[II.2313] Analyse de données - TP4

Rémi Biolley – Thierry Lincoln

A. Analyzing Fisher's Iris with the K-Means algorithm

1. First of all, let’s open the file iris. To do so we use the read.table() command specifying that the file contains a header

```
tab=read.csv(“iris.data”, header=T, sep=”,”)
```

2. The last column in your data contains the labels matching with the Iris specie to which each data belongs. Remove these labels from the main set and store them in another vector.

We copy the column containing the labels in a vector named *eyeclass.*

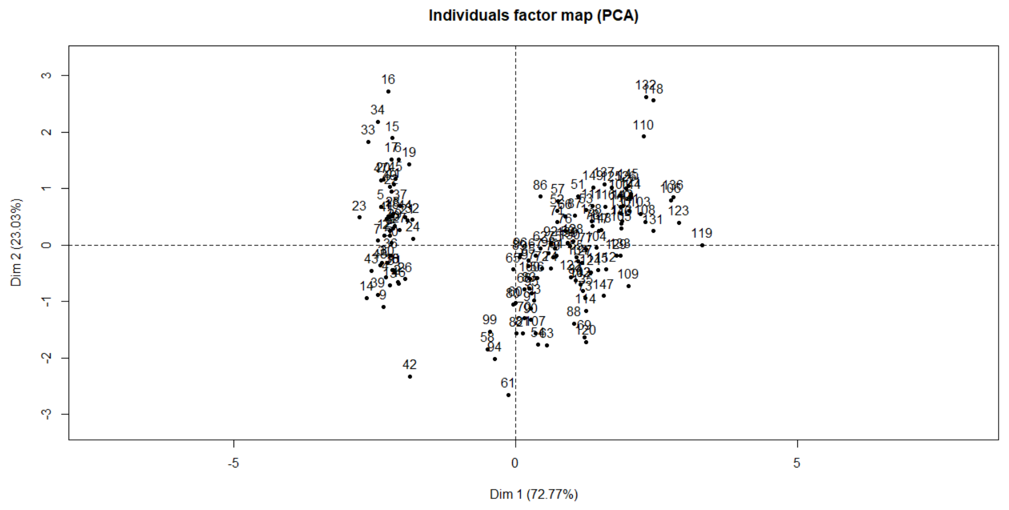
```
eyeclass=tab[,5]
tab=tab[,-5]
```

3. Use the command pca=princomp(data, cor=T) to do a Principal Component Analysis on your data. Then use the following lines to retrieve the dataset projected on the two principal components:

We want to reduce the dimension of our dataset so that we can plot and visualize our data.

The function *princomp()* returns a list. This list contains a lot of information like the standard deviation of every variable, the score of every value of our dataset, the mean of the variables ... We are interested in the scores as it helps us construct our 2 dimensions. We’ll keep the 2 first components and will combine them to create a new dataset which is defined in 2 dimensions. We can explain 95% of the variance of our dataset in this 2-dimension space.

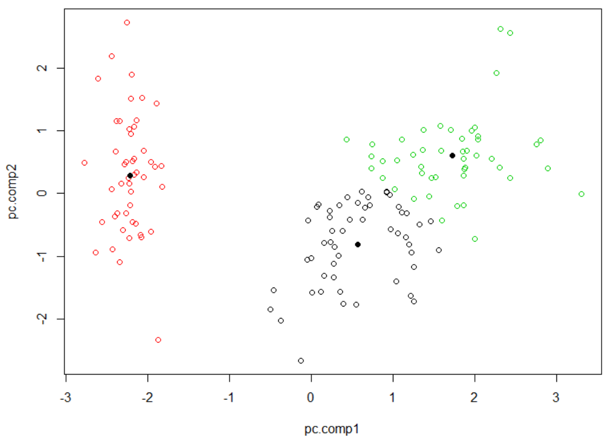
```
pca=princomp(tab, cor=T)
pc.comp = pca$scores
pc.comp1 = pc.comp[,1]
pc.comp2 = pc.comp[,2]
X = cbind(pc.comp1, pc.comp2)
```



4. Use the K-Means algorithm on your data X to obtain a partition with 3 clusters and visualize your results.

We use the *kmeans()* function to do so, indicating that we want to use the modified data we obtained previously and that we expect to find 3 clusters (indeed we know that 3 different species are represented in our dataset). Then, we plot the result in our 2 dimensions space precising that we want a different color for each detected cluster. We finally indicate that we want to see the center of every cluster.

```
cl=kmeans(X,3)
plot(pc.comp1, pc.comp2,col=cl$cluster)
points(cl$centers, pch=16)
```

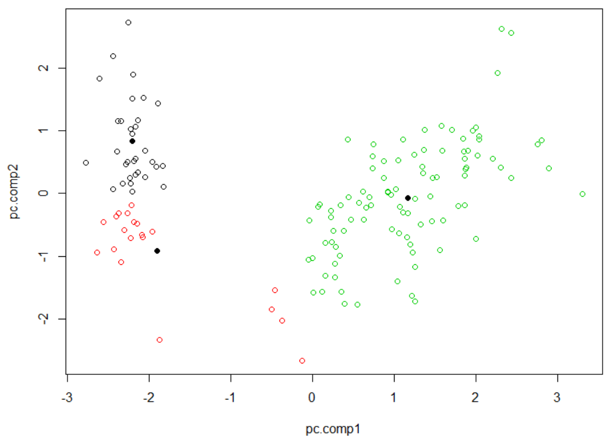


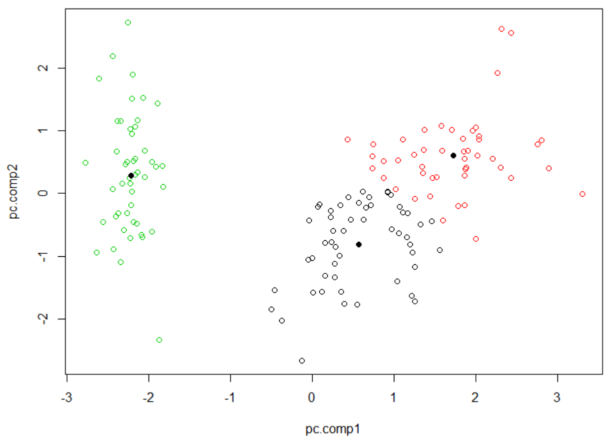
The 3 clusters found by the K-Means algorithm aren't looking absurd. There is a cluster which is isolated (the red one). This one is probably correctly defined. Nevertheless, the 2 other clusters are close to each other. Thus, we should be careful and not conclude that these clusters are well defined.

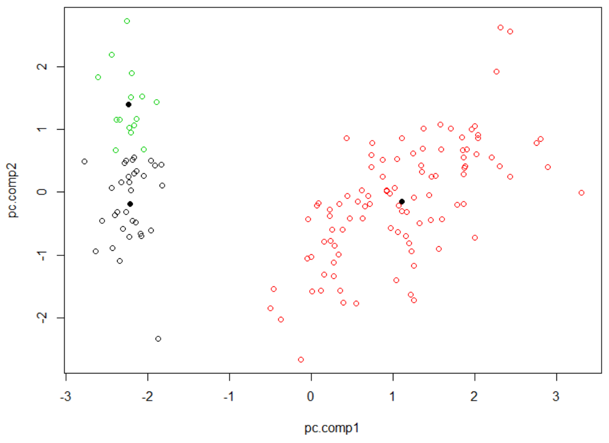
5. Repeat question 4) several times. What happens? Comment.

We wonder if the resulted clusters are likely to change if we repeat the K-means algorithm. We're going to verify this assumption.

There are 3 other results obtained repeating the previous steps:







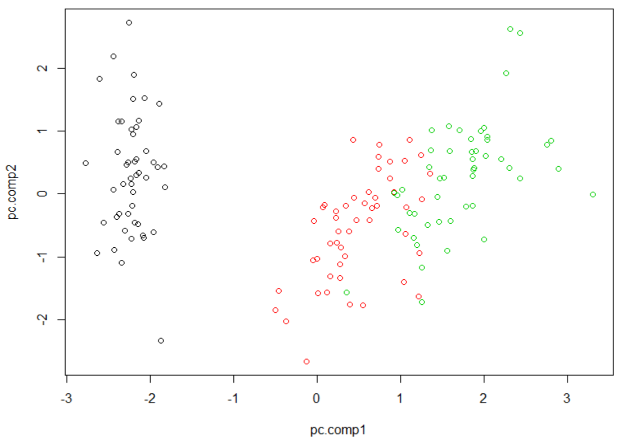
We notice that the results are often differing from our first result. Only 2 of them are the same. In the other cases, the block of points on the left is considered as being composed of 2 clusters, the rest of the points being considered as a unique cluster.

That's not surprising to get such different results considering how the K-Means algorithm works. Indeed, the algorithm first randomly chooses 3 different points which will be used to start the rest of the operations. The final result depends on the choice of these 3 points and can change dramatically.

