Partition Around Medoids (PAM) and Silhouette plots

By Vinícius Manvailer

In Lab 5 we learned about clustering techniques and distances in order to visualize our dataset in different ways. This lab introduces a new clustering technique based on the k-means algorithm. While hierarchical clustering techniques progressively group data points that are closer to each other, k-means-based algorithms uses ***medoids*** to group data points. These medoids represents the centers of gravity of your dataset, that is, where your data points are more grouped together (*i.e*. where they are more similar to each other).

In the first step of the analysis, a *k* number of medoids are introduced into the dataset – you must choose a priori how many clusters you want and I will explain later how to select the best number. Data points closest to each medoid are assigned as belonging to that cluster. Now there are two alternating steps: In the first step, the center point (mean) of all data points in a single cluster is calculated, and the medoid shifted to that point. Next, the distances of all points to the nearest medoid are recalculated and the data points reassigned to their closest medoids. These alternating steps continue until medoids stop moving and data points stay the same. **The objective of the function is to reduce the sum of the total within distances in all clusters**. The within distance is the distance of each point to its respective medoid.

The PAM function is considered to be a more robust version of the k-means because, contrary to k-means the PAM function does not introduce medoids randomly. Instead, it uses data points as medoids. This makes it less sensitive to outliers. This function also requires you to input the number of cluster (*k*) a priori and then it proceeds as follows: The first medoid is assigned as the data point that has the smallest distance to all other data points, so it is in the center of everything. The next medoid is introduced in a way to reduce the total distance of each data point to their nearest medoid. The third medoid is the same and so on. This is called the building phase. After that, different data points are tested as medoids and if a different point reduces the total within cluster distance the algorithm swaps the medoid with that point. All possible combinations are tested and thus only one solution is possible. This is also more advantageous to the k-means because you can have *n* number of possible solutions depending on where the medoids start.

Finally we can visualize our results by performing a discriminant analysis. Thus, after each data point are assigned to their respective cluster, a discriminant analysis is performed and the new scores of the data points are plotted – i.e. the new coordinates after rotation.

1. Partition Around Medoids in R

Let's use the Iris dataset as an example. Start by setting yourself as usual and importing the dataset "iris.csv" from the website. Check if the import was successful.

```
iris <- read.csv("iris.csv")

head(iris)

str(iris)
```

Now let's plot a PCA to check the general distribution of data and see how the data is organized.

```
pcairis <- princomp(iris[1:4])

plot(pcairis$scores,col=rainbow(3)[iris$Species], asp=1)
```

While we are at it, lets also plot the vectors (loadings) to understand what is driving the differences among observations. For that we will use the vegan package.

```
#install.packages("vegan")

library(vegan)
```

```
veciris <- envfit(pcairis$scores[,1:2], iris[,1:4], permutations =0)

plot(veciris, col="black")
```

