RAJAOBELINA_RAMADANI_MAD_Projet_2023

Fitahiry RAJAOBELINA & Dorart RAMADANI

2023-12-17

Projection Orthogonale

Soit v un vecteur directeur de la droite $\mathcal{D}$, la projection orthogonale de $\mathbf{x} \in \mathbb{R}^p,\ p \in \mathbb{N}$ est donnée par:

$$\Pi_{\mathcal{D}}(\mathbf{x}) = \frac{\langle \mathbf{x}, v \rangle}{\langle v, v \rangle}.v$$

où $\langle ., . \rangle$ est le produit scalaire.

```
projection <- function(x, v) {
  return ((sum(x * v) / sum(v * v)) * v)
}
```

La distance entre $\mathbf{x}$ et $\mathcal{D}$ est:

$$d(\mathbf{x}, \mathcal{D}) = \|\mathbf{x} - \Pi_{\mathcal{D}}(\mathbf{x})\|$$

```
dist <- function(x, v) {
  return (sqrt(sum(x - projection(x,v))^2))
}
```

Soient K droites $\mathcal{D}_1, ..., \mathcal{D}_K$ et $\mathbf{x} \in \mathbb{R}^p$

```
closest_dist <- function(x, D) {
  # D: liste des vecteurs directeurs

  # Calcul des distances entre x et chaque droite
  distances <- sapply(D, function(d) dist(x, d))

  # Trouver l'indice de la droite la plus proche
  closest_indices <- which.min(distances)

  # En cas d'égalité de distances, sélection aléatoire
  if (length(closest_indices) > 1) {
    closest_index <- sample(closest_indices, 1)
  }
  else {
    closest_index <- closest_indices
  }

  # La droite la plus proche et sa distance à x
  closest_distance <- distances[closest_index]
```

```
  return (closest_distance)
}
```

Les nuées dynamiques ou les kmeans généralisées

Objectif

Soit $C = (c_{ik})$ un matrice de classification et $D = \{\mathcal{D}_1, ..., \mathcal{D}_K\}$

Le but est de minimiser le critère suivant:

$$J(C, D) = \sum_i \sum_{\mathcal{D}_k \in D} c_{ik} d(\mathbf{x}_i, \mathcal{D}_k)$$

avec d la distance entre un point et une droite.

On cherche alors $\hat{C}$ et $\hat{D}$ tel que:

$$(\hat{C}, \hat{D}) = argmin_{(C,D)} J(C, D)$$

Pour cela, on s'inspire de l'algorithme des K-means, l'objectif étant de regrouper les données en K clusters en minimisant la distance entre chaque point de données et la droite représentant son cluster.

Étapes de l'algorithme

Initialisation : Définir le nombre de clusters K. Initialiser aléatoirement les droites représentant les classes. Chaque droite est définie par ses coefficients (m, b) (pour une droite $y = mx + b$).

Itérations : L'algorithme effectue les étapes suivantes tant que le nombre d'itérations n'atteind pas max_iter :

Classification :

Pour chaque point de données, on calcule la distance entre ce point et chaque droite représentant les clusters. Puis on affecte chaque point au cluster avec la droite la plus proche (plus petite distance).

Mise à jour des droites :

Pour chaque cluster : On sélectionne les points appartenant à ce cluster. On vérifie si le cluster contient suffisamment de points pour être mis à jour (pas vide et contenant plus d'un point). Si le cluster remplit les conditions : On calcule une nouvelle droite en utilisant une régression linéaire simple avec les points du cluster pour obtenir les coefficients de la droite (m, b). Si le cluster est vide ou ne contient qu'un seul point, on génère de nouvelles droites aléatoires.

Résultats :

Les résultats finaux retournent les clusters attribués à chaque point de données et les coefficients des droites représentant chaque cluster.

