task-2

Ngoc Duong

4/23/2020

Load curves.Rdata

```
load("./curves.RData")
```

```
#fake data
set.seed(123123)
Sigma1 = matrix(c(1, 0.5, 0.5,0.5,1,0.5,0.5, 0.5,1), ncol = 3, nrow = 3)
x1 = mvrnorm(n = 100, mu = c(0 ,0, 0), Sigma1)
Sigma2 = matrix(c(2, 0.5, 0.5,0.5,2,0.5,0.5, 0.5,2), ncol = 3, nrow = 3)
x2 = mvrnorm(n = 100, mu = c(1, 3, 2), Sigma2)
Sigma3 = matrix(c(3, 0.5, 0.5,0.5,3,0.5,0.5, 0.5,3), ncol = 3, nrow = 3)
x3 = mvrnorm(n = 100, mu = c(4 ,1, -2), Sigma3)
data =  rbind(x1, x2, x3)
#standardize data
standardize = function(col) {
  mean = mean(col)
  stdev = sd(col)
  return((col - mean)/stdev)
}
data = as_tibble(data) %>% map_df(.x = ., standardize)
```

```
## Warning: `as_tibble.matrix()` requires a matrix with column names or a `.name_repair` argument. Using
## This warning is displayed once per session.
```

K-means For each data point x, compute d, the distance between x and the nearest center that has already been chosen. Choose one new data point at random as a new center, using a weighted probability distribution where a point x is chosen with probability proportional to d^2 Repeat steps 2 and 3 until k centers have been chosen.

```
#partition of data such that squared error between empirical mean and points in each cluster/partition
km_func <- function(data, k){
  p <- ncol(data)  # number of parameters
  n <- nrow(data)  # number of observations
  diff = 1
  iter = 0
  itermax = 50
  while(diff > 1e-4 && iter <= itermax){
    #initial centroids
    if(iter == 0){
      centroid = data[sample(nrow(data), k),]
      centroid_mem = centroid
    }

    #assign to cluster
    d = sapply(1:k, function(c) sapply(1:n, function(i) {sum((centroid[c,] - data[i,])^2)}))

    cluster = apply(d, 1, which.min)

    #recalculate cluster
    centroid = t(sapply(1:k,
                        function(c) {apply(data[cluster == c,], 2, mean)}
                        ))

    #recalculate distance
    diff = sum((centroid - centroid_mem)^2)
    iter = iter + 1
    centroid_mem = centroid
  }
  return(list(centroid = centroid, cluster = cluster))
}
```

```
#test on simulated data
set.seed(7)
km_sim <- km_func(data, 3)
colnames(data) = c("a","b","c")
data_new = cbind(data, cluster = km_sim$cluster)


#crosstab with true underlying distribution
data_true = data %>% mutate(cluster = c(rep(1,100),rep(2,100),rep(3,100)))

#check misclassification
table(km_sim$cluster,data_true$cluster) %>% knitr::kable()
```

1	2	3
0	1	86
4	77	5
96	22	9

Misclassification is not too high, reason for misclassification is likely due to MVN RV's are actually kind of close to each other.

```
plot_ly(x=data_new[,1], y=data_new[,2], z=data_new[,3], type="scatter3d", mode="markers", color = data_
```

