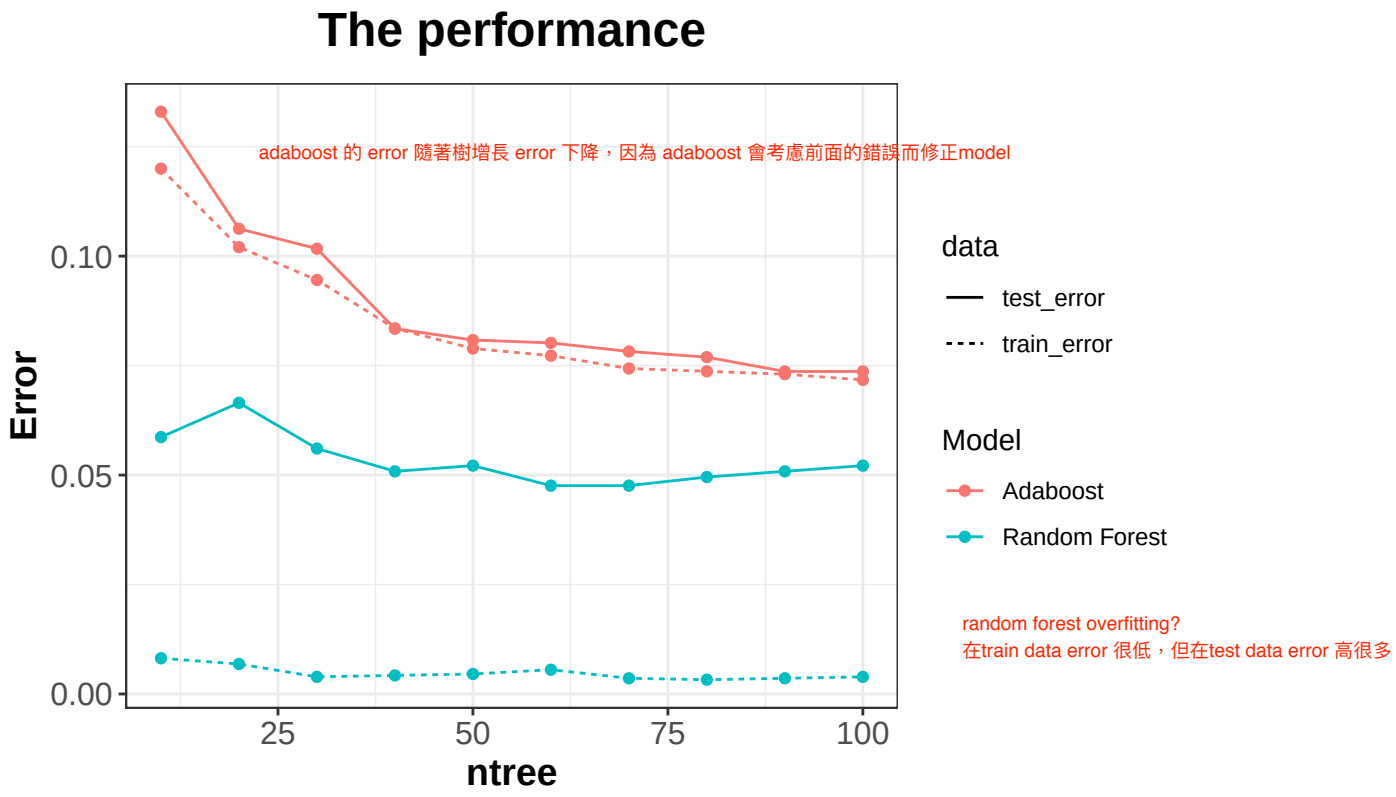
```
library(mboost)
ada.df <- data.frame(ntree = numeric(),
                     train_error = numeric(), test_error = numeric())
trees <- seq(10, 100, 10)
# run trees from 10 to 100
for (tree in trees) {
  # ada.model <- blackboost(Spam ~ ., data = train, family = Binomial(type = "adaboost"), control = boo
  ada.model <- blackboost(Spam ~ ., data = train, family = AdaExp(),
                          control = boost_control(mstop = tree))
  train.pred <- predict(ada.model, newdata = train, type = "class")
  test.pred <- predict(ada.model, newdata = test, type = "class")
  tb <- table(train.pred, train$Spam)
  train_error_rate <- (tb[1, 2] + tb[2, 1])/sum(tb)
  tb <- table(test.pred, test$Spam)
  test_error_rate <- (tb[1, 2] + tb[2, 1])/sum(tb)
  ada.df <- rbind(ada.df, data.frame(ntree = tree,
                                     train_error = train_error_rate,
                                     test_error = test_error_rate))
}
```