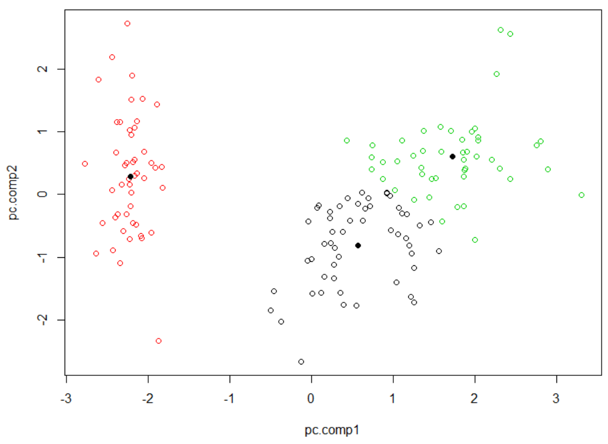
6. Use the command plot() to project the labels that you stored in a separate vector in question 2). Compare these results with the partitions from your K-Means experiments. Comment.

To determine which result of the K-mean results are the closest to the reality we decide to project the labels we stored in a separate vector at the beginning of this part.

```
plot(pc.comp1, pc.comp2,col=eyeclass)
```



Visualization of the projected labels



Visualization of the k-means clustering

As we visualize the projected labels, we notice that the block on the left was definitely specific to a specie. We obtained the same result after most of our K-Means experiments.

What's interesting is the division of the second block. We've never managed to get a result exactly similar to the projected labels using the K-Means algorithm. Nevertheless, the result is really close to the reality. As expected, the K-Means algorithm can't assign to the right cluster the observations which are "lost" in the middle of the block of the wrong specie.

7. Use the command table to prompt the contingency table comparing your results with the theoretical labels. Comment.

It is possible to visualize the differences between the resulted clusters and the species using a contingency table.

```
table(eyeclass,cl$cluster)

eyeclass      1  2  3
Setosa        0  0 50
Versicolor   11 39  0
Virginica    36 14  0
```

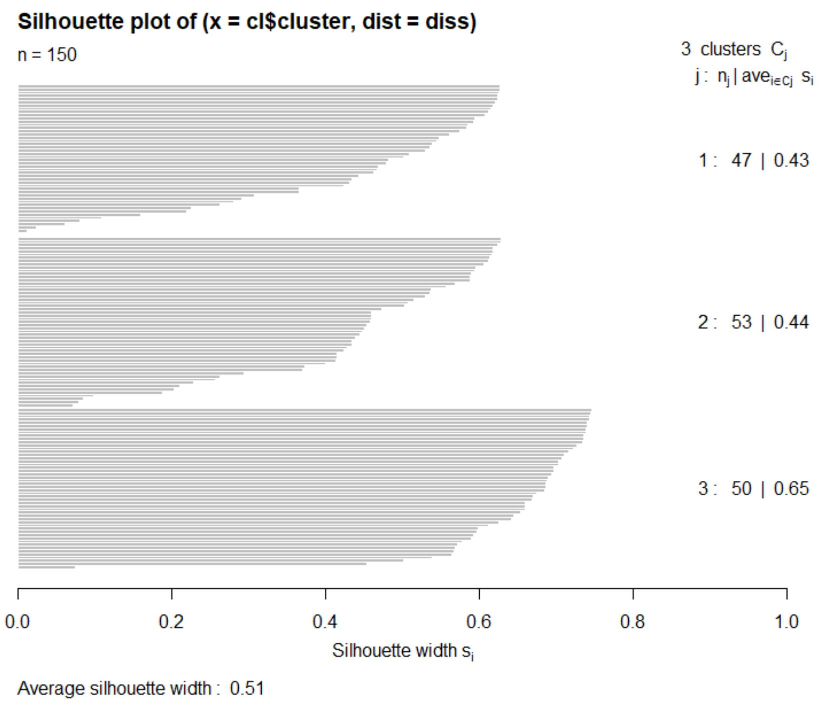
Thanks to this table, we can see that the specie *Setosa* is perfectly detected by the K-Means algorithm (it is the block of points on the left). On the contrary, approximately a quarter of the 2 other species are attributed to a wrong cluster by the K-Means algorithm.

8. Choose a solution that seems good enough for you and use the following code lines to compute the silhouette index. Comment.

We want to evaluate the quality of our clustering. To do so, we use the silhouette index. We can compute this index using the function *silhouette()*. But first of all, we have to calculate the distances between all the observations of the dataset with the *daisy()* function.

```r
library(cluster)
diss=daisy(X)
s = silhouette(cl$cluster,diss)
plot(s)
```

We obtain the following graph:



This graph tells us that there are 47 observations in the first cluster, 53 in the second one and 50 in the last one (we could already read it in the contingency table).

What is interesting in this graph is the information about the silhouette width. We can read the average silhouette width of our clustering, the average silhouette width per cluster and the silhouette width attributed to every observation. The silhouette index can take its values between -1 and 1. A silhouette width close to 1 means that our clustering is good. On the contrary, a negative value means that the quality of the clustering has to be questioned.

We see that the 3rd cluster (which is the one representing the points on the left of the K-Means representation) has a high silhouette index value. It is more likely to be well represented. On the other hand, the 2 other clusters contain observations which have silhouette width values very close to 0. This means that these observations might not have been attributed to the right cluster.

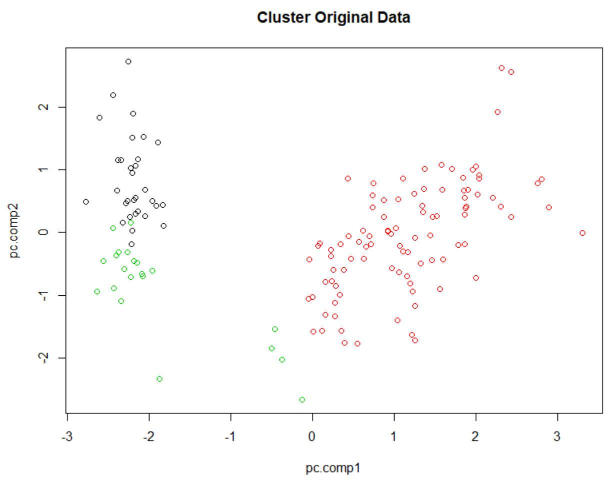
Nevertheless, the average silhouette width is equal to 0.51. The quality of our clustering is, on average, quite important.

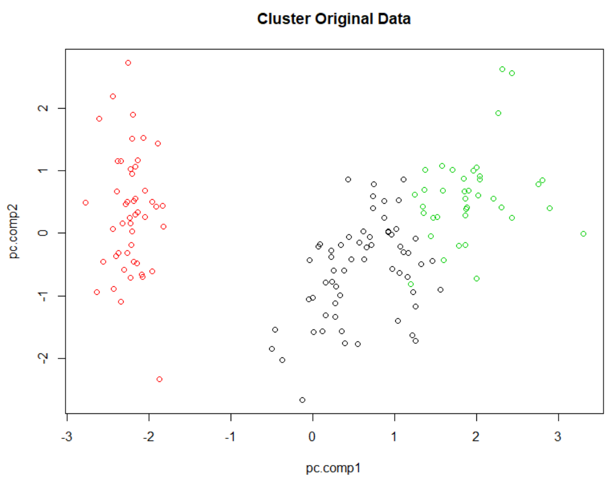
9. Start again questions 4) to 8) using the original data (data") instead of the projected ones. How different are the results? Explain the pros and cons of using the projected data or the original ones.

We want to see what our results would be if we keep the original data. Indeed, with 2 dimensions we can only explain 95% of the variance of our data. That is not bad, but we still are losing information.

We use our K-Means algorithm with our original data then we project the result in the 2 dimensions space, so we can visualize the result correctly:

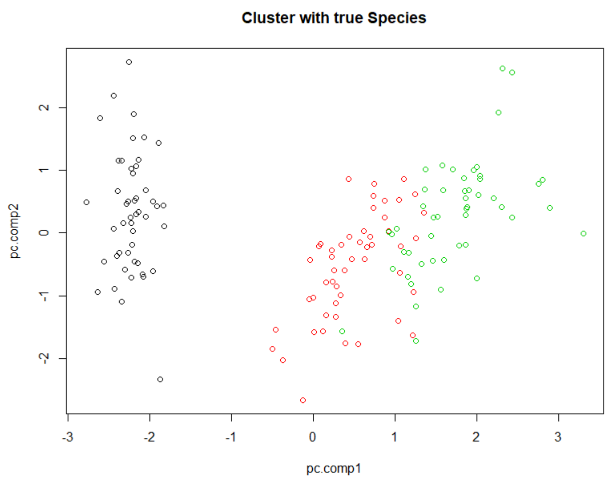
```
cl=kmeans(tab,3)
plot(pc.comp1, pc.comp2,col=cl$cluster,main="Cluster Original Data")
points(cl$centers, pch=16)
```



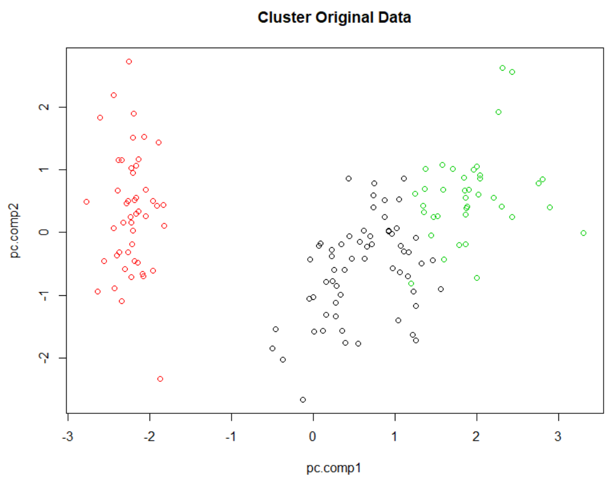


First, we notice that the random choice of the points at the beginning of the algorithm keeps being a source of error (figure on the left). Nevertheless, it seems like we get better clusters than with the projected data when the random points are well chosen (figure on the right). Indeed, some points are attributed to a cluster while they are "lost" in the middle of another cluster! This would not have been possible with the previous method considering how the algorithm is working.

