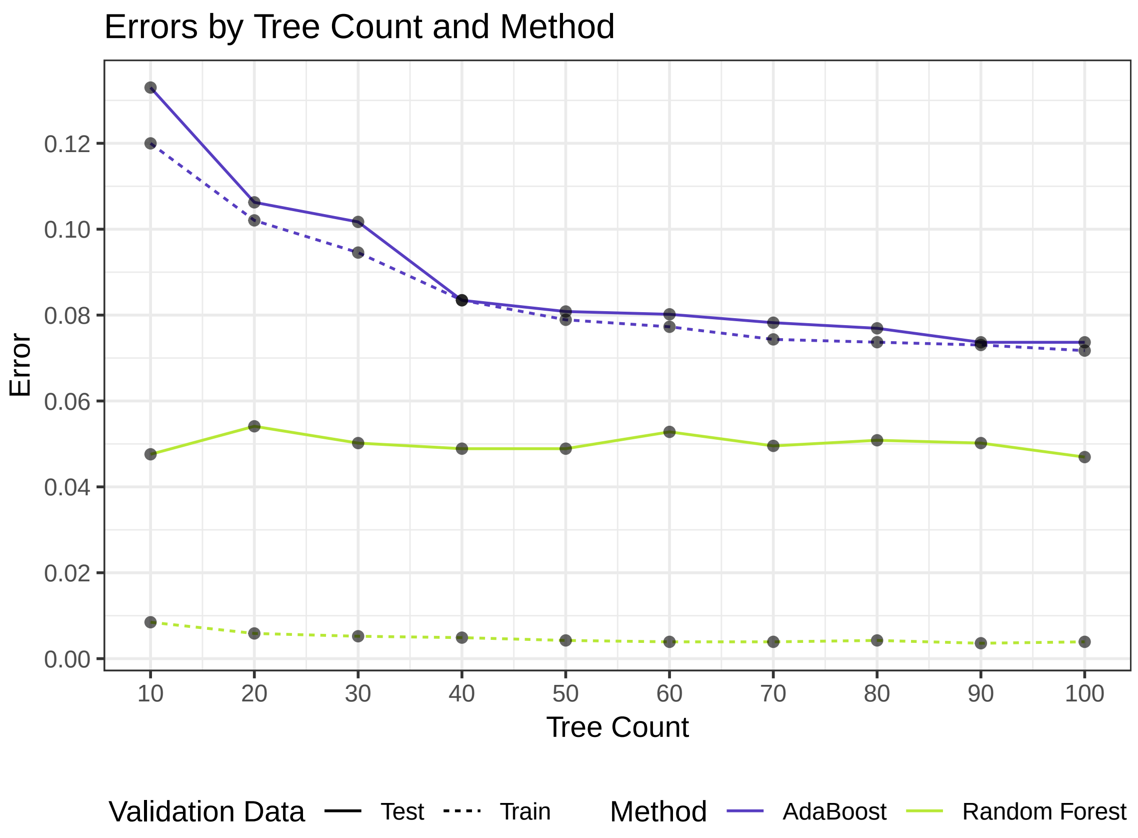
lab1_bl2_mimte666

mimte666 - Mim Kemal Tekin

11/25/2018

Contents

Assignment 1: Ensemble Methods



In this task, we use Random Forest and AdaBoost algorithms with same train and test data, different tree counts. Random Forest (RF) uses bagging method with a difference which is, RF algorithm creates N three, but uses m feature where m < M (M: total feature count). This provides less correlation between predictions of each trees. In other hand, the aim of AdaBoost is different. AdaBoost tries to create a stronger model from weak models. It gives weights to all observations equally and fits the model. After predictions it changes weights in order to make correct to mispredicted observations. By this way, it increases its prediction quality in every step.
We used 10-100 trees to run both of algorithms. As we can see in AdaBoost, while tree training iteration count increases, our misclassification decreases. It started decrease rapidly, but after a while it decrease slowlier than the beginning. We can observe that after a point count of iteration does not effect to model as much as before. In addition, we can say train data and test data misclassification values are really close in all attempt. This means our model is more stable if we compare with Random Forest algorithm.
In Random Forest algorithm, misclassification values are so close to zero when train data is used for validation. But validations with test data have more misclassification. The gap is much more than AdaBoost algorithm but it is still less misclassification error. But the closeness to zero of validations with train data seems like low bias and may occur overfit. The gap between test and train data is big, it may sign of high variance. Additionally as we can see the lines of two data has same path with small changes, so we can say that increasing tree count does not effect to our model quality.

Assignment 2: Mixture Models

In this assignment we will run a EM algorithm simulation. We have real data which is from Bernoulli Distribution and has 1000x10 size. We know that 3 component mixture model can be fit to this multivariate Bernoulli Distribution and probabilities of being from any component (π_k) are equal to $1/3 = 0.33$ and we can see real μ_k distribution at the plot which named "True Mu Values, K=3". We can see μ_1 and μ_2 follow opposite trends for each feature of data, while μ_3 is stable at 0.5. μ_3 shows us, probability of having any feature in component 3 is always 50% and it is not certain to predict whether a observation belongs to component 3. In other words we can say being in component 3 has not good characteristic features. We will simulate EM algorithm with different component counts (K = 2,3,4) and compare this results.

