MAD Project 2023 - Generalized K-means

Richard CHEAM

2023-12-09

Orthogonal Projection

Consider a line D and a point $x \in \mathbb{R}^{|}$. Write the explicit formula and the associated computer code that gives:

- The coordinates of the orthogonal projection of x onto D

Let $O = (O_1, O_2, ..., O_p)$ be the origin and $x = (x_1, x_2, .., x_p)$ is the point to be projected. The orthogonal projection formula of x onto a line D with parametric equation $\vec{x}_i = \vec{O}_i + \vec{v}t$ for $i = 1, 2, ..., p$ passing through O with a directional vector $v = (v_1, v_2, ..., v_p)$ can be computed as:

$$Proj_D(x) = O + \frac{\vec{Ox}.\vec{v}}{||\vec{v}||^2}\vec{v}$$

Where

$$\vec{Ox} = (x_1 - O_1, ..., x_p - O_p)$$

```
ortho_proj <- function(O, x, v){
  Ox = x - O
  proj = O + as.vector(((Ox %*% v)/(v %*% v))) * v
  return (proj)
}
```

- The distance between x and D:

$$d(x, D) = ||x - Proj_D(x)||$$

```
distance <- function(O, x, v){
  proj = ortho_proj(O, x, v)
  dis = x - proj
  return (sqrt(dis %*% dis))
}
```

- Let K lines $D_1, ..., D_K$ and a point x in $\mathbb{R}^{|}$. Write the code that determines the closest line to x. In case of equal distances, a random selection will be used.

```r
#D is a set of lines
closest_line <- function(x, D){
  K = length(D)
  set_dis = c()
  dim = length(x)
  for (k in 1:K){
    set_dis[k] = distance(D[[k]][1:dim], x, D[[k]][(dim+1):length(D[[k]])])
  }
  min_val = min(set_dis)
  #handle case which there are many min values, so we want to randomly select one of the lines
  #store smallest index
  ind_of_min = which(set_dis == min_val)
  #check if there are more than 1
  if (length(ind_of_min) > 1){
    #randomly select one of them
    rand_ind = sample(ind_of_min, 1)
  } else {
    rand_ind = ind_of_min
  }
  return (rand_ind)
}
```

Dynamic Clouds or Generalized k-means

The dynamic clouds algorithm or generalized k-means replaces the notion of class center with the notion of class representative and minimizes the following criterion:

$$J(C, D) = \sum_{i} \sum_{k=1}^{K} c_{ik} d^2(x_i, D_k)$$

Generalized k-means implementation

Describe and code a generalized k-means algorithm in $\mathbb{R}^1$ where the class representative is a line and d^2 is the distance between a point and its orthogonal projection onto D_k.