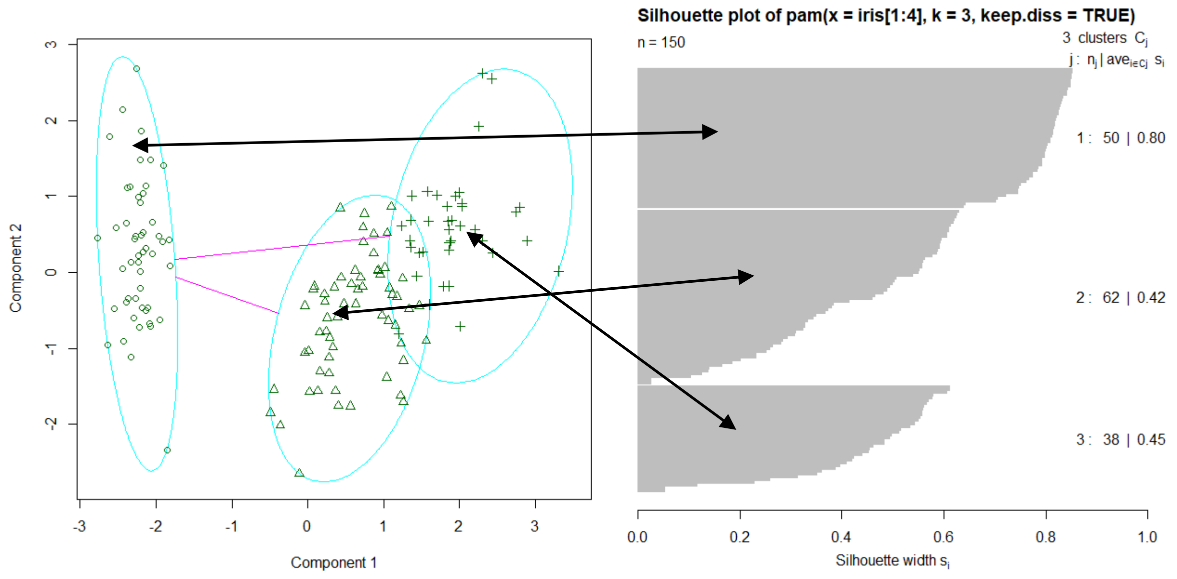
Now let’s install and load the *cluster* package. We will apply the Partition Around Medoids function on Sepal and Petal measurements and check if it will group the observations according to the species. After executing the plot command, click on the console and hit the Enter key to see the plots. In doubt just look at the console message.

```
install.packages("cluster")

library(cluster)

pamiris <- pam(iris[1:4], 3, keep.diss = TRUE)

plot(pamiris)
```

For the pam function we input our variables (columns 1 through 4) and the number of cluster we want as an output, in this case, we want three to check the performance of classification of the function. Now check the first graph to see how observations were grouped. Is there an overlap between the different groups? Can you find a clear structure in your data? This is a discriminant analysis and shows how similar/dissimilar observations are.

Now let’s take a look at the silhouette graph. How well is this data set structured? Check the average silhouette value/width for the whole plot. You can check the Kaufman and Rousseeuw (1990) table to tell if data is well or poorly structured. Also you can see how well structured each cluster is by looking at their individual silhouette values/widths.

S_i	**Proposed Interpretation**
0.71 to 1.00	A strong structure has been found.
0.51 to 0.70	A reasonable structure has been found.
0.26 to 0.50	Structure is weak and could be artificial. Try other methods on this database.
-1 to 0.25	No substantial structure has been found.



The Si value is calculated for each observations and the left, center, right clusters on the ordination on the left correspond to the top, middle, bottom bars on the silouett plot on the right. High bars for individual observations mean that the cluster is well separated from the rest. So the first group with the highest Si values (an average of 0.8) has the best support for a clustered structure.

Now let’s see how well the algorithm classified the clusters compared to the original species identifications. For visualization, we are using a regular pca and coloring by species or cluster. To do that we will extract the clustering information and attached it to our original table.

```
iris <- cbind(iris, pamiris$clustering)

colnames(iris)[6] <- "cluster"

plot(pcairis$scores,col=rainbow(3)[iris$Species], asp=1) #Species coded

plot(pcairis$scores,col=rainbow(3)[iris$cluster], asp=1) #Cluster coded
```

You can see that the result is a fairly reliable grouping with only a few observations classified differently, given the criteria that you want three species. Try different numbers of pre-determined species groups (for example 2 or 4) to see how the silhouette and Si values change.

2. Better visualization with GGPLOT

We have seen in previous classes the power of ggplot to visualize data. It provides clean graphs that are easily customizable and more visually appealing. We can plot our results using the ggplot package following the tutorial below.

2.1 Discriminant Analysis plot with probability ellipsis

Let’s start by installing and loading the *ggplot2* package and the *ggfortify*. The later allow us to visualize multivariate analysis using ggplot.

```
#install.packages("ggplot2")

#install.packages("ggfortify")

library(ggplot2)

library(ggfortify)
```

Now simply run the function below to obtain the graph. The first part of the code is the analysis. The arguments frame and frame.type draw a probability ellipsis around the clusters.

```
autoplot(pam(iris[1:4],3), frame=TRUE, frame.type = "norm")
```

