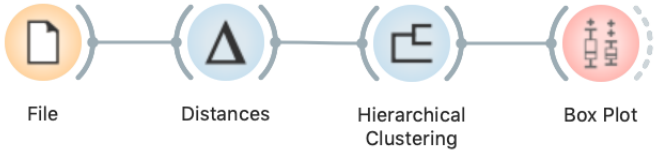
Animal Kingdom

Your lecturers spent a substantial part of their youth admiring a particular Croatian chocolate called Animal Kingdom. Each chocolate bar came with a card—a drawing of some (random) animal, and the associated album made us eat a lot of chocolate.

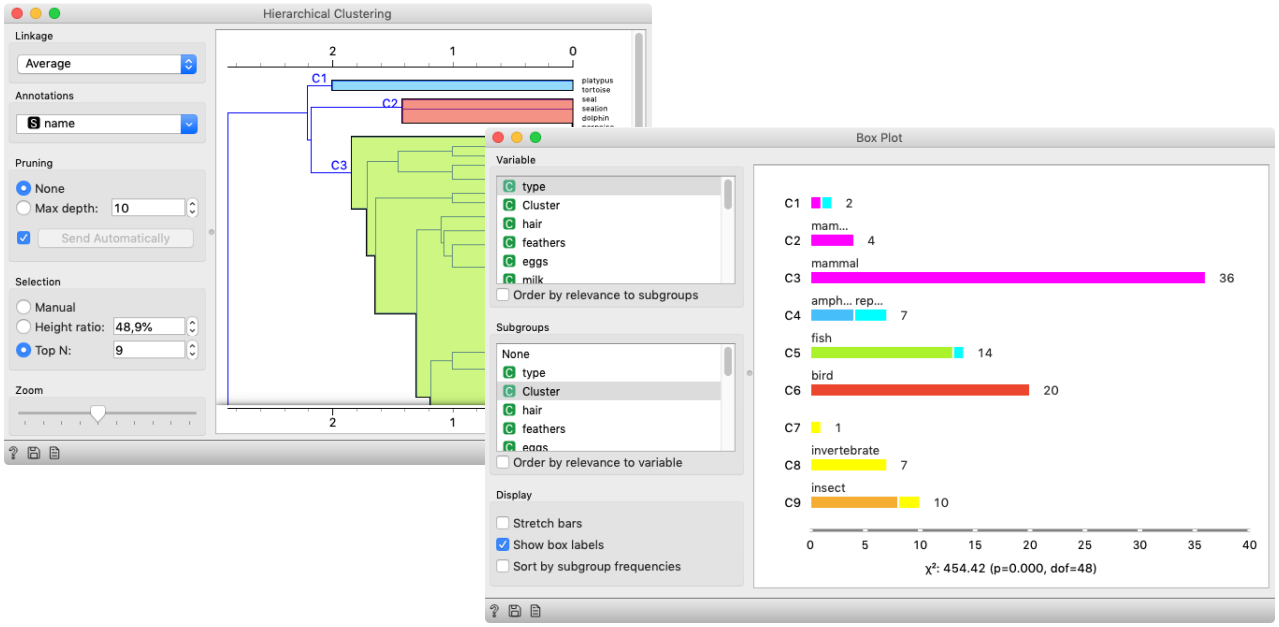
Funny stuff was we never understood the order in which the cards were laid out in the album. We later learned about taxonomy, but being more inclined to engineering we never mastered learning it in our biology classes. Luckily, there's data mining and the idea that taxonomy simply stems from measuring the distance between species.

Here we use the *zoo* data (from the documentation data sets) with attributes that report on various features of animals (has hair, has feathers, lays eggs). We measure the distance and compute the clustering. Animals in this data set are annotated with type (mammal, insect, bird, and so on). It would be cool to know if the clustering re-discovered these groups of animals.



Hierarchical clustering works fast for smaller data sets. But for bigger ones it fails. Simply, it cannot be used. Why?

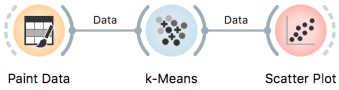
To split the data into clusters, let us manually set a threshold by dragging the vertical line left or right in the visualization. Can you say what is the appropriate number of groups?