1. Arbitrarily choose an initial k representatives (lines) from $D = \{D_1, ..., D_k\}$
2. For each $i \in \{1, ..., k\}$, minimize the criterion $J(C, D)$ with respect to C, meaning, set the cluster C_i to be the set of points in $\mathcal{X}$ that are closer to D_i than they are to D_j for all $j \neq i$, where $\mathcal{X}$ be a set of all data points. In other words, assigning data points to the closest representative class (line).
3. For each $i \in \{1, ..., k\}$, set D_i to be the class representative in which it minimizes the criterion $J(C, D)$ with respect to D, precisely, given that $d(x, D) = \sqrt{(x_1 - (O_1 + v_1 t))^2 + ... + (x_p - (O_p + v_p t))^2}$, find the partial derivatives with respect to parameters $\vec{O_k}$ and $\vec{v_k}$, then by taking $\frac{dJ}{d\vec{O_k}} = 0$, $\frac{dJ}{d\vec{v_k}} = 0$ as a result, the optimized parameters will be defined and set as the new class representative.(Resemble to EM for PCA)
4. Iterate step (2) and (3) until stationarity (convergence) is attained.

```r
# Function to compute optimal O_k for a given k
compute_Ok <- function(k, C, X, Vk) {
  numerator <- apply(C, 1, function(ci) sum(ci * (X - Vk) / sum(Vk^2) * Vk))
  denominator <- rowSums(C)
  Ok <- numerator / denominator
  return(Ok)
}

# Function to compute optimal v_k for a given k
compute_vk <- function(k, C, X, Ok, Vk) {
  numerator <- apply(C, 1, function(ci) sum(ci * (X - Ok) / sum(Vk^2) * Vk))
  denominator <- rowSums(C)
  vk <- numerator / denominator
  return(vk)
}
```

```r
generalized_kmeans <- function(X, k, max_iter = 100){
  X <- as.matrix(X)
  n <- nrow(X)
  p <- ncol(X)
  C <- matrix(0, nrow = n, ncol = k)
  # D = list()
  #for (i in 1:n){
  #  sample_O = rnorm(p)
  #  sample_v = rnorm(p)
  #  D[[i]] = c(sample_O, sample_v)
  #}

  #randomly pick (k lines) from (n lines) for (k cluster)
  #index_of_chosen_line = sample(1:n, k)
  #initialize lines
  #Dk = list()
  #for (i in 1:k){
  #  Dk[[i]] = D[[index_of_chosen_line[i]]]
  #}

  #choose Dk
  S <- (t(X)%*%X)/n
  eig = eigen(S)
  Dk = list()
  for(i in 1:k){
    Dk[[i]] = c(rep(0,p),eig$vectors[i,])
  }

  Dk_new = list()
  V = list()
  O = list()
  for (iter in 1:max_iter) {
    # Step 2: Assign each data point to the closest line
    closest_line_to = sapply(1:n, function(i) closest_line(X[i,], Dk))

    # Cluster each data point x to the closest line among k lines
    for (i in 1:n) {
```

```
      C[i, ] = rep(0, k)
      C[i, closest_line_to[i]] = 1
    }

    # Step 3: Update representative class based on current cluster C
    for (i in 1:k){
      O <- Dk[[i]][1:p]
      V <- Dk[[i]][(p + 1):(2 * p)]
    }
    new_O <- compute_Ok(k, C, X, V)
    new_v <- compute_vk(k, C, X, O, V)

    for (i in 1:k){
      Dk_new[[i]] = c(new_O[1:p], new_v[(p + 1):(2 * p)])
    }

    if (identical(Dk, Dk_new)){
      break
    }

    Dk = Dk_new

  }
  return (C)
}
```

Test and compare on iris dataset

```
data <- iris[,1:4]
true_class <- iris[,5]
```

```
res_gkm <- generalized_kmeans(data, 3)
gkm_class <- apply(res_gkm, 1, function(row) which(row == 1))
gkm_class
```

```
##   [1] 3 1 3 2 3 2 3 1 3 2 3 3 3 2 2 2 3 3 2 2 1 3 1 2 2 2 2 3 3 3 3 2 2 1 1 2 1
##  [38] 1 1 2 1 3 3 2 3 3 1 1 1 1 3 1 1 2 2 2 1 2 2 1 2 3 2 2 2 2 3 2 3 2 1 2 2 2
##  [75] 1 3 1 2 2 3 3 3 3 1 1 2 3 3 1 1 3 3 2 3 2 2 3 2 2 2 3 1 3 2 3 1 1 1 1 3 3
## [112] 2 3 1 1 1 3 1 1 2 3 2 3 1 1 3 1 1 3 3 1 1 3 3 2 1 3 1 1 1 2 3 2 2 1 2 1 1
## [149] 2 2
```

```
res_km <- kmeans(as.matrix(data), 3)
km_class <- res_km$cluster
km_class
```

```
##   [1] 2 3 3 3 2 2 2 2 3 3 2 2 3 3 2 2 2 2 2 2 2 2 2 2 3 3 2 2 2 3 3 2 2 2 3 2 2
##  [38] 2 3 2 2 3 3 2 2 3 2 3 2 2 1 1 1 1 1 1 1 3 1 1 3 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [75] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 3 1 1 1 1 3 1 1 1 1 1 1 1 1 1 1 1
## [112] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
## [149] 1 1
```

```
res_gmm <- Mclust(data, G = 3)
gmm_class <- res_gmm$classification
gmm_class
```