Run k-means algorithm on estimated parameters

```
#import data
param_df = param_df1 %>% as.data.frame()

#standardize data
param_names = c("a_std","b_std","c_std")
param_standard = NULL
for (i in 2:4) {
col = (param_df[,i] - mean(param_df[,i]))/sd(param_df[,i])
param_standard = cbind(param_standard, col)
}
colnames(param_standard) = param_names
param_both = cbind(param_standard, param_df)
```

```r
set.seed(2020)
#apply K-means to param estimates data
km_2 <- km_func(param_standard, 2)
km_3 <- km_func(param_standard, 3)
km_4 <- km_func(param_standard, 4)
km_5 <- km_func(param_standard, 5)

#prepare final k-means data
param_km2_final = cbind(param_both, cluster = km_2$cluster) %>%
  group_by(cluster) %>% arrange(desc(cluster))

param_km3_final = cbind(param_both, cluster = km_3$cluster) %>%
  group_by(cluster) %>% arrange(desc(cluster))

param_km4_final = cbind(param_both, cluster = km_4$cluster) %>%
  group_by(cluster) %>% arrange(desc(cluster))

param_km5_final = cbind(param_both, cluster = km_5$cluster) %>%
  group_by(cluster) %>% arrange(desc(cluster))
```

```r
#plot on standardized data
plot_ly(x=param_km3_final$a_std, y=param_km3_final$b_std, z=param_km3_final$c_std,
        type="scatter3d", mode="markers", color = param_km3_final$cluster)

#visualize on original data - 2 clusters
plot_ly(x=param_km2_final$a, y=param_km2_final$b, z=param_km2_final$c,
        type="scatter3d", mode="markers", color = param_km2_final$cluster)

#visualize on original data - 3 clusters
plot_ly(x=param_km3_final$a, y=param_km3_final$b, z=param_km3_final$c,
        type="scatter3d", mode="markers", color = param_km3_final$cluster)

#visualize on original data - 4 clusters
plot_ly(x=param_km4_final$a, y=param_km4_final$b, z=param_km4_final$c,
        type="scatter3d", mode="markers", color = param_km4_final$cluster)

#visualize on original data - 5 clusters
plot_ly(x=param_km5_final$a, y=param_km5_final$b, z=param_km5_final$c,
        type="scatter3d", mode="markers", color = param_km5_final$cluster)
```

Summary table for kmeans clustering results

```r
param_km3_final %>% group_by(cluster) %>%
  summarise(`Mean a` = mean(a),
            `Mean b` = mean(b),
            `Mean c` = mean(c),
            `Max a` = max(a),
            `Max b` = max(b),
            `Max c` = max(c),
            `Min a` = min(a),
            `Min b` = min(b),
            `Min c` = min(c)) %>%
  knitr::kable()
```

cluster	Mean a	Mean b	Mean c	Max a	Max b	Max c	Min a	Min b	Min c
1	24196.053	0.0815789	73.55159	610000	0.18	89.99994	100	0.03	51.0419683
2	7970.455	0.1261364	35.22784	110000	0.22	61.16436	100	0.06	0.0000783
3	198895.455	0.2609091	39.61083	1000000	0.50	89.79950	100	0.17	5.9623879

Grouped boxplots of a, b, and c for 3 clusters (k = 3)

