ML Block 2 Lab 1

Group A14

12/3/2018

Contents

Assignment 1: Ensemble Methods

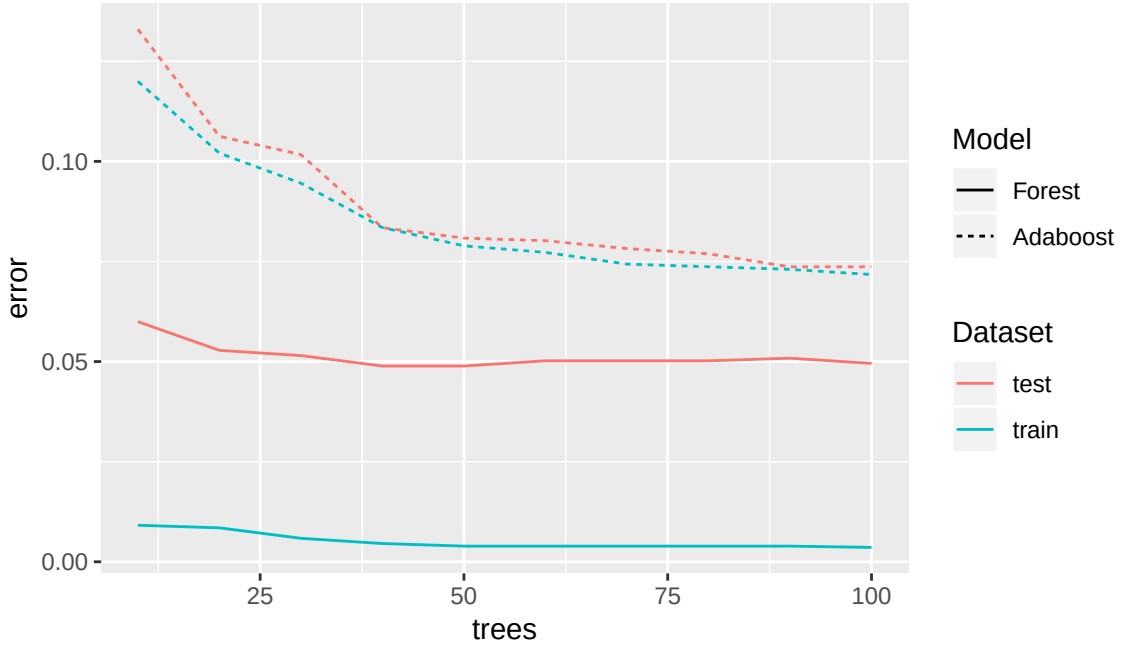

Figure 1: Performances of the two models (differentiated by linetype) on the train and test datasets (differentiated by color) versus the number of trees used during the estimation process

Analysis

In this task, we perform two different ensemble methods for the same data, called Adaboost classification and random forest. We estimate the error rates for each method for both training and test data and finally we plot the number of trees used in each model vs the error rates for the models. Although both methods

try to converge weak learners to strong learners, they target different things and this is the reason why the results bellow for each model are different. So every time that an observation misclassified, Adaboost change it weight, as a result to make predictions based on the weight. On the contrary, random forests combine many difference tries, which may not fit the data well, but with this way the predictions will be much more accurate.

From the plot, one can easily observe that regardless of the number of trees, Adaboost methods has always higher error rate both for training and test data. For Adaboost, the difference between training and test data is not that important. More specific, Adaboost's error rates start at around 0.13 for the test and 0.12 for the train and it is reducing exponentially as the number of trees increases. It can be also said that for more than 70 trees, Adaboost's error seems to be stabilized at around 0.07. On the contrary, for the random forest method, the test error rate starts at approximately 0.06 for 10 trees, decreases slightly until we use 30 trees and finally it fluctuates with a very low range between 0.05 and 0.055. The random forest's error rate for the training data, starts close to 0.01, slightly decreases until we use 30 trees and after that stabilized close to 0.005. In conclusion, the performance of the random forest methods is always more efficient than the Adaboost regardless of the number of trees we fit the model. Moreover , we can choose to fit the model for more than 30 trees using random forest and the difference in the error rate between the two methods would probably be huge. Therefore, the best model should be a random forest using more than 30 trees.