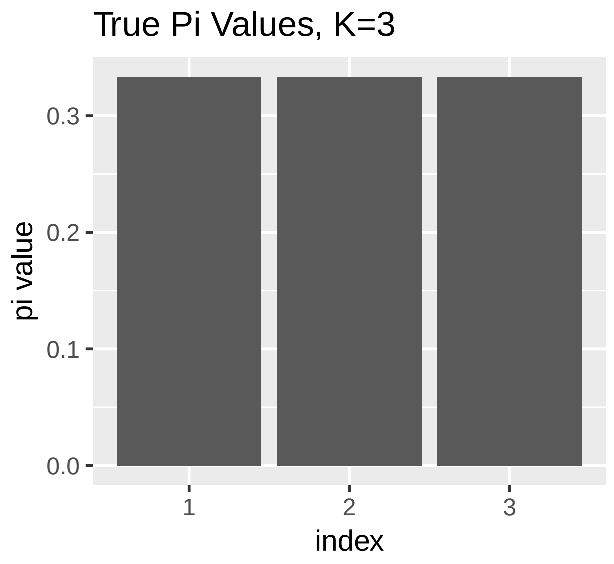
When K=2, pi values are so close to 0.5. Data is divided to 2 class with equal probabilities. This likelihood maximization is completed in 12 iteration and we can see in the beginning likelihood increases exponentially and when it reached 8 iteration increasement goes slow and stop in the end. Because the algorithm stops when likelihood change is less then minimum change (0.1). It means we cannot make this model better after this point. We can see two final mu values are flactuating with same trend as real μ_1 and μ_2 values. As a conclusion, μ_3 component is distributed equally between μ_1 and μ_2, because in real classifications, component 3 does not have characteristic features. It can have any features because all probabilities are equal 0.5 in real mu values.

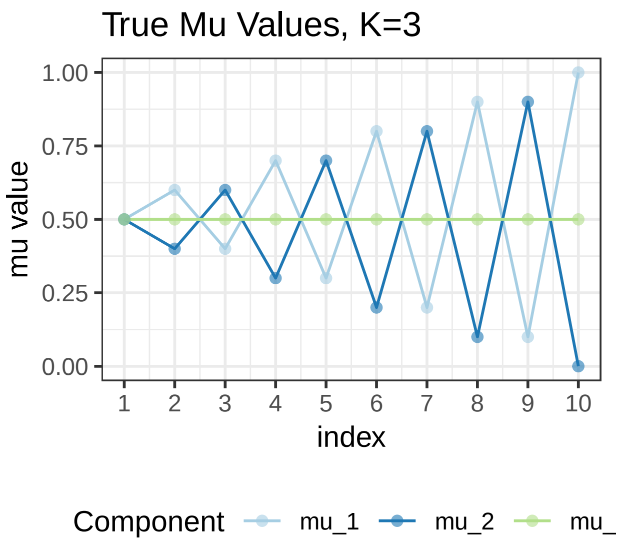
When K=3, pi values are quite close each other except μ_1. Log likelihood is in same trend with before. It increases exponentially, after interation 8 incresement goes slower and when it does not change enough, algorithm stops. In the final mu values plot, the result is quite similar with real data. Especially, it is clear to see that μ_1 and μ_3 follows so similar trend as μ_1 and μ_2 in the real latent parameters.

When K=4, if we examine final mu plot, we can see pair of $\{\mu_1, \mu_2\}$, and pair of $\{\mu_3, \mu_4\}$ follows same movement. However it is still clear to see μ_1 and μ_2 behave same as μ_1 and μ_2 in the real latent parameters. μ_3 and μ_4 have lowest probability in π_k plot.

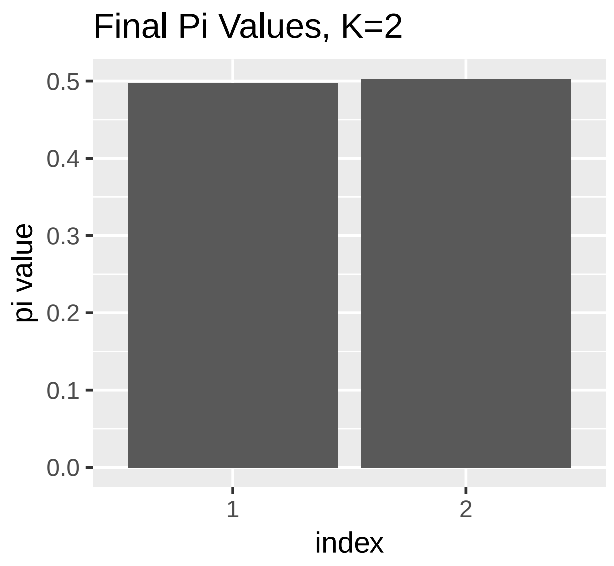
Finally we can say that, if we increase our component count, complexity will increase and some redundant components which have same behaviours can be captured like happened in K=4. Without knowing the real data, it is hard to choose between K=2 and K=3. Because the extra component could either be present in real data or not. It would be necessary to run further analysis.