Implémentation

On décide de prendre Sepal.Length et Sepal.Width dans le jeu de données iris.

```
# Chargement du jeu de données Iris
data(iris)

# Sélection des deux premières colonnes (longueur et largeur des sépales)
iris_data <- iris[, c("Sepal.Length", "Sepal.Width")]
```

```r
# Normalisation des données
iris_data <- scale(iris_data)
```

```r
# Fonction pour l'algorithme des nuées dynamiques avec droites comme représentants de classe
kmeans_generalized <- function(data, K, max_iter = 100) {
  lines <- matrix(runif(K * 2), ncol = 2)  # Initialisation aléatoire des droites représentant les clas

  for (iter in 1:max_iter) {
    clusters <- rep(0, nrow(data))  # Classification des points selon les droites actuelles

    for (i in 1:nrow(data)) {
      min_distance <- Inf
      for (k in 1:K) {
        dist_to_line <- abs(data[i, 2] - lines[k, 1] * data[i, 1] - lines[k, 2]) / sqrt(1 + lines[k, 1]
        if (dist_to_line < min_distance) {
          min_distance <- dist_to_line
          clusters[i] <- k
        }
      }
    }

    for (k in 1:K) {
      cluster_points <- data[clusters == k,]  # Sélection des points appartenant au cluster k
      if (nrow(cluster_points) > 0) {
        line_fit <- lm(cluster_points[,2] ~ cluster_points[,1])  # Régression linéaire pour mettre à jo
        lines[k,] <- c(coef(line_fit)[2], coef(line_fit)[1])
      } else {
        lines[k,] <- matrix(runif(2), ncol = 2)  # Génération de nouvelles droites aléatoires si le clu
      }
    }
  }

  return(list(clusters = clusters, lines = lines))
}

# Application

K <- 3  # Nombre de clusters
result_iris <- kmeans_generalized(iris_data, K)

# Affichage des clusters
print(result_iris$clusters)
```

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

```r
# Affichage des droites représentant les classes
print(result_iris$lines)
```

```
##           [,1]       [,2]
## [1,] 1.3308178  2.3041342
```

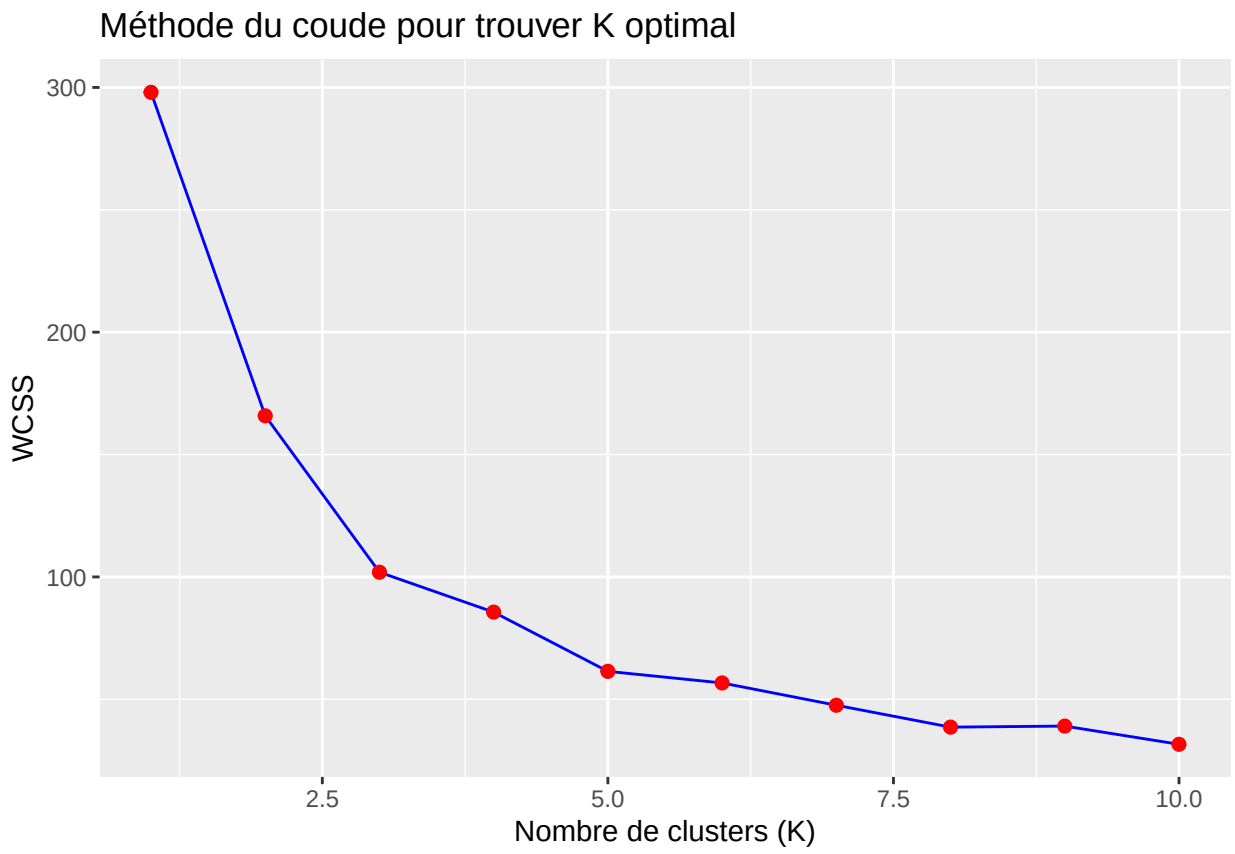
```
## [2,] 0.3809262 -1.1520617
## [3,] 0.3980220 -0.1196231
```

