R Tutorial: Clustering A Mixed Dataset

Introduction

This article shows how to implement the k-prototypes clustering algorithm *(Huang, 1998)* via the R software. This article is far from exhaustive and omits many concepts and prerequisites. For an introduction to R, principal component analysis and unsupervised learning (especially the k-means method), see *Begin'R* (in French) and/or these videos: *Introduction to R*, *PCA* and *k-means* (all in English).

We assume here that you have access to a complete and operational dataset (although the algorithm used here must be able to handle missing values a priori). You will need the following libraries: *clustMixType, ggeffects, ggplot2, dplyr.*

The objective of clustering methods is to group subgroups of individuals in a data set. Concretely, the underlying algorithm will **group the individuals who are most similar**, i.e. who have the same characteristics (we'll what it means).

Let's take an example: the iris dataset. I have access to the data of different length and width measure, as well as the species. It is quite possible to use other methods of data analysis than clustering, but the latter has the advantage of **treating observations according to all their characteristics** and not "all things being equal", as would be the case with linear regression for example.

The main problem is that classical clustering methods (k-means, k-modes...) do not allow to cluster mixed data sets. To this end, *Z. Huang (1998)* will propose an extension of the k-means method allowing the clustering of a mixed data set: the k-prototypes method. For a more detailed description of the algorithm, it is strongly recommended to read the original article by Huang and the article by *Gero Szepannek* on the subject. We specify here that the proposed algorithm is a slight extension of the original algorithm but keeps the general principle.

K-prototypes algorithm

The calculation and the interest of the objective function of the model will not be really detailed here. However, there is one important thing to know: the k-prototype algorithm **requires to give a weight to the categorical variables compared to the continuous variables**. For the latter, the algorithm computes the distance between the points (in the geometrical sense of the term), while for the categorical variables it is the simple measure of dissimilarity (the number of different elements between X and Y, two distinct parameters). For a more exhaustive description of the k-mode method, see this article by *A. Aprilliant.*

Objective function:

$$E = \sum_{i=1}^{n} \sum_{j=1}^{k} u_{ij} d(x_i, \mu_j)$$

The distance function corresponds here to the **weighted sum of the Euclidean distances** between two points (for the q continuous variables), and the simple dissimilarity measure (for the $p-q$ categorical variables). Categorical variables come with an important coefficient: λ (called lambda).

Distance function:

$$d(x_i, \mu_j) = \sum_{m=1}^{q} (x_i^m - \mu_j^m)^2 + \lambda \sum_{m=q+1}^{p} \delta(x_i^m, \mu_j^m)$$

The argument λ is important since it will determine the weight we give to categorical variables over continuous variables. In the `kproto()` function that we will see just after, the "lambda" argument corresponds precisely to the variable λ. If no value is specified, then the algorithm will estimate it via the `lambdaest()` function. If you have no idea what to do with this (important) parameter, you can simply use the `lambdaest()` function with all the default arguments. See *G. Szepannek's article* for a more in-depth explanation of the `lambdaest()` function.

In order to check if you have understood the general idea, try to guess what would happen if the lambda argument was set to $= 0$. If you have the right answer, we can continue. In fact, setting $\lambda = 0$ means giving zero weight to the categorical variables and thus having **results very similar** to those we would get via the k-means method.