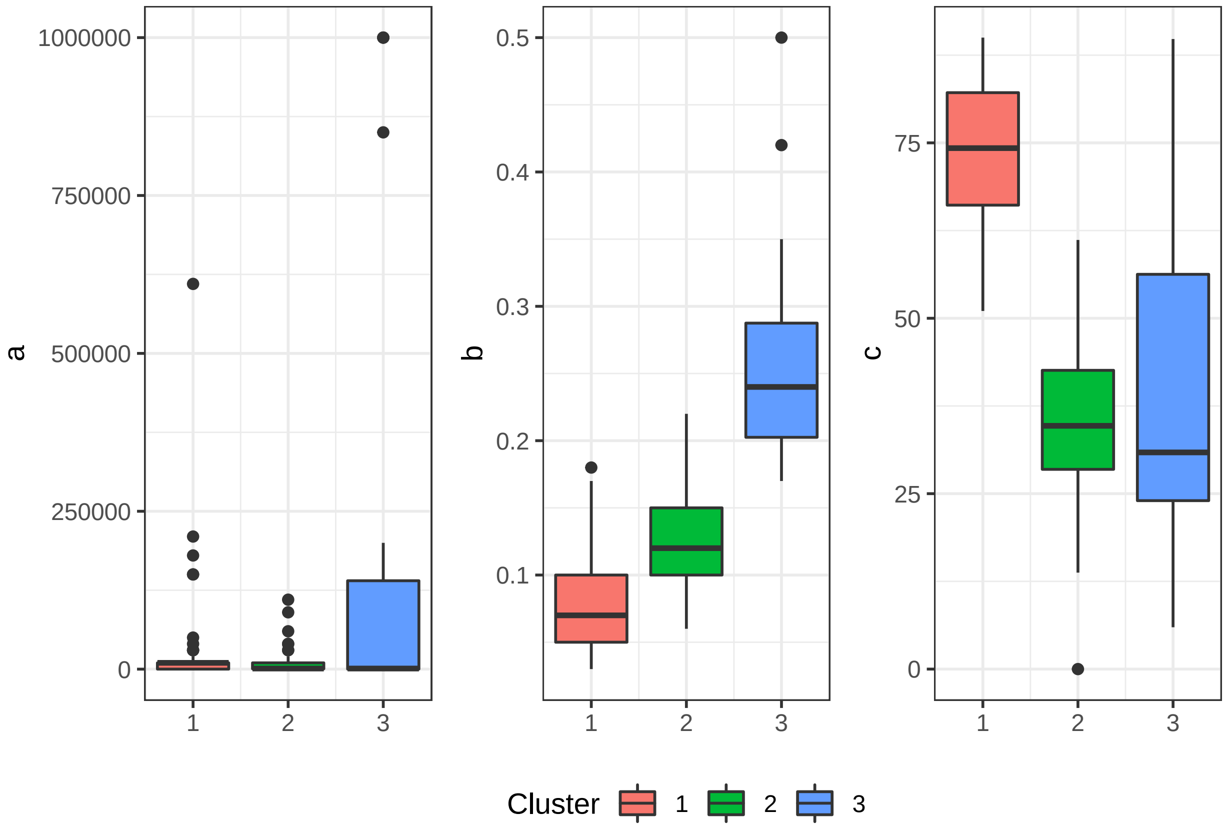
On original scale

```
a = param_km3_final %>% group_by(cluster) %>%
  ggplot(aes(x = cluster, y = a, fill = factor(cluster))) +
  geom_boxplot() + theme_bw() +
  theme(legend.position = "none", axis.title.x=element_blank())

b = param_km3_final %>% group_by(cluster) %>%
  ggplot(aes(x = cluster, y = b, fill = factor(cluster))) +
  geom_boxplot() + theme_bw() + labs(fill = "Cluster") +
  theme(legend.position = "bottom", axis.title.x=element_blank())

c = param_km3_final %>% group_by(cluster) %>%
  ggplot(aes(x = cluster, y = c, fill = factor(cluster))) +
  geom_boxplot() + theme_bw() +
  theme(legend.position = "none", axis.title.x=element_blank())

a+b+c
```