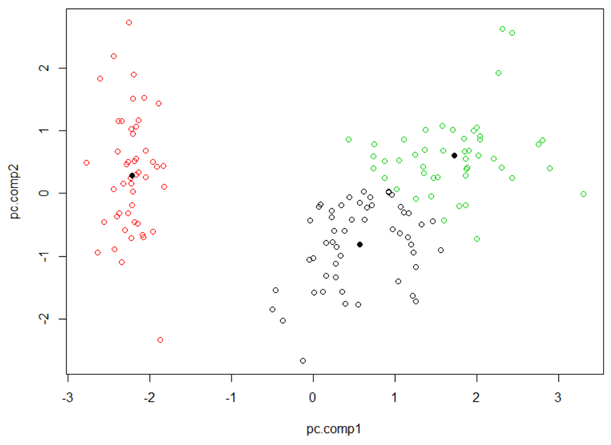
We can compare the 2 methods and the projected labels below:



Visualization of the projected labels



Visualization of the k-means clustering using ***original*** *data*



Visualization of the k-means clustering using ***projected*** *data*

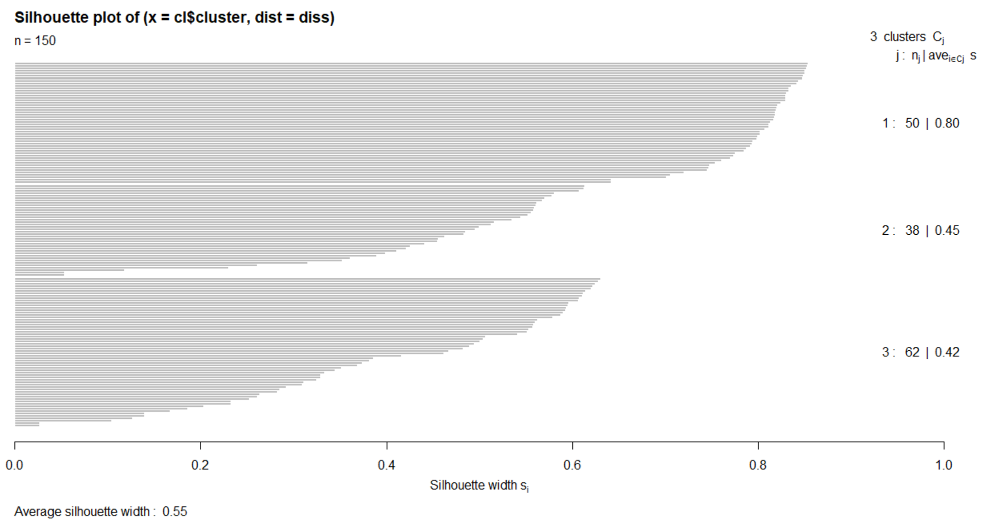
The difference isn't obvious at first glance. We decide to calculate the contingency table so we are able to see how much using the original data increases our precision:

```
table(eyeclass,cl$cluster)
eyeclass      1  2  3
setosa       50  0  0
versicolor    0 48  2
virginica     0 14 36
```

Like with projected data, the algorithm is able to construct a cluster that perfectly represents the *Setosa* specie. The main difference is that the observations of the *Versicolor* specie are almost all regrouped in the second cluster. Only 2 of them are placed in the wrong cluster – while there were 11 of them with the previous method.

The last step of our clustering evaluation is the calculation of the silhouette indexes:

```
library(cluster)
diss=daisy(tab)
s = silhouette(cl$cluster,diss)
plot(s)
```



The average silhouette width of our 3 clusters, which is equal to 0.55, is higher than previously (0.51). The silhouette index doesn't consider the changes in the clustering as important changes.

The most important change is the value of the silhouette index for the cluster containing the observations of *Setosa.* The index value jumped from 0.65 to 0.8. That could be surprising, considering that this cluster contains the same points as in the clustering which uses projected data. This shows how a change in one cluster can impact the silhouette value of the other clusters.

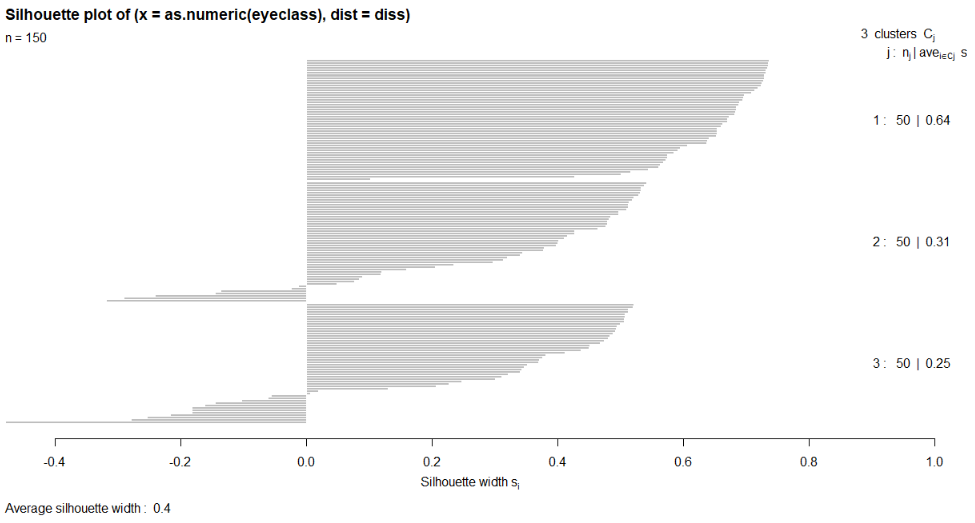
In the end, we get a better clustering using the original data instead of the projected ones. This can be explained by the fact that we are losing some information when we project our data in a 2-dimension space (5% of the variance isn't represented in this space). To use the original data seems to be a better option. Nevertheless, the calculation of the clusters is heavier when you use the original data. We don't feel it in this example as we have a relatively small dataset. But if you consider that we may have a dataset with dozens of variables and millions of observations, you can have a glimmering of the complexity of the calculation with not projected data (even more if you have to repeat different clusterings).

To conclude, to use projected data is more interesting if we have a huge amount of data. Else, you can use the original data.

BONUS: the limits of the interpretation of the silhouette index

We propose to plot the silhouette graph of the labels themselves:

```
library(cluster)
diss=daisy(tab)
s = silhouette(as.numeric(eyeclass),diss)
plot(s)
```



What do we notice? The silhouette indexes are lower than with the calculated clusters, especially the indexes of the second and the third cluster (which represent *Virginica* and *Versicolor*). The index is even telling us that some observations (the ones with negative values) should not be placed in their current category. Indeed, the silhouette index favors spherical clusters. As a consequence, this index can't consider the "real" clusters as good clusters.

This reminds us that the silhouette index has to be interpreted carefully.

B. Hierarchical clustering on the Iris data set

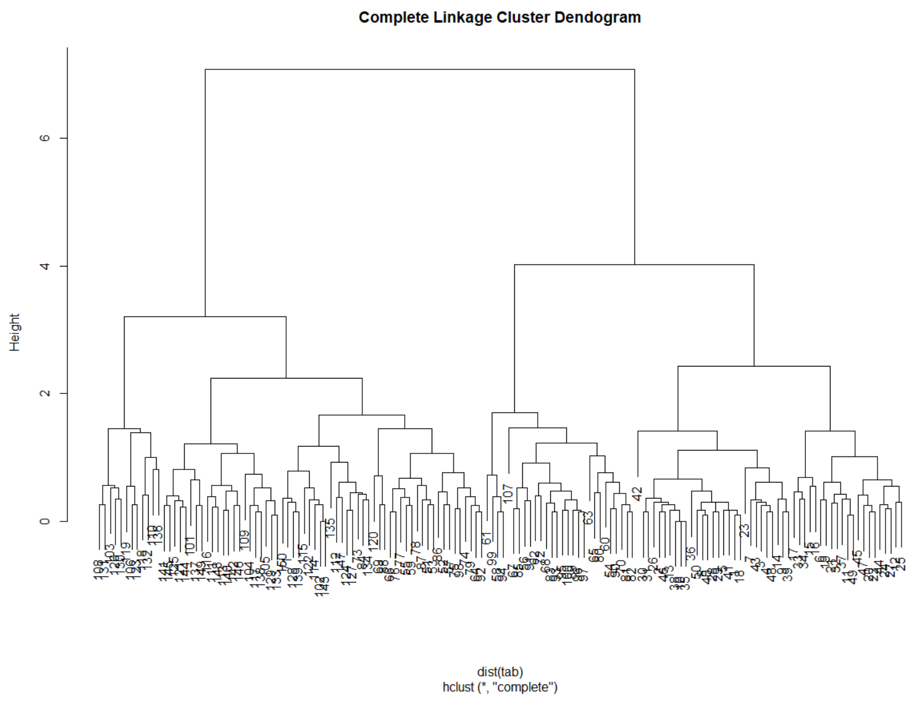
1. The command hclust(…) can be use for HCA in R. Use the help from R to determine how to use this function with your data and with what parameters.

