Projet CA

TANIEL Rémi - GIS2A4

Contents

1 Introduction

Le but de ce projet est de comparer les différentes méthodes de partitionnement vu en cours sur un jeu de données `cameras.csv`, pour chacune des méthodes, nous allons déterminer le nombre de classes / clusters optimal à séléctionner puis calculer l'inertie intraclasse, interclasse et la part d'inertie expliquée afin de comparer ces méthodes entres elles.

Nous allons avoir besoins des packages suivants pour le traitement et l'analyse de nos données :

```
library(factoextra)
library(cluster)
library(knitr)
```

2 Analyse préalable des données

On commence par charger notre jeu de données et de visualiser les premières entrées de notre jeu de données:

```
data <- read.csv('Camera.csv', sep = ';')
kable(head(data))
```

Model	Release.date	Max.resolution	Low.resolution	Effective.pixels	Zoom.wide	Zoom.tele	Nor
Agfa ePhoto 1280	1997	1024	640	0	38	114	
Agfa ePhoto 1680	1998	1280	640	1	38	114	
Agfa ePhoto CL18	2000	640	0	0	45	45	
Agfa ePhoto CL30	1999	1152	640	0	35	35	
Agfa ePhoto CL30 Clik!	1999	1152	640	0	43	43	
Agfa ePhoto CL45	2001	1600	640	1	51	51	

On décide ensuite de visualiser les différentes variables de notre jeu de données, on utilise donc la fonction suivante :

```
str(data)
```

```
## 'data.frame':    1038 obs. of  13 variables:
##  $ Model             : chr  "Agfa ePhoto 1280" "Agfa ePhoto 1680" "Agfa ePhoto CL18" "Agfa ePhoto CL
##  $ Release.date      : int  1997 1998 2000 1999 1999 2001 1999 1997 1996 2001 ...
##  $ Max.resolution    : num  1024 1280 640 1152 1152 ...
##  $ Low.resolution    : num  640 640 0 640 640 ...
##  $ Effective.pixels  : num  0 1 0 0 0 1 1 0 0 1 ...
##  $ Zoom.wide         : num  38 38 45 35 43 51 34 42 50 35 ...
##  $ Zoom.tele         : num  114 114 45 35 43 51 102 42 50 105 ...
##  $ Normal.focus.range: num  70 50 0 0 50 50 0 70 40 76 ...
##  $ Macro.focus.range : num  40 0 0 0 0 20 0 3 10 16 ...
##  $ Storage.included  : num  4 4 2 4 40 8 8 2 1 8 ...
##  $ Weight            : num  420 420 0 0 300 270 0 320 460 375 ...
##  $ Dimensions        : num  95 158 0 0 128 119 0 93 160 110 ...
##  $ Price             : num  179 179 179 269 1299 ...
```

Notre jeu de données contient donc 1038 observations de 13 variables, on remarque que toutes ces variables sont quantitatives et que la variable `Model` représente le nom de chaque observation, nous pouvons donc l'enlever de notre jeu de données pour nommer nos observations :

```
row.names(data) <- data[,1]
data <- data[,-1]
```

On voudrait ensuite savoir si notre jeu de données contient des données manquantes, pour cela on utilise la fonction `summary` de R:

```
summary(data)
```

```
##   Release.date  Max.resolution Low.resolution Effective.pixels   Zoom.wide
##  Min.   :1994   Min.   :   0   Min.   :   0   Min.   : 0.000   Min.   : 0.00
##  1st Qu.:2002   1st Qu.:2048   1st Qu.:1120   1st Qu.: 3.000   1st Qu.:35.00
```

```
##  Median :2004   Median :2560   Median :2048   Median : 4.000   Median :36.00
##  Mean   :2004   Mean   :2475   Mean   :1774   Mean   : 4.596   Mean   :32.96
##  3rd Qu.:2006   3rd Qu.:3072   3rd Qu.:2560   3rd Qu.: 7.000   3rd Qu.:38.00
##  Max.   :2007   Max.   :5616   Max.   :4992   Max.   :21.000   Max.   :52.00
##
##    Zoom.tele     Normal.focus.range Macro.focus.range Storage.included
##  Min.   :  0.0   Min.   :  0.00     Min.   : 0.000    Min.   :  0.00
##  1st Qu.: 96.0   1st Qu.: 30.00     1st Qu.: 3.000    1st Qu.:  8.00
##  Median :108.0   Median : 50.00     Median : 6.000    Median : 16.00
##  Mean   :121.5   Mean   : 44.15     Mean   : 7.788    Mean   : 17.45
##  3rd Qu.:117.0   3rd Qu.: 60.00     3rd Qu.:10.000    3rd Qu.: 20.00
##  Max.   :518.0   Max.   :120.00     Max.   :85.000    Max.   :450.00
##                                     NA's   :1         NA's   :2
##      Weight         Dimensions        Price
##  Min.   :   0.0   Min.   :  0.0   Min.   :  14.0
##  1st Qu.: 180.0   1st Qu.: 92.0   1st Qu.: 149.0
##  Median : 226.0   Median :101.0   Median : 199.0
##  Mean   : 319.3   Mean   :105.4   Mean   : 457.4
##  3rd Qu.: 350.0   3rd Qu.:115.0   3rd Qu.: 399.0
##  Max.   :1860.0   Max.   :240.0   Max.   :7999.0
##  NA's   :2        NA's   :2
```

On remarque que notre jeu de données contient quelques données manquantes dans les variables `Macro.focus.range` ou `Weight` par exemple. On décide donc d'enlever les observations pour lesquelles une variable est manquante :

```
data_w_na <- na.omit(data)
dim(data_w_na)
```

```
## [1] 1036   12
```

Seules 2 observations ont été supprimé de notre jeu de données.