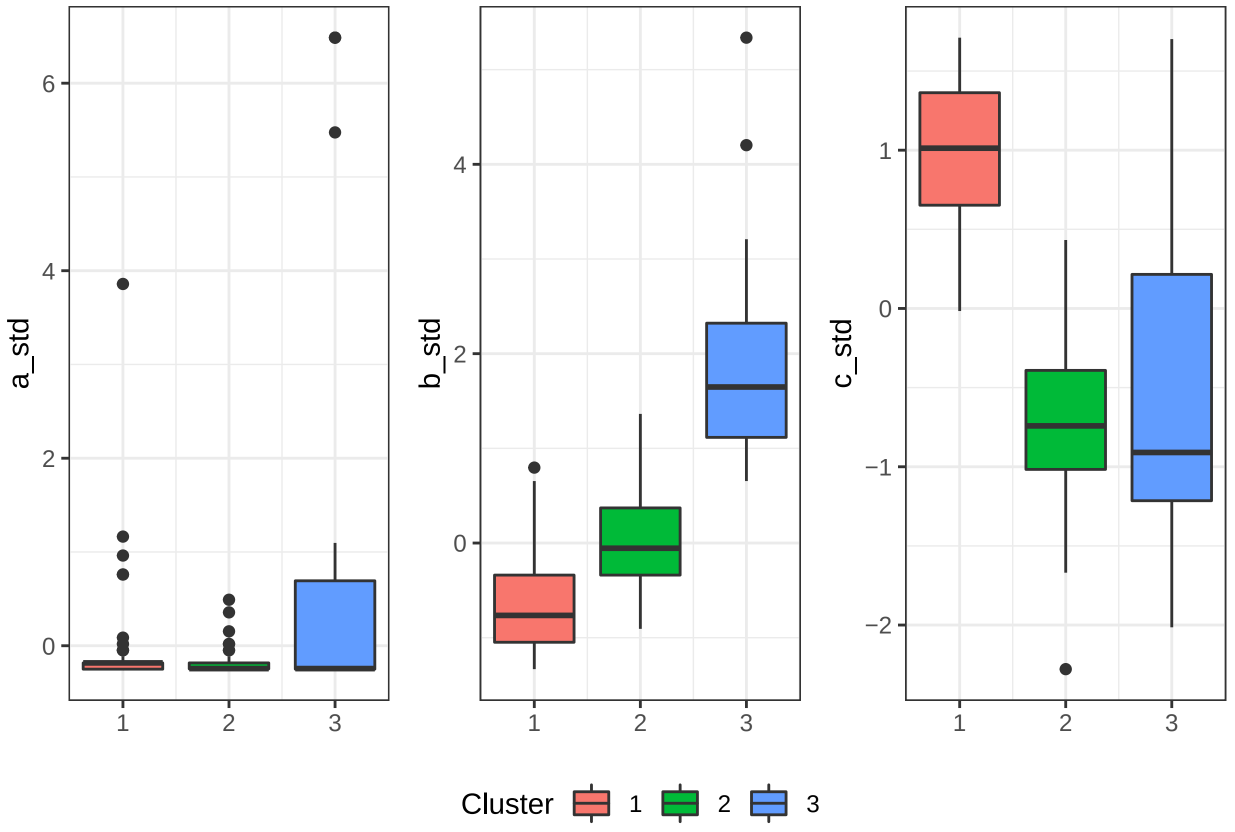
On standardized scale

```r
a1 = param_km3_final %>% group_by(cluster) %>%
  ggplot(aes(x = cluster, y = a_std, fill = factor(cluster))) +
  geom_boxplot() + theme_bw() +
  theme(legend.position = "none", axis.title.x=element_blank())

b1 = param_km3_final %>% group_by(cluster) %>%
  ggplot(aes(x = cluster, y = b_std, fill = factor(cluster))) +
  geom_boxplot() + theme_bw() + labs(fill = "Cluster") +
  theme(legend.position = "bottom", axis.title.x=element_blank())

c1 = param_km3_final %>% group_by(cluster) %>%
  ggplot(aes(x = cluster, y = c_std, fill = factor(cluster))) +
  geom_boxplot() + theme_bw() +
  theme(legend.position = "none", axis.title.x=element_blank())

a1+b1+c1
```



EM algorithm for Gaussian mixtures

```r
gmm_func <- function(X, k){
  #setting
  data <- as.matrix(X)
  #%>% scale()
  N <- nrow(data)
  q <- ncol(data)
  p_j <- rep(1/k, k)
  mu <-  data[sample(N, k),] %>% as.matrix()
  covmat <- diag(ncol(data))
  covList <- list()
  for(i in 1:k){
    covList[[i]] <- covmat
  }

  count=1
  while(count <100){
    mu0 <- mu

    # E-step: Evaluate posterior probability, gamma
    gamma <- c()
    for(j in 1:k){
      gamma2 <- apply(data,1, mvtnorm::dmvnorm, mu[j,], covList[[j]])
      gamma <- cbind(gamma, gamma2)
    }

    # M- step: Calculate mu
    tempmat <- matrix(rep(p_j,N),nrow=N,byrow = T)
    r <- (gamma * tempmat) / rowSums(gamma*tempmat)
    mu <- t(r) %*% data / colSums(r)

    # M- step: Calculate Sigma and p
    for(j in 1:k){
      sigma <- matrix(rep(0,q^2),ncol=q)
      for(i in 1:N){
        sigma = sigma + r[i,j] * (data[i,]-mu0[j,]) %*% t(data[i,]-mu0[j,])
      }
      covList[[j]] <- sigma/sum(r[,j])
    }
    p_j <- colSums(r)/N
    count = count + 1
  }

  cluster <- which(r == apply(r, 1, max), arr.ind = T)
  cluster <- cluster[order(cluster[,1]),]
  return(list(cluster = cluster))
}
```