Assignment 2: Mixture Models

Analysis

In this assignment we will run a EM algorithm simulation. The simulated data has been originated from Bernoulli Distributions and has 1000×10 size. We know that a 3 component mixture model can be fit to this multivariate Bernoulli Distribution and the probabilities of each point of being from any component (π_k) are all equal to $1/3 = 0.33$. We can see the real μ parametrs of the distributions in the plot in figure 4A. We can observe how μ_1 and μ_2 follow opposite trends for each feature of the data and have the 10^{th} variable with probability equal to either one or zero, resulting in a deterministic event, while μ_3 is stable at 0.5. In other words the probability of having any feature in component 3 is always 50% and may be unclear to predict whether a observation belongs to it since no characteristic features emerge. We will simulate EM algorithm with different component counts (K = 2,3,4) and compare this results.

When $K = 2$, $\hat{\pi}_{.,2}$ values are close to 0.5 (figure 2). Data is divided into 2 class with equal probabilities. Its likelihood maximization is completed within 12 iteration, as displayed in the graph in figure 3. Even though the convergence has been reached way before the other two models, the final value of the log-likelyhood doesn't seem to change between different choiches of K. It also shows how in the beginning the likelihood, starting at -69500 for $K = 2$, increases exponentially until iteration 8 and then remains stable around -63600. The algorithm stops when the change in likelihood is less then a pre-defined threshold, equal to 0.1: after reaching this point the model will not improve significantly, having the log-likelyhood function reached a maximum, at least locally.
We can see two final $\hat{\mu}_{.,2}$ values are following the same trend as the real μ_1 and μ_2 (figure 4). Finally component μ_3 is distributed equally between $\hat{\mu}_{1,2}$ and $\hat{\mu}_{2,2}$ (figure 5) since it does not have peculiar feature: they are all equally likely, resulting in the maximum variance possible for a Bernoulli distribution ($Var(x) = \mu \cdot (1 - \mu) = 0.5 \cdot 0.5 = 0.25$). Moreover, being feature number 10 always present/absent for μ_1 and μ_2, component μ_3 has probably been assigned to one of the remaing two according to the outcome of last variable (figure 4).

When $K = 3$, $\hat{\pi}_{.,3}$ values fluctuate around 0.33. Log-likelihood follows the same trend as before. It increases exponentially, after interation 8 the incresement is almost irrelevant and when the change does not exceed the threshold, the algorithm stops. In the final $\hat{\mu}_{.,3}$ values plot, the result resamble closely the hidden phenomena behind the data. In particular $\hat{\mu}_{.,1}$ and $\hat{\mu}_{.,2}$ appear to follow trends similar to the ones of μ_1 and μ_2 of the real latent parameters.

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 our component count, with the increase of complexity some redundant components with similar behaviours can be estimated, as in the case of $K = 4$. Without knowing the real distribution of the data, it is hard to choose between $K = 2$ and $K = 3$: the extra component could either be present in real data or not. It would be necessary to run further analysis.