Random Forests

```
library(randomForest)
rf.df <- data.frame(ntree = numeric(),
                    train_error = numeric(), test_error = numeric())
for (tree in trees) {
  rf.model <- randomForest(Spam ~ ., data = train, ntree = tree)
  train.pred <- predict(rf.model, newdata = train, type = "class")
  test.pred <- predict(rf.model, newdata = test, type = "class")
  tb <- table(train.pred, train$Spam)
  train_error_rate <- (tb[1, 2] + tb[2, 1])/sum(tb)
  tb <- table(test.pred, test$Spam)
  test_error_rate <- (tb[1, 2] + tb[2, 1])/sum(tb)
  rf.df <- rbind(rf.df, data.frame(ntree = tree,
                                     train_error = train_error_rate,
                                     test_error = test_error_rate))
}
```

Plot the performance for Adaboost and Random Forest

```r
library(ggplot2)
library(tidyr)

ada.df <- ada.df %>%
  gather(data, score, -ntree)
ada.df$Model <- "Adaboost"

rf.df <- rf.df %>%
  gather(data, score, -ntree)
rf.df$Model <- "Random Forest"

df <- rbind(ada.df, rf.df)

ggplot(df, aes(x = ntree, y = score)) +
  geom_line(aes(linetype = data, colour = Model)) +
  geom_point(aes(colour = Model)) +
  theme_bw() +
    labs(title = "The performance", x = "ntree", y = "Error") +
    theme(plot.margin = margin(.5, .7, .5, .3, "cm"), # graph margin
        axis.text = element_text(size = rel(1.1)),  # axis labels size
        axis.title = element_text(size = rel(1.3), face = "bold"), # axis names size
        plot.title = element_text(size = rel(1.6), face = "bold",
                                  hjust = 0.5, margin = unit(c(1, 0, 4, 0), "mm")))
```