Test on simulated data

```
set.seed(7)
gmm_test = gmm_func(data, 3)
sim_data_gmm = cbind(data,gmm_test$cluster) %>% dplyr::select(-row) %>% rename(cluster = col)

#see clustering performance -- ideally 100 for each cluster
table(gmm_test$cluster[,2])
```

```
##
##   1   2   3
##  99  94 107
```

```
#check misclassification
table(gmm_test$cluster[,2],data_true$cluster)
```

```
##
```

```
##       1  2  3
##   1   2  1 96
##   2   7 85  2
##   3  91 14  2
```

Some misclassification (clustered into incorrect underlying distributions), but misclassification rate is not too high (10%) so it's ok

```r
plot_ly(x=sim_data_gmm$a, y=sim_data_gmm$b, z=sim_data_gmm$c,
        type="scatter3d", mode="markers", color = sim_data_gmm$cluster)
```

Now, we can try run GMM on estimated parameter data, picking k = 3

```r
set.seed(2020)
param_gmm <- gmm_func(param_standard, 3)

#prepare data
param_gmm_final = cbind(param_both, cluster= param_gmm$cluster[,2]) %>%
  group_by(cluster) %>% arrange(desc(cluster))

table(param_gmm_final$cluster) %>% knitr::kable()
```

Var1	Freq
1	103
2	21
3	62

```r
param_gmm_final %>% group_by(cluster) %>%
  summarise(`Mean a` = mean(a),
            `Mean b` = mean(b),
            `Mean c` = mean(c),
            `Max a` = max(a),
            `Max b` = max(b),
            `Max c` = max(c),
            `Min a` = min(a),
            `Min b` = min(b),
            `Min c` = min(c)) %>%
  knitr::kable()
```

cluster	Mean a	Mean b	Mean c	Max a	Max b	Max c	Min a	Min b	Min c
1	445.6311	0.1183495	44.28682	1e+03	0.42	89.99994	100	0.03	0.0000783
2	289052.3810	0.1800000	61.94758	1e+06	0.50	89.91041	100	0.10	18.1315000
3	12903.2258	0.1140323	59.66076	3e+04	0.23	89.99992	10000	0.05	14.1520460

```r
#plot on standardized data
plot_ly(x=param_gmm_final$a_std, y=param_gmm_final$b_std, z=param_gmm_final$c_std,
        type="scatter3d", mode="markers", color = param_gmm_final$cluster)

#visualize on original data k =3
plot_ly(x=param_gmm_final$a, y=param_gmm_final$b, z=param_gmm_final$c,
        type="scatter3d", mode="markers", color = param_gmm_final$cluster)
```

Grouped boxplots of a, b, and c for 3 clusters (k = 3)