We use HCA thanks to the *hclust()* function. This function takes the distances in our data as first parameter (we can get it using *dist("our data")*) and the used method as a second parameter. We can either choose to use the complete method, the average method, the min method … We should choose our method according to our dataset (that's what we'll see in this part).

2. Execute an HCA using a complete linkage, and display the result using the command plot.

We plot the result of our HCA with the complete method:

```
hIris=hclust(dist(tab),"complete")
plot(hIris)
```

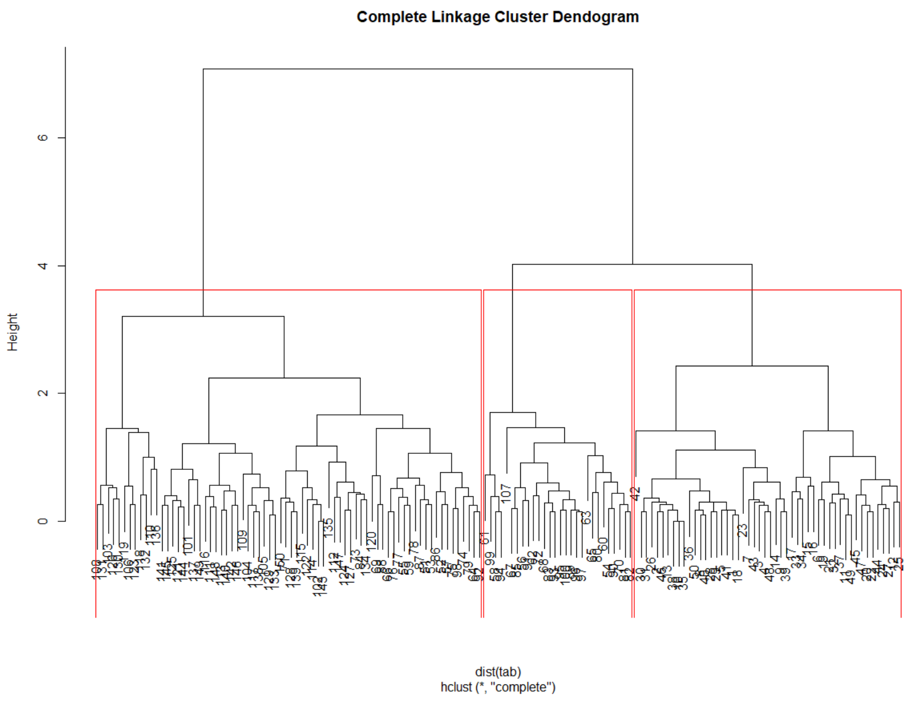


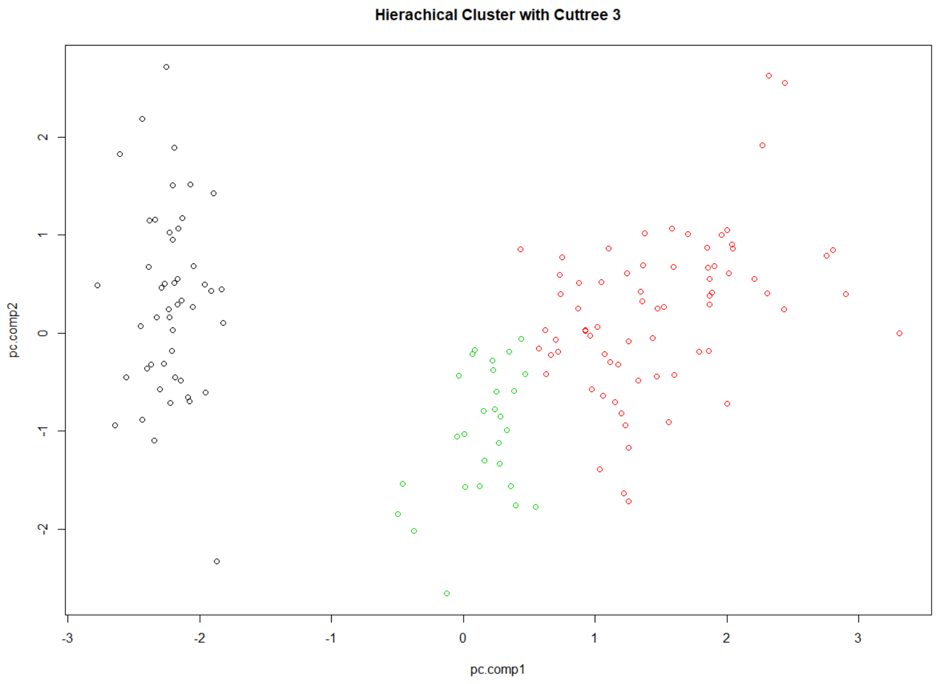
We obtain a tree constructed on the distance existing between the observations. The problem is that we can't visualize the 3 desired clusters easily.

3. The command cutree(clust,K) can be used to prune a tree so that only K branches are left. Use this command on your HCA result to get the final partition with 3 clusters.

```
hIriscut=cutree(hIris, 3)
rect.hclust(hIris,k=3,border="red")
table(eyeclass,hIriscut)
plot(pc.comp1, pc.comp2,col=hIriscut,main="Hierachical Cluster with Cuttree 3")
```

Thanks to the *cutree()* function we achieved to get the final partition with 3 clusters. We can either visualize these clusters on our dendrogram with *rect.hclust()* or visualize them in our 2-dimension space:





We're going to pay attention to the last graph as it is easier to analyze. Like with the K-Means algorithm, the HCA manages to recognize the cluster on the left. Nevertheless, it has hard times recognizing the right clusters for the rest of the data.

4. Display the contingency table comparing your partition with the theoretical labels. Comment.

```
table(eyeclass,hIriscut)
hIriscut
eyeclass      1  2  3
  setosa     50  0  0
  versicolor  0 23 27
  virginica   0 49  1
```

The cluster 1 is representing the *Setosa* family perfectly.

Almost every *Virginica* observation is in the second cluster. Unfortunately, there are also a lot of *Versicolor* observations that have been put in this cluster (half of them).

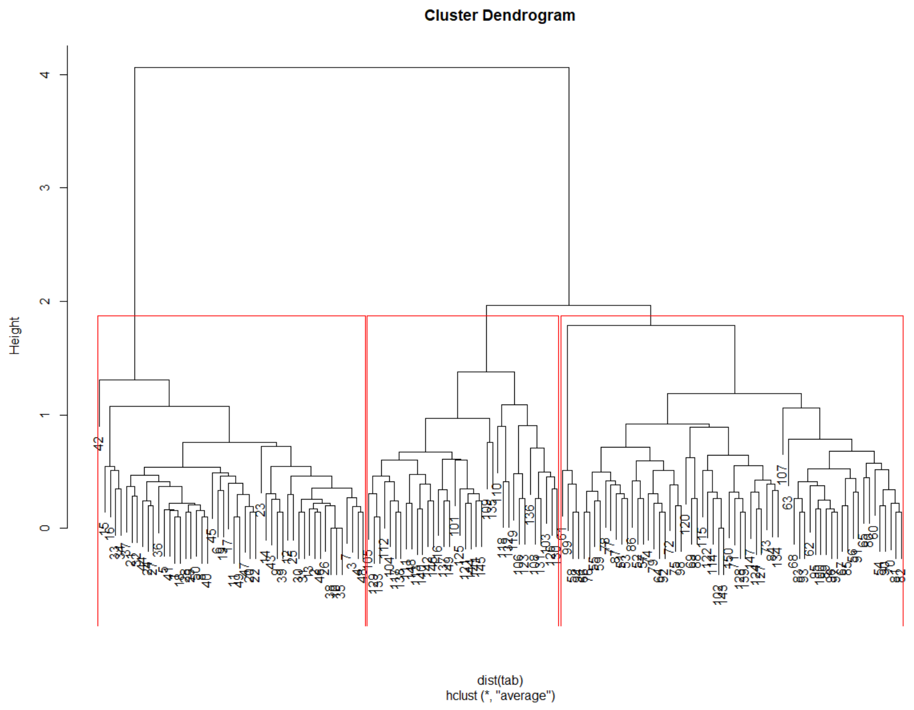
This clustering isn't satisfying at all.

Is it specific to the HCA or can it be corrected using another method than the "complete" method. ?