Plots

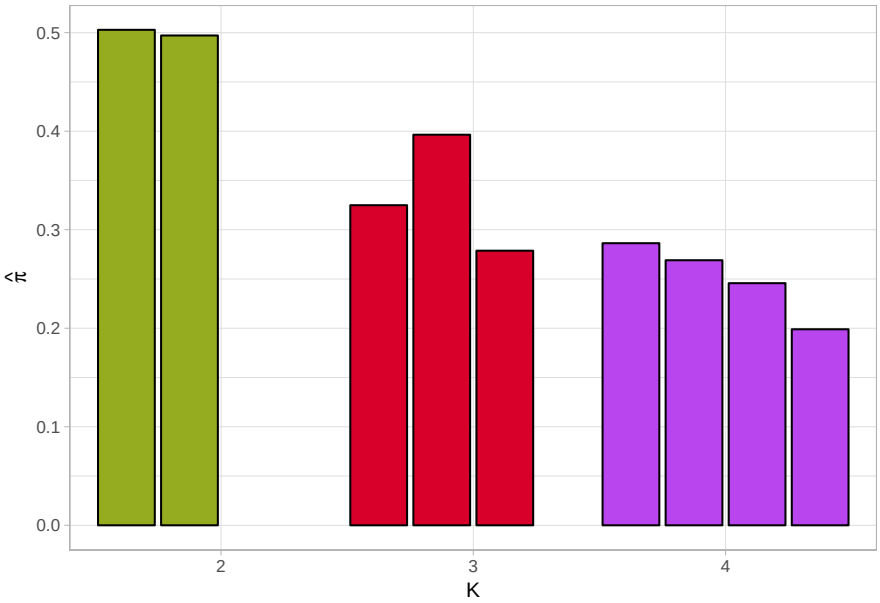

Figure 2: Estimated mixing coefficients for all the choices of K.

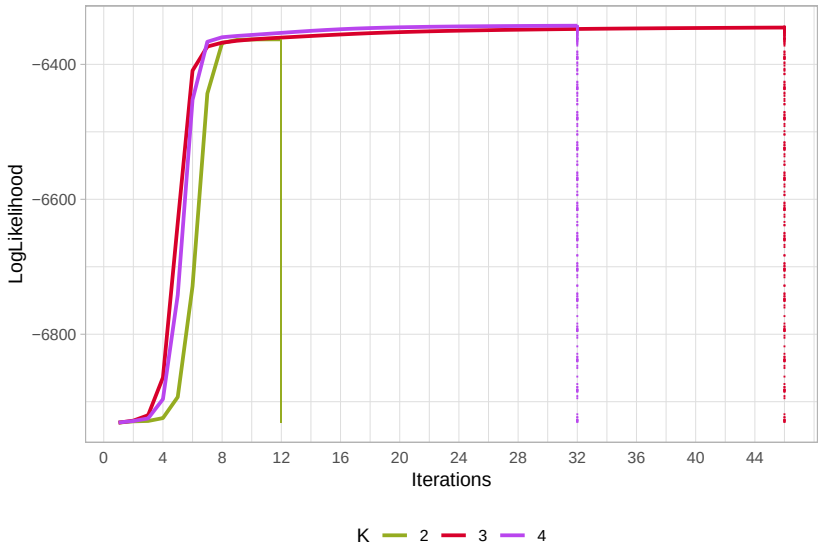

Figure 3: Computed Log-Likelyhood for all the choices of K.

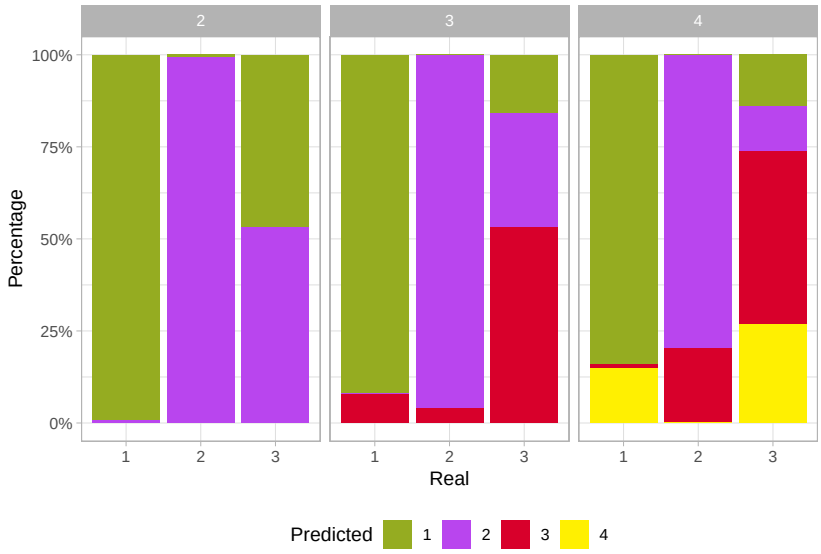

Figure 4: Stacked barplot of real vs predicted components, with K as faceting variable.