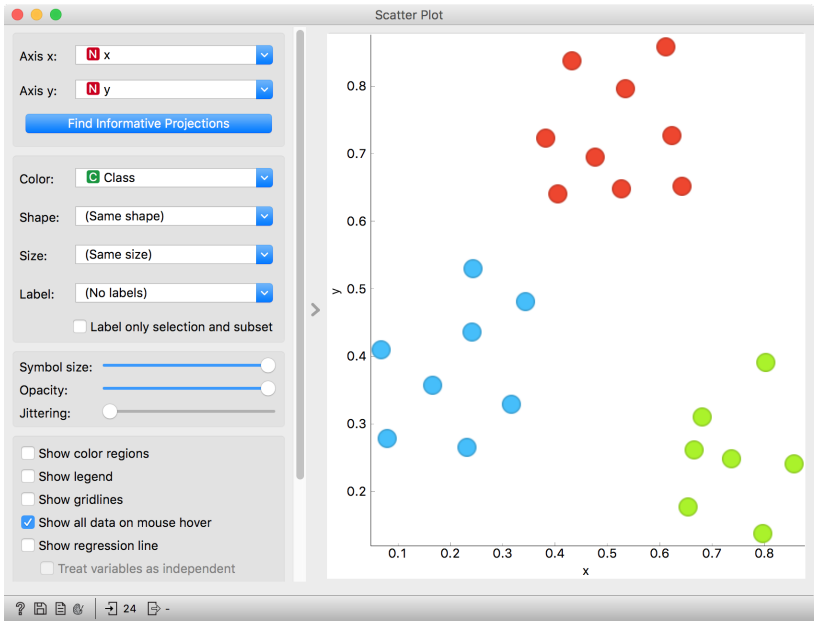
What is wrong with those mammals? Why can't they be in one single cluster? Two reasons. First, they represent 40% of the data instances. Second, they include some weirdos. Who are they?

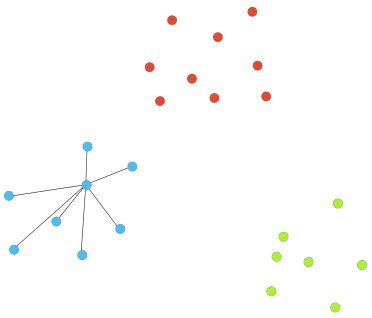
Silhouettes

Don't get confused: we paint data and/or visualize it with Scatter plots, which show only two features. This is just for an illustration! Most data sets contain many features and methods like k-Means clustering take into account all features, not just two.



Consider a two-feature data set which we have painted in the *Paint Data* widget. We send it to the k-means clustering, tell it to find three clusters, and display the clustering in the scatter plot.





Average distance A.

The data points in the green cluster are well separated from those in the other two. Not so for the blue and red points, where several points are on the border between the clusters. We would like to quantify the degree of how well a data point belongs to the cluster to which it is assigned.

We will invent a scoring measure for this and we will call it a silhouette (because this is how it's called). Our goal: a silhouette of 1 (one) will mean that the data instance is well rooted in the cluster, while the score of 0 (zero) will be assigned to data instances on the border between two clusters.

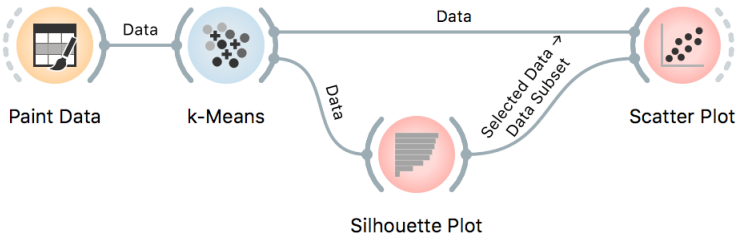
For a given data point (say the blue point in the image on the left), we can measure the distance to all the other points in its cluster and compute the average. Let us denote this average distance with A. The smaller the A, the better.