True Values

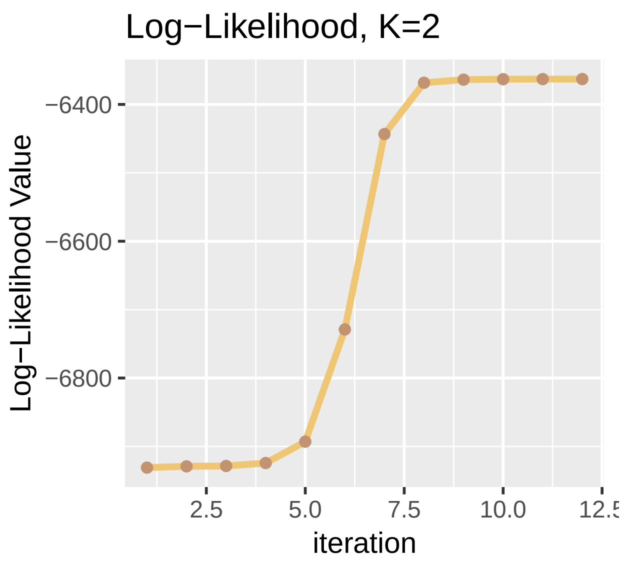




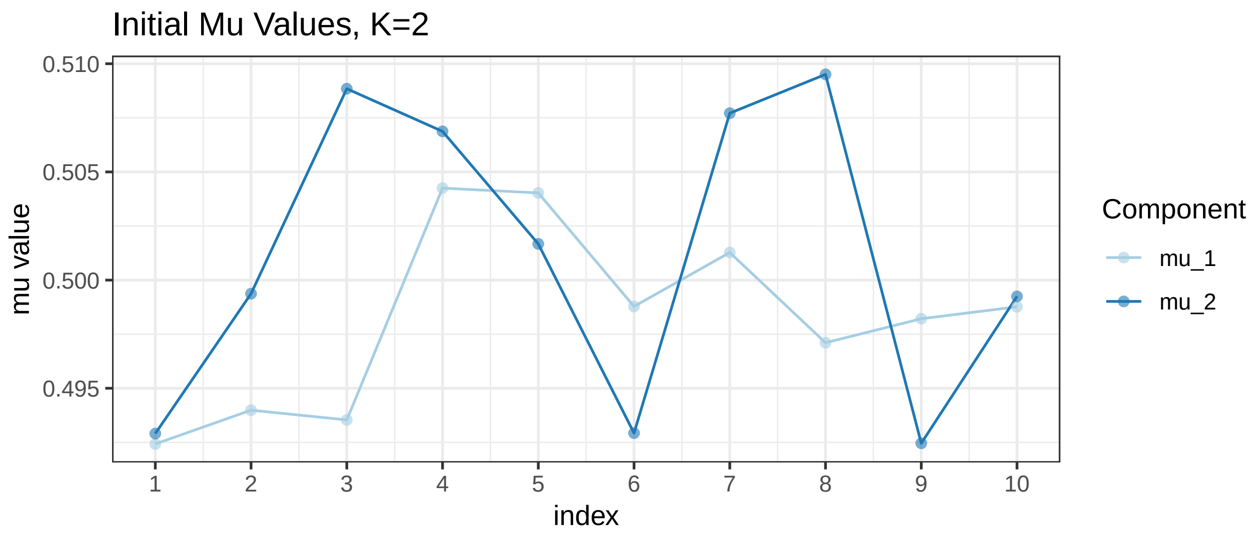
K=2



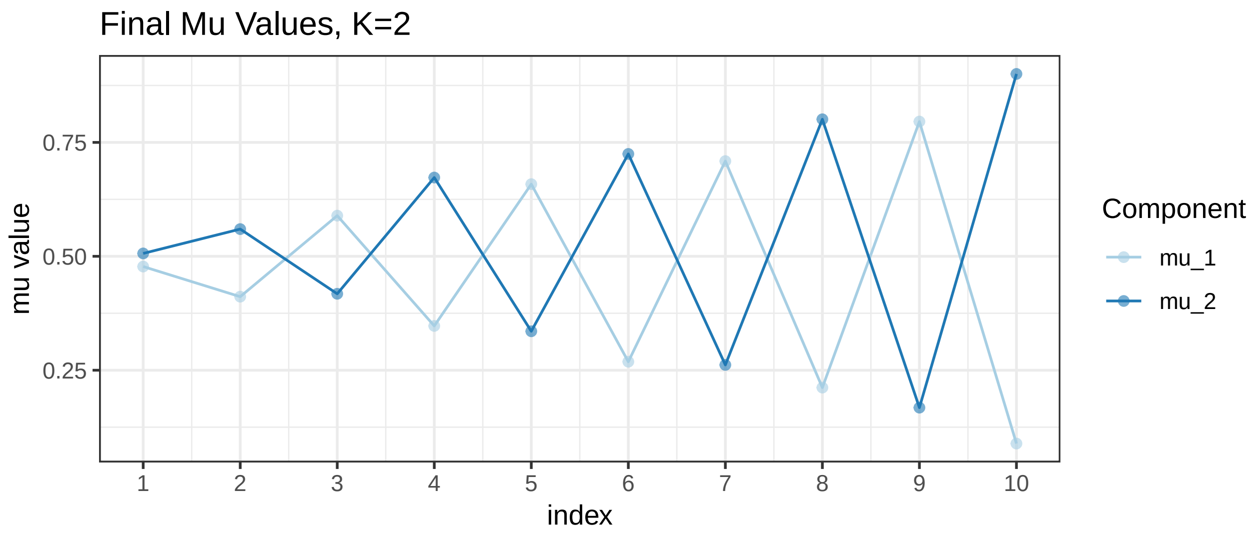
Final Pi Values, K=2
pi value
0.5
0.4
0.3
0.2
0.1
0.0
1
2
index