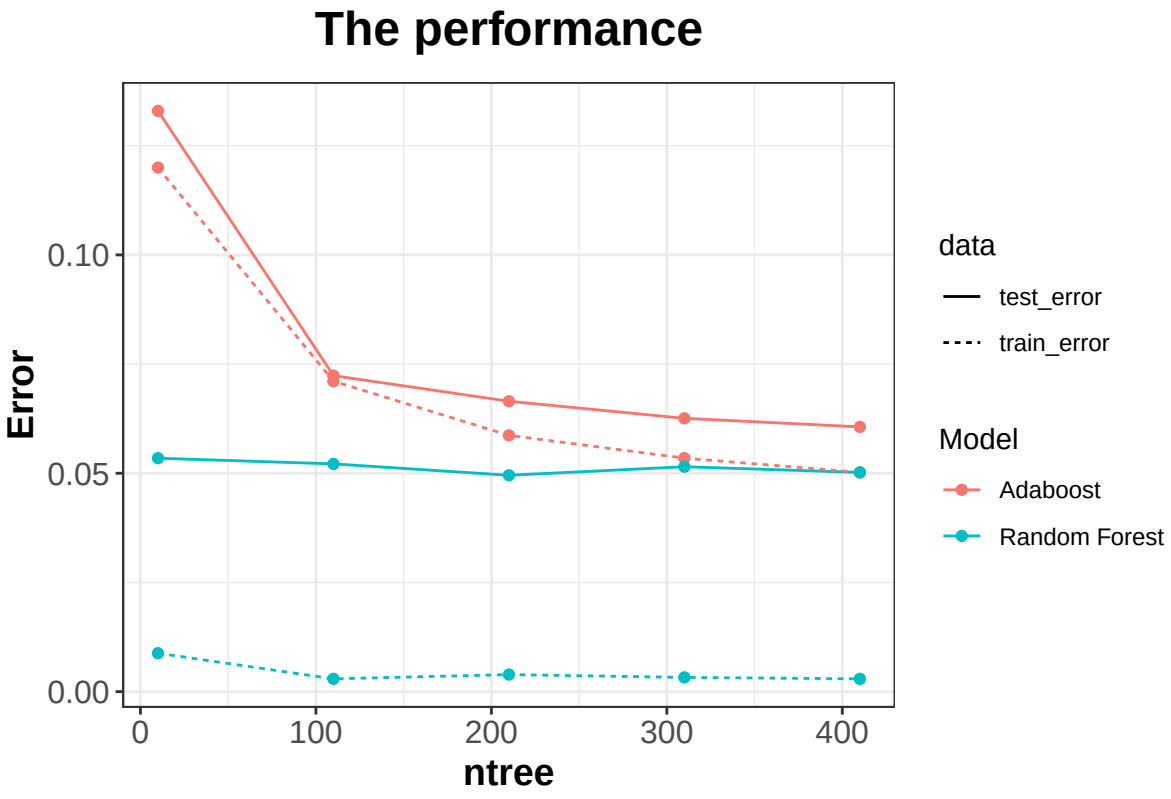


Comment

From the plot above we can see Random Forest has lower test error and train error, Adaboost has higher test error and train error. For Random Forest, test error and train error has a steady gap, and the error of train data is close to 0. Hence, this model might be overfitting. On the other

hand, as the number of trees increases, the error of both testing data and train remain constant. For AdaBoost, the error between testing and train data is not much different when the number of trees grows. However, the error in both testing data and train data decrease significantly when the number of trees from 10 to 40. After the tree is 50, the error goes down slightly.

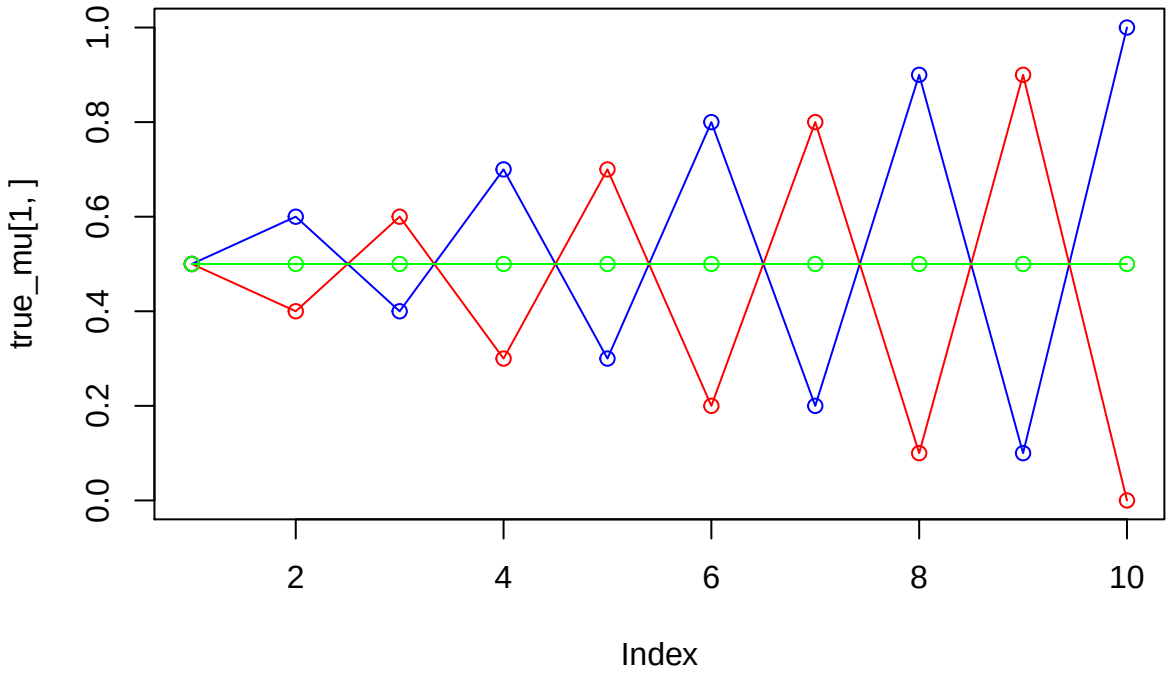
In the graph, Random Forest performs better than AdaBoost, but technically, AdaBoost should have better performance. Hence, I try to increase the number of trees to see if AdaBoost would play better than Random Forest as the trees grow. However, the disadvantage of AdaBoost is it takes too much time. The plot below is the result of 500 trees, we can see, the error for testing and training decreases. We might guess, it will perform better than Random Forest if we could add more trees.