Avant d'employer la méthode k-means, nous devons standardiser notre jeu de données, c'est à dire les centrer et reduire :

```
data_sc <- scale(data_w_na)
kable(head(data_sc))
```

	Release.date	Max.resolution	Low.resolution	Effective.pixels	Zoom.wide	Zoom.tele	No
Agfa ePhoto 1280	-2.4171211	-1.908233	-1.368157	-1.614496	0.4877952	-0.0806507	
Agfa ePhoto 1680	-2.0500532	-1.571119	-1.368157	-1.262736	0.4877952	-0.0806507	
Agfa ePhoto CL18	-1.3159173	-2.413905	-2.139724	-1.614496	1.1646973	-0.8182699	
Agfa ePhoto CL30	-1.6829852	-1.739676	-1.368157	-1.614496	0.1976942	-0.9251712	
Agfa ePhoto CL30 Clik!	-1.6829852	-1.739676	-1.368157	-1.614496	0.9712967	-0.8396502	
Agfa ePhoto CL45	-0.9488494	-1.149725	-1.368157	-1.262736	1.7448991	-0.7541291	

3 Implémentation de l'algorithme k-means

3.1 Principe

Le principe de l'algorithme que nous voulons implémenté est le suivant :

- Choisir `k` observations au hasard parmis le jeu de données qui seront considéres comme les centres

initiaux des clusters

- Pour chaque observation, calculer la distance entre tous les centres des clusters et ajouter cette observation au cluster qui a son centre le plus proche, calculer le nouveau centre du cluster séléctionné
- Recommencer à partir du point précédent jusqu'à avoir une stabilisation des centres des clusters

3.2 Implémentation

```
dist_eucli <- function(x, y) {
  return(sqrt(sum((x - y) ** 2)))
}

get_cluster_index <- function(point, centers) {
  # calcul des distances entre chaque centre
  distances <- c()
  for (j in 1:nrow(centers)) {
    distances <- c(distances, dist_eucli(point, centers[j,]))
  }
  # selection de l'index de la plus petite distance
  return(match(min(distances), distances))
}

calculate_center_of <- function(cluster) {
  if (!is.null(dim(cluster))) {
    return(apply(cluster, 2, mean))
  } else {
    return(cluster)
  }
}

cluster_assignment <- function(data, centers) {
  clusters <- c()

  # pour chaque observation
  for (i in 1:nrow(data)) {
    # index du cluster dans lequel assigné l'observation
    index <- get_cluster_index(data[i,], centers)
    # ajout de l'index
    clusters <- c(clusters, index)
    # recalcul du centre du cluster
    clusters_points <- data[which(clusters == index, arr.ind = TRUE),]
    centers[index,] <- calculate_center_of(clusters_points)
  }

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

my_kmeans <- function(data, k, max_iteration=20) {
  i <- 0
  # étape 0
  centers <- data[sample(1:nrow(data), k, replace = F),]
  old_centers <- centers
```

```
  result <- cluster_assignment(data, centers)
  centers <- result$centers
  clusters <- result$clusters
  # étape 1 à n
  while(i < max_iteration && old_centers != centers) {
    old_centers <- centers
    result <- cluster_assignment(data, centers)
    centers <- result$centers
    clusters <- result$clusters
    i <- i + 1
  }
  return(list(centers=centers, clusters=clusters, iteration=i, data=data))
}
```

3.3 Calcul d'inertie

Maintenant que nous avons notre répartition on peut calculer l'inertie totale, intraclasse, interclasse et expliquée par notre répartition :

```
calculate_inertie_total_of <- function (points) {
  n <- nrow(points)
  barycentre <- calculate_center_of(points)
  distances <- c()

  for (i in 1:nrow(points)) {
    distances <- c(distances, dist_eucli(barycentre, points[i,]) ** 2)
  }

  return(sum(distances) / n)
}

calculate_inertie_inter <- function (my_kmeans_result) {
  centers <- my_kmeans_result$centers
  clusters <- my_kmeans_result$clusters
  barycentre <- calculate_center_of(my_kmeans_result$data)
  n <- length(clusters)
  distances <- c()

  for(i in 1:nrow(centers)) {
    indexes <- which(clusters == i)
    nr <- length(indexes)
    distances <- c(distances, nr * (dist_eucli(barycentre, centers[i,]) ** 2))
  }

  return(sum(distances) / n)
}

calculate_inertie_intra <- function (my_kmeans_result) {
  clusters <- my_kmeans_result$clusters
  n <- length(clusters)
  inerties <- c()

  for (i in 1:nrow(my_kmeans_result$centers)) {
```

```
    indexes <- which(clusters == i)
    nr <- length(indexes)
    inerties <- c(inerties, nr * calculate_inertie_total_of(my_kmeans_result$data[indexes,]))
  }

  return(sum(inerties) / n)
}

calculate_inertie <- function(my_kmeans_result) {
  total <- calculate_inertie_total_of(my_kmeans_result$data)
  inter <- calculate_inertie_inter(my_kmeans_result)
  intra <- calculate_inertie_intra(my_kmeans_result)

  expli <- 100 * (1 - intra / total)

  return(list(total=total, inter=inter, intra=intra, expli=expli))
}
```

4 Application

Nous allons maintenant appliquer la méthode que nous avons implémenté sur notre jeu de données