Here are the first line of a mixed dataset

```
#clean environment
rm(list=ls())

#load a dataset and add 120 (~15%) NaN randomly
data("iris")
df = data.frame(iris)
head(df)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

Determine K, the number of clusters

The algorithm does not allow us to define the number of clusters that our dataset "contains". It is up to us to decide, in a more or less arbitrary way, what is the optimal number of clusters. For this, there are a number of different methods. We decide here to use the most common one in the field of clustering: the **"elbow method"**.

This method consists in calculating the cost (or error) of the model according to the number of clusters chosen. Ideally, we want a low cost for a relatively low number of clusters. Indeed, as we will see, increasing the number of clusters in the model decreases the cost but makes us **lose in parsimony**. It is indeed quite useless to have 30 clusters in a dataset of size n=30 (1 cluster = 1 individual, total absence of parsimony!). I take the opportunity to specify that it is better to do clustering on relatively large samples (a few hundred is probably a good minimum).

The elbow method is in fact the same as looking for, from a curve of the cost function, the number k which represents the "elbow" (thus the **break between the arm and the forearm**). Don't panic, we will see a concrete example.

Important clarification: having a model with, for example, 5 clusters instead of 4 does not mean adding a 5th cluster among the 4 existing ones, but in fact dividing the dataset into 5 subgroups INSTEAD of dividing

it into 4 subgroups. The 4 "first" subgroups of the 5-cluster model are not the same subgroups as the 4 subgroups of the 4-cluster model.

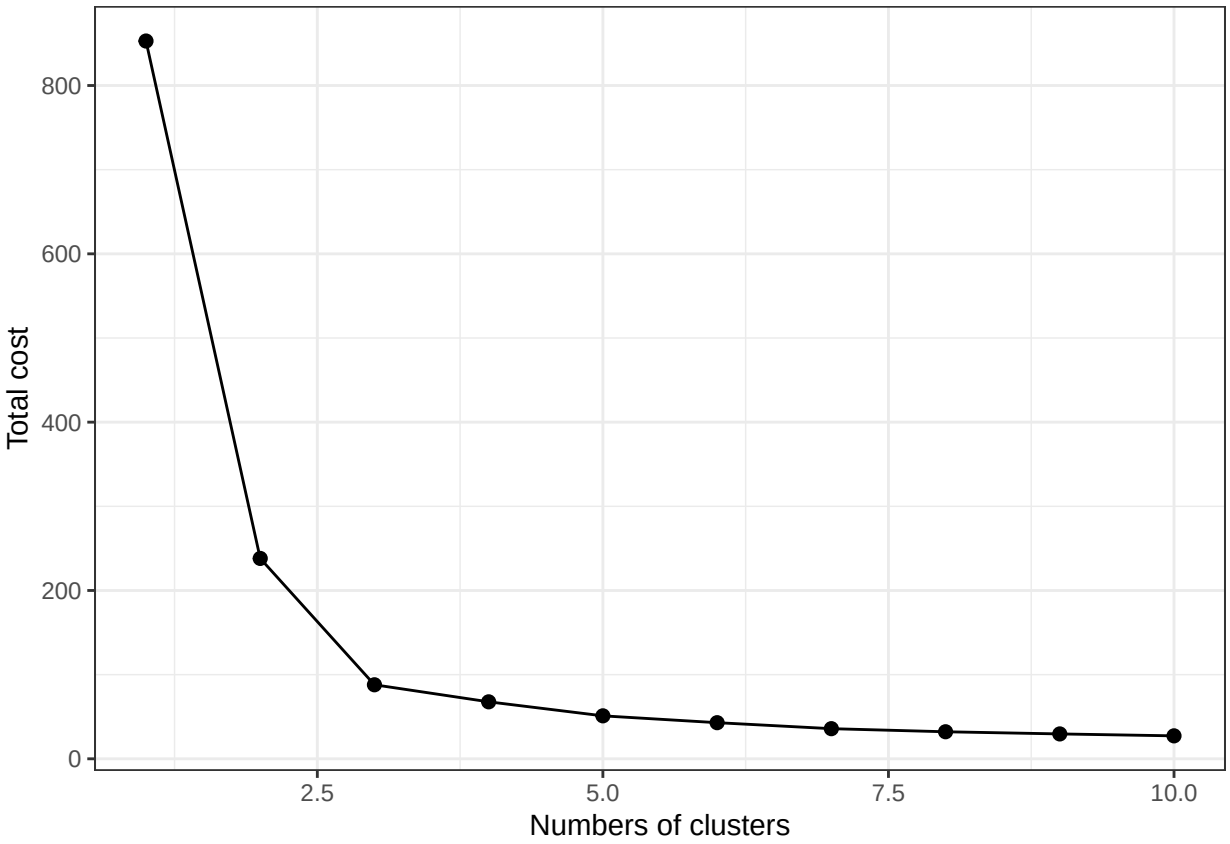
Creating the cost plot

There is no predefined function in R for the k-prototype algorithm that allows to display the "elbow" graph. So we will have to create this function ourselves. First, we want to create a vector that contains the cost of the model for each number of clusters. To do this, I create a loop that will **calculate the cost of the model at 1 cluster, then 2 clusters, etc, up to 10** (it is rare to want more than 10 clusters). Then we display this vector as a graph against the number of clusters in the model. It is possible that the code will move if you copy and paste it. So remember to check the alignment once pasted.

This computation step can take quite some time depending on the size of your dataset (several hours...). To have an estimate of the time that the algorithm will take, I recommend having a line of code in the loop that will be used only to display in the R console that the i-th iteration of the for loop is finished (see the line starting with cat). I am not sure that the time is constant between the different models. The argument "nstart=25" is the **number of iterations to be performed for each model**. The most important thing is that this value is big enough (in general we advise nstart>20) in order to increase the "chances" to be in a global minimum (or at least a **better** local minimum) and not in a local minimum.