2 MIXTURE MODELS

Set ture data and initialize variables

```
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")
```



Implement EM algorithm

```
EM_algorithm <- function(K) {
  # set.seed(1234567890)
  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)
```

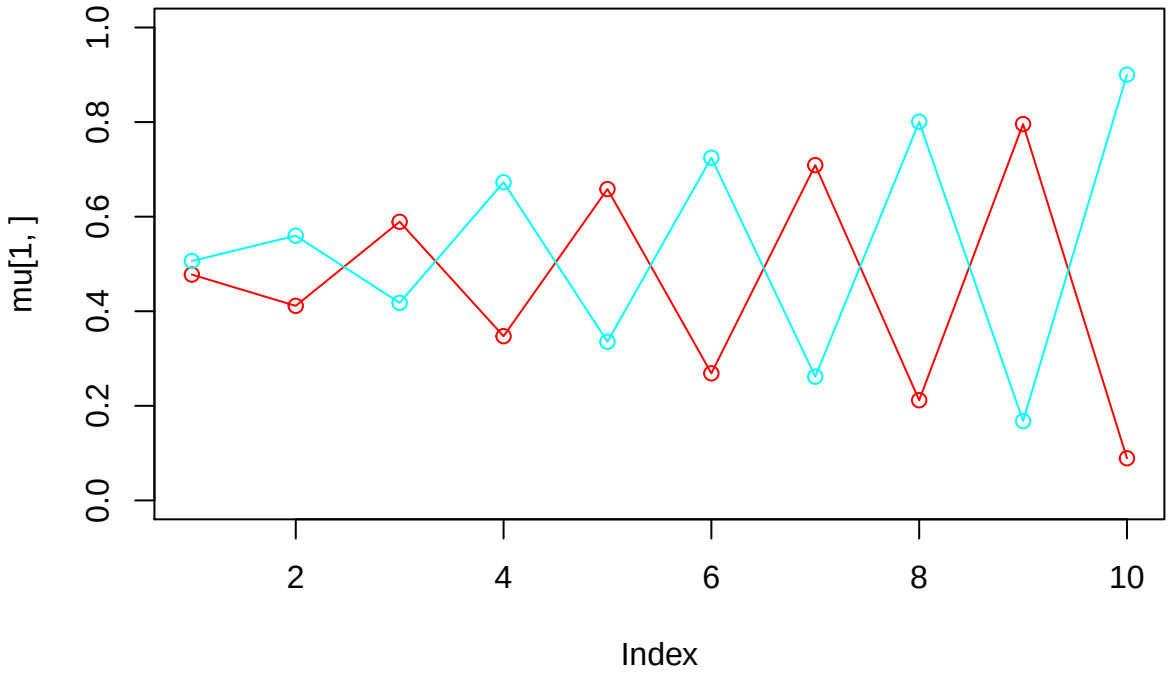
```
  }

  for(it in 1:max_it) {
    Sys.sleep(0.5)

    # --------------------------------
    # E-step: Computation of the fractional component assignments
    p <- z
    for (i in 1:nrow(x)) {
      # Bernoulli pmf
      bernoulli <- matrix(ncol = D, nrow = K)
      for (j in 1:K) {
        bernoulli[j, ] <- (mu[j,]^x[i, ])*((1-mu[j,])^(1-x[i, ]))
      }
      bernoulli <- apply(bernoulli, 1, prod)
      p[i, ] <- bernoulli
    }

    for (i in 1:K) {
      p[, i] <- pi[i]*p[, i]
    }

    z <- p/apply(p, 1, sum)

    # Log likelihood computation.
    logllik <- sum(log(rowSums(p)))
    llik[it] <- logllik
    cat("iteration: ", it, "log likelihood: ", llik[it], "\n")
    flush.console()
    # --------------------------------
    # Stop if the log likelihood has not changed significantly
    if (it != 1) {
      if ((logllik - llik[it-1]) < min_change) {
        cols <- rainbow(K)
        plot(mu[1,], type="o", col=cols[1], ylim=c(0,1))
        for (i in 2:(K)) {
          points(mu[i,], type="o", col=cols[i])
        }
        llik <- data.frame(llik)
        llik$K <- K
        llik$iteration <- 1:nrow(llik)
        llik <- llik[which(llik$llik != 0), ]
        return (llik)
      }
    }

    # M-step: ML parameter estimation from the data and fractional component assignments
    pi <- colSums(z)/N
    mu <- t(z)%*%x/colSums(z)
  }
}
```