You can even incorporate other analysis in the graph by using the code below.

```
autoplot(pamiris,

      frame=TRUE,

      frame.type = "norm",

      loadings = TRUE,

      loadings.colour = "black",

      loadings.label = TRUE,

      loadings.label.colour = "black")
```

You can also use the same function to plot the pca graphs we use to check our classification. One minor modification is needed: cluster values need to be factors. The code to do that is also below.

```
autoplot(pcairis, data = iris, colour = 'Species')
```

```
iris$cluster <- as.factor(iris$cluster)

autoplot(pcairis, data = iris, colour = 'cluster')
```

You can also add the arguments frame and frame.type on the pca plot to easily visually how the results change.

```
autoplot(pcairis, data = iris, colour = 'Species', frame=TRUE)

autoplot(pcairis, data = iris, colour = 'cluster', frame=TRUE)

autoplot(pcairis, data = iris, colour = 'Species', frame=TRUE, frame.type = "norm")

autoplot(pcairis, data = iris, colour = 'cluster', frame=TRUE, frame.type = "norm")
```

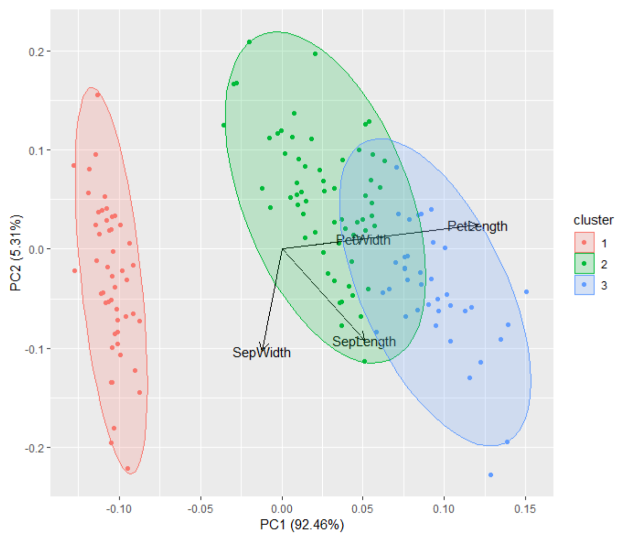
2.2 Silhouette plots with GGPLOT

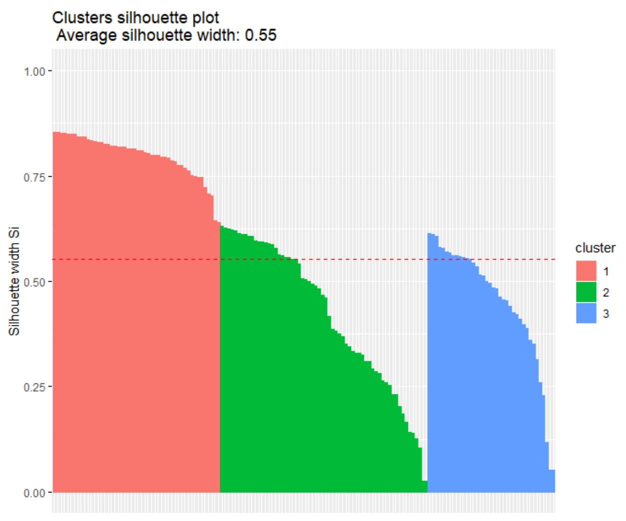
Finally, to plot our silhouette plot with ggplot we will need the *factoextra* package and we will use the function fviz_silhouette(). You can set labels=TRUE to better investigate which observation have low or even negative silhouette value.

```
#install.packages("factoextra")

library(factoextra)

fviz_silhouette(pamiris, label=TRUE)
```

You can find that value in the discriminant analysis graph by also placing labels on it.

```
autoplot(pamiris,

     frame=TRUE,

     frame.type = "norm",

     loadings = TRUE,

     loadings.colour = "black",

     loadings.label = TRUE,

     loadings.label.colour = "black",

     label= TRUE)
```


PetLength
PetWidth
SepWidth
SepLength
PC1 (92.46%)
PC2 (5.31%)
cluster


Clusters silhouette plot
Average silhouette width: 0.55
Silhouette width Si
cluster