Log−Likelihood, K=2
Log−Likelihood Value
−6400
−6600
−6800
2.5
5.0
7.5
10.0
12.5
iteration

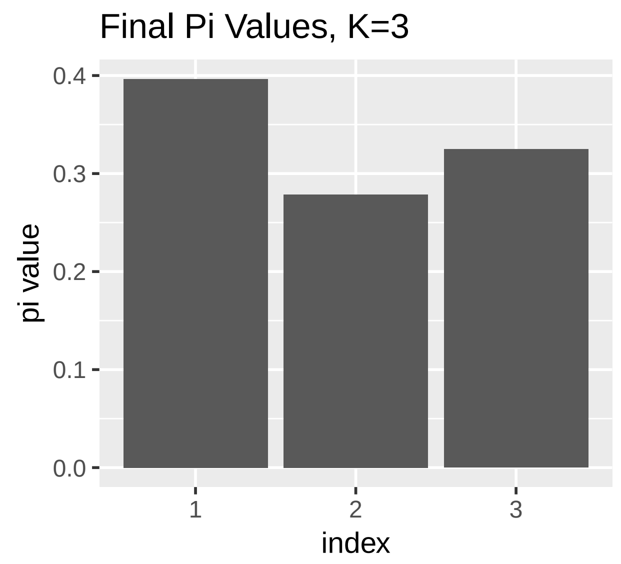


Initial Mu Values, K=2
mu value
0.510
0.505
0.500
0.495
1
2
3
4
5
6
7
8
9
10
index
Component
mu_1
mu_2

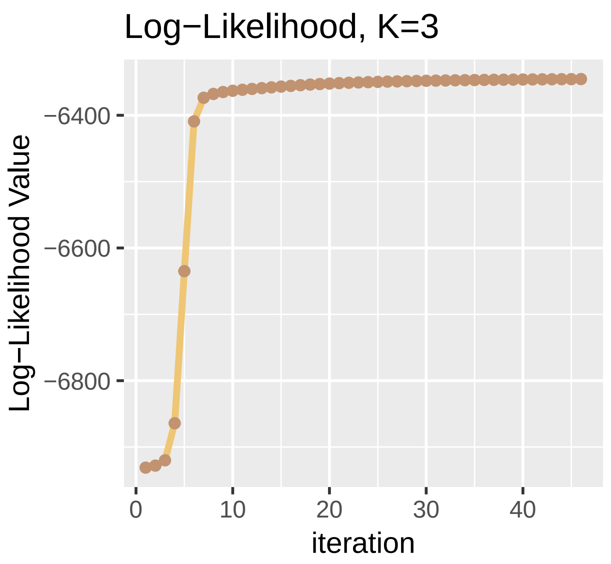


Final Mu Values, K=2
mu value
0.75
0.50
0.25
1
2
3
4
5
6
7
8
9
10
index
Component
mu_1
mu_2