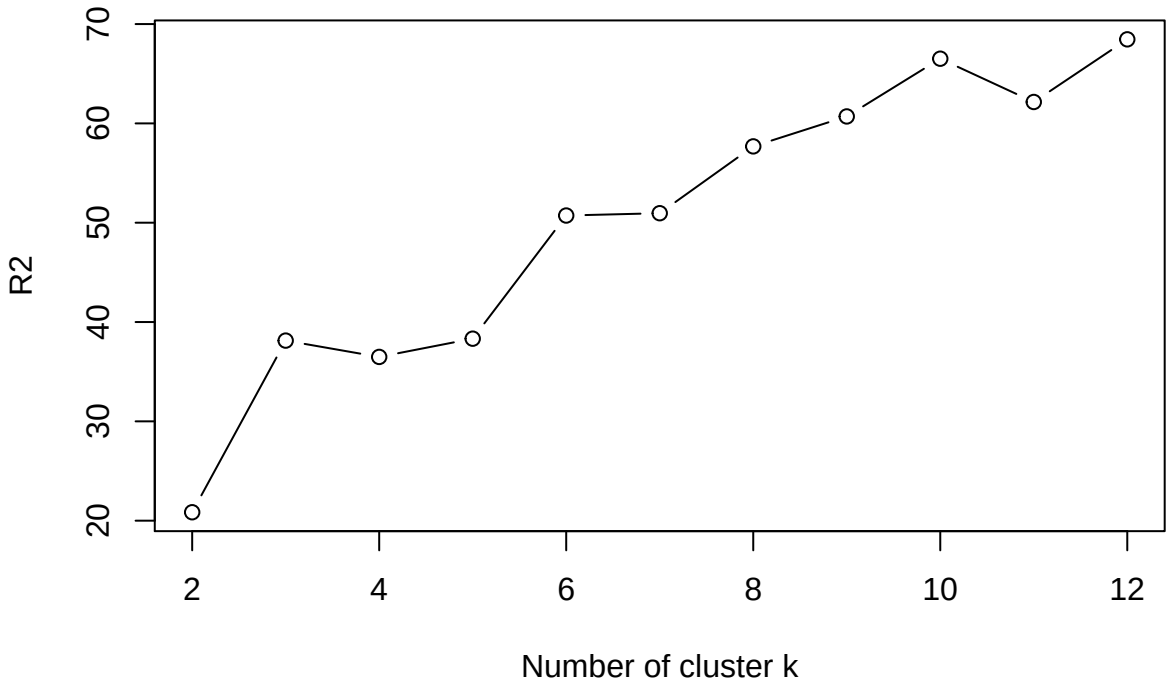
4.1 Choix du nombre de clusters

On détermine dans un premier temps le nombre de clusters optimales à séléctionner, pour cela, nous allons utiliser la méthode de la base du R2

```
res <- c()
n_col <- ncol(data_sc)

for (i in 2:n_col) {
  m <- my_kmeans(data_sc, i)
  i <- calculate_inertie(m)
  res <- c(res, i$expli)
}

plot(2:n_col, res, type = 'b', xlab = 'Number of cluster k', ylab = 'R2')
```



Le coude apparaissant à `k=3`, le nombre de clusters à séléctionner est donc de 3:

```
k <- 3
```

4.2 Application de la méthode implémentée

On applique donc la méthode précedemment implementée avec un nombre de clusters de 3 :

```
kmeans.1 <- my_kmeans(data_sc, k)
```

Et on calcule l'inertie intraclasse, interclasse et expliquée de cette partition :

```
inerties <- calculate_inertie(kmeans.1)
print(inerties)
```

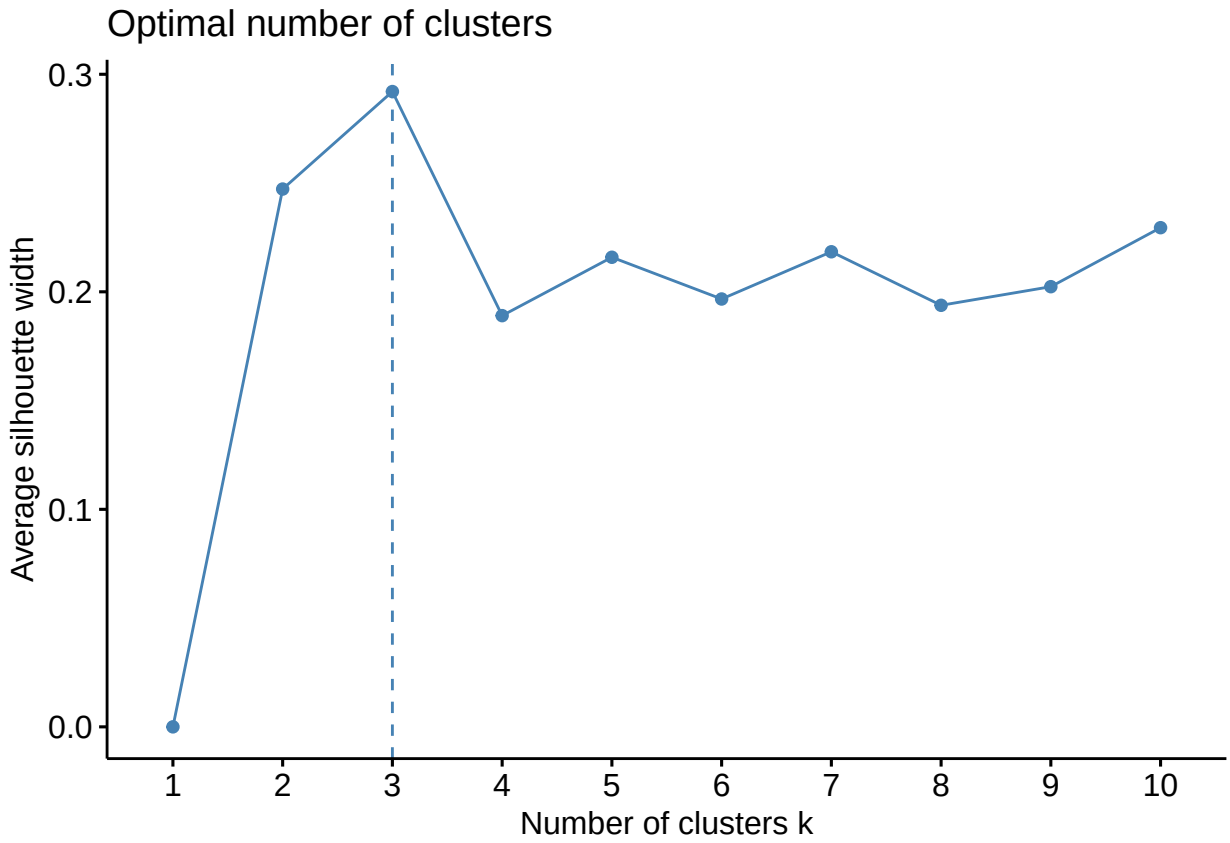
```
## $total
## [1] 11.98842
##
## $inter
## [1] 4.571062
##
## $intra
## [1] 7.417355
##
## $expli
## [1] 38.12899
```