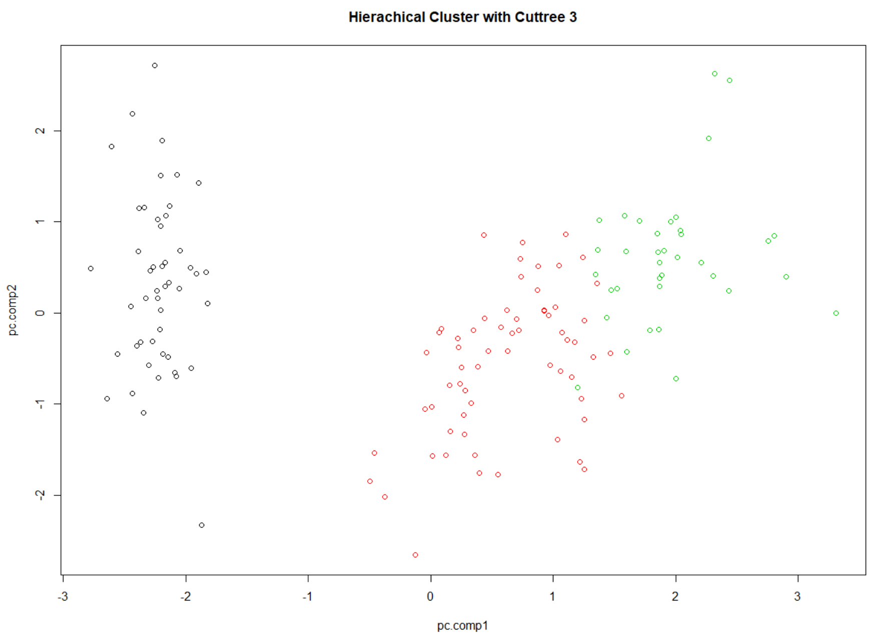
5. Do again questions 2) to 4) using a HCA with an average linkage. Compare the results. Which linkage seems to work best ?

We decide to try to use the HCA with the "average" method. We precise it in the *hclust()* function. Then we display what we need to analyze the clustering.

```
hIris=hclust(dist(tab),"average")
plot(hIris,main="Average Linkage Cluster Dendrogram")
cut=cutree(hIris,3)
rect.hclust(hIris,k=3,border="red")
table(eyeclass,cut)
plot(pc.comp1, pc.comp2,col=cut,main="Hierachical Cluster with Cuttree 3")
```



First, we see that it is easier to distinguish the 3 main branches on the dendrogram.



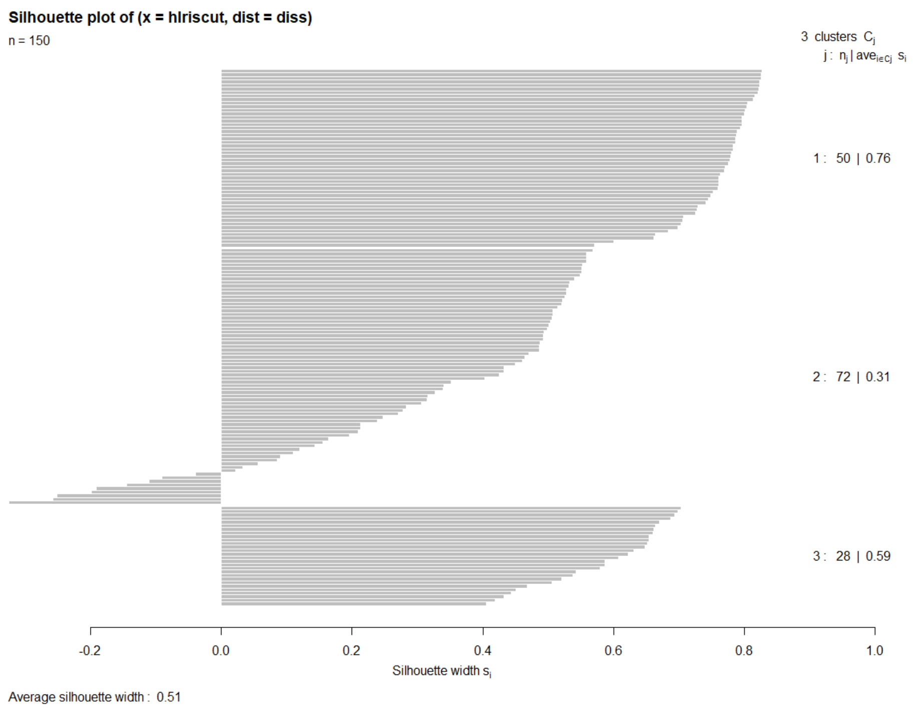
In our projected space, we notice that the different clusters are closer to the ones we are expecting to obtain. The division of the right block of points is more precise than with the complete method.

```
cut
eyeclass      1  2  3
setosa       50  0  0
versicolor    0 50  0
virginica     0 14 36
```

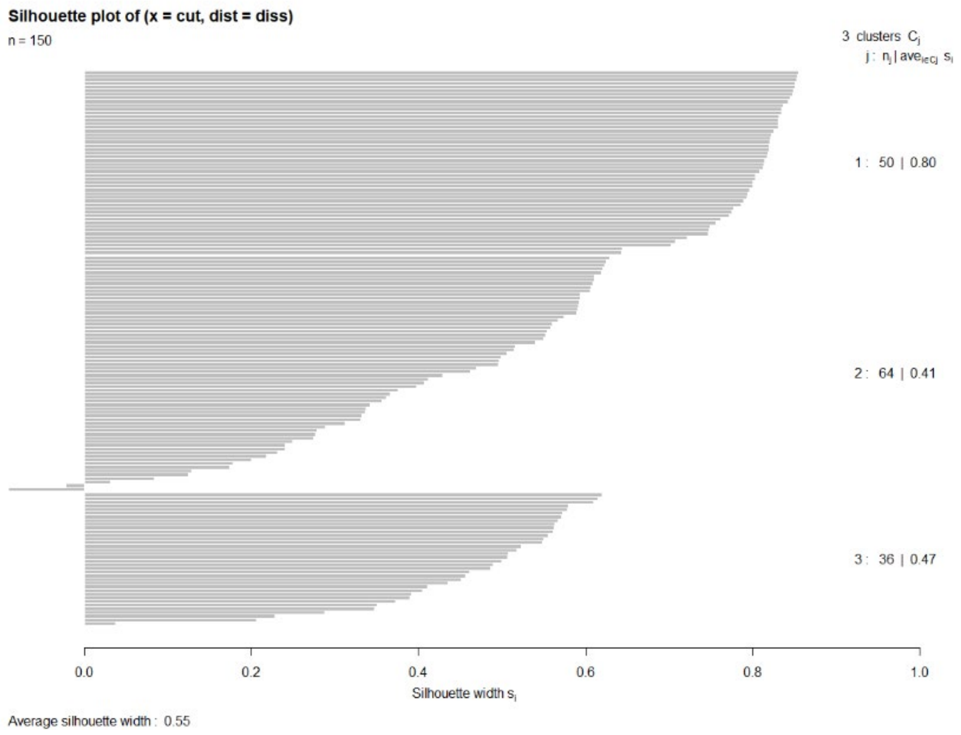
With the table of contingency, we notice that the first cluster perfectly represents the *Setosa* family. Furthermore, the second cluster contains every single observation of the *Versicolor* family. Finally, the third cluster contains approximately 70% of the observations of the *Virginica* family.

We can easily say that the average method is more appropriated than the complete method for our dataset.

6. Compute the silhouette index for both partitions (complete linkage and average linkage). Does it confirm your results from question 5) ? Comment.



Silhouette indexes for "complete" method



Silhouette indexes for "average" method

We notice that the average silhouette width is slightly higher with the "average" method. However, the difference isn't that significant. The most important improvement is the fact that we have much fewer negative values with the "average" method.

The silhouette graph confirms what we saw previously, even if it is not so significant.

C- Optimal cluster number in exoplanet data

The purpose of this exercise is to determine the optimal clustering number in exoplanet data.

1. Open the le exo4_atm_extr.csv and remove the last column containing the labels.

The easiest way to declare the column as null. But as we saw in Part A, it is important to keep the data to compare our clustering results with the true results.

```
  labels=tabexo$Type
  tabexo$Type=NULL
```

2. Write down the different properties of the Davies-Bouldin index.

At the beginning we were supposed to use the Davies-Bouldin Index, unfortunately it did not work when we tried to install the packages (with the commands below) it was unable to install one package on R.

```
Install.packages(“clusterSim”, dependencies=T)
Index.DB(data, clusters)$DB
```

So, we will use another index method called Calinhara

```
Install.packages(“fpc”, dependencies=T)
Calinhara(data, clusters)
```

Here is the formula and the different properties of Calinhara index:

This index is defined as follows:

$$CH(k) = \frac{\frac{B}{k-1}}{\frac{W}{N-k}}$$

Where B is the sum of squares among the clusters W is the sum of squares within clusters k is the number of clusters

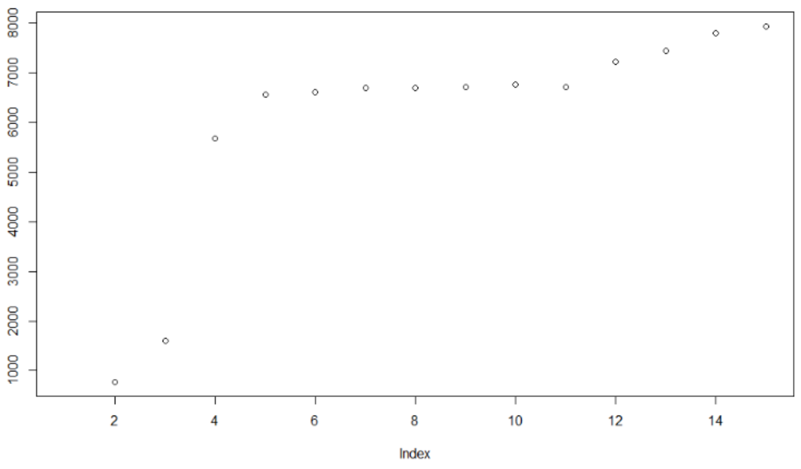
Properties of the CalinHara-Index

- Not normalized
- Better when higher
- With balanced clusters, the CH index is generally a good criterion to indicate the correct number of clusters.