Choix de K

Cette section vise à choisir le K optimal.

Pour procéder, on décide d'utiliser la méthode du coude.

```r
library(ggplot2)

# Fonction pour calculer la somme des carrés des distances intra-cluster (WCSS)
calculate_wcss <- function(data, k) {
  wcss <- numeric(length = k)
  for (i in 1:k) {
    kmeans_result <- kmeans(data, centers = i)
    wcss[i] <- kmeans_result$tot.withinss
  }
  return(wcss)
}

# Calcul de WCSS pour différentes valeurs de K
max_k <- 10
wcss_values <- calculate_wcss(iris_data, max_k)

# Tracer le graphique du coude
elbow_plot <- ggplot() +
  geom_line(aes(x = 1:max_k, y = wcss_values), color = "blue") +
  geom_point(aes(x = 1:max_k, y = wcss_values), color = "red", size = 2) +
  labs(x = "Nombre de clusters (K)", y = "WCSS") +
  ggtitle("Méthode du coude pour trouver K optimal")

# Afficher le graphique
print(elbow_plot)
```



On remarque que le "coude" semble atteint en $K = 3$, donc $K_{opt} = 3$

Comparaison avec K-mean et l'algorithme de mélanges avec des gaussiennes

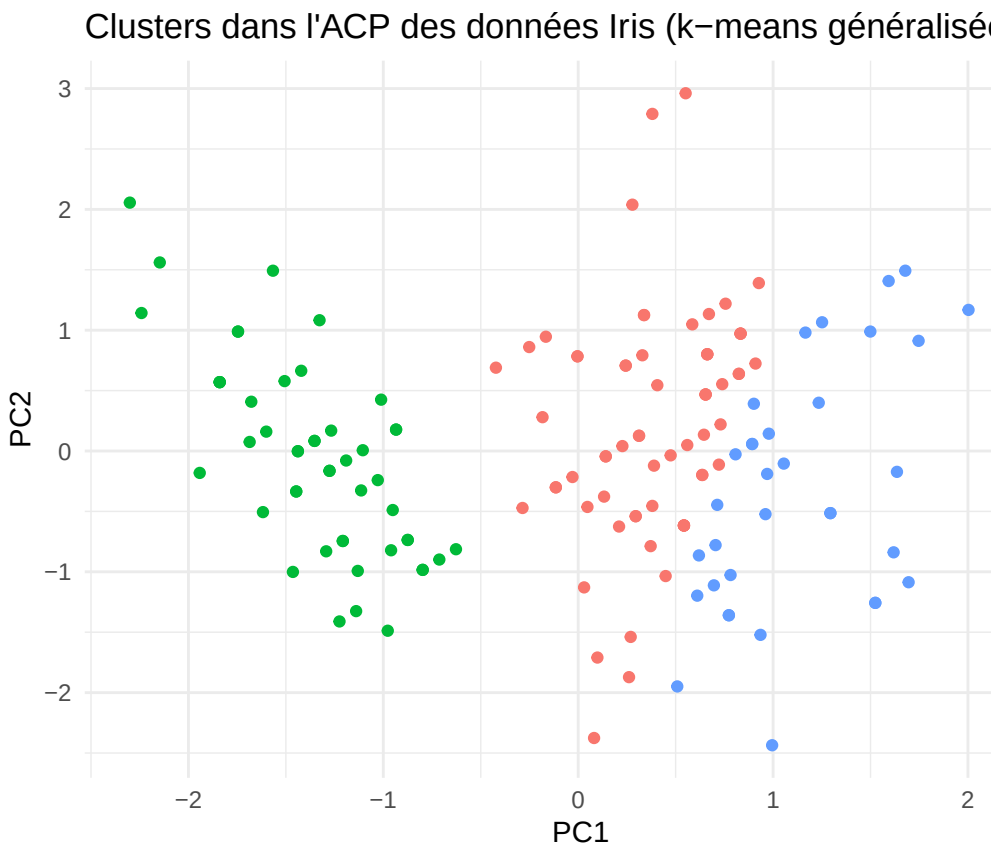
On effectue tout d'abord une ACP sur les jeux de données

```
# Effectuer l'ACP sur les données Iris
iris_pca <- prcomp(iris_data, scale. = TRUE)

# Obtenir les composantes principales
iris_components <- iris_pca$x[, 1:2]  # Utilisation des deux premières composantes principales
```

```
# Appliquer l'algorithme des k-means généralisée
K <- 3
result_iris <- kmeans_generalized(iris_data, K)
clusters <- result_iris$clusters
```

```
library(ggplot2)

# Créer un data frame pour la visualisation
iris_vis <- data.frame(PC1 = iris_components[, 1], PC2 = iris_components[, 2], Cluster = as.factor(clus

# Plot
ggplot(iris_vis, aes(x = PC1, y = PC2, color = Cluster)) +
  geom_point() +
```