On the other hand, we would like a data point to be far away from the points in the closest neighboring cluster. The closest cluster to our blue data point is the red cluster. We can measure the distances between the blue data point and all the points in the red cluster, and again compute the average. Let us denote this average distance as B.

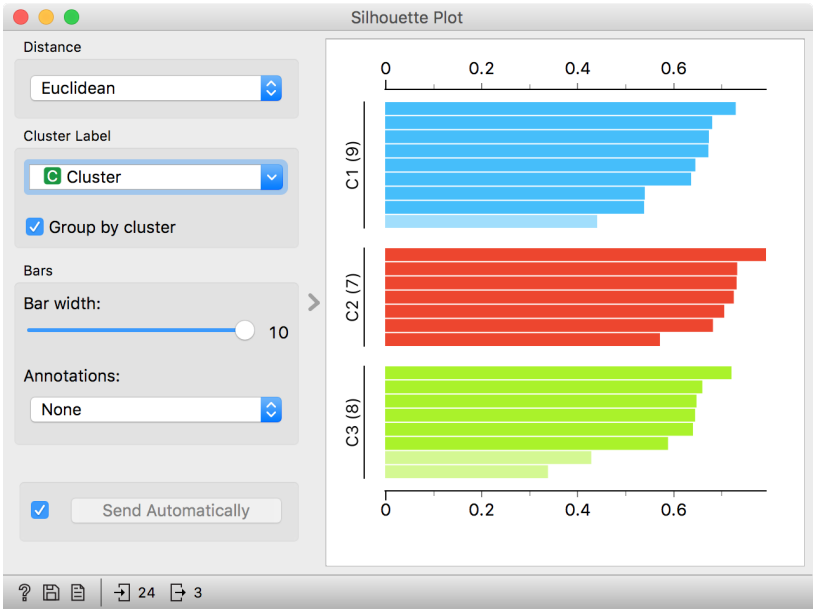
The larger the B, the better.

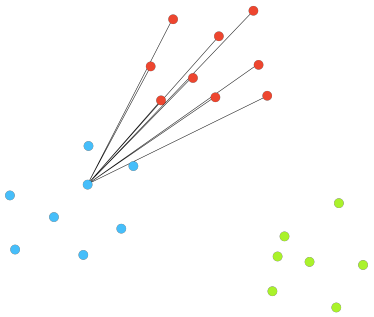
The point is well rooted within its own cluster if the distance to the points from the neighboring cluster (B) is much larger than the distance to the points from its own cluster (A), hence we compute B-A. We normalize it by dividing it with the larger of these two numbers, S = (B -A) / maxA, B. Voilá, S is our silhouette score.

Orange has a *Silhouette Plot* widget that displays the values of the silhouette score for each data instance. We can also choose a particular data instance in the silhouette plot and check out its position in the scatter plot.



This of course looks great for data sets with two features, where the scatter plot reveals all the information. In higher-dimensional data, the scatter plot shows just two features at a time, so two points that seem close in the scatter plot may be actually far apart when all features - perhaps thousands of gene expressions - are taken into account.

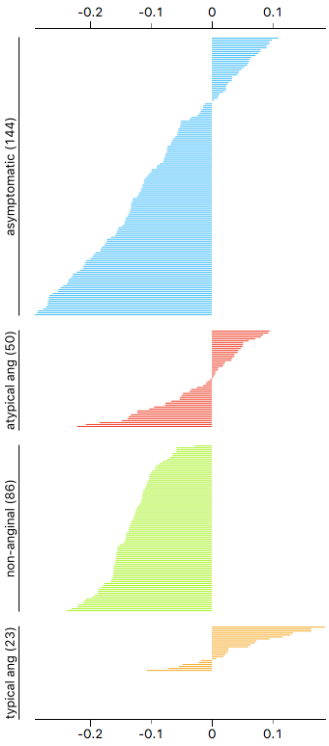




Average distance B.