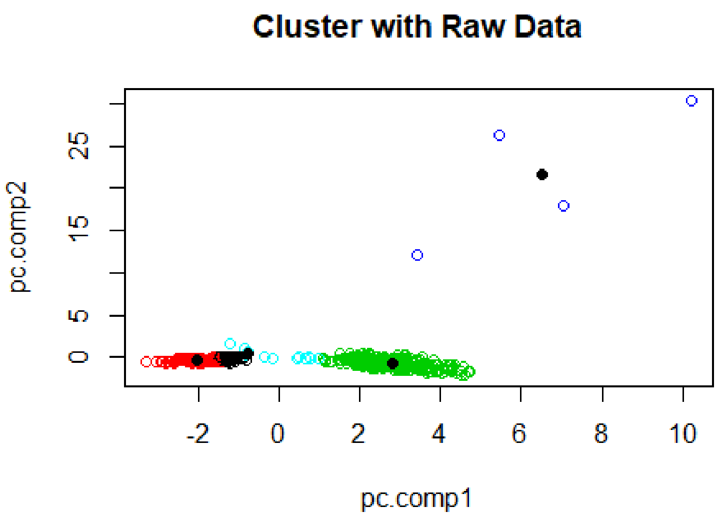
3. Using exercise A and the course, propose a method that can help guessing the right number of clusters in this dataset based on the Davies-Bouldin index (index.DB(data,clusters)) and the K-Means algorithm.

So the method we use is as follow: we process an average of 500 K-Means per Cluster index (ranging from 2:15), and for each measure we score with Calinhara method each cluster index. Here is the algorithm:

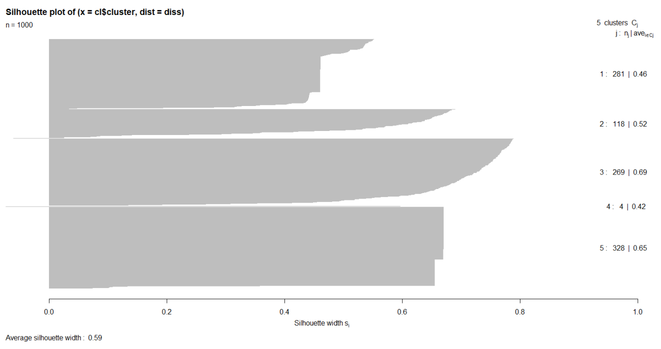
```
library(fpc)
count=0;
num=0;
means=numeric();
for (i in 2:15){
  count=0;
  mean=0;
  num=0

while (count<500)
{
    cl=kmeans(tabexo,i)
    count=count+1
    num=num+calinhara(tabexo,cl$cluster)
}
  mean=num/500
  means[i] <- mean
}
plot(means)
```



It gave us this graph, let's analyze it! Between [5-11] index cluster values we see a stall in the Calinhara score. After 12 cluster the score increase again, this is easily explained, the less data in the cluster, the better the score. We can also see that with 2-3 clusters the score is terrible. We can conclude by this graph that the ultimate cluster score must be 5 according to the Calinhara score. Let's visualize these clusters using a K-Means algorithm then:



The clustering above with the original data is unreadable on this plot, but we can't say that it is bad.



When we process the silhouette, we have an average score of 0.59, which is not too bad. Some values are negative though.

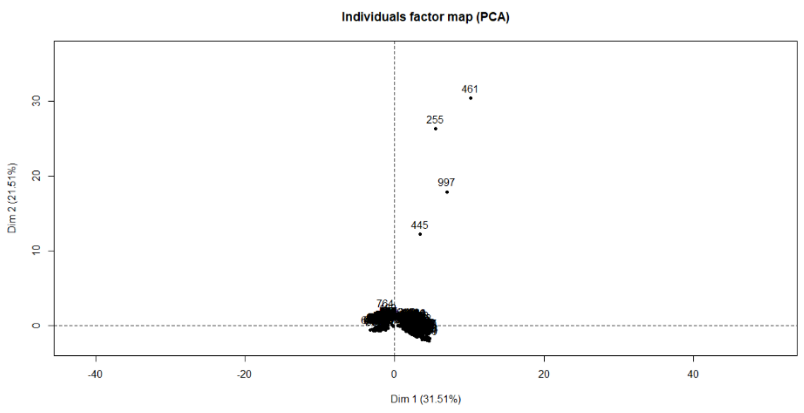
/!\ The number of data is too high so we have to use the *windows()* function before plotting. Otherwise we can't visualize the silhouette index of the observations indidually. /!\

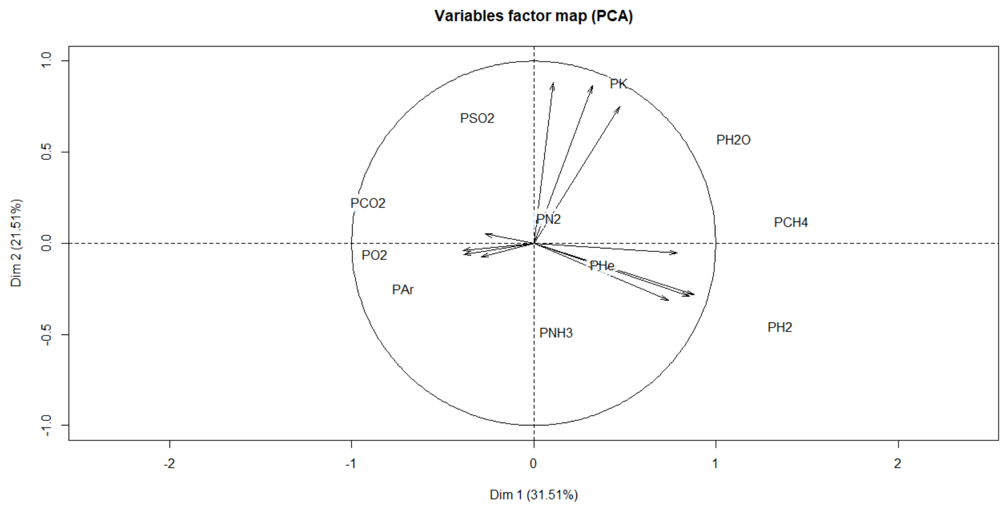
Let's complete our observation with the consistency table:

```
labels
      d   g   i   l   r
  1   0   0   0   4   0
  2  13   0   1   0 460
  3   0   0  95   0   0
  4 128   0  30   0   0
  5   2 267   0   0   0
```

The planets of the types l are perfectly represented by its cluster. Planet r & g do have all their values however some misplaced values are noticeable. d & i planets are splitted among many clusters.

We'd like to get rid of the 4 observations that make our clusters unreadable. We know that the *PCA()* function gives the index of the unusual observations; furthermore, using this function will also give us some information about the precision of our 2-dimension space. So that's what we do:





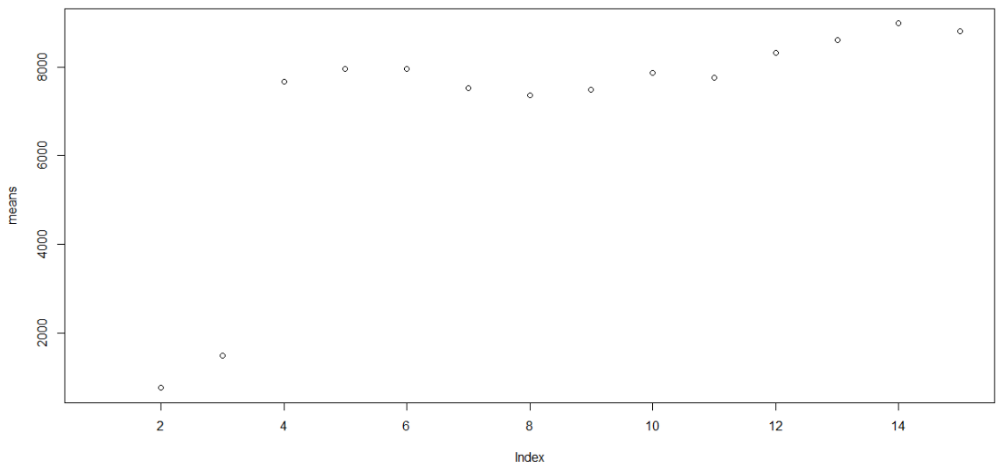
We notice two important things. First, we get the indexes of the observations we want to get rid of: 445, 997, 255 and 461. Then we also see that we can only explain 53% of the variance of our dataset in a 2-dimension space! We are losing plenty of information and we have to keep that in mind for the conclusion of our analysis.

Now we remove the values that we found:

We better understand the values now, the quality of the 2-dimensional PCA is weak: 31.51% + 21.5% = 53%, which means that 47 % of the information is lost. Let's then erase the following points which correspond to the L type of planets.

