Audio transcription of the clustering with FactoMineR video

In this video, we're going to see how to do clustering with FactoMineR. In FactoMineR, clustering is done after first running some principal component method. When we have a data table with quantitative variables, we can use the individual's coordinates from the PCA to run a clustering method. When the variables are qualitative, we can do an MCA and then clustering using the individual's coordinates from the MCA. If instead we have a contingency table, we can do a CA and use the coordinates of the rows, or the columns, to do clustering, depending on what exactly we want to do the clustering.

The clustering function in FactoMineR works according to the following steps. First up, there's an optional step. If we have lots of individuals, it's possible to construct a very rough partitioning of the data, for example with a hundred classes, using k-means, before running the main clustering algorithm. In this way, the final clustering will be with respect to these pre-processed classes. After this optional first step, the other steps of the function can run. First, this involves running a principal component method, which in turn requires choosing the number of dimensions to retain in this principal component method.

This is because, essentially, the distances that will be used to build the clustering will be calculated using only these first dimensions. The later axes are put to the side because they are considered to represent noise. There is therefore a choice to be made for the number of dimensions to keep; an important choice. By default, the principal component method will keep the first five. We can modify this number using the ncp option in the PCA, CA or MCA. And if we want to do the clustering based on the actual distances, we have to keep ALL of the axes; for this, we have to write: ncp = Inf. After this has been set, the function will build a hierarchical tree and cut it at a certain height, therefore defining the clusters.

We will choose this height. In the next step, the fourth, we will be able to make the partitioning more robust, using k-means. And lastly, when the clusters have been finalized, we can describe what defines each cluster, using active or supplementary variables, either quantitative or qualitative, and also take a closer look at what individuals are grouped together and why.

To have a closer look at how the clustering function in FactoMineR works, we're going to use the decathlon dataset once more. Just a little reminder about this data. It involves forty-one athletes and thirteen variables. The first ten variables involve the athlete's performances in each of the ten decathlon events, including the 100 meters, the long jump, shot-put, etc.

The next two variables are quantitative, and treated as supplementary ones. They are: the rank of each athlete after the decathlon is over, and their total number of points. The thirteenth and final variable is the qualitative one, stating whether a particular athlete participated in the decastar competition or the Olympic games. We are going to do the clustering step using only the performance data, that is, the first ten variables. If you want any more details about this dataset, feel free to go back and have a look at the PCA video.

Let's now see how to put clustering into practice using FactoMineR. The first thing we're going to do is import the dataset, like this...... Ok, so far so good, the data seems to be imported correctly.

Let's check on this. The first ten variables are indeed quantitative, and we have two supplementary quantitative variables: ranks, and the number of points. And the last variable, the competition, is indeed qualitative. Next, let's load the FactoMineR library. Ok, that's done. Now, we'll run a PCA on this dataset. We make sure that the function understands that the 11th and 12th variables are supplementary and quantitative, and that the 13th is supplementary and qualitative. We can also signal that we want to keep ALL the axes, by putting ncp = Inf. Two plots pop out. The first one is of the variables, and the second is of the individuals. We can look at the percentages of inertia for each axis, and see for example that the first eight allow us to get ninety-six percent of the information, which is quite enough for us. We will therefore choose to do the clustering using these eight components. This time, in the res object, we get the results for the first eight dimensions.

Now we can do the clustering. We take the object output by the PCA and do clustering on it with the HCPC function. Here, kk=Inf means that we don't want to do any initial partitioning before running the clustering. If we DID put a number there, like 100 for example, first the function would divide the data up into 100 classes, then do the clustering. As our dataset isn't so big -- there are only forty-one individuals -- we can simply do the clustering directly. Here, the choice of min and max means that we want to find the optimal number of classes between three and ten. consol = TRUE means that we want to make the classes robust at the output of the clustering. So, here's the output: a hierarchical tree with a graphical output showing the loss of inertia.

Look at the first bar in the graph, which tells us that there is a big loss in inertia when passing from two classes to one, which means that it's not a good idea to group together the two classes. In the same way, we see that there's a loss of inertia when passing from three classes to two, and from four classes to three. In contrast, there's barely any loss of inertia when passing from five classes to four. Therefore, it's a good idea to retain four classes here. And similarly for the hierarchical tree, the cutting height proposed here gives four classes. I'm going to click here on four classes.

Several plots are produced when we run this function. The first one shows the individuals on the first two axes of the PCA, that is, the first two dimensions. The individuals are colored in terms of which classes they belong to. We also get a three-dimensional plot showing the individuals in the first two dimensions for the PCA, and in the third plot, the hierarchical tree. This tree shows the relative proximity of the individuals to each other. For example, like for the previous plot, the blue and black classes are well-separated, whereas the green and red ones, not so much.

A good question is: why are these two classes tangled together? Well, let's go and have a look at the plot for the third and fourth dimensions. If we output this plot, we can see that now, the green and red classes are well-separated. What this means is that we have the first two axes which separate the blue and black classes well, and a second plane, given by the third and fourth axes, which separates the green and red classes well.