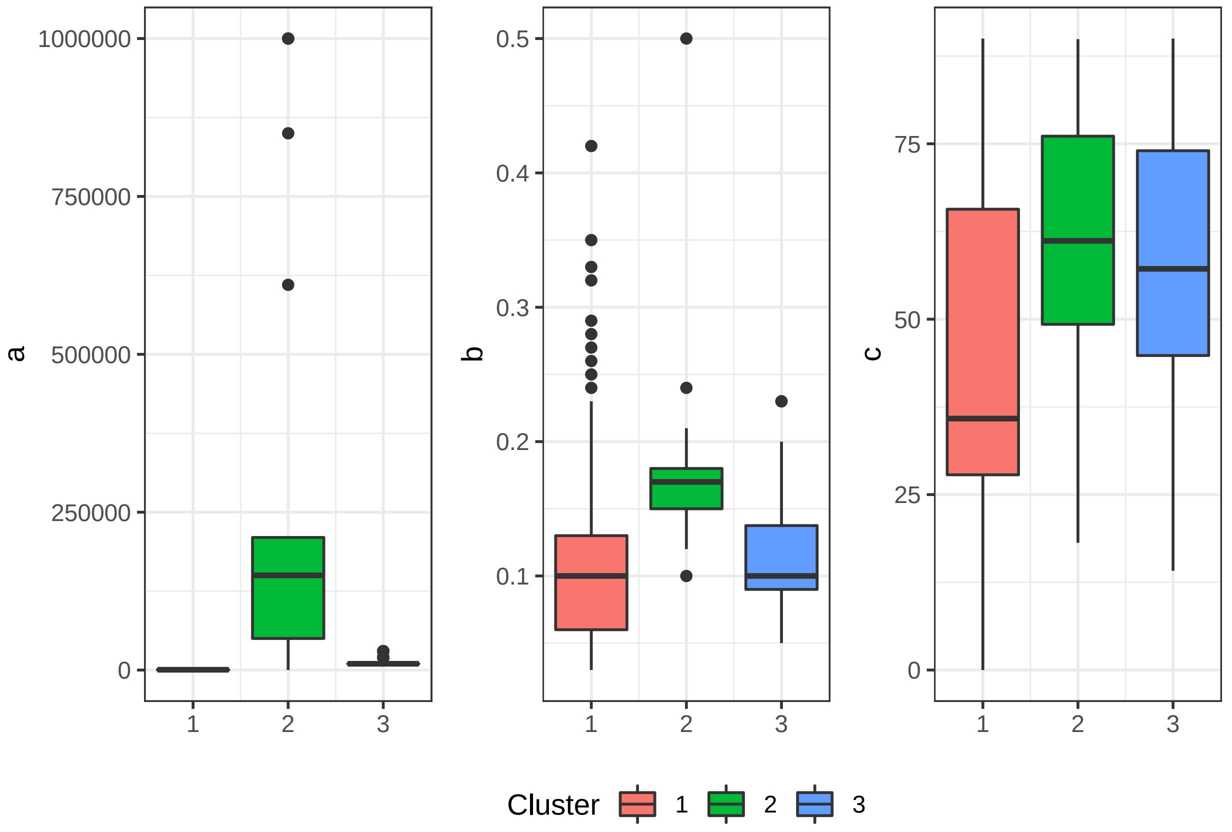
On original scale

```
d = param_gmm_final %>% group_by(cluster) %>%
  ggplot(aes(x = cluster, y = a, fill = factor(cluster))) +
  geom_boxplot() + theme_bw() +
  theme(legend.position = "none", axis.title.x=element_blank())

e = param_gmm_final %>% group_by(cluster) %>%
  ggplot(aes(x = cluster, y = b, fill = factor(cluster))) +
  geom_boxplot() + theme_bw() + labs(fill = "Cluster") +
  theme(legend.position = "bottom", axis.title.x=element_blank())

f = param_gmm_final %>% group_by(cluster) %>%
  ggplot(aes(x = cluster, y = c, fill = factor(cluster))) +
  geom_boxplot() + theme_bw() +
  theme(legend.position = "none", axis.title.x=element_blank())

d+e+f
```