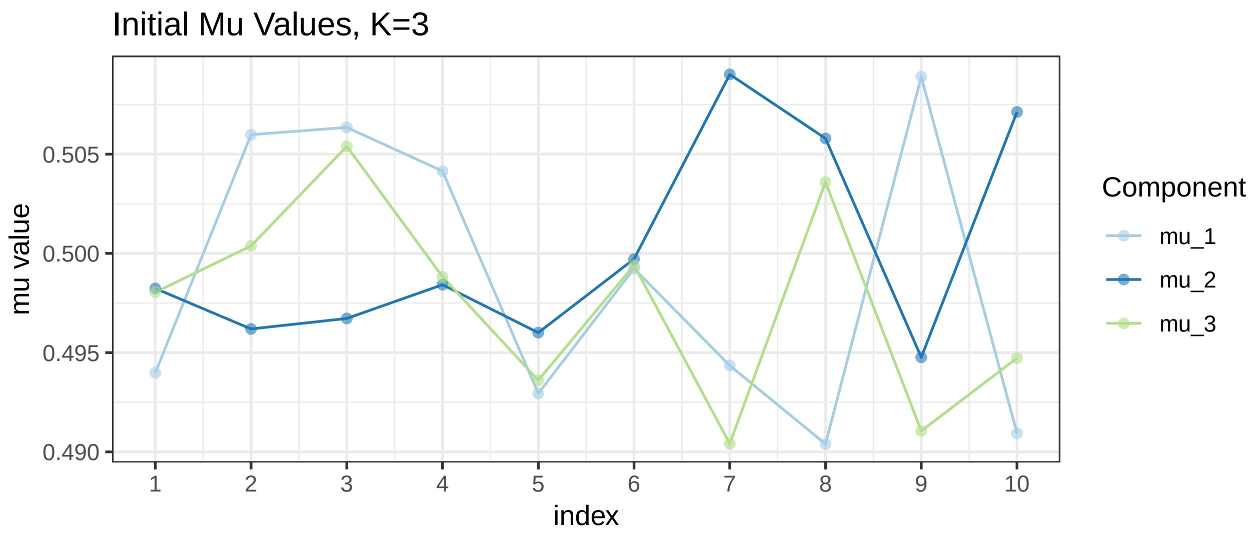
K=3



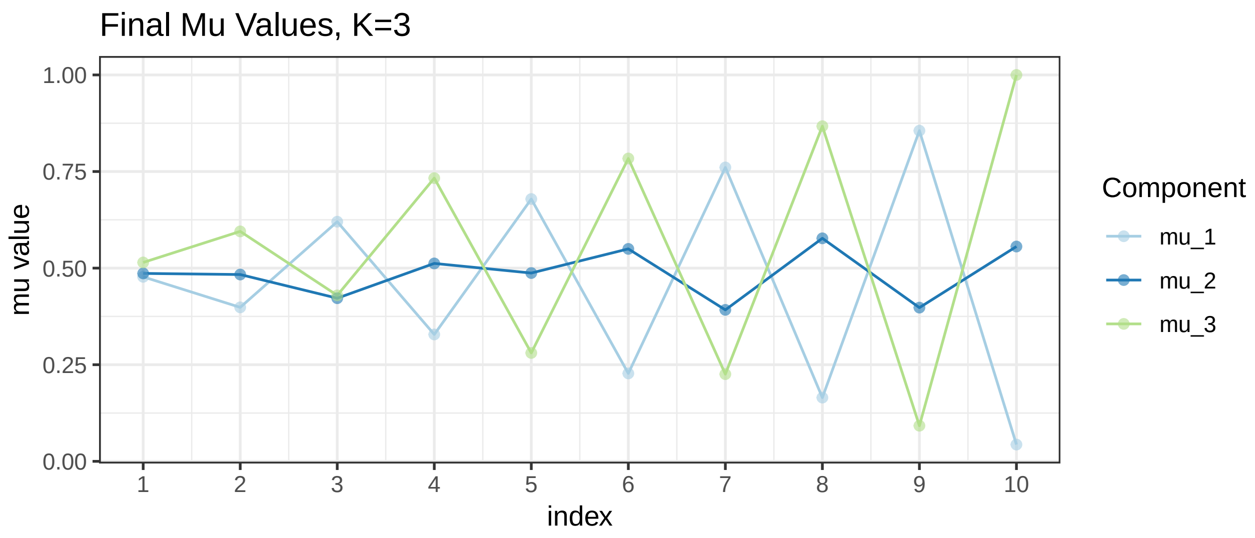
Final Pi Values, K=3
pi value
0.4
0.3
0.2
0.1
0.0
1
2
3
index



Log−Likelihood, K=3
Log−Likelihood Value
−6400
−6600
−6800
0
10
20
30
40
iteration

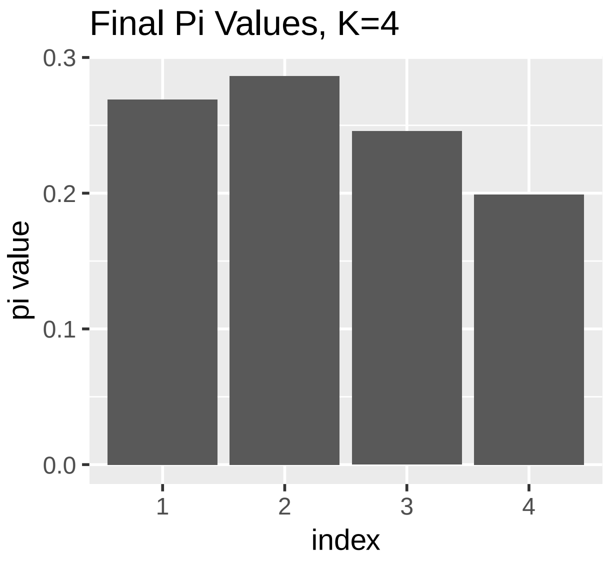


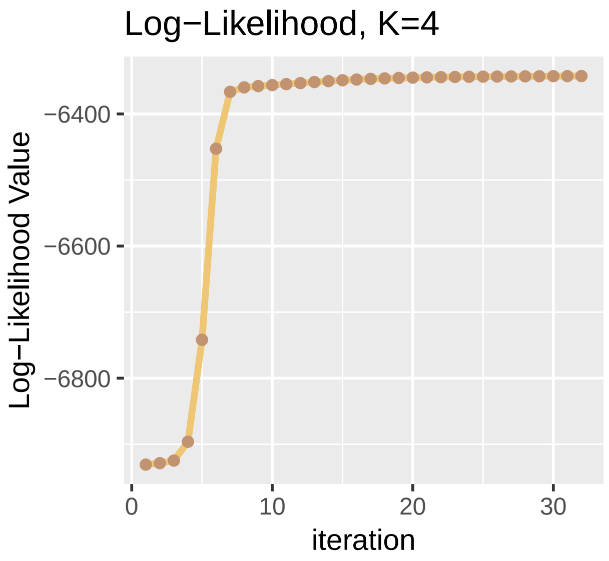
Initial Mu Values, K=3
mu value
0.505
0.500
0.495
0.490
index
Component
mu_1
mu_2
mu_3