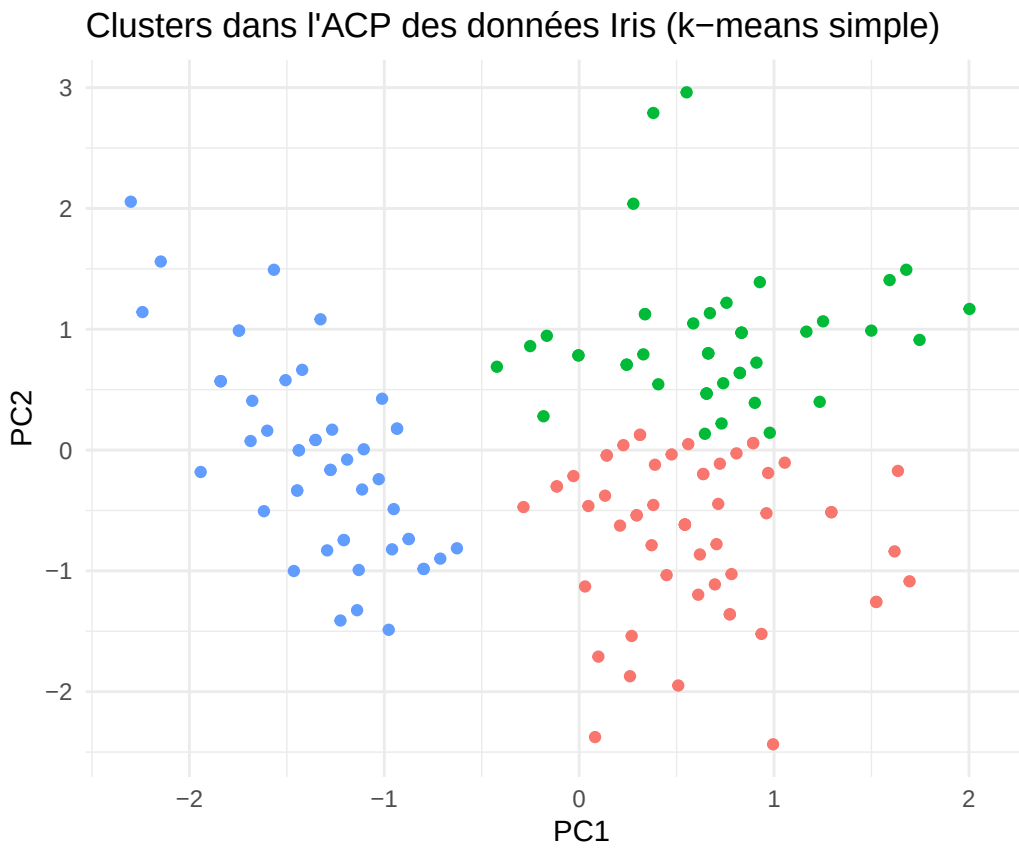
```
  labs(title = "Clusters dans l'ACP des données Iris (k-means généralisée)") +
  theme_minimal()
```



Visualisation Kmean généralisée

```
# Appliquer l'algorithme des k-means standards
K <- 3
result_kmeans <- kmeans(iris_data, centers = K)
clusters_kmeans <- result_kmeans$cluster
```

```
library(ggplot2)

# Créer un data frame pour la visualisation
iris_vis_kmeans <- data.frame(PC1 = iris_components[, 1], PC2 = iris_components[, 2], Cluster = as.facto

# Plot
ggplot(iris_vis_kmeans, aes(x = PC1, y = PC2, color = Cluster)) +
  geom_point() +
  labs(title = "Clusters dans l'ACP des données Iris (k-means simple)") +
  theme_minimal()
```