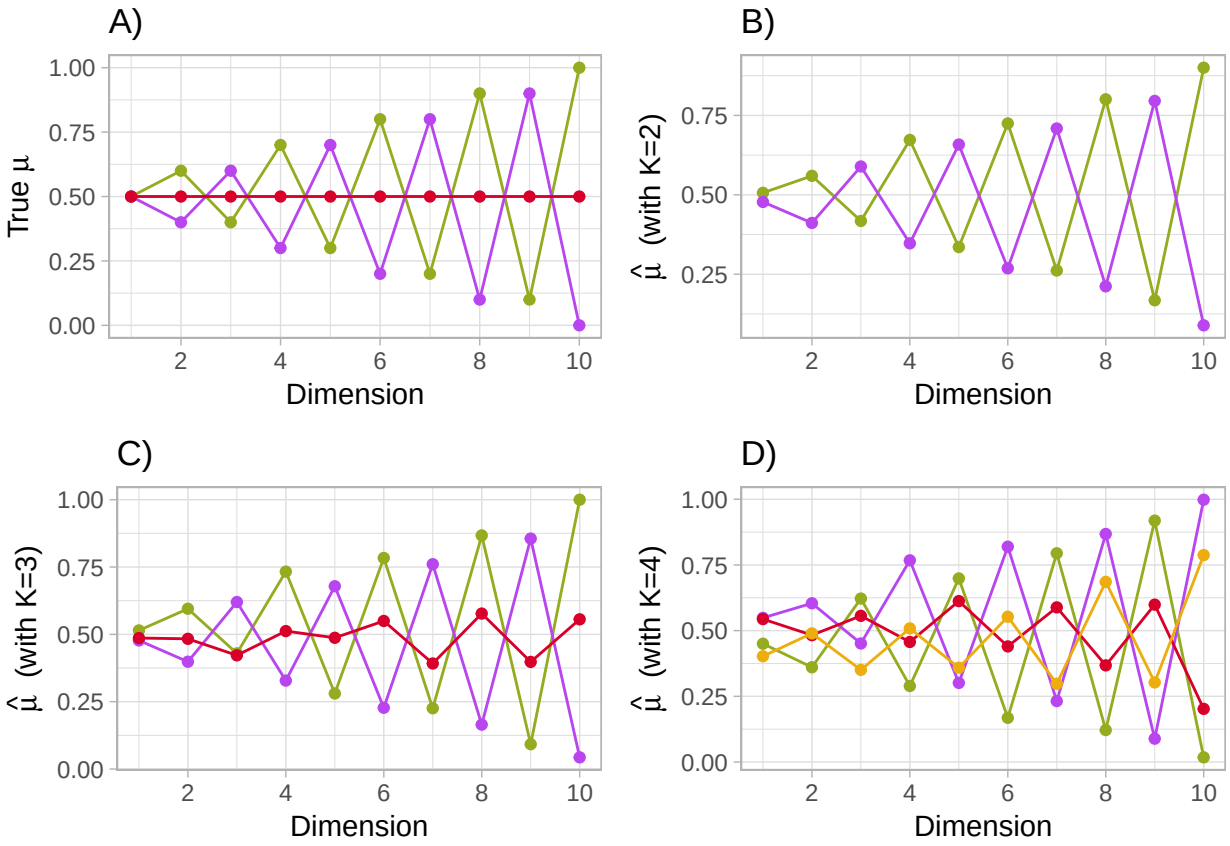

Figure 5: Line plot of the reals and of the all the estimated μ.