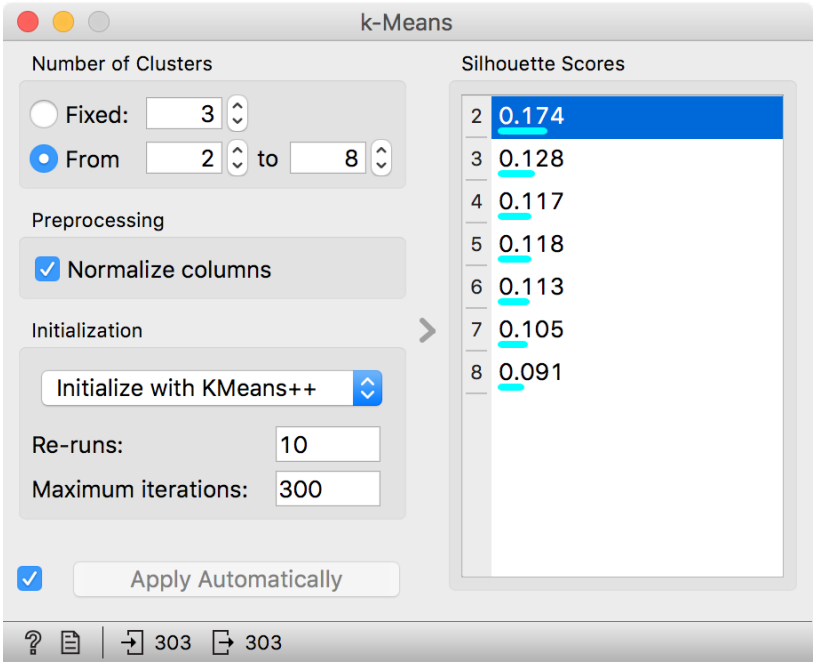
C3 is the green cluster, and all its points have large silhouettes. Not so for the other two.

We selected three data instances with the worst silhouette scores. Can you guess where they lie in the scatter plot?



The total quality of clustering - the silhouette of the clustering - is the average silhouette across all points. When the *k-Means* widget searches for the optimal number of clusters, it tries a different number of clusters and displays the corresponding silhouette scores. Ah, one more thing: Silhouette Plot can be used on any data, not just on data sets that are the output of clustering. We could use it with the iris data set and figure out which class is well separated from the other two and, conversely, which data instances from one class are similar to those from another.

We don't have to group the instances by the class. For instance, the silhouette on the left would suggest that the patients from the heart disease data with typical anginal pain are similar to each other (with respect to the distance/similarity computed from all features), while those with other types of pain, especially non-anginal pain are not clustered together at all.



k-Means Clustering

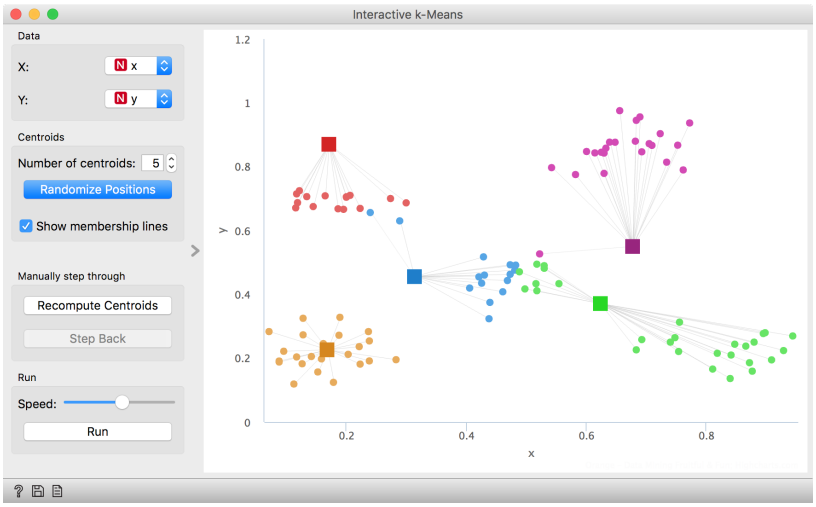
Hierarchical clustering is not suitable for larger data sets due to the prohibitive size of the distance matrix: with 30 thousand objects, the distance matrix already has almost one billion elements. An alternative approach that avoids using the distance matrix is k-means clustering.