Visualisation Kmean standard

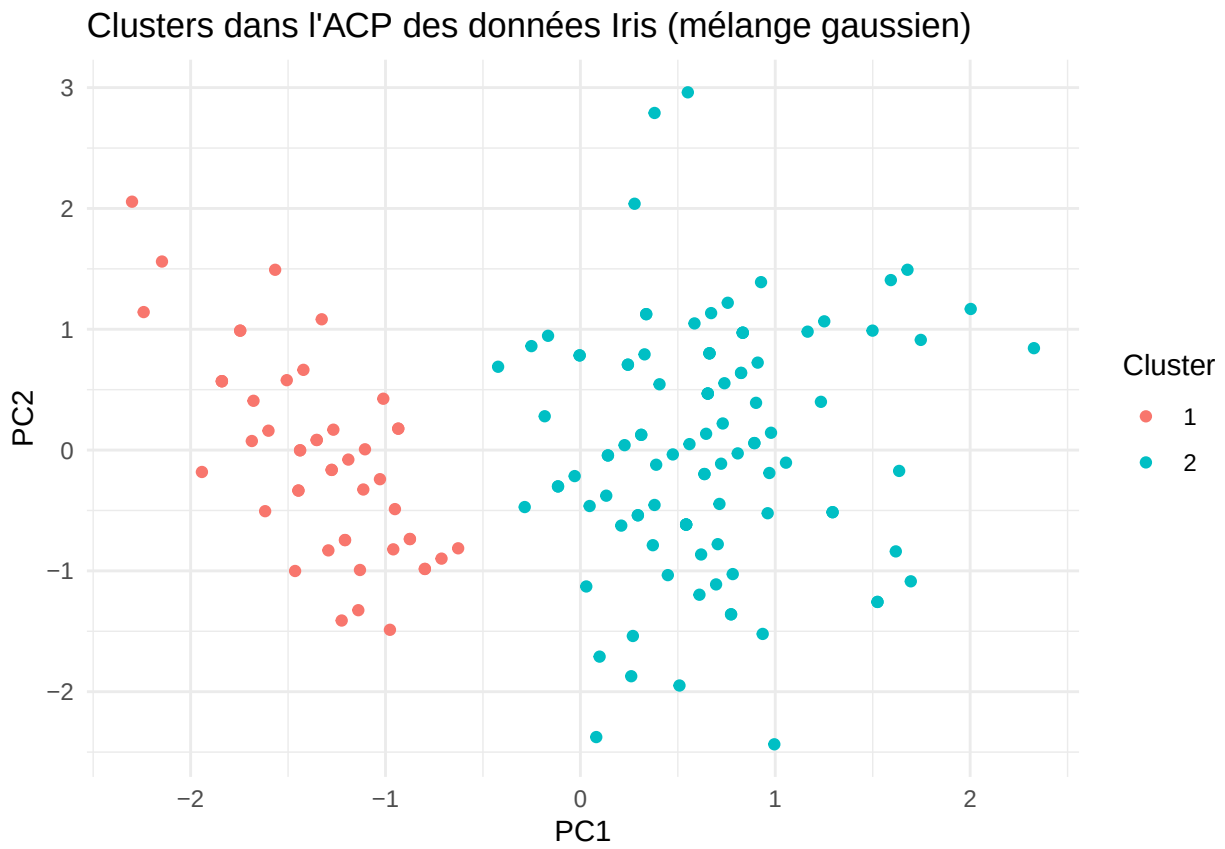
```
# Appliquer l'algorithme des mélanges gaussiens
library(mclust)
```

Visualisation pour mélange de gaussiennes

```
## Package 'mclust' version 6.0.0
## Type 'citation("mclust")' for citing this R package in publications.
```

