Practicals 4 : Unsupervised learning

1 Framework

To illustrate the two unsupervised learning methods studied during the lecture, we propose to handle road data from the french institute IGN. This dataset includes distances between 47 french cities and border places by road.

To load these data and some useful functions used in the sequel, we begin by loading the script *tp4.R*,

```
source("http://www.math.univ-toulouse.fr/~xgendre/ens/m2se/tp4.R")
```

We always manipulate the same data in the sequel, thus we now put them in the data matrix `X`,

```
X <- DataVilles()
```

2 k-means

The k-means clustering allows us to classify the individuals into k groups for some fixed k. With the R software, we have at our disposal a function called `kmeans` to compute this clustering. The manual page `help(kmeans)` is brief but we read that to get the clustering into $k = 5$ classes, we just need the following command,

```
cmob <- kmeans(X, 5)
```

The obtained object `cmob` contains various informations. Among them, we have :
`cmob$cluster` vector of integers between 1 and k that gives the group index of each individual,
`cmob$centers` matrix of the distances between the individuals and the center of each class,
`cmob$size` sizes of the classes.
We can print the result of the clustering on a map with

```
AfficheVilles(cmob$cluster)
```

Run again the command that gave you `cmob` and display the result on a map. What do you notice ? Explain why the result is not the same at each time.

To stabilise this variability, it is possible to give to `kmeans` the initial centers of gravity. We will use this approach in Section 4. An other way amounts to repeat several times the procedure and to take the clustering with the minimal criterion (see details in the lecture). To try this, we can deal with the variable `cmob$tot.withinss`. With the help of the manual page, understand the meaning of this value. Compute several times the clustering and print the corresponding value `cmob$tot.withinss`. Explain with your own words why we do that and how it can help us.

As we have seen during the lecture, we also can consider a robust variant of the k-means called PAM (*Partitioning Around Medoids*). The R package `cluster` provides a function `pam` to use this procedure,

```
library(cluster)
help(pam)
```

The function `pam` is similar to `kmeans` and we get a clustering into 5 classes with the next commands,

```
cpam <- pam(X,5)
AfficheVilles(cpam$clustering)
```

Do some tests and compare the results obtained by `kmeans` and `pam`. What is the content of `cpam$id.med`? Use this variable and function `VillePos` to locate the medoids on the map.

3 Hierarchical clustering

The main drawback of procedures like k-means and its variants is the needed prior knowledge of the number of classes. To bypass this problem, various methods exist and, among them, we have considered the hierarchical clustering. To compute it, the R software has the function `hclust`,

```
help(hclust)
```

Here, we only use the two first parameters of this function :

`d` data given by a matrix of dissimilarities (we use `as.dist` to convert our matrix `X`),

`method` method to compute the similarity between two clusters (*single linkage*, *complete linkage*, ...).

We can display the obtained dendrogram with the classical function `plot` applied to the object returned by `hclust`. Thus, we get the following result for the hierarchical clustering computed with the Ward's method,

```
cah <- hclust(as.dist(X),method="ward.D2")
plot(cah)
```

The instance `cah` contains several informations whose the heights of the branches. We can see the scree of these heigths,

```
plot(rev(cah$height),type="b")
```

With the help about `hclust`, compute the hierarchical clustering for the various methods seen during the lecture. Compare the results you obtain.

To get the actual clustering, we also need to cut the tree at some height. The function `rect.hclust` can be used to plot this cut of the tree for a given number `k` of clusters or a fixed height `h`.

```
plot(cah)
rect.hclust(cah, k=5, border="red")
rect.hclust(cah, h=1500, border="blue")
```

The function `cutree` is used to get the cluster indices,

```
help(cutree)
```

This function can return the clustering by choosing a number `k` of classes or by cutting at a given height.

```
cluster1 <- cutree(cah, k=5) # 5 classes
cluster2 <- cutree(cah, h=1000) # height 1000
```

Display the clusterings obtained in `cluster1` and `cluster2`. Make vary the values `k` and `h` and the aggregation method. Comment the various results that you get.

4 Stabilisation

During the lecture, we have seen that the scree of the heights given by the hierarchical clustering can help us to select a number of classes. Recall how we did that and discuss about the number of clusters that is choosen for each aggregation method.

In particular, to get a stable result that profit by the two methods studied in these practicals, we can use the number of clusters given by the hierarchical clustering in any of the variants of k-means. In such a case, we also can initialize the procedure with the clusters from the hierarchical clustering. Explain and run the few next commands,

```
cah <- hclust(as.dist(X),method="ward.D2")
cluster <- cutree(cah, h=2000)
AfficheVilles(cluster)

nc <- max(cluster)
init <- c()
for (i in 1:nc) init <- rbind(init, colMeans(X[cluster==i,]))
cmob <- kmeans(X,init)
AfficheVilles(cmob$cluster)
```

Do other tests around this procedure. Try to use PAM, to change the aggregation method, to change `h`, ...