5 Application de la méthode native et comparaison

5.1 Choix du nombre de clusters

Pour le chosisir le nombre de clusters optimales, nous allons utiliser une autre méthode que celle vu précédemment, nous allons utiliser la méthode de la silhouette :

```
fviz_nbclust(data_sc, kmeans, method = 'silhouette')
```



Comme précédemment le nombre de clusters `k` à sélectionner est de :

```
k <- 3
```

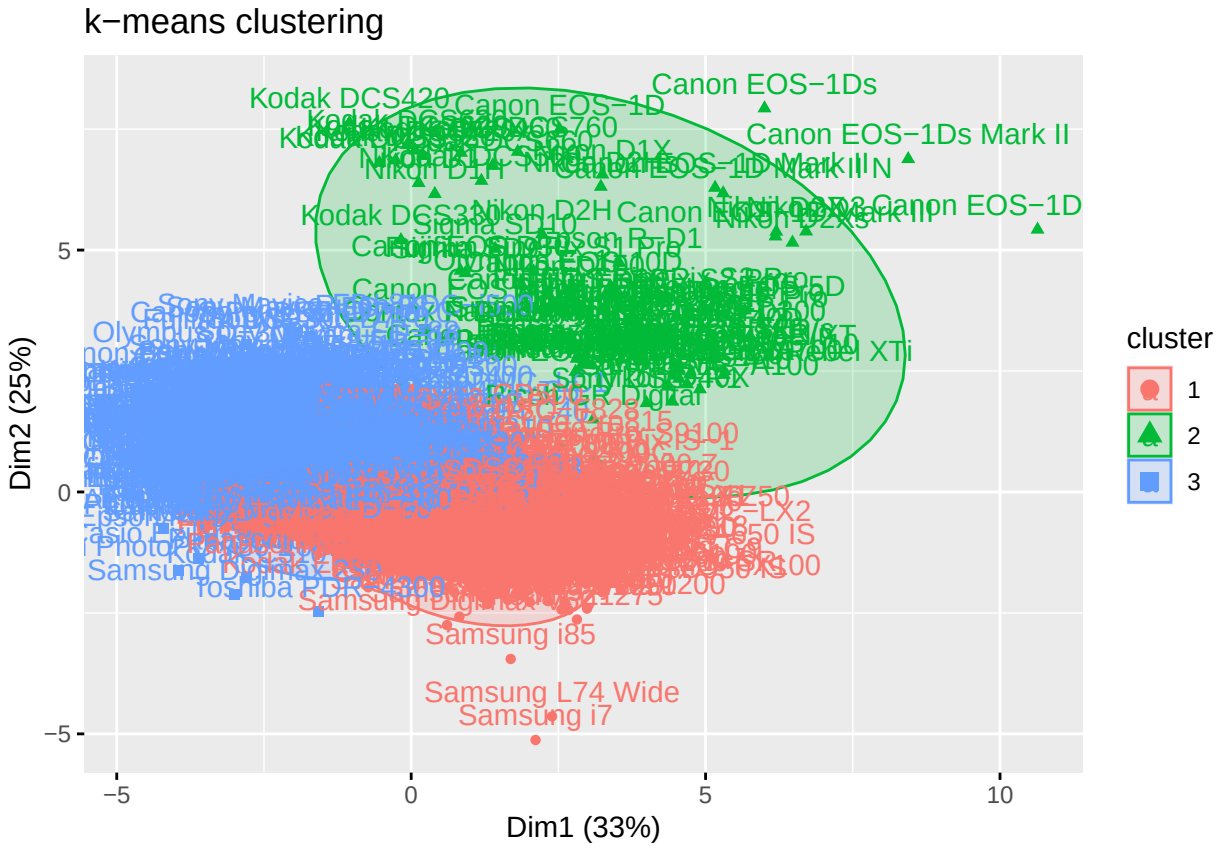
5.2 Calcul

On génère donc une partition avec la méthode native de R :

```
kmeans.2 <- kmeans(data_sc, centers = k)
```

On obtient alors la segmentation suivante :

```
fviz_cluster(kmeans.2, data_sc, ellipse.type = 'norm') + ggtitle('k-means clustering')
```



Et calcule les différentes valeurs des inerties de cette partition :

```
inerties <- calculate_inertie(list(data=data_sc, centers=kmeans.2$centers, clusters=kmeans.2$cluster))
print(inerties)
```

```
## $total
## [1] 11.98842
##
## $inter
## [1] 4.990722
##
## $intra
## [1] 6.997695
##
## $expli
## [1] 41.62954
```

5.3 Comparaison

6 Autres méthodes de segmentation

Ils existent d'autres méthodes de segmentation dites `k-medoids` comme les méthodes PAM ou CLARA, on décide donc pour chacune de ces méthodes de les présenter avec leurs avantages / inconvénients puis d'appliquer ces méthodes à notre jeu de données `data_sc`

6.1 Présentation des méthodes

6.1.1 PAM

La méthode PAM (Partition Around Medoids) est une méthode similaire au k-mean mais qui au lieu de se baser sur les barycentres des clusters se base sur les médoides (ou points centrales), c'est à dire que par rapport à l'algorithme des k-means qui peut définir un centre qui n'est pas un des points, la méthode PAM choisit une observation en tant que point central, elle cherche donc à minimiser l'erreur quadratique moyenne.