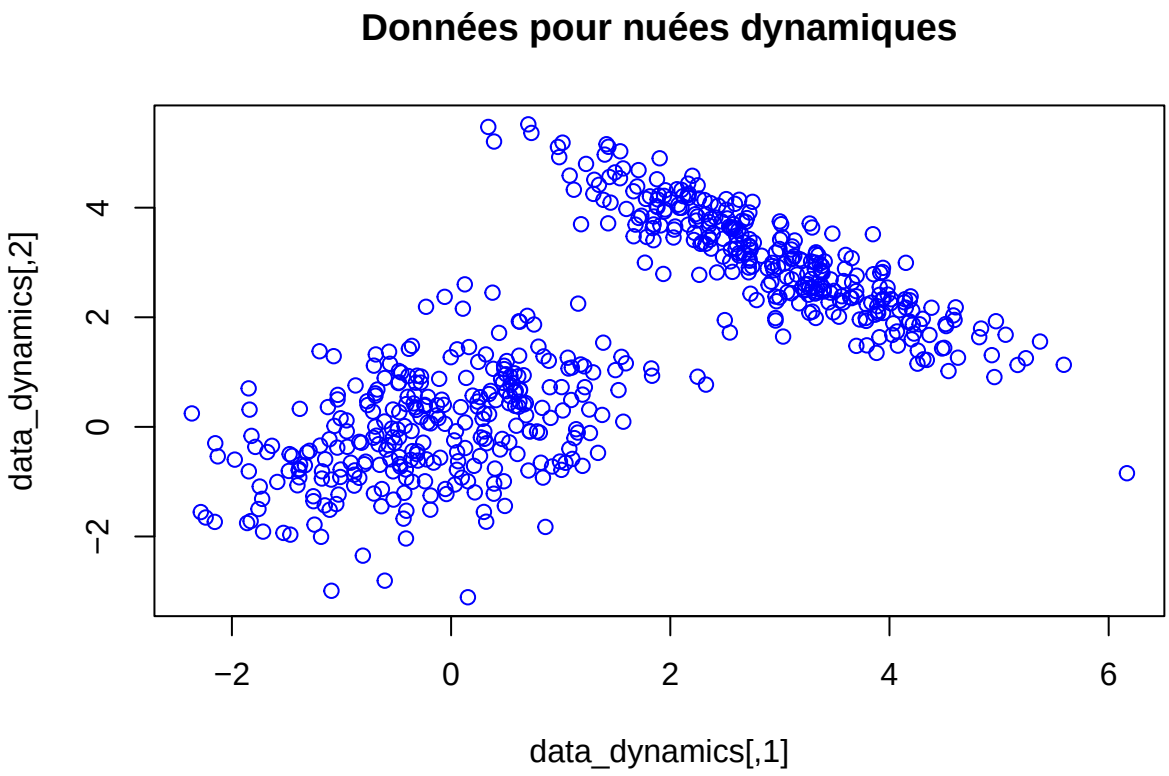
```
iris_mclust <- Mclust(iris_data)
clusters_mclust <- iris_mclust$classification
```

```
library(ggplot2)

# Créer un data frame pour la visualisation
iris_vis_mclust <- data.frame(PC1 = iris_components[, 1], PC2 = iris_components[, 2], Cluster = as.facto

# Plot
ggplot(iris_vis_mclust, aes(x = PC1, y = PC2, color = Cluster)) +
  geom_point() +
  labs(title = "Clusters dans l'ACP des données Iris (mélange gaussien)") +
  theme_minimal()
```

