# Lesson 22: Mapping the Data

Imagine a foreign visitor to the US who knows nothing about the US geography. He doesn't even have a map; the only data he has is a list of distances between the cities. Oh, yes, and he attended the Introduction to Data Mining.

If we know distances between the cities, we can cluster them.

For this example we retrieved data from http://www.mapcrow.info/united_states.html, removed the city names from the first line and replaced it with "31 labelled".

The file is available at http://file.biolab.si/files/us-cities.dst.zip. To load it, unzip the file and use the File Distance widget from the Prototypes add-on.
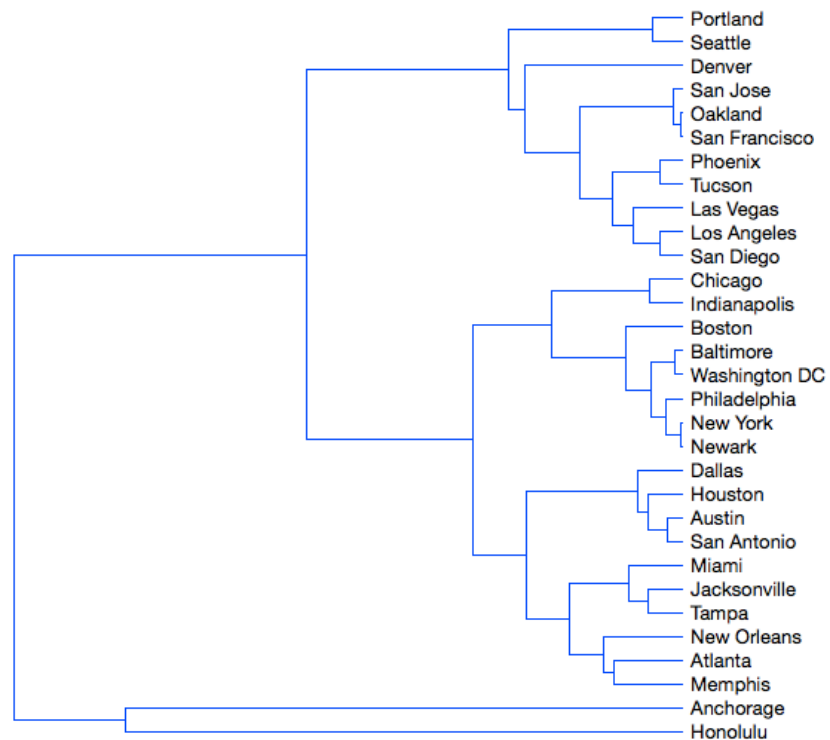
How much sense does it make? Austin and San Antonio are closer to each other than to Houston; the tree is then joined by Dallas. On the other hand, New Orleans is much closer to Houston than to Miami. And, well, good luck hitchhiking from Anchorage to Honolulu.

As for Anchorage and Honolulu, they are left-overs; when there were only three clusters left (Honolulu, Anchorage and the big cluster with everything else), Honolulu and Anchorage were closer to each other than to the rest. But not close — the corresponding lines in the dendrogram are really long.

The real problem is New Orleans and San Antonio: New Orleans is close to Atlanta and Memphis, Miami is close to Jacksonville and

Tampa. And these two clusters are suddenly more similar to each other than to some distant cities in Texas.

In general, two points from different clusters may be more similar to each other than to some points from their corresponding clusters.

We can't run k-means clustering on this data, since we only have distances, and k-means runs on real (tabular) data. Yet, k-means would have the same problem as hierarchical clustering.