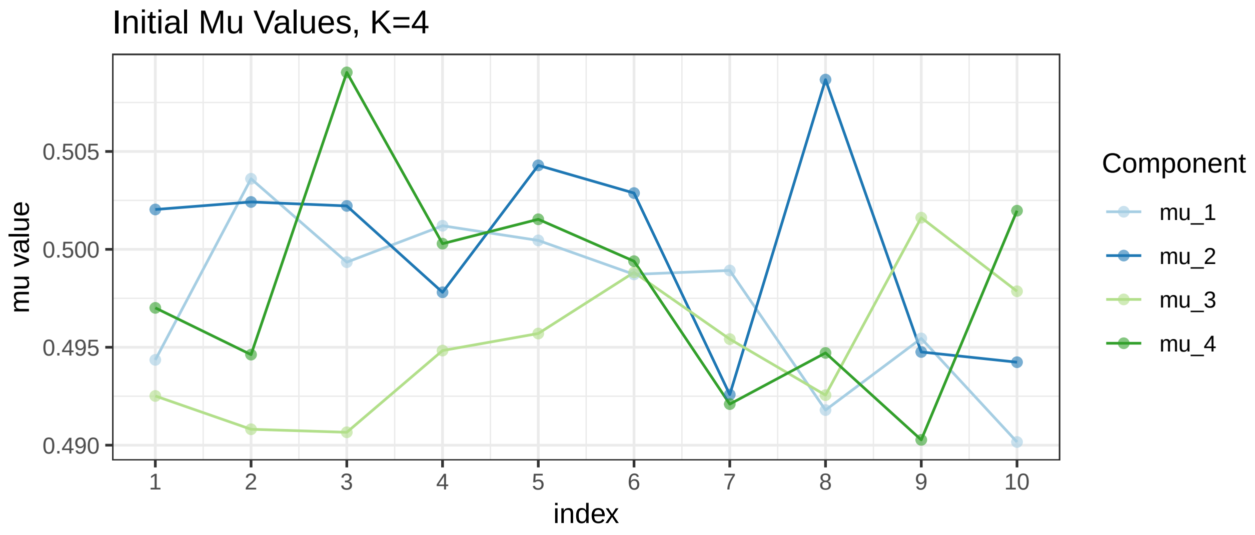


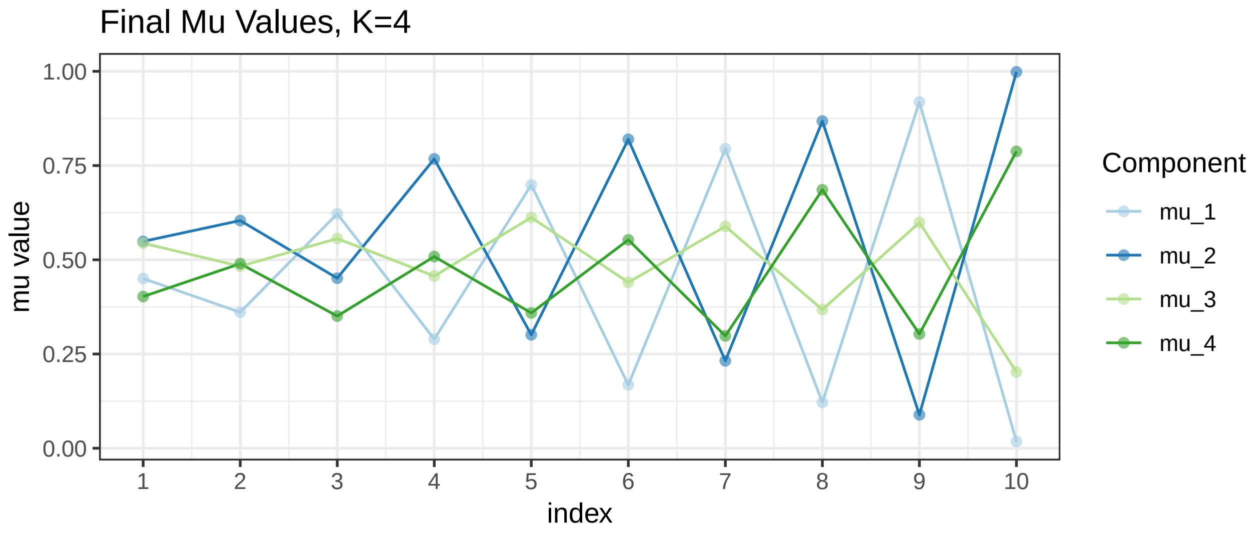
Final Mu Values, K=3
mu value
1.00
0.75
0.50
0.25
0.00
index
Component
mu_1
mu_2
mu_3

K=4


Final Pi Values, K=4
pi value
0.0
0.1
0.2
0.3
1
2
3
4
index


Log−Likelihood, K=4
Log−Likelihood Value
−6400
−6600
−6800
0
10
20
30
iteration


Initial Mu Values, K=4
mu value
0.490
0.495
0.500
0.505
1
2
3
4
5
6
7
8
9
10
index
Component
mu_1
mu_2
mu_3
mu_4