L'avantage de cette méthode est sa meilleure résistance aux valeurs abbérantes et son principal inconvénient est sa compléxité et donc son temps de calcul qui est relativement long

6.1.2 CLARA

La méthode CLARA (Clustering LARge Application), par rapport aux autres méthodes vus précédemments ne cherche pas à trouver des centres pour tout le jeu de données, considère déjà un petit échantillon sur lequel on va appliquer la méthode PAM afin de générer les meilleurs médoides pour notre échantillon, on va ensuite recommencer ce processus un nombre pre-défini de fois dans le but de minimiser le biais

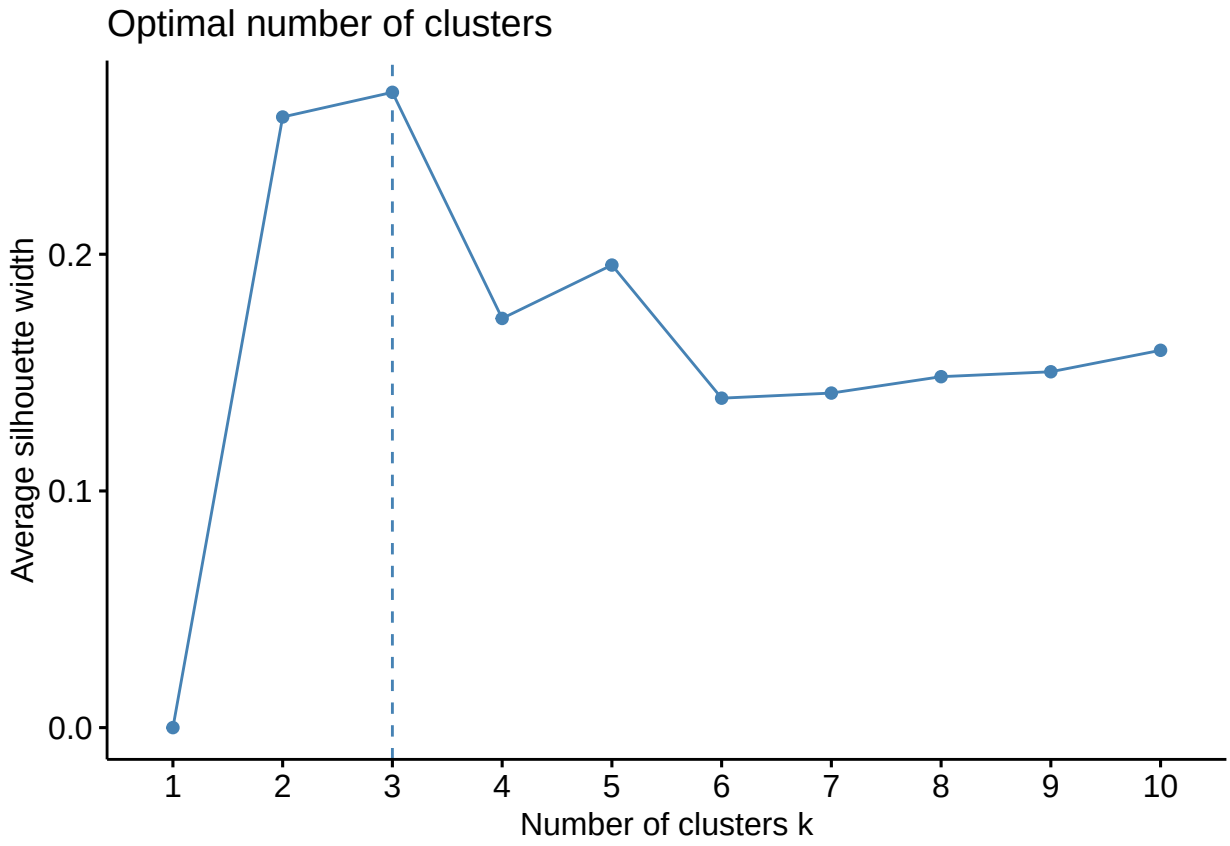
Etant donné que c'est une "extension" de la méthode PAM, l'avantage est son temps d'execution sur un grand nombre de données, seulement

6.2 Application des méthodes

6.2.1 PAM

On commence par estimer le nombre optimal de cluster à sélectionner, comme précédemment on utilise la méthode de la silhouette grâce à la fonction suivante :

```
fviz_nbclust(data_sc, pam, method = 'silhouette')
```



On choisit encore un nombre de cluster fixé à :