```r
#call packages
library(clustMixType)
library(ggeffects)
library(ggplot2)
library(dplyr)

#create an empty vector
tot_cost = c()

#calculate the cost
for (i in 1:10) {
      kprot = kproto(x = df, k = i, nstart = 25, verbose=FALSE)
      tot_cost[i] = kprot$tot.withinss
}
```

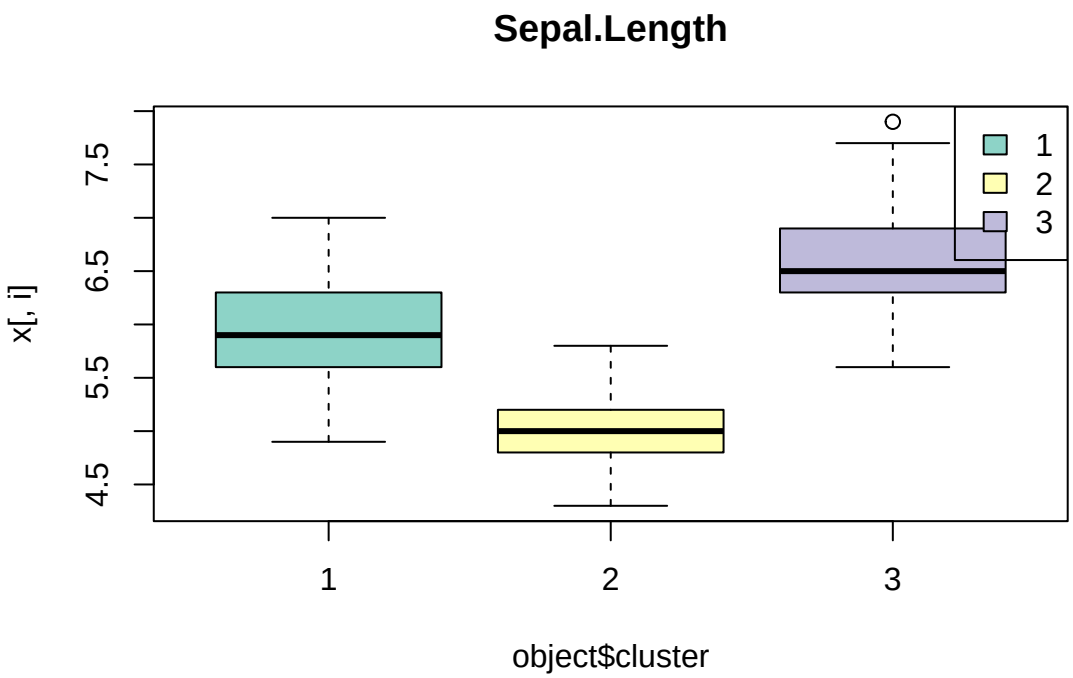
```r
#plot cost
tibble(k = 1:length(tot_cost), total_error = tot_cost) %>%
      ggplot(aes(x=k, y=total_error)) +
      geom_point(size=2) + geom_line() + theme_bw() +