Appendix

```
knitr::opts_chunk$set(echo = FALSE, message = FALSE, warning = FALSE, error = FALSE,
                      fig.pos = 'h')

library(ggplot2)
library(randomForest)
library(mboost)


# ------------------------------------------------------------------------------
# Assignment 1
# ------------------------------------------------------------------------------

# Load the data
sp <- read.csv2("../dataset/spambase.csv")
sp$Spam <- as.factor(sp$Spam)

# Spliting the data to training and test
n = dim(sp)[1]
set.seed(12345)
id = sample(1:n, floor(n*(2/3)))
train = sp[id, ]
test = sp[-id, ]


# Fit the model for different number of trees(10,20,...,100)
misclass_rf_test = c()
misclass_rf_train = c()

for (i in seq(from = 10, to = 100,by = 10)) {

  set.seed(1627)

  # fit the model
  forest = randomForest(formula = Spam ~., data = train, ntree = i)

  #predict for training data
  predict_train = predict(forest,newdata = train)

  #confusion mattrix for training data
   conf_table_train = table(predict_train , train$Spam)

  #Misclassification rate for training data
  misclass_rf_train =
    c(misclass_rf_train, (conf_table_train[1,2] +
                          conf_table_train[2,1])/sum(conf_table_train))

  # Predict for test data
  predict_test = predict(forest,newdata = test)

  # Confusion matrix
   conf_table_test = table(predict_test , test$Spam)
```

```
  # Misclassification rate for test data
  misclass_rf_test =
    c(misclass_rf_test, (conf_table_test[1,2] +
                           conf_table_test[2,1])/sum(conf_table_test))
}

# Data frame with all misclassification rates for dif. number of trees
rf_errors_test = data.frame("trees" = seq(from = 10,to = 100,by = 10),"error" = misclass_rf_test,"Datas
rf_errors_train = data.frame("trees" = seq(from = 10,to = 100,by = 10),"error" = misclass_rf_train,"Data
rf_errors = rbind(rf_errors_test,rf_errors_train)


# Adaboosting Model
# Loop for all different trees
misclass_boost_test = c()
misclass_boost_train = c()

for (i in seq(from = 10, to = 100,by = 10)){

  # Fit the model with i number of trees
  boost = blackboost(formula = Spam ~., data = train, family = AdaExp(),
                     control = boost_control(mstop = i))

  # Misclassification rate for training data
  pred_train = predict(object = boost, newdata = train, type = "class")
  conf_boost_table_train = table( pred_train , train$Spam)
  misclass_boost_train =
    c(misclass_boost_train, (conf_boost_table_train[1,2] +
                              conf_boost_table_train[2,1])/sum(conf_boost_table_train))

  # Prediction for Adaboost
  predict_f = predict(boost, newdata = test,type = "class")

  # Confusion matrix for test data
  conf_boost_table = table( predict_f , test$Spam)

  # Misclassification rates for test data
  misclass_boost_test =
    c(misclass_boost_test, (conf_boost_table[1,2] +
                             conf_boost_table[2,1])/sum(conf_boost_table))
}

# Data frame with all misclassification rates for dif. number of trees
boost_errors_test = data.frame("trees" = seq(from = 10,to = 100,by = 10),
                               "error" = misclass_boost_test, "Dataset" = "test",
                               "Model" = "Adaboost")
boost_errors_train = data.frame("trees" = seq(from = 10,to = 100,by = 10),
                                "error" = misclass_boost_train, "Dataset" = "train",
                                "Model" = "Adaboost")
boost_errors = rbind(boost_errors_test,boost_errors_train)


# All the data we need for the plot (n_trees,errors,dataset,method)
```

```r
errors = rbind(rf_errors,boost_errors)

# Plot the errors for random Forest and Adaboost
ggplot(errors) + geom_line(aes_string(x="trees",
                                      y = "error",
                                      colour="Dataset",
                                      linetype="Model"))