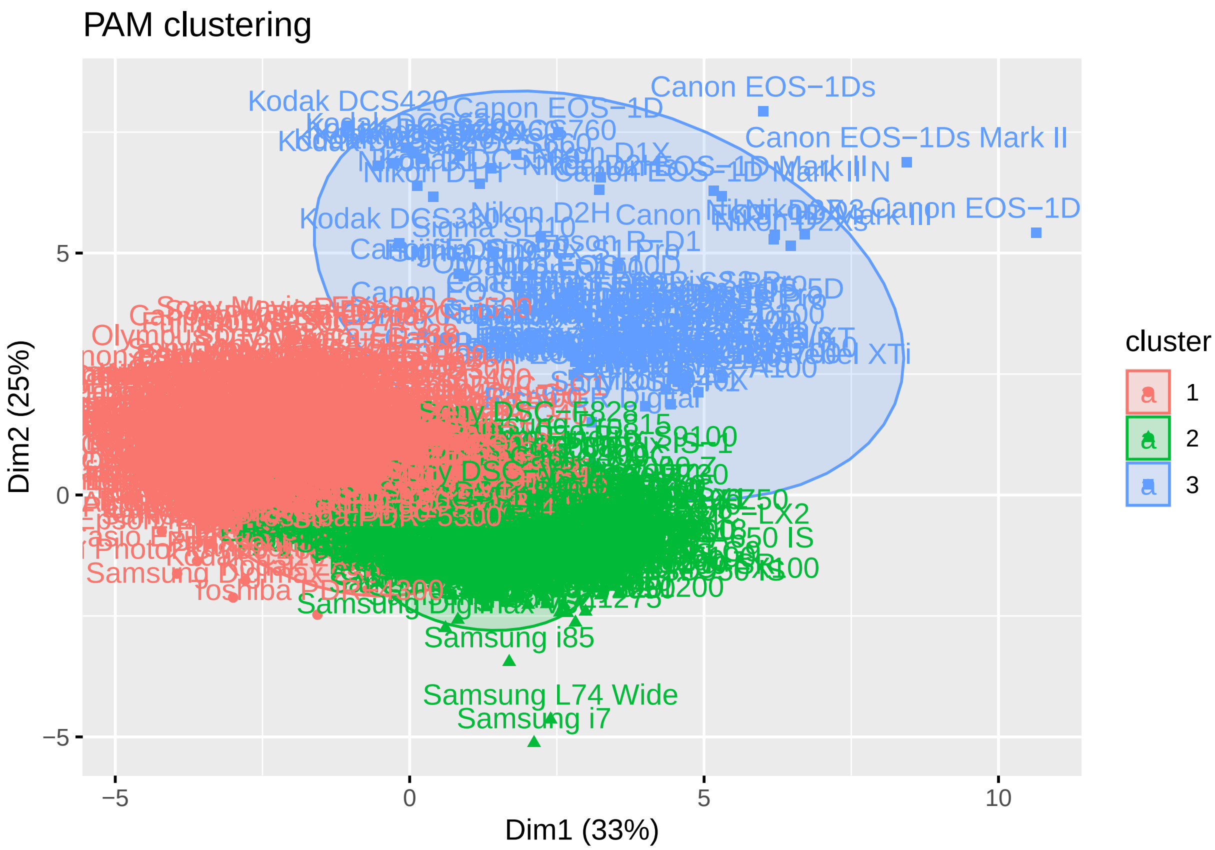
```
k <- 3
```

On applique la méthode PAM à notre jeu de données grâce aux lignes suivantes :

```
pam.res <- pam(data_sc, k)
```

On obtient alors la classification suivante :

```
fviz_cluster(pam.res, data_sc, ellipse.type = 'norm') + ggtitle('PAM clustering')
```



On décide de regarder l'inertie de cette partition :

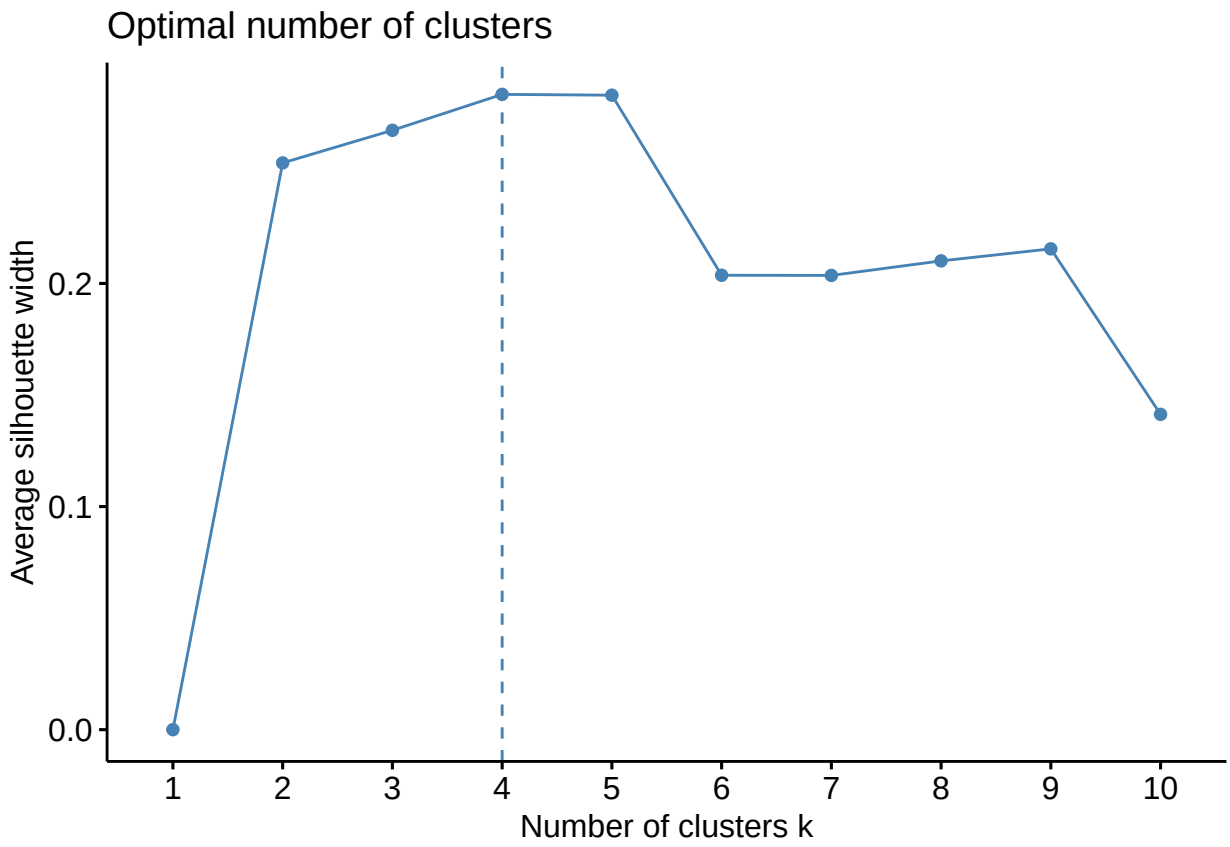
```
inerties <- calculate_inertie(list(data=pam.res$data, centers=pam.res$medoids, clusters=pam.res$clusteri
print(inerties)
```

```
## $total
## [1] 11.98842
##
## $inter
## [1] 4.393158
##
## $intra
## [1] 7.106372
##
## $expli
## [1] 40.72302
```