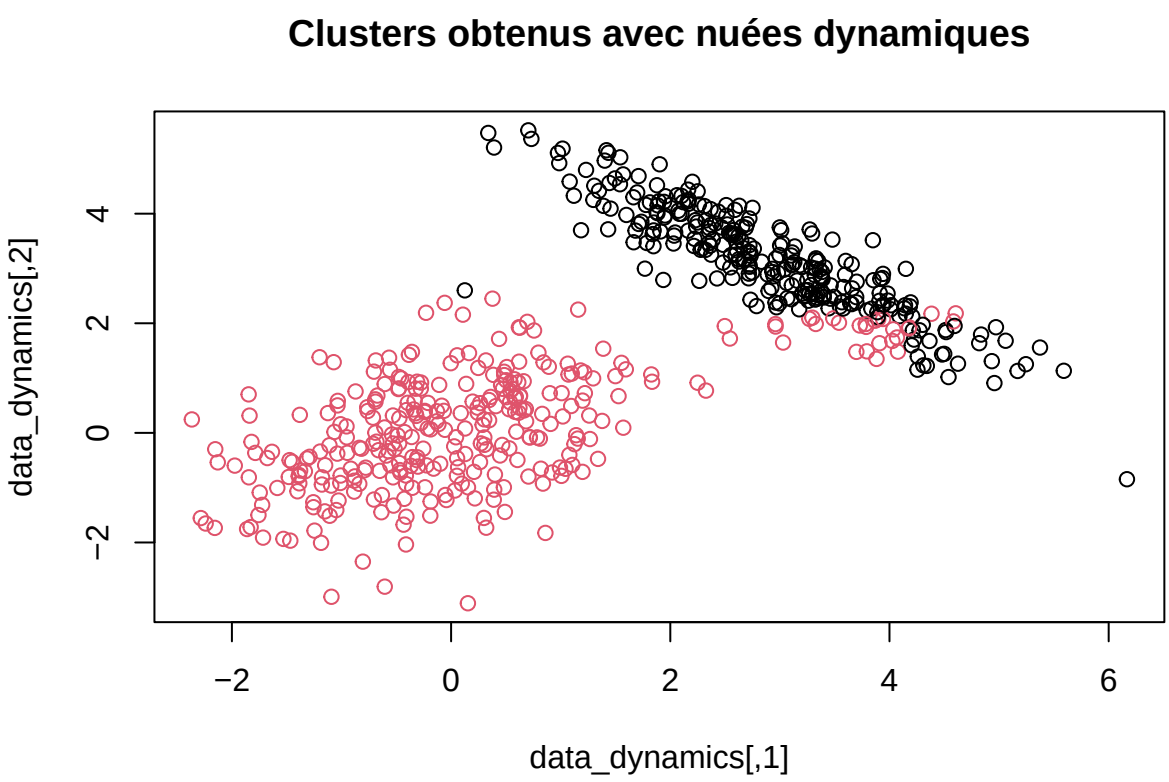


Données plus adaptées à l'algorithme des K-means généralisé

```
library(MASS)

# Génération de données
data_dynamics <- mvrnorm(300, mu = c(0, 0), Sigma = matrix(c(1, 0.5, 0.5, 1), nrow = 2))
data_dynamics <- rbind(data_dynamics, mvrnorm(300, mu = c(3, 3), Sigma = matrix(c(1, -0.9, -0.9, 1), nr


# Visualisation
plot(data_dynamics, col = 'blue', main = "Données pour nuées dynamiques")
```




```r
# Appliquer l'algorithme des nuées dynamiques révisé sur les données simulées pour les nuées dynamiques
result_dynamics <- kmeans_generalized(data_dynamics, K = 2)

# Visualiser les clusters obtenus
plot(data_dynamics, col = result_dynamics$clusters, main = "Clusters obtenus avec nuées dynamiques")
```



On remarque que notre algorithme classifie efficacement les points lorsques ces derniers forment des nuées, les points semblent en effet se regrouper tout autour d'une droite.

Commentaires

Les nuées dynamiques présentent plusieurs avantages.

Flexibilité de la forme des clusters :

Contrairement au k-means qui suppose des clusters de forme plus ou moin sphérique, les nuées dynamiques peuvent modéliser des clusters de formes arbitraires, linéaires ou non linéaires. L'agorithme semble être plus adapté pour des données présentant des clusters complexes et de formes variées.

Réduction des biais géométriques :