      labs(x = 'Numbers of clusters', y = 'Total cost')
```

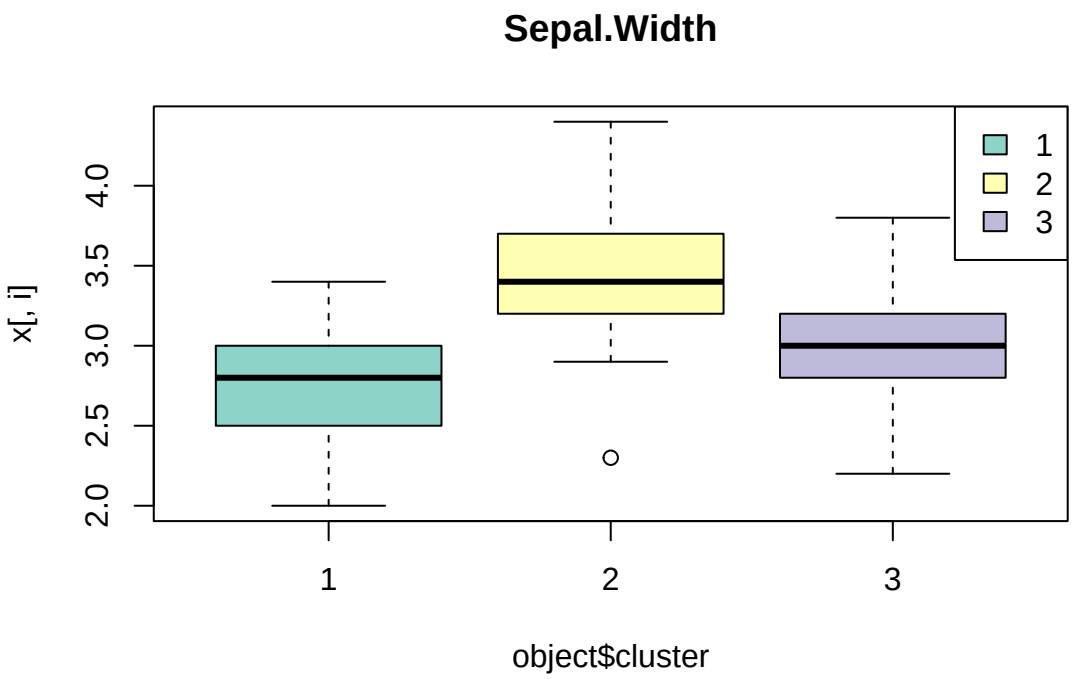


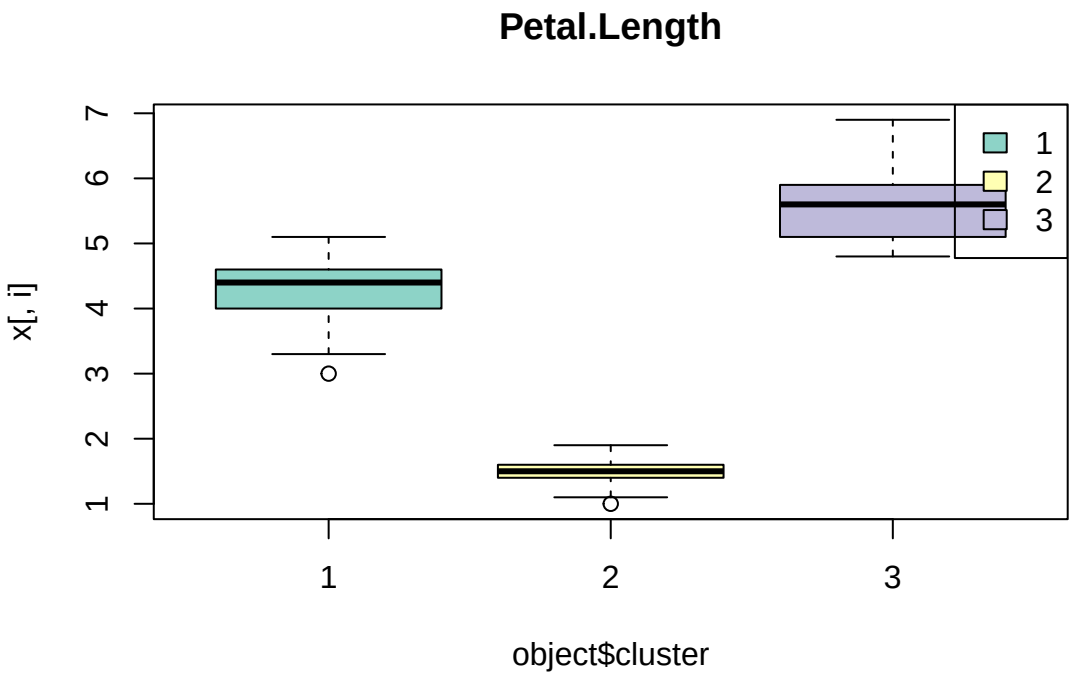
We can see here that increasing the number of clusters decreases the cost. However, it is contrary to the principle of parsimony to decrease the cost at any cost. Therefore, we try to **make a "reasonable" choice**.

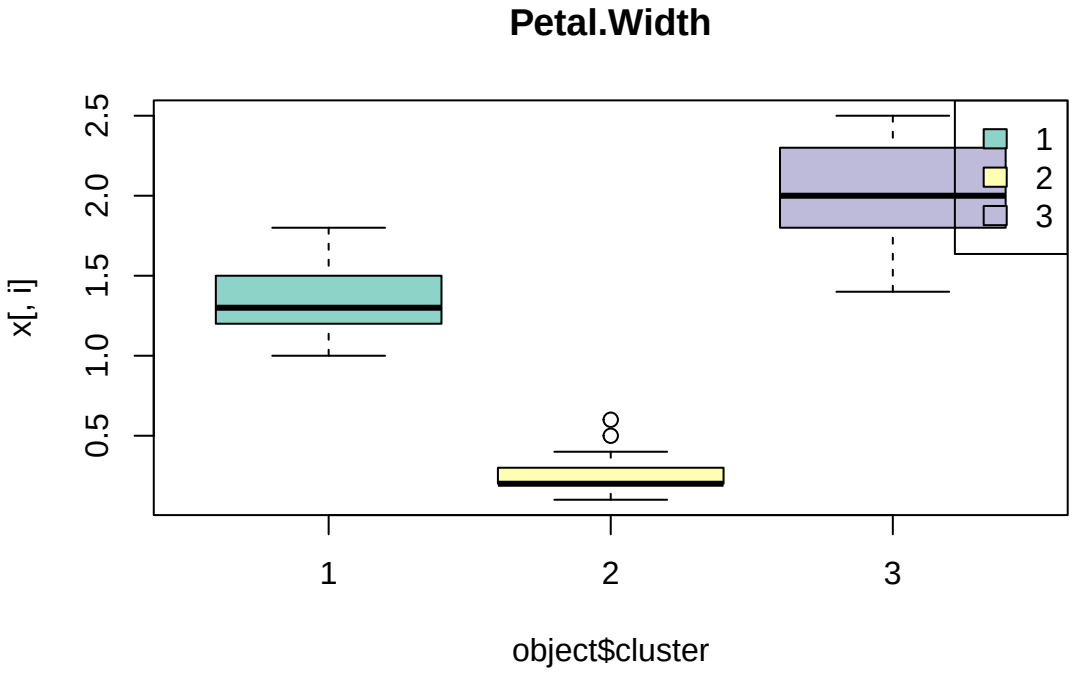
In our case, I consider that $k = 3$ is a good choice because beyond that, increasing the number of clusters does not significantly