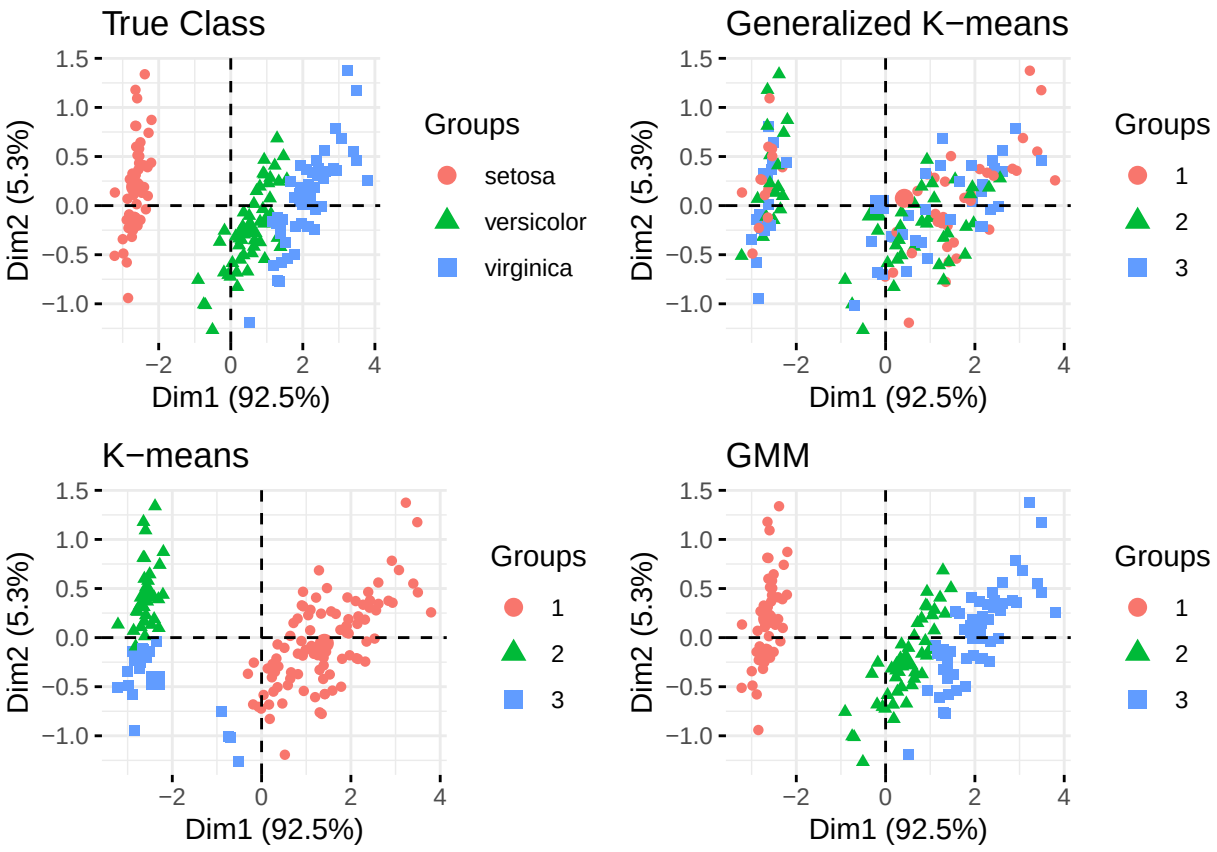
```
##   [1] 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
##  [38] 1 1 1 1 1 1 1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 2 3 2 3 2
##  [75] 2 2 2 3 2 2 2 2 2 3 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3 3 3 3 3
## [112] 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3 3
## [149] 3 3
```