Final Mu Values, K=4
mu value
0.00
0.25
0.50
0.75
1.00
1
2
3
4
5
6
7
8
9
10
index
Component
mu_1
mu_2
mu_3
mu_4

Appendix

```r
knitr::opts_chunk$set(echo = FALSE, fig.width=5, fig.height=3.5,
                      fig.align = "center", message = F, warning = F, error = F)
library(kableExtra)
library(ggplot2)
library(gridExtra)
library(mboost)
library(randomForest)
calculate_rate = function(conf_matrix){
  conf_matrix = as.matrix(conf_matrix)
  return(1 - sum(diag(conf_matrix[1:2,1:2]))/sum(conf_matrix[1:2,1:2]))
}

conf_matrix = function(real_data, predicted_data){
  ct = table(real_data, predicted_data)
  df = data.frame(c(as.vector(ct[,1]), sum(ct[,1])),
                  c(as.vector(ct[,2]), sum(ct[,2])),
                  c(sum(ct[1,]), sum(ct[2,]), sum(ct)))
  rownames(df) = c("Non-Spam", "Spam", "Frequencies")
  colnames(df) = c("Non-Spam", "Spam", "Frequencies")
  return(df)
}

kable_cm = function(cm, capture){
  return(kable(cm, "latex", booktabs = T, align = "c",
            caption = capture) %>%
            row_spec(2, hline_after = T) %>%
            column_spec(c(1,3), border_right = T) %>%
            kable_styling(latex_options = "hold_position"))
}

create_tuple = function(tree_count, error, method, validation){
  return(data.frame(tree_count = tree_count,
                    error = error,
                    method = method,
                    validation = validation, stringsAsFactors = FALSE))
}

# import data
df_spam = read.csv2("../dataset/spambase.csv")
df_spam$Spam <- as.factor(df_spam$Spam)
# split data
set.seed(12345)
n = dim(df_spam)[1]
id = sample(1:n, floor(n*(2/3)))
train = df_spam[id, ]
test = df_spam[-id, ]

# declare container variables
rf_conf_matrixes = list()
plot_df = data.frame(tree_count = numeric(),
                     error = double(),
                     method = character(),
                     validation = character(), stringsAsFactors = FALSE)

for(tree_count in 1:10*10){
  # cat(tree_count)
  ##### RANDOM FOREST #####
  # fit random forest
  rf_i = randomForest(formula = Spam ~., data = train, ntree=tree_count)

  # predict
  pred_train = predict(rf_i, train)
  pred_test = predict(rf_i, test)

  # conf matrix
  cm_train = conf_matrix(train$Spam, pred_train)
  cm_test = conf_matrix(test$Spam, pred_test)
  cms = list(train = cm_train,
             test = cm_test)

  # calculate rates
  rate_train = calculate_rate(cm_train)
  rate_test = calculate_rate(cm_test)

  # store results
  rf_conf_matrixes[[paste("rf_", tree_count, sep="")]] = cms
  plot_df = rbind(plot_df, create_tuple(tree_count, rate_train,
                                        "rf", "Train"))
  plot_df = rbind(plot_df, create_tuple(tree_count, rate_test,
                                        "rf", validation = "Test"))

  ##### ADABOOST #####
  # fit adaboost
  ada_i = blackboost(formula = Spam~., data = train,
                     family = AdaExp(),
                     control = boost_control(mstop = tree_count))

  # predict
  pred_train = predict(ada_i, newdata = train, type = "class")
  pred_test = predict(ada_i, newdata = test, type = "class")

  # conf matrix
  cm_train = conf_matrix(train$Spam, pred_train)
  cm_test = conf_matrix(test$Spam, pred_test)
  cms = list(train = cm_train,
             test = cm_test)

  # calculate rates
  rate_train = calculate_rate(cm_train)
  rate_test = calculate_rate(cm_test)

  # store results
  rf_conf_matrixes[[paste("ab_", tree_count, sep="")]] = cms
  plot_df = rbind(plot_df, create_tuple(tree_count, rate_train,
```

```r
                                        "ab", "Train"))
  plot_df = rbind(plot_df, create_tuple(tree_count, rate_test,
                                        "ab", validation = "Test"))
}

plot = ggplot(plot_df) +
  geom_line(aes(x = tree_count, y = error, linetype = validation,
                color = method)) +
  geom_point(aes(x = tree_count, y = error), alpha=0.6) +
  labs(title = "Errors by Tree Count and Method", x = "Tree Count", y = "Error",
       linetype = "Validation Data", color = "Method") +
  scale_color_manual(labels = c("AdaBoost", "Random Forest"),
                     values = c("#573DC1","#B7E836")) +
  scale_x_continuous(breaks = seq(0, 100, by = 10)) +
  scale_y_continuous(breaks = round(seq(min(plot_df$error), max(plot_df$error),
                                        by = 0.02), 2)) +
  theme_bw() +
  theme(legend.position = "bottom")

plot


######################
### PLOT FUNCTIONS ###
######################

plot_mu = function(mu, title, legend_pos="right"){
  # mu should be a matrix which is KxD size
  plot_df = NULL
  # create df for plot
  for(k in 1:dim(mu)[1]){
    plot_df = rbind(plot_df, data.frame(index = c(1:dim(mu)[2]),
                                        value = mu[k, ],