decrease the cost of the model (i.e. the slope becomes almost linear). However, it could have been **perfectly justified to choose another value of k**. As said before, there are no really good answers to this problem and the decision is sometimes quite subjective. Once this choice is made, we can run the algorithm again with a constant for the k argument (3 in our case).

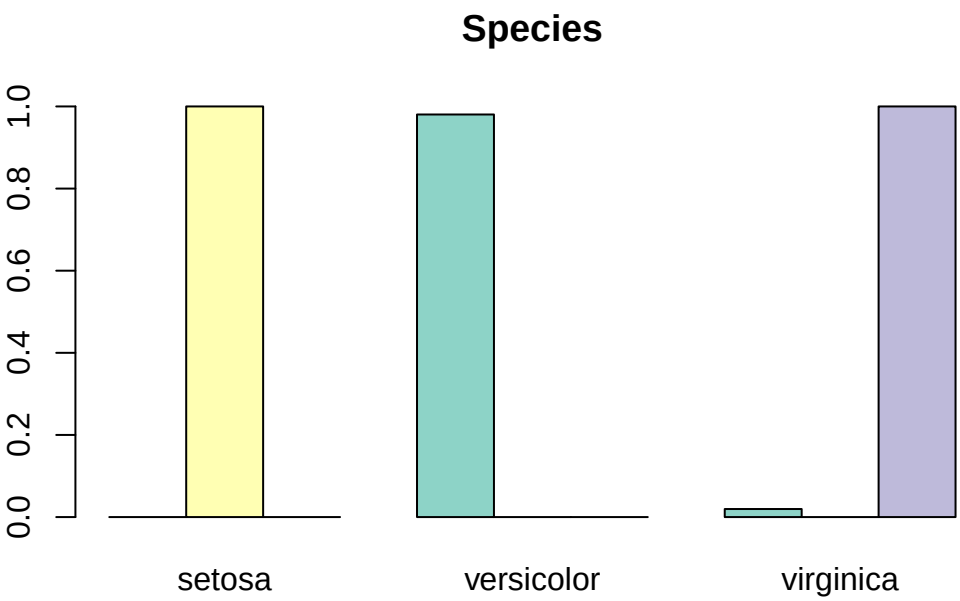
```
#create a model with 3 clusters
k3prot = kproto(x = df, k = 3, nstart = 25, verbose=FALSE)