Producing the training data

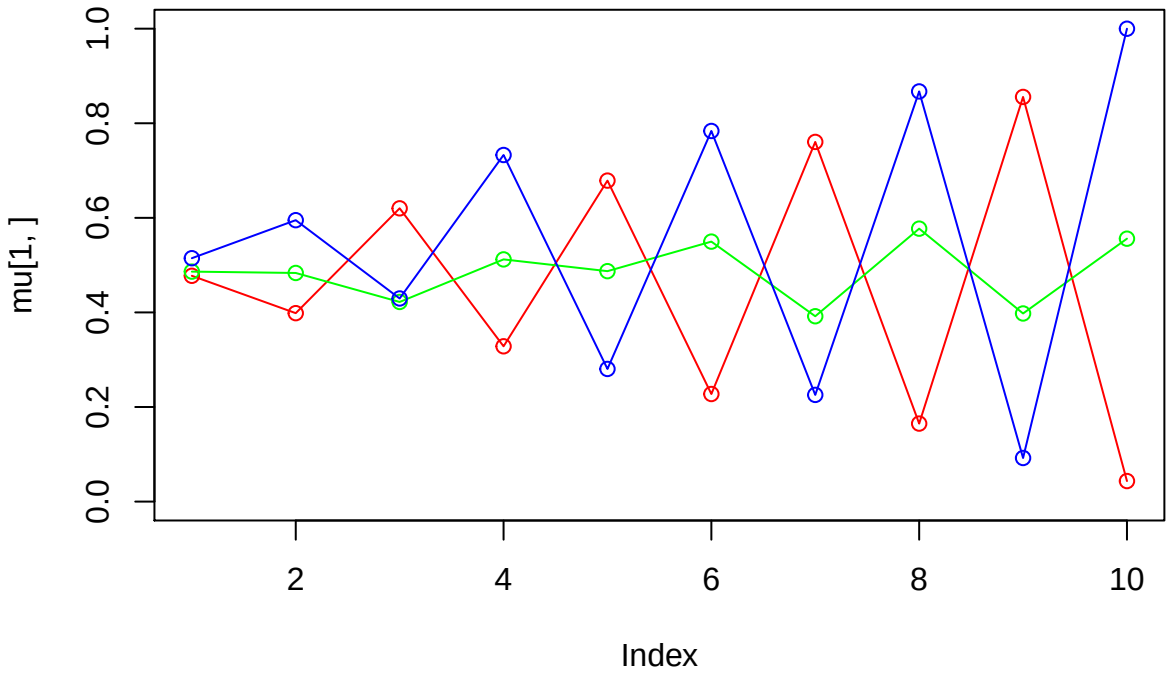
```
# Producing the training data
for(n in 1:N) {
  k <- sample(1:3,1, prob = true_pi)
  for(d in 1:D) {
    x[n, d] <- rbinom(1, 1, true_mu[k, d])
  }
}
```