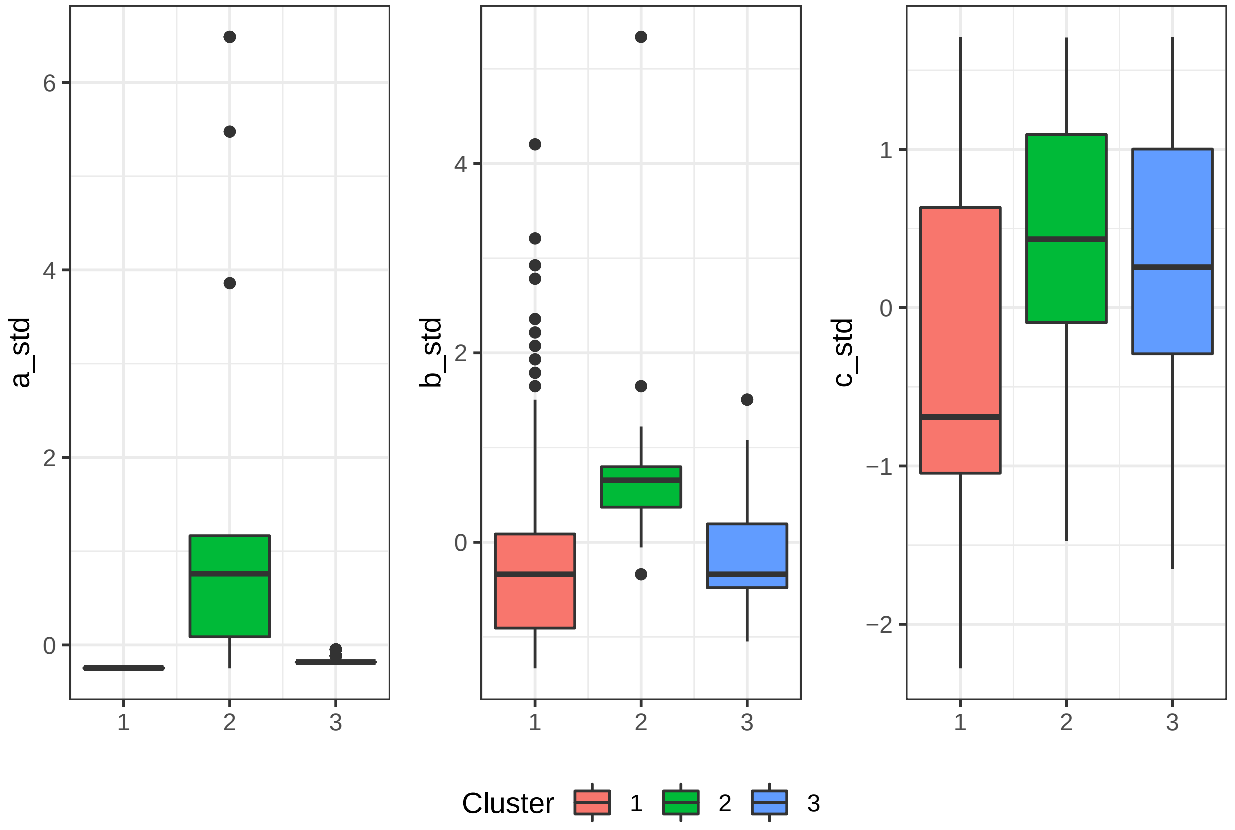
On standardized scale

```
d1 = param_gmm_final %>% group_by(cluster) %>%
  ggplot(aes(x = cluster, y = a_std, fill = factor(cluster))) +
  geom_boxplot() + theme_bw() +
  theme(legend.position = "none", axis.title.x=element_blank())

e1 = param_gmm_final %>% group_by(cluster) %>%
  ggplot(aes(x = cluster, y = b_std, fill = factor(cluster))) +
  geom_boxplot() + theme_bw() + labs(fill = "Cluster") +
  theme(legend.position = "bottom", axis.title.x=element_blank())

f1 = param_gmm_final %>% group_by(cluster) %>%
  ggplot(aes(x = cluster, y = c_std, fill = factor(cluster))) +
  geom_boxplot() + theme_bw() +
  theme(legend.position = "none", axis.title.x=element_blank())

d1+e1+f1
```



Math behind clustering methods

For both methods, we first standardize the data as follows:

$$standardized(x_i) = \frac{x_i - \bar{x}}{std(x)}$$

for $i = 1, 2, ..., n$

K-means

First, we produce a set of initial centroids (sample means) $\mu_1^{(0)}, \mu_2^{(0)}, ..., \mu_k^{(0)}\}$. The k-means algorithm is based on that initialized setup to iterate the following two steps:

Find optimal cluster assignment given fixed centroids

For each data point x_i, compute d_i, the Euclidean distance between x_i and the centroids, then assign x_i to cluster with the smallest d_i. This is equivalent to:

$$r_{i,j}^{(v+1)} = I\{j = \arg\min_j \|\mathbf{x}_i - \mu_j^{(v)}\|\}$$

Calculate cluster centers using the cluster assignment in the previous step

This steps involves minimizing $J(r, \mu)$ over μ:

$$\mu_j^{(v+1)} = \frac{\sum_{i=1}^n \mathbf{x}_i r_{i,j}^{(v+1)}}{\sum_{i=1}^n r_{i,j}^{(v+1)}}$$

In other words, we update the new cluster means by taking the average of all datapoints that belong to that cluster using a weighted probability distribution where point x_i is chosen. This reflects the new cluster assignments of the datapoints.

Then, we repeat these two steps until k centers have been chosen/converged.

Gaussian mixtures

Let $\{$x1,x2,...,xn$\} \in R^3$ be a collection of 3-dimensional datapoints. Assuming they come from underlying Gaussian distributions, we can implement the EM algorithm to identify what distribution each points belongs to, which consists of three steps: initialization, E-step, and M-step. In short, the algorithm randomly selects k different clusters and starts with finding weights, mean, and covariance matrix for each cluster. Then, E-step can be applied on those values. These two steps are then reiteratively applied back-and-forth until centroids converge or maximum iteration threshold is reached.

Initialization

Gaussian mixture models assume that each latent class has different means and covariances. However, since each class is unknown, we begin by intializing the parameters and iteratively updating. Here, we chose to initialize 3 (random) different classes.

Then, we initialize a weight (posterior probability) matrix γ with size, where each row contains the mean value for each of the three Gaussians. Then, the values for covariance matrix list, where each element in the list represents a covariance matrix for each Gaussian. Finally, we can randomly assign data to each Gaussian with the weight matrix, where each column shows the probability each point belongs to each of the three clusters.

E-step In E-step, each observation is assigned a weight (responsibility) for each cluster, based on the likelihood of each of the corresponding Gaussian using the initialized parameters (means and covariance matrix and the function dmvnorm). The responsibility is defined as:

$$\hat{\gamma}_{i,k}^{(t)} = P(r_{i,k} = 1 | \mathbf{x}_i, \theta^{(t)}) = \frac{p_k^{(t)} f(\mathbf{x}_i | \boldsymbol{\mu}_k^{(t)}, \Sigma_k^{(t)})}{\sum_{j=1}^K f(\mathbf{x}_i | \boldsymbol{\mu}_j^{(t)}, \Sigma_j^{(t)})}$$

M-step

In the M-step, we aim to maximize the complete log-likehood function and thus revising the parameters. In this step, each observation contributes to the weighted means and covariances for every cluster. Here, the mixing component is $p_k = \frac{n_k}{n}(1)$, also a prior for each class estimated from the data. Let $n_k = \sum_i^n \gamma_{ik}(2)$, we have

$$\mu_k = \frac{1}{m_k} \sum_i \gamma_{ik} x_i (3)$$

, and

$$\Sigma_k = \frac{1}{n_k} \sum_i \gamma_{ik} (x_i \breve{} \mu_k)^T (x_i - \mu_k)(4)$$

We start by updating the mixing components (i.e. prior probabilities). Each datapoint contributes to a $\gamma_{i,k}$ which corresponds to one column vector each class. Given the available information, using (1) and (2), we can update the prior probabilities. Following the formulas (3) and (4), we can update μ_k^{t+1} and Σ_k^{t+1} as our new Gaussian density parameters. This process involves standard calculation of mean and variance for a

Gaussian distribution. However, during this process, we also use weights γ_{ik} and $\frac{1}{m_k}$ to weight each point in our mean and covariance estimation.

Once the likelihood has been maximized in this step, loop back and iterate the process until a maximum is found/an iteration threshold has been reached.

Discussion for clustering

Despite the relative similarity in the decision boundaries of the two clustering methods, Gaussian mixtures model produced a somewhat smoother decision boundary/plane. (3D plot?)

We also might want to take into account some considerations: Gaussian mixture models converge well when the class densities are better separated, so this also depends on how fine the parameter estimates are.