6.2.2 CLARA

Comme pour les autres méthodes de partitionnement, nous devons d'abord estimer le nombre optimal de cluster à séléctionner avant d'appliquer la méthode CLARA, on utilise toujours la même fonction :

```
fviz_nbclust(data_sc, clara, method = 'silhouette')
```



Et on séléctionner un nombre de clusters `k` de :

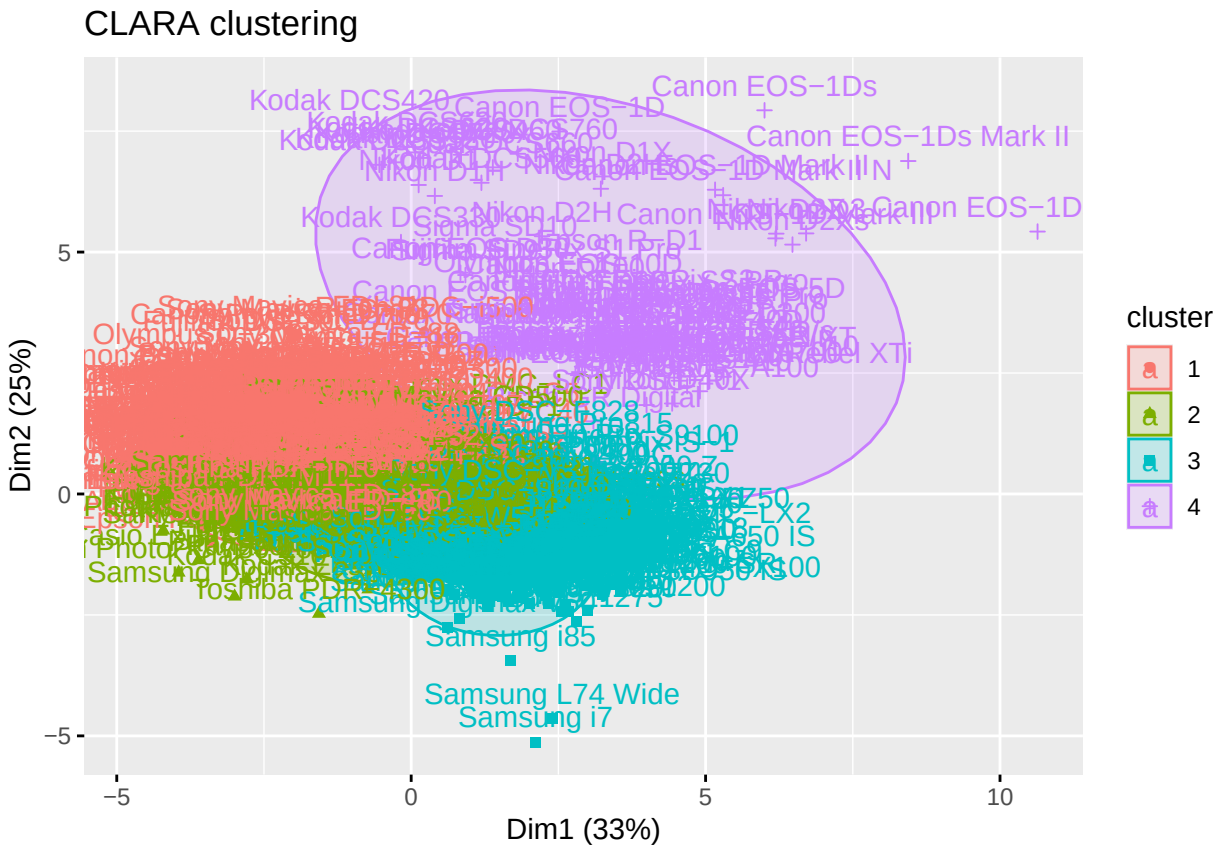
```
k <- 4
```

Et on applique la méthode CLARA grace à la ligne suivante :

```
clara.res <- clara(data_sc, k, samples = 50)
```

On obtient alors la représentation suivante :

```
fviz_cluster(clara.res, data_sc, ellipse.type = 'norm') + ggtitle('CLARA clustering')
```




```
inerties <- calculate_inertie(list(data=clara.res$data, centers=clara.res$medoids, clusters=clara.res$cl
print(inerties)
```

```
## $total
## [1] 11.98842
##
## $inter
## [1] 5.147479
##
## $intra
## [1] 6.569188
##
## $expli
## [1] 45.20387
```

7 Classification mixte

Nous allons maintenant effectuer une classification mixte sur nos données et voir si cela améliore nos résultats.