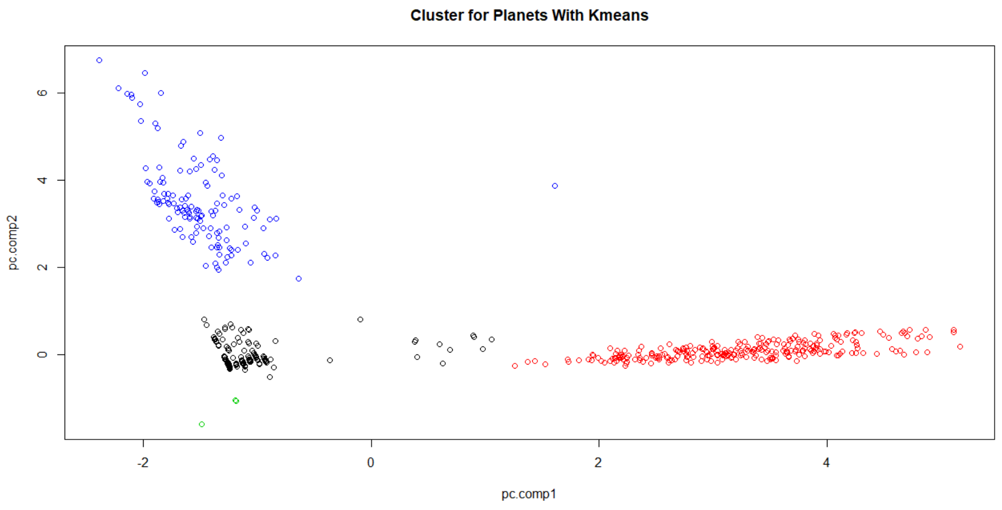
```
tabexo2=tabexo[-c(445,997,255,461),]
```

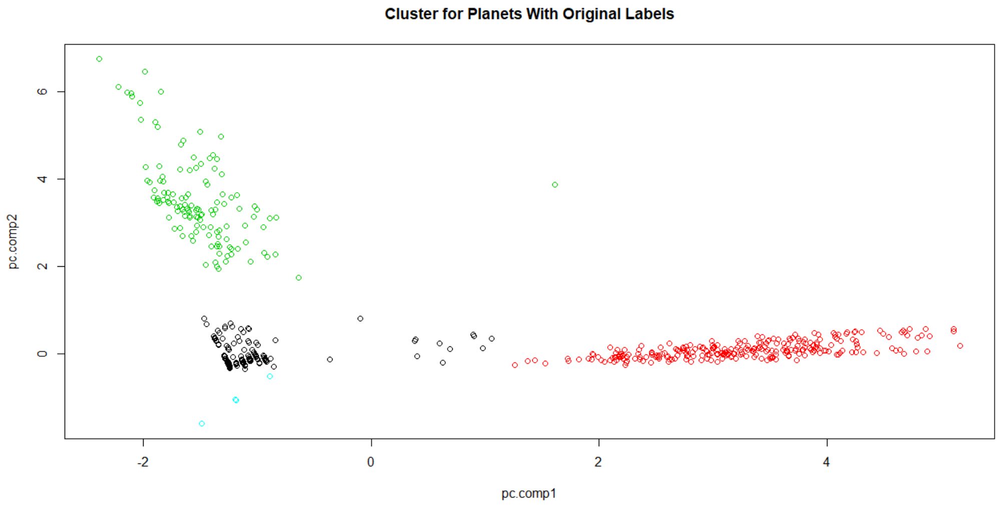
We modified our dataset by removing 4 really extreme values. We have to be careful and to recalculate the number of clusters we want to find. We get the following graph:



We notice that the right number of clusters changed. It seems like we are expecting to find 4 clusters instead of 5 previously.

The K-Means algorithm, used on projected values, gives us the following graph (we compare it to the projected labels):





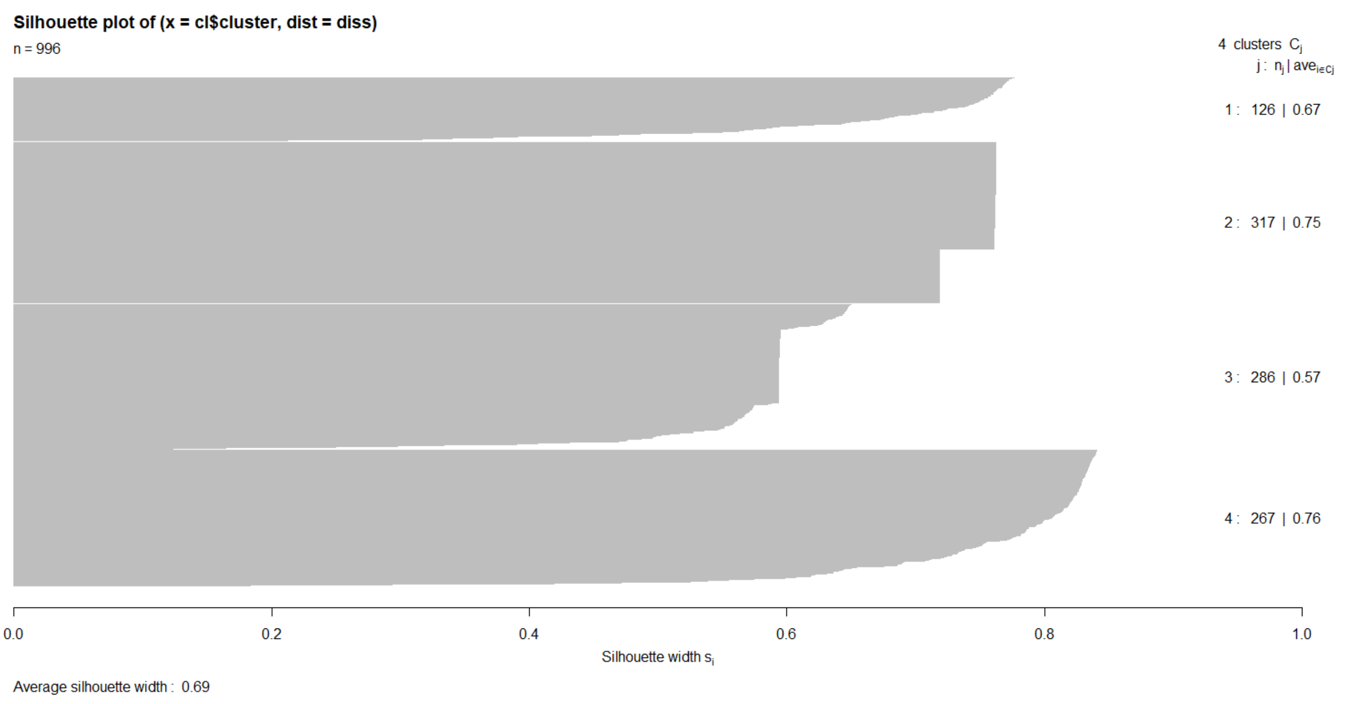
The clustering is quite satisfying at a first glance. It seems like just one observation has been placed in the wrong cluster (bottom-left). However, we have to be careful. It is possible that we are missing some information by just analyzing this graph.

Let's complete our observation with the consistency table:

```
labels
    d   g   i   l   r
1 143   0   0   0 143
2   0 267   0   0   0
3   0   0   0   0 317
4   0   0 126   0   0
```

The planets of the types g and i are perfectly represented by their cluster. Every value of the type d are in the first cluster, but a huge part of the r-type planets has been placed in this cluster too. Apparently the "point" we saw in the 2-dimension representation was the agglomeration of several observations.

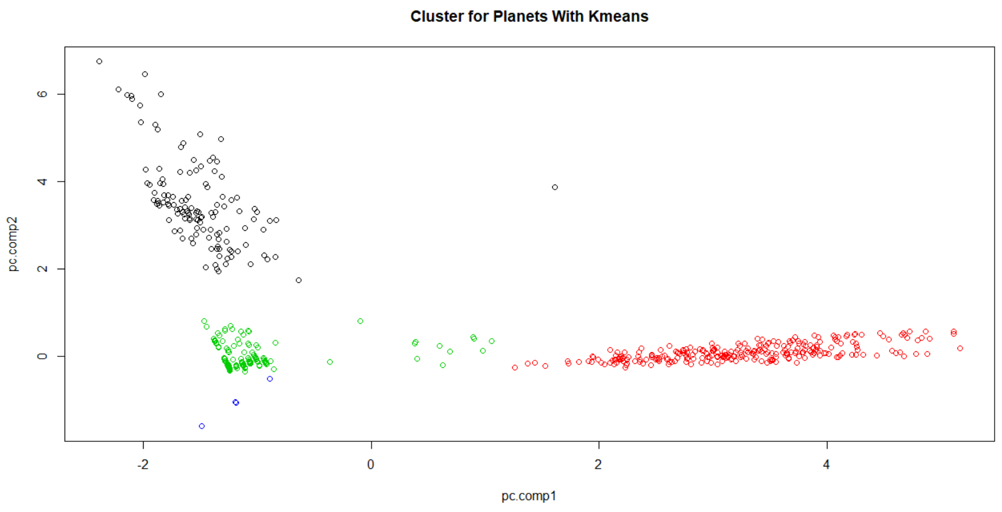
We also notice that there are no l-type planets left in our dataset. The 4 values that we removed were all attached to l-type planets.