# ------------------------------------------------------------------------------
# Assignment 2
# ------------------------------------------------------------------------------

set.seed(1234567890)              # Set seed for sake of reproducibility

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_class <- vector(length = D)  # True population of each observation
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)

# 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])
  }
  true_class[n] <- k
}

# Number of guessed components:
K <- c(2,3,4)
# Vectors where to store all pi:
all_pi <- c()
all_pi_names <- c()
# Data frame where to store all mu
df_mu <- data.frame(matrix(NA, D, sum(K)))
colnames(df_mu) <- paste("mu", unlist(sapply(K, function(x) 1:x)),
                         paste("_", unlist(sapply(K, function(x)
                           rep(x, x), simplify = F)), sep = ""),
                         sep = "")
# Matrix where to store all log likelihoods
all_llik <- matrix(nrow = 0, ncol = 3)
colnames(all_llik) <- c("LogLikelihood", "Iterations", "K")
```

```r
# Vector where to store the most probable Z once convergence is reached
pred_class <- matrix(nrow = 0, ncol = 2)
colnames(pred_class) <- c("Predicted", "K")


# EM for all the K components
for(try in K) {

  # Initialization of the parameters
  z <- matrix(nrow=N, ncol=try)          # Fractional component assignments
  pi <- vector(length = try)             # Mixing coefficients
  mu <- matrix(nrow=try, ncol=D)         # Conditional distributions
  llik <- vector(length = max_it)        # Log-likelihood of the EM iterations

  # Random initialization of the paramters
  pi <- runif(try,0.49,0.51)
  pi <- pi / sum(pi)
  for(k in 1:try) {
    mu[k,] <- runif(D,0.49,0.51)
  }

  # EM with "try" guessed components
  for(it in 1:max_it) {

    z <- exp(x %*% t(log(mu)) + (1-x) %*% t(log(1-mu)))   # P(x | mu)
    m_pi <- matrix(pi, N, k, byrow = T)
    llik[it] <- sum(log(rowSums(m_pi * z)))                 # Log-Likelyhood
    z <- (m_pi * z) / rowSums(m_pi * z)                       # Latent variable Z

    message(paste("iteration: ", it, "; log likelihood: ", llik[it], "\n", sep = ""))

    # Stop if the lok likelihood has not changed significantly
    if(it != 1  &&  llik[it]-llik[it-1] < min_change) {
      break
    }

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

  }

  # Save the results
  for(k in 1:try) {
    df_mu[[paste("mu", k, "_", try, sep = "")]] <- mu[k,]
  }
  if(try != max(K)) {
    pi[(try+1):max(K)] <- NA
  }
  all_pi <- c(all_pi, pi)
  all_pi_names <- c(all_pi_names, rep(paste(try), max(K)))
  all_llik <- rbind(all_llik, cbind(llik[llik!=0], 1:it, try))
  temp_class <- apply(z, 1, function(x) which.max(x))
```

```
  if(try==2) {
    temp_class <- ifelse(temp_class==1, 2, 1)
  }
  if(try==3) {
    temp_class <- ifelse(temp_class==3, 1, ifelse(temp_class==2, 3, 2))
  }
  if(try==4) {
    temp_class <- ifelse(temp_class==1, 2, ifelse(temp_class==2, 1,
                                                  ifelse(temp_class==3, 3, 4)))
  }
  pred_class <- rbind(pred_class, cbind(temp_class, try))

}


real_order <- c(2,1,3,4,7,5,6,8,10,9,11,12)
df_pi <- data.frame(Pi = all_pi[real_order], K = all_pi_names[real_order],
                    pop = rep(1:max(K), length(K)))

ggplot(df_pi, aes(K, Pi)) +
  geom_bar(aes(fill = K, group = pop), position = position_dodge(width=1),
           stat="identity", col = "black") +
  scale_fill_manual(values = c("#95ac21", "#d8002b", "#b945ee")) +
  labs(y = expression(paste(hat(pi)))) +
  theme_light()  +
  theme(legend.position="none")


all_llik <- as.data.frame(all_llik)
all_llik$K <- as.factor(all_llik$K)

ggplot(all_llik, aes(x = Iterations)) +
  geom_line(aes(y = LogLikelihood, col = K, group = K), size = 1) +
  geom_segment(aes(x = 12, y = min(all_llik$LogLikelihood),