Visualize the different partitions

```
data_scale <- scale(data, center = TRUE, scale = FALSE)
iris_pca <- PCA(data, ncp = 2, scale.unit = FALSE, graph = FALSE)
```

```
iris_plot <- fviz_pca_ind(iris_pca, label="none",
                          habillage = as.factor(true_class), title="True Class")
iris_plot_gkm <- fviz_pca_ind(iris_pca, label="none",
                  habillage = as.factor(gkm_class), title = "Generalized K-means")
iris_plot_km <- fviz_pca_ind(iris_pca, label="none",
                                habillage = as.factor(km_class), title="K-means")
iris_plot_gmm <- fviz_pca_ind(iris_pca, label="none",
                                habillage = as.factor(gmm_class), title="GMM")
```

```
library(gridExtra)
grid.arrange(iris_plot, iris_plot_gkm, iris_plot_km, iris_plot_gmm, ncol = 2)
```



- Based on the visualization above, there is still a mixture of three classes clustered by GKM, whereas, the others seem to be comparable, and GMM is the one identical to the true class.

Simulate data for Generalized K-means

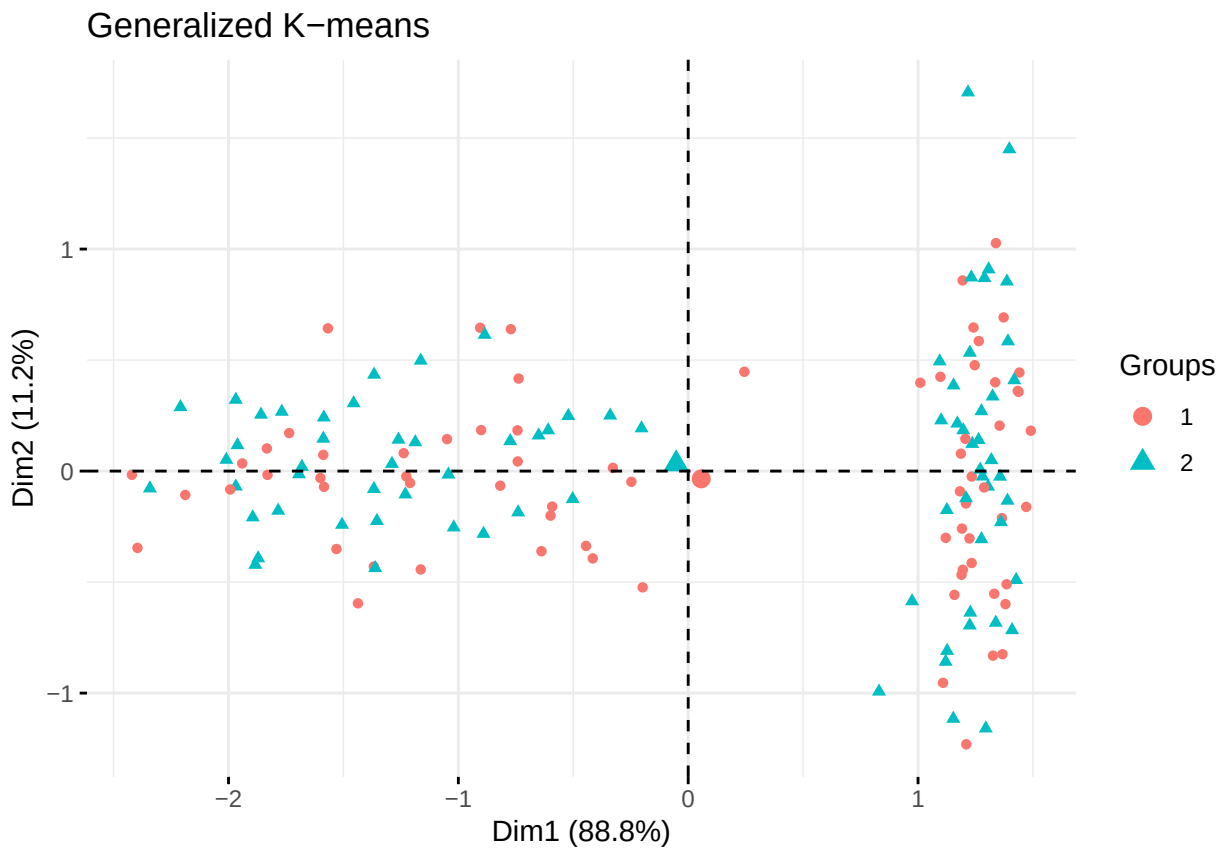
```
n <- 150
mu1 <- c(1, 1)
mu2 <- c(5, 5)
sigma1 <- matrix(c(1, 0.5, 0.5, 1), nrow = 2)
sigma2 <- matrix(c(1, -0.9, -0.9, 1), nrow = 2)

data1 <- mvrnorm(n/2, mu1, sigma1)
data2 <- mvrnorm(n/2, mu2, sigma2)
simulated_data <- rbind(data1, data2)
sim_pca <- PCA(simulated_data, ncp = 2, graph = FALSE)

# Apply generalized k-means
res_gkm <- generalized_kmeans(simulated_data, k = 2)
gkm_class <- apply(res_gkm, 1, function(row) which(row == 1))

iris_plot_gkm <- fviz_pca_ind(sim_pca, label = "none",
                              habillage = as.factor(gkm_class), title = "Generalized K-means")

iris_plot_gkm
```