#plot the results for each variable
clprofiles(object = k3prot, x = df)
```


Sepal.Length
x[, i]
4.5
5.5
6.5
7.5
1
2
3
object$cluster
1
2
3


Sepal.Width
x[, i]
2.0
2.5
3.0
3.5
4.0
1
2
3
object$cluster
1
2
3


Petal.Length
x[, i]
1
2
3
4
5
6
7
1
2
3
object$cluster


Petal.Width
x[, i]
0.5
1.0
1.5
2.0
2.5
1
2
3
object$cluster



Graph of the results

We have access to a function that allows us, for each variable in the original data set, to create a graph that shows us how the clusters are distributed in each variable (see above). These graphs are very important because they allow us to see the differences between the profiles. In the obtained graphs, each cluster is associated with a color and keeps this color for all the graphs. We can see that each cluster is very different from others, which means that our model works great!

Below you will find the code to save the results, i.e. to add a feature to the dataset containing, for each individual, the cluster associated with him. This feature can be useful to perform **a posteriori analysis** of the data.

```
#create the clusters column
df = mutate(df, clusters = k3prot$cluster)
df$clusters = as.factor(df$clusters)

#display a random subset
head(df[sample(nrow(df)), ])
```

```
##     Sepal.Length Sepal.Width Petal.Length Petal.Width    Species clusters
## 95           5.6         2.7          4.2         1.3 versicolor        1
## 71           5.9         3.2          4.8         1.8 versicolor        1
## 141          6.7         3.1          5.6         2.4  virginica        3
## 10           4.9         3.1          1.5         0.1     setosa        2
## 39           4.4         3.0          1.3         0.2     setosa        2
## 106          7.6         3.0          6.6         2.1  virginica        3
```