To get a better impression about the physical layout of cities, people have invented a better tool: a map! Can we reconstruct a map from a matrix of distances? Sure. Take any pair of cities and put them on paper with a distance corresponding to some scale. Add the third city and put it at the corresponding distance from the two. Continue until done. Excluding, for the sake of scale, Anchorage, we get the following map.
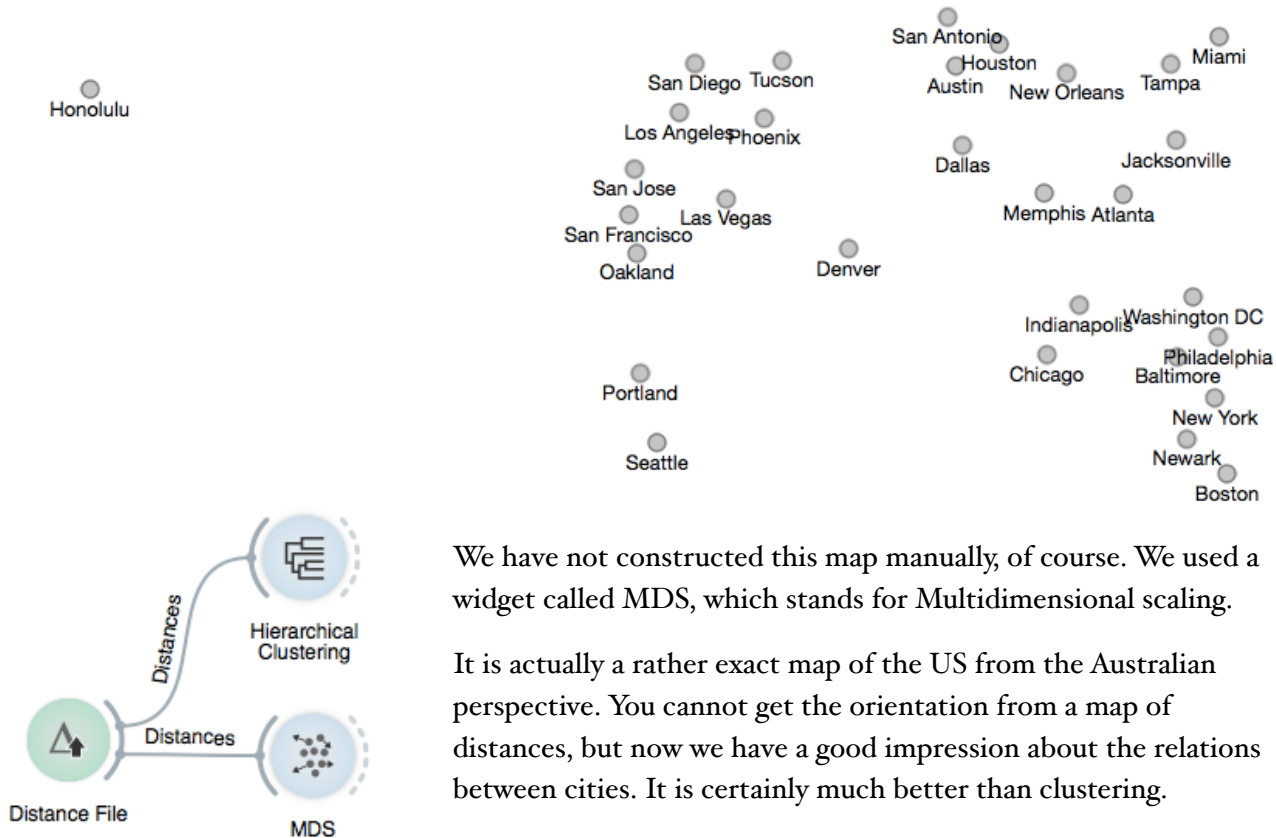
We have not constructed this map manually, of course. We used a widget called MDS, which stands for Multidimensional scaling.

It is actually a rather exact map of the US from the Australian perspective. You cannot get the orientation from a map of distances, but now we have a good impression about the relations between cities. It is certainly much better than clustering.

Remember the clustering of animals? Can we draw a map of animals?
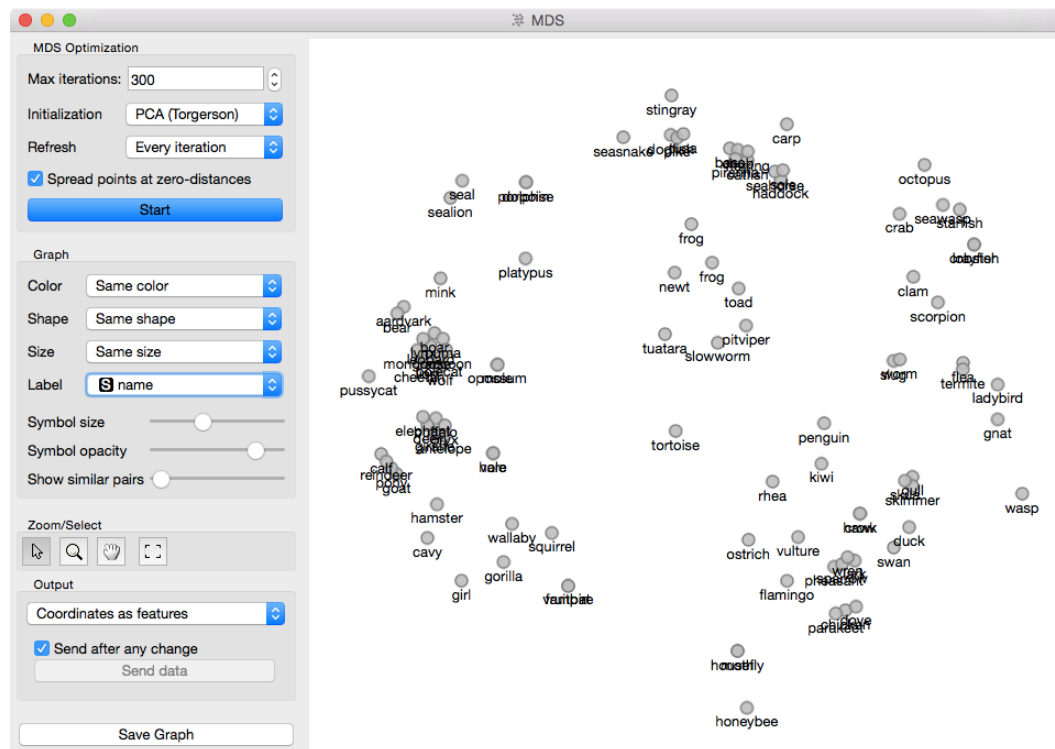