The average silhouette index is equal to 0.69. That's very satisfying. The silhouette index doesn't show that the cluster 1 is containing values which shouldn't be there (there is no negative value of silhouette index). This can be explained by the fact that the observations which have been placed in the wrong cluster are close to the other values present in this cluster.

We are satisfied of our clustering but we may get an even better one if we were using the original data instead of the projected ones. The dataset still is quite small so we can afford it.

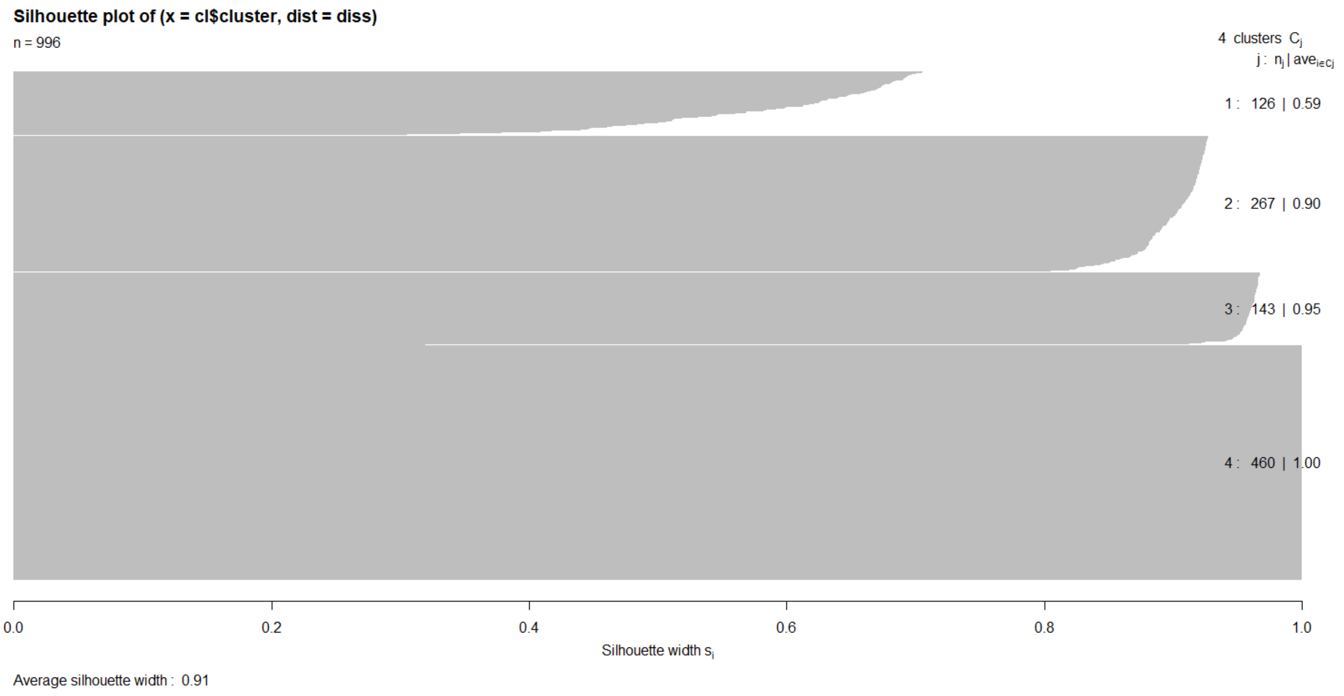
Here is the K-Means of the original data, projected in the 2-dimension space:



The clusters that we obtain seem to be exactly the same as expected. We know that we can be fooled by what we see, so we display the table of contingency to be sure:

```
labels
  d   g   i   l   r
1   0   0 126   0   0
2   0 267   0   0   0
3 143   0   0   0   0
4   0   0   0   0 460
```

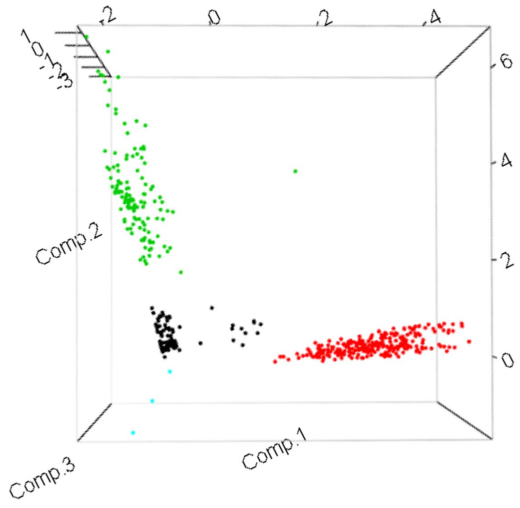
Every cluster is perfectly matching a specific type of planet. Our clusters are perfect.



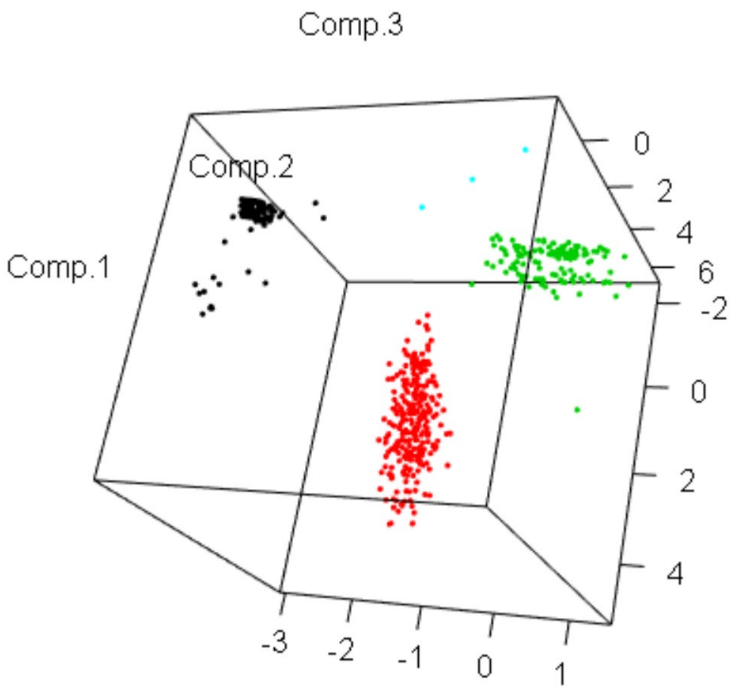
Even if we didn't have the type of each planet to see if our clustering is good, we would have been able to say that are reaching the target as the silhouette index is extremely close to 1. Indeed we have an average silhouette index of 0.91!

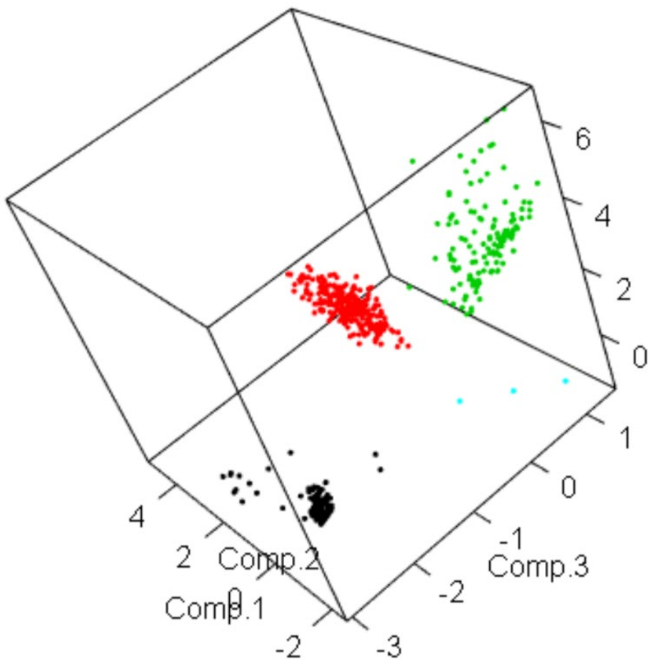
Let's go further, by doing a 3D PCA, this way we'll be able to add another layer of information, which in this case is not exactly well represented by our 2D PCA. We've use the plot3d() function from the rgl library, to plot the output of the PCA.

```
library(rgl)
pca=princomp(tabexo, cor=TRUE, scores=TRUE)
plot3d(pca$scores, col=as.numeric(labels))
```



On the above figure, we've changed the angle so that we can see almost the same graph as we had in 2D before.





With those angles we can clearly see the Euclidian distances between the clusters. Especially the distance between the blue clusters and the rest.

⚠ In the future, we should not try to plot the values to judge the effectiveness of our clusters, and only rely on the mathematical scores. Why ? Because we can't see further than 3 dimensions (4D if we try hard). And after all clustering is supposed to be unsupervised.

To conclude: We can surely say that clustering is not an easy problem. Evaluation (or "validation") of clustering results is as difficult as the clustering itself, it often requires human analysis: For this process to be correct, it involves synthetizing different scores provided by many tools. Relying on a single tool is not optimal and may lead to incorrect or largely false clusters. If the data can be well projected using a PCA in 2D or 3D, it is relevant to visualize the clusters, which gives us a sense of what the clustering process has done. However, the visualization often falls short when too many dimension are involved.