Les nuées dynamiques sont moins susceptibles d'être influencées par la géométrie des données que le k-means. Elles ne supposent pas de formes géométriques spécifiques pour les clusters, ce qui peut être bénéfique lorsque les clusters ne sont pas bien délimités géométriquement.

Robustesse aux valeurs aberrantes :

Par ailleurs, les nuées dynamiques peuvent être plus robustes aux valeurs aberrantes comparées au k-means, car elles utilisent la distance des points à une représentation de classe plutôt que la moyenne, qui peut être sensible aux valeurs extrêmes.

Cependant, l'algorithme présente quelques inconvénients.

Complexité:

Il est plus complexe sur le plan computationnel que le k-means, ce qui ne s'améliore pas avec de grands ensembles de données. La recherche des représentants de classe peut être coûteuse en termes de temps de

calcul.

Sensibilité importante:

Tout au long du projet, nous avons rencontrés plusieurs difficultés, dont l'une d'elles étaient que la fonction "kmeans_generalized" donnait, une fois sur deux, des erreurs. Et ceci, en gardant les mêmes données et le même code.

Conclusion

En résumé, les nuées dynamiques offrent une approche plus souple et adaptable pour détecter des structures complexes dans les données, surtout lorsque les clusters ne sont pas de forme régulière. Cependant, l'implémentation de l'algorithme est plus coûteuses en termes de complexité et de temps de calculs.