K-means clustering randomly selects k centers (with k specified in advance). Then it alternates between two steps. In one step, it assigns each point to its closest center, thus forming k clusters. In the other, it recomputes the centers of the clusters. Repeating these two steps typically converges quite fast; even for big data sets with millions of data points it usually takes just a couple of ten or hundred iterations.

Orange's Educational add-on provides a widget *Interactive k-Means*, which illustrates the algorithm.

Use the *Paint Data* widget to paint some data - maybe five groups of points. Feed it to Interactive k-means and set the number of centroids to 5. You may get something like this.

Try rerunning the clustering from new random positions and observe how the centers conquer the territory. Exciting, isn't it?



Keep pressing Recompute Centroids and Reassign Membership until the plot stops changing. With this simple, two-dimensional data it will take just a few iterations; with more points and features, it can take longer, but the principle is the same.

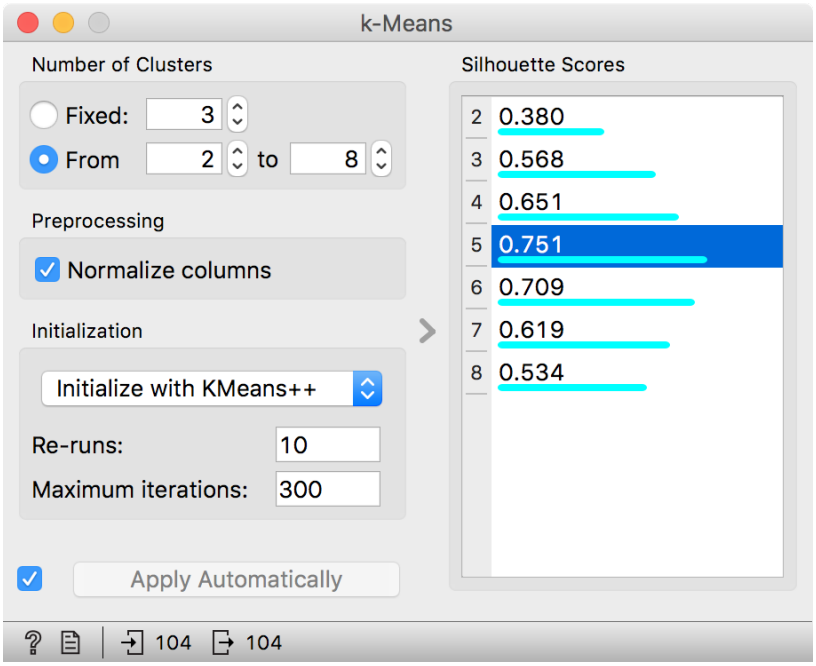
How do we set the initial number of clusters? That's simple: we choose the number that gives the optimal clustering.

Well then, how do we define the optimal clustering? This one is a bit harder. We want small distances between points in the same cluster and large distances between points from different clusters. Pick one

point, and let A be its average distance to the data points in the same cluster and let B represent the average distance to the points from the closest other cluster. (The closest cluster? Just compute B for all other clusters and take the lowest value.) The value (B - A) / max(A, B) is called silhouette; the higher the silhouette, the better the point fits into its cluster. The average silhouette across all points is the silhouette of the clustering. The higher the silhouette, the better the clustering.

Now that we can assess the quality of clustering, we can run k-means with different values of parameter k (number of clusters) and select k which gives the largest silhouette.

For this, we abandon our educational toy and connect Paint Data to the widget k-Means. We tell it to find the optimal number of clusters between 2 and 8, as scored by the Silhouette.



Works like a charm.

Except that it often doesn't. First, the result of k-means clustering depends on the initial selection of centers. With unfortunate selection, it may get stuck in a local optimum. We solve this by re-running the clustering multiple times from random positions and using the best result. Second, the silhouette sometimes fails to correctly evaluate the clustering. Nobody's perfect.

Time to experiment. Connect the Scatter Plot to k-Means. Change the number of clusters. See if the clusters make sense. Could you paint the data where k-Means fails? Or where it works really well?