Run EM algorithm for different K

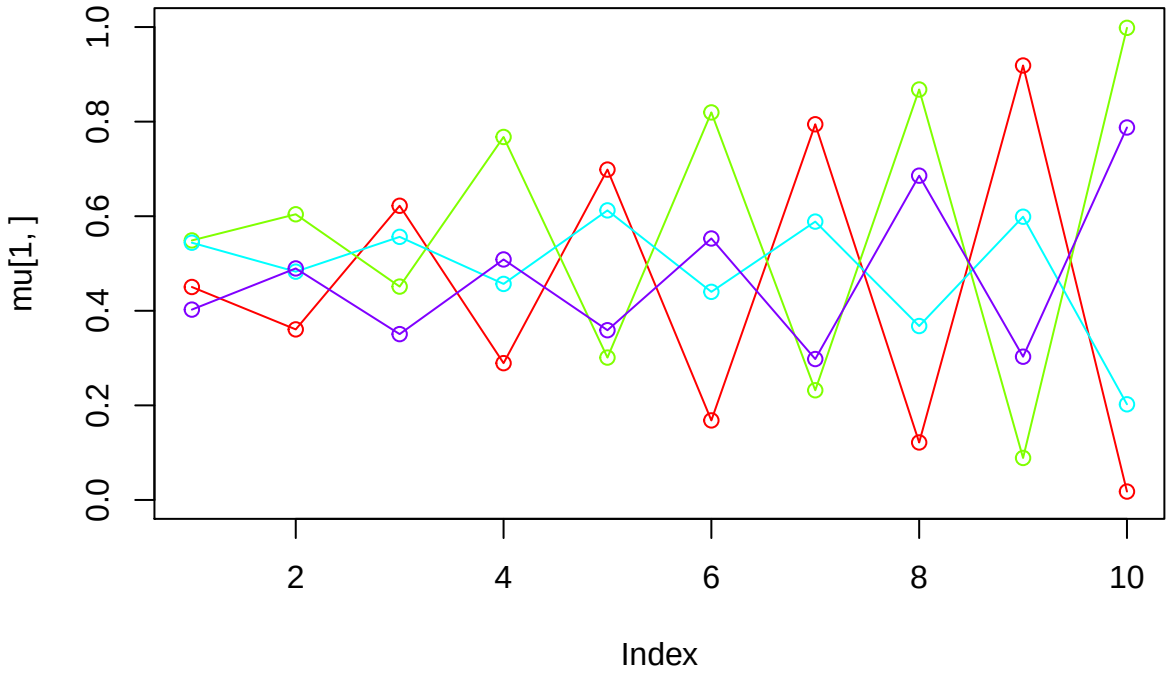
```
K <- 2
llik.df <- EM_algorithm(K)
```




```
K <- 3
llik.df <- rbind(llik.df, EM_algorithm(K))
```