Therefore, we need four dimensions to separate the classes well. Now, let's go back to the plot for the first two dimensions. Here it is. We can now look at the results, starting by typing names (res.hcpc). We can see that the results include several objects: data.clus, desc.var, desc.axes, call, and desc.ind, the results for the individuals. Let's start by looking at the results in "call". In particular, we can get results on the hierarchical tree using res$call$t, which is the most informative object in "call", showing first the principal component method results, in our case the PCA, followed by results on the hierarchical clustering. Here, the hierarchical clustering function used is called hclust.

All of the results are in the "tree" object. Then, we have the number of classes retained, in our case, four. Followed by the within-class inertia. So, we can see that the within-class inertia is 9.6 here, when all of the individuals are in the same class. So, this isn't equal to the number of variables, ten. Why is this, you may ask? Well, as the variables were centered and standardized, the total variability to begin with was ten, but because we only kept 8 dimensions, it turns out that in those eight, the total inertia is 9.6. Here, the clustering into one class shows the total inertia of the dataset we are clustering, so, for the first eight dimensions only.

Next up, we have the within-class inertia when we partition the data into two classes, three classes, etc. Here, we have the gain in inertia when we go from N classes to N plus one. So, when we go from one class to two, we have an increase in inertia of 1.78, which is a lot. So, we're going to want to keep at least two classes. Same thing when we go from two to three classes, we gain a lot. Again when we go from three to four. And then, from then on, we don't gain so much. The output $quot gives us the ratio of successive within-class inertias. This can help us choose an optimal number of classes.

So, let's take stock of the various results. data.clust returns the initial data set along with all the active and supplementary variables, and adds an extra variable, the class variable, which has four categories in our case. Next, we can characterize the classes found in terms of the variables, with desc.var. This desc.var first sorts the quantitative variables, starting from those which best characterize the partition, moving to those that only partially characterize the partition, but still significantly. In fact, only the variables with a statistically significant link to the class variable are retained here.

To find out whether a link is strong or not, we calculate the correlation ratio, and look to see if it's significantly different to zero. That is, the correlation ratio between the quantitative variable and the class variable. For example, we see here that the "number of points" variable is the one that's most linked to the class variable. Next up, we have a kind-of summary for each class. For example, for the first class, the most linked variables are the 100 metres, the 400 metres, the number of points, and the shot put.

For the 100 metres, the individuals from the first class correspond to values which are significantly different to zero, and higher than the mean. That is, significantly different to zero because the t-statistic is greater than two in absolute value, and higher than the mean because the t-statistic is positive.

We can check this out, and we find that indeed, the individuals in the first class, run the 100 metres in an average time of 11.27 seconds, while the average over all athletes is 10.99. The first class is clearly, on average, a group of slower sprinters.

And these are the individuals who have significantly less points than the others. Significantly, because the test-value is quite negative, and less points, because we can see that on average they got 7596 points, while the average across all athletes was 8500. Moving on, we then have a description of the second class, then the third, and then the fourth.

In this example the qualitative variable does not to characterize the classes. We can also describe the classes using the axes, that is, the PCA dimensions in this example. These dimensions are quantitative variables, so exactly the same method as before can be used. We see that it's the first and third dimensions that best characterize the classes here.

Indeed, the first class has significantly smaller coordinate values than the other classes in the first dimension. For the individuals in class two, they have significantly smaller coordinate values than the other classes in the third dimension. Individuals in the third class have significantly larger coordinate values in the third dimension, and slightly smaller ones in the second dimension. Lastly, the individuals in the fourth class have significantly larger coordinate values in the first dimension. Having seen all this, it makes sense to show the hierarchical tree using the plane defined by the first and third dimensions, which is what we do here.

We can see clearly that when we show the hierarchical tree on the one-three plane, the classes are really well-separated. Let's now stop for a moment to take note of something. The plane defined by the first and second dimensions gives the best visual idea of the distances between individuals. But if we add the colors corresponding to which classes they are in, we can see that there is a separation between green class and red class individuals, that is, a large distance between these two classes. Therefore, the clustering helps us to see what is happening in the third and fourth dimensions. Overall, this means we get an optimal impression of distances via the PCA plane, which is enriched by the clustering, indicated by the colors.

Now, we can have a look at the results for the individuals. For this, we have two objects. One object, called para, concerns the "model" individuals, that is, the ones closest to the center of each class. For class 1, this model individual is Uldal, who is at a distance of 1.43 from the barycenter of the class 1 individuals. This is the closest individual to the barycenter. The second closest is Barras, then Karlivans, etc. For the class 2 individuals, the closest is Hernu, and so on, and so on. There is a second measure that comes from the individuals, which we call the specificity. The results of this are in the "dist" object.

So what is this exactly? Well, for the "dist" object, the distance of individuals to the barycenters of the other classes has been calculated. For the class 1 individuals, Casarsa is the one that is furthest from the other classes, that is, the furthest from the barycenters of the other classes. Its distance to the closest other barycenter is 5.08. From this point of view, Carsara is very clearly representative of the first class; there's no way it could be in any of the other classes. As for class 2, the most typical individual in this sense is Smith, as they are very far from the barycenters of the other classes.

So, now we've seen much of what we can do with clustering in FactoMineR. We can characterize classes using individuals, using quantitative and/or qualitative variables, active and/or supplementary variables, and even by using dimensions. And what's more, we can get interesting information and results from the hierarchical tree output. Now it's your turn to put clustering into practice using FactoMineR. Good luck!