                   xend = 12, yend = all_llik$LogLikelihood[12]),
               size = 0.1, linetype = "twodash", col = "#95ac21") +
  geom_segment(aes(x = 46, y = all_llik$LogLikelihood,
                   xend = 46, yend = all_llik$LogLikelihood[58]),
               size = 0.5, linetype = "1F", col = "#d8002b") +
  geom_segment(aes(x = 32, y = all_llik$LogLikelihood,
                   xend = 32, yend = all_llik$LogLikelihood[90]),
               size = 0.5, linetype = "1F", col = "#b945ee") +
  scale_color_manual(values = c("#95ac21", "#d8002b", "#b945ee")) +
  scale_x_continuous(breaks = seq(0, max(all_llik$Iterations), by = 4)) +
  theme_light()  +
  theme(axis.ticks.x = element_blank(), legend.position="bottom")


pred_class <- as.data.frame(pred_class)
pred_class$K <- as.factor(pred_class$K)
pred_class$Predicted <- as.factor(pred_class$Predicted)
pred_class$Real <- as.factor(rep(true_class, 3))
plot_class <- data.frame(Iteration = 0, Real = 0, Predicted = 0, Percentage = 0)
```

```r
for(k in levels(pred_class$K)) {
  temp <- prop.table(table(pred_class[pred_class$K==k,c(1,3)]), 2)
  k <- as.numeric(k)
  for(i in 1:nrow(temp)) {
    for(j in 1:ncol(temp)) {
      plot_class <- rbind(plot_class, c(k, j, i, temp[i,j]))
    }
  }
}
plot_class <- plot_class[-1,]
plot_class$Iteration <- as.factor(plot_class$Iteration)
plot_class$Real <- as.factor(plot_class$Real)
plot_class$Predicted <- as.factor(plot_class$Predicted)

ggplot(plot_class, aes(x = Real, y = Percentage, fill = Predicted)) +
  geom_bar(stat = 'identity', position = 'stack') +
  facet_grid(~ Iteration) +
  scale_y_continuous(labels = scales::percent_format()) +
  scale_fill_manual(values = c("#95ac21", "#b945ee", "#d8002b", "#fff001")) +
  theme_light()  +
  theme(legend.position = "bottom")


df_mu <- cbind(df_mu, t(true_mu))
colnames(df_mu)[10:12] <- c("true_m1", "true_m2", "true_m3")
df_mu$Dimension <- 1:10

p1 <- ggplot(df_mu, aes(x = Dimension)) +
  geom_line(aes(y = true_m1), col = "#95ac21") +
  geom_point(aes(y = true_m1), col = "#95ac21") +
  geom_line(aes(y = true_m2), col = "#b945ee") +
  geom_point(aes(y = true_m2), col = "#b945ee") +
  geom_line(aes(y = true_m3), col = "#d8002b") +
  geom_point(aes(y = true_m3), col = "#d8002b") +
  scale_x_continuous(breaks = seq(2, 10, 2)) +
  labs(y = expression(paste("True ", mu)), title = "A)") +
  theme_light()

p2 <- ggplot(df_mu, aes(x = Dimension)) +
  geom_line(aes(y = mu2_2), col = "#95ac21") +
  geom_point(aes(y = mu2_2), col = "#95ac21") +
  geom_line(aes(y = mu1_2), col = "#b945ee") +
  geom_point(aes(y = mu1_2), col = "#b945ee") +
  scale_x_continuous(breaks = seq(2, 10, 2)) +
  labs(y = expression(paste(hat(mu), "  (with K=2)")), title = "B)") +
  theme_light()

p3 <- ggplot(df_mu, aes(x = Dimension)) +
  geom_line(aes(y = mu3_3), col = "#95ac21") +
  geom_point(aes(y = mu3_3), col = "#95ac21") +
  geom_line(aes(y = mu1_3), col = "#b945ee") +
  geom_point(aes(y = mu1_3), col = "#b945ee") +
  geom_line(aes(y = mu2_3), col = "#d8002b") +
```

```
  geom_point(aes(y = mu2_3), col = "#d8002b") +
  scale_x_continuous(breaks = seq(2, 10, 2)) +
  labs(y = expression(paste(hat(mu), "  (with K=3)")), title = "C)") +
  theme_light()

p4 <- ggplot(df_mu, aes(x = Dimension)) +
  geom_line(aes(y = mu1_4), col = "#95ac21") +
  geom_point(aes(y = mu1_4), col = "#95ac21") +
  geom_line(aes(y = mu2_4), col = "#b945ee") +
  geom_point(aes(y = mu2_4), col = "#b945ee") +
  geom_line(aes(y = mu3_4), col = "#d8002b") +
  geom_point(aes(y = mu3_4), col = "#d8002b") +
  geom_line(aes(y = mu4_4), col = "darkgoldenrod2") +
  geom_point(aes(y = mu4_4), col = "darkgoldenrod2") +
  scale_x_continuous(breaks = seq(2, 10, 2)) +
  labs(y = expression(paste(hat(mu), "  (with K=4)")), title = "D)") +
  theme_light()

gridExtra::grid.arrange(p1, p2, p3, p4, ncol = 2, nrow = 2)
```