7.1 Principe

Le principe d'une classification mixte est de :

- Effectuer une première partition en utilisant la méthode des kmeans avec un nombre de classes assez élévé de manière à limiter le risque de fusion

- Effectuer une CAH sur les centres de la partition précédente
- Determiner le nombre de clusters à séléctionner avec le dendogramme
- Appliquer la segmentation avec le nombre de classes séléctionné à nos données de départ (avec la méthode du kmeans)

7.2 Application

On commence par effectuer une partition avec un nombre élévé de cluster, on considère qu'un nombre de cluster de 10% du nombre d'observations est correcte :

```
k <- round(nrow(data_sc) / 10)
print(k)
```

```
## [1] 104
```

On genère la partition avec nos 104 classes :

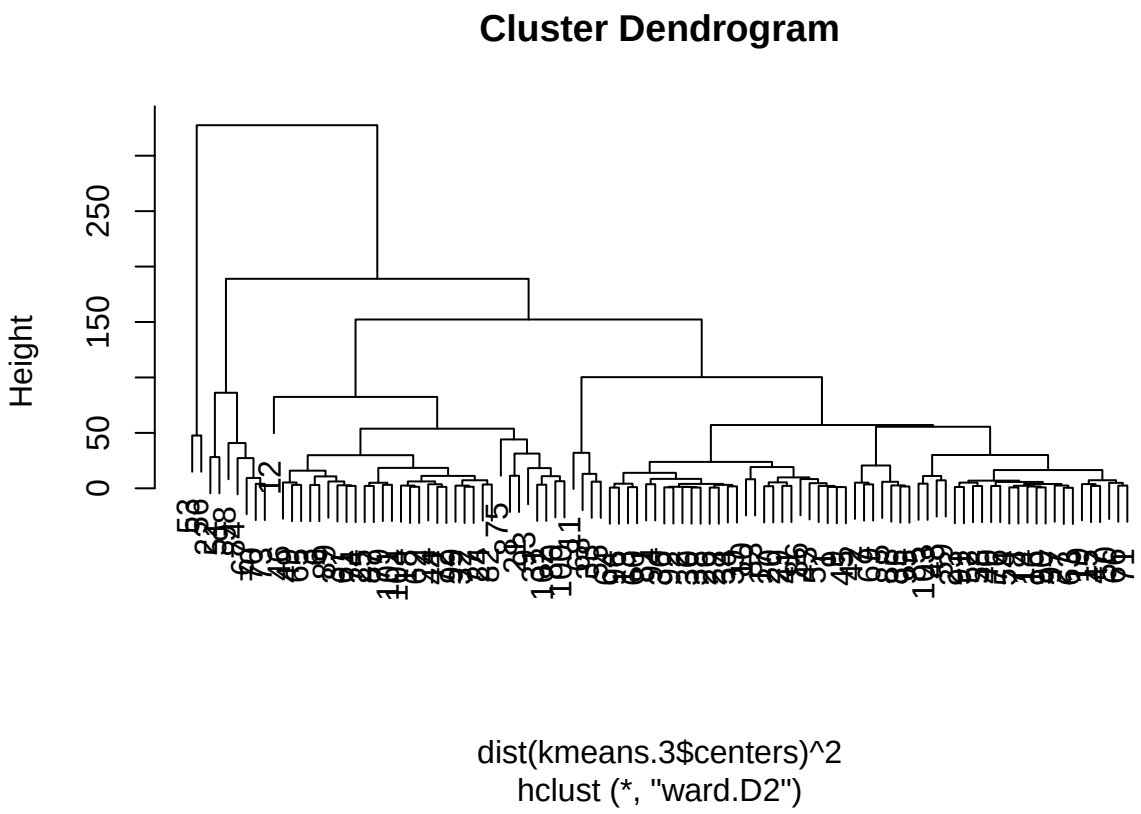
```
kmeans.3 <- kmeans(data_sc, k)
id_clusters <- factor(kmeans.3$cluster)
```

Puis on effectue une CAH avec la méthode de Ward sur les centres de cette partition :

```
cah <- hclust(dist(kmeans.3$centers)^2, method = "ward.D2")
```

On affiche le dendogramme pour pouvoir déterminer le nombre de classes à séléctionner :

```
plot(cah)
```



Une partition en 4 classes semble encore une fois pertinente :

```
k.new <- 4
```

On applique donc la segmentation sur nos 104 centres de départ :

```
group <- cutree(cah, k.new)
```

En appliquant à nos données :

```
clusters <- numeric(nrow(data_sc))
for (j in 1:nlevels(id_clusters)) {
  clusters[id_clusters == j] <- group[j]
}
```

On va maintenant calculer les centres de chacune des classes précédentes :

```
centers <- matrix(nrow = k.new, ncol = ncol(data_sc))
for (j in 1:k.new) {
  points <- data_sc[which(clusters == j),]
  centers[j,] <- calculate_center_of(points)
}
```

Puis on calcule l'inertie intraclasse, interclasse et la part d'inertie expliquée

```
inerties <- calculate_inertie(list(data=data_sc, centers=centers, clusters=clusters))
print(inerties)
```

```
## $total
## [1] 11.98842
##
## $inter
## [1] 5.334367
##
## $intra
## [1] 6.65405
##
## $expli
## [1] 44.496
```

8 Comparaison des méthodes

Nous allons maintenant comparer les différentes méthodes que nous avons utilisées jusqu'ici, pour cela nous allons nous baser sur le tableau suivant qui récapitule l'inertie intraclasse, interclasse et expliquée de chacune des méthode précédentes :

```
kable(results)
```

	total	inter	intra	expli	k
my kmeans	11.98842	4.571062	7.417355	38.12899	3
native kmeans	11.98842	4.990722	6.997695	41.62954	3
PAM	11.98842	4.393158	7.106372	40.72302	3
CLARA	11.98842	5.147479	6.569188	45.20387	4
CAH mixte	11.98842	5.334366	6.654051	44.49600	4

On remarque qu'on obtient des valeurs similaires pour les méthodes PAM, kmeans natif de R et l'implémentation de l'algorithme kmeans vu en cours. Les méthodes CLARA et CAH mixte ont un nombre de

classes séléctionné supérieure donc une part d'inertie expliquée meilleure que les autres méthodes, seulement leurs inerties interclasses et intraclasses sont également meilleurs.

La méthode CLARA semble la plus pertinente à retenir étant donné que sa part d'inertie expliquée est plus élevée que les autres méthodes, son inertie intraclasse est la plus faible et son inertie interclasse la plus élevée