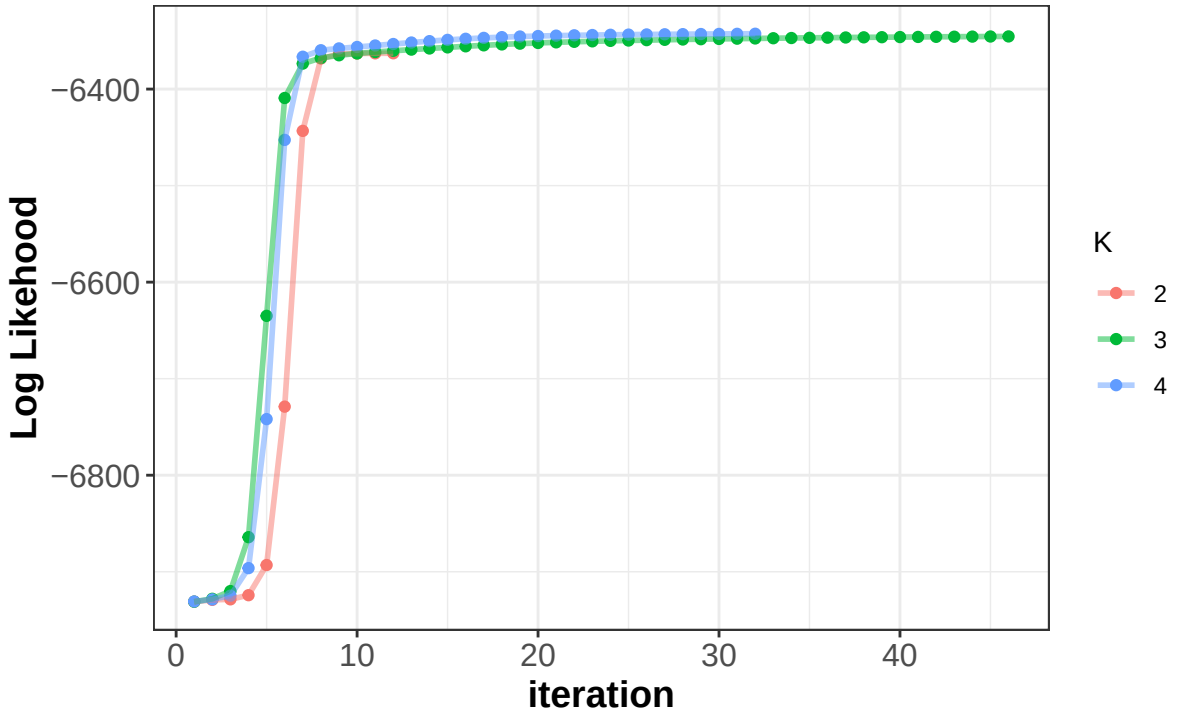

```
K <- 4
llik.df <- rbind(llik.df, EM_algorithm(K))
```



Plot log likelihood for each K

```
library(ggplot2)

llik.df$K <- as.factor(llik.df$K)
ggplot(llik.df, aes(y = llik, x = iteration, colour = K)) +
  geom_point() +
  geom_line(size=1, alpha = 0.5) +
  theme_bw() +
```

```
    labs(title = "", y = "Log Likehood") +
    theme(plot.margin = margin(.5, .7, .5, .3, "cm"), # graph margin
        axis.text = element_text(size = rel(1.1)),  # axis labels size
        axis.title = element_text(size = rel(1.3), face = "bold"), # axis names size
        plot.title = element_text(size = rel(1.6), face = "bold",
                                  hjust = 0.5, margin = unit(c(1, 0, 4, 0), "mm")))
```



Comment

We know the real K is 3. From this Log Likehood iteration plot, all of the likelihood are very close. When K is 4 has the highest log likelihood. If we did not know the real K, we might guess K is 4. However, these three likelihood are too closes, when we set different seed it has distinct result. Which might be better to try many times to predict the K. For the mu plots, we can see when K equals 3, the plot is quite similar as the true one.