                                        component =
                                          rep(paste("mu_", k, sep = ""))))
  }

  plot = ggplot(plot_df, aes(x = index, y = value, color = component)) +
    geom_line() +
    geom_point(alpha = 0.6) +
    labs(title = title, x = "index", y = "mu value",
         color = "Component") +
    theme_bw() +
    theme(legend.position = legend_pos) +
    scale_color_brewer(palette="Paired") +
    scale_x_continuous(breaks = seq(1,10,1))

  return(plot)
}

plot_pi = function(pi, title){
  plot_df = data.frame(index = factor(c(1:length(pi))),
                       pi = pi)
  plot = ggplot(plot_df, aes(x=index, y=pi)) +
    geom_bar(stat = "identity") +
    labs(title = title, x = "index", y = "pi value")

  return(plot)
}

plot_llik = function(llik, title){
  plot_df = data.frame(iteration = c(1:length(llik)),
                       llik = llik)
  plot = ggplot(plot_df, aes(x = iteration, y= llik)) +
              geom_line(color = "#eec674", size=1.2) +
              geom_point(color = "#c19371", size = 1.5) +
              labs(title = title, x = "iteration", y = "Log-Likelihood Value")
  return(plot)
}

############################
### REAL DATA GENERATION ###
############################

set.seed(1234567890)
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 = plot_mu(true_mu, "True Mu Values, K=3", legend_pos = "bottom")
plot_true_pi = plot_pi(true_pi, "True Pi Values, K=3")
############################
### SIM DATA GENERATION  ###
############################

# 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])
  }
}

# number of guessed components
K=3
# max number of EM iterations
max_it <- 100
# min change in log likelihood between two consecutive EM iterations
min_change <- 0.1

###################
```

```r
### EM FUNCTION ###
###################

em_bernoulli = function(x, K=3, max_it=100, min_change=0.1){
  N = dim(x)[1]
  D = dim(x)[2]
  ### declare random variable
  z <- matrix(nrow=N, ncol=K) # fractional component assignments
  pi <- vector(length = K) # mixing coefficients
  mu <- matrix(nrow=K, ncol=D) # conditional distributions

  llik <- c() # log likelihood of the EM iterations
  # Random initialization of the latent parameters
  pi <- runif(K,0.49,0.51)
  pi <- pi / sum(pi)
  for(k in 1:K) {
    mu[k,] <- runif(D,0.49,0.51)
  }
  # create initialized mu plot
  plot_init_mu = plot_mu(mu, paste("Initial Mu Values, K=", K, sep=""))
  ### main loop
  for(it in 1:max_it) {
    # E-step: Computation of the fractional component assignments
    ### calculate Z matrix
    # P(X|mu) = \prod{i} mu_ki^x_i * (1 - mu_ki)^(1-x_i)
    # We cannot calculate powers with matrix multiplication
    # We can take log of whole probability formula, and to reach real values
    # we will take exp of the result.
    # exp(log( P(X|mu) )) = exp(\sum{i} [x_i*log(mu_ki) + (1-x_i)*log(1-mu_ki)])
    p = exp(x %*%t (log(mu)) + (1-x) %*% t(log(1-mu)))
    # there is no way to calculate row wised in R. We have to create N redundant
    # rows to multiply matrixes element wised.
    pi_multiplier = matrix(rep(pi, N), nrow=N, byrow = T)
    # weighted probability
    weighted_p = pi_multiplier * p
    # latent variable calculation
    z_probs = weighted_p / rowSums(weighted_p)

    ### Log likelihood computation.
    llik[it] = sum(log(rowSums(weighted_p)))

    # Stop if the log likelihood has not changed significantly
    # value is a parameter of the function
    if(it!=1){
      if(llik[it]-llik[it-1] <= min_change)
        break
    }
    # M-step: ML parameter estimation
    # from the data and fractional component assignments
    # calculate new latent parameters
    ### calculate pi
    pi = colSums(z_probs) / N
    ### calculate mu
    mu = t(z_probs) %*% x / colSums(z_probs)
  }
  # create final plots
  plot_final_mu = plot_mu(mu, paste("Final Mu Values, K=", K, sep=""))
  plot_final_pi = plot_pi(pi, paste("Final Pi Values, K=", K, sep=""))
  plot_llik = plot_llik(llik, paste("Log-Likelihood, K=", K, sep = ""))
  result = list(llik = llik,
                plots = list(
                  init_mu = plot_init_mu,
                  final_mu = plot_final_mu,
                  final_pi = plot_final_pi,
                  llik = plot_llik)
                )
  return(result)
}

grid.arrange(grobs=list(plot_true_pi, plot_true_mu), ncol=2)

em_2 = em_bernoulli(x, K=2)
grid.arrange(grobs=list(em_2$plots$final_pi, em_2$plots$llik), ncol=2)
em_2$plots$init_mu
em_2$plots$final_mu
em_3 = em_bernoulli(x, K=3)
grid.arrange(grobs=list(em_3$plots$final_pi, em_3$plots$llik), ncol=2)
em_3$plots$init_mu
em_3$plots$final_mu
em_4 = em_bernoulli(x, K=4)
grid.arrange(grobs=list(em_4$plots$final_pi, em_4$plots$llik), ncol=2)
em_4$plots$init_mu
em_4$plots$final_mu
```