- Visibly, the data is still yet to be well clustered.

Additional Question

In a p-dimensional space, a hyperplane is a flat affine subspace of hyperplane dimension p. To illustrate, in two dimensions, a hyperplan is a flat one-dimensional subspace, also known as, a line. Whereas, in three dimensions, a hyperplane is a flat two-dimensional subspace which is a plane.

For some reaons, it is difficult to visualize a hyperplane for any dimension p $>$ 3, however, with that being said, a hyperplane is a $p-1$-dimensional flat subspace in dimension p is still applies.

Consider a hyperplane D and a point $x \in \mathbb{R}^{\text{l}}$. Write the explicit formula and the associated computer code that gives:

- The coordinates of the orthogonal projection of x onto D

In any arbitrary p dimension, where $p \geq 2$, the projection of a point x=$(x_{1c}, x_{2c}, ..., x_{pc})$ on a hyperplane D: $a_1x_1 + a_2x_2 + ... + a_px_p + b = 0$ is given as:

$$x_{i_{proj}} = x_{ic} - d(x, D)\hat{a}_i, \quad i = \{1, .., n\}$$

Where $d(x, D)$ is the distance from point x to hyperplane D and

$$(\hat{a_1}, \hat{a_2}, ..., \hat{a_p}) = (\frac{a_1}{\sqrt{a_1^2 + a_2^2 + ... + a_p^2}}, \frac{a_2}{\sqrt{a_1^2 + a_2^2 + ... + a_p^2}}, ..., \frac{a_p}{\sqrt{a_1^2 + a_2^2 + ... + a_p^2}})$$

are unit vectors corresponding to coefficients $(a_1, a_2, \ldots, a_p)$ of hyperplane D

- The distance between x and D:

$$d(x, D) = \frac{|a_1 x_{1c} + a_2 x_{2c} + ... + a_p x_{pc} + b|}{\sqrt{a_1^2 + a_2^2 + ... + a_p^2}}$$

- GKM algorithm:

By following the steps as the algorithm above except the formula of distance, partially derive for each coefficient instead and setting them to zero to find the new coefficients of the hyperplane for step 3 (update the representative class)

$$d\frac{J(C, D)}{d(a_1, .., a_p)} = d\frac{\sum_i \sum_{k=1}^{K} c_{ik} \frac{(|a_1 x_{1c} + a_2 x_{2c} + ... + a_p x_{pc} + b|)^2}{a_1^2 + a_2^2 + ... + a_p^2}}{d(a_1, .., a_p)} = 0$$

$$d\frac{J(C, D)}{da_j} = \frac{\sum_i \sum_{k=1}^{K} -2 c_{ik} x_{jc} (|a_1 x_{1c} + a_2 x_{2c} + ... + a_p x_{pc} + b|)^2}{(a_1^2 + .. + a_p^2)^{3/2}} = 0 \quad , \quad j = 1, ..., p$$

```
#D is a c() contains all coefficients, x is a point
distance <- function(x, D){
  n = length(x)
  num = 0
  #coefficients a1,...,ap
  for (i in 1:n){
    num = num + (x[i] * D[i])
  }
  #add coef b
  num = num + D[n+1]
  #D[-(n+1)] remove the last element from D, there is no need for coef b
  den = sqrt(sum(D[-(n+1)]^2))
  return (num/den)
}

#D is a c() contains all coefficients, x is a point
projection <- function(x, D){
  coordinate = c()
  n = length(x)
  dis = distance(x, D)
  #normalize directional vector
  unit_vec = c()
  for (i in 1:n){
    unit_vec[i] = D[i]/sqrt(sum(D[-(n+1)]^2))
  }
  for (i in 1:n){
    coordinate[i] = x[i] - (dis * unit_vec[i])
  }
  return (coordinate)
}
```

```
#D is a list() where each element is a c() containing coefficients
#x is a point
closest <- function(x, D){
  K = length(D)
  set_of_distance = c()
```

```
  #store distance between x to each line
  for (k in 1:K){
    set_of_distance[k] = distance(x, D[[k]])
  }

  #get the smallest distance
  min_distance = min(set_of_distance)

  #handle case which there are many min values
  #store smallest index
  index_of_min = which(set_of_distance == min_distance)
  #randomly select if there are more than one minimum distance
  if (length(index_of_min) > 1){
    rand_index = sample(index_of_min, 1)
  } else {
    rand_index = index_of_min
  }
  #return index of a line in set D that is closest to x
  return (rand_index)
}
```

```
update_representative <- function(p, Dk){
  new_D <- list()
  for (i in 1:length(Dk)){
    # Define the objective function (negative of the distance function)
    objective_function <- function(a) {
      -distance(p, c(a[1:p], Dk[[i]][(p + 1):(2 * p)]))
    }

    # Initialize starting values for optimization
    initial_values <- rep(1, p)

    # Perform optimization
    opt_result <- optim(par = initial_values, fn = objective_function, method = "BFGS")

    # Extract the optimized coefficients
    new_a <- opt_result$par

    # Combine the optimized coefficients with the existing directional vector
    new_D[[i]] <- c(new_a, Dk[[i]][(p + 1):(2 * p)])
  }
  return (new_D)
}
```

```
generalized_kmeans <- function(X, k, max_iter = 100){
  X <- as.matrix(X)
  n <- nrow(X)
  p <- ncol(X)
  C <- matrix(0, nrow = n, ncol = k)

  S <- (t(X)%*%X)/n
  eig = eigen(S)
  Dk = list()
```

```
  for(i in 1:k){
    Dk[[i]] = c(rep(0,p),eig$vectors[i,])
  }

  Dk_new = list()
  V = list()
  O = list()
  for (iter in 1:max_iter) {
    # Step 2: Assign each data point to the closest line
    closest_line_to = sapply(1:n, function(i) closest(X[i,], Dk))

    # Cluster each data point x to the closest line among k lines
    for (i in 1:n) {
      C[i, ] = rep(0, k)
      C[i, closest_line_to[i]] = 1
    }

    # Step 3: Update representative class based on current cluster C
    Dk_new = update_representative(p, Dk)

    if (identical(Dk, Dk_new)){
      break
    }

    Dk = Dk_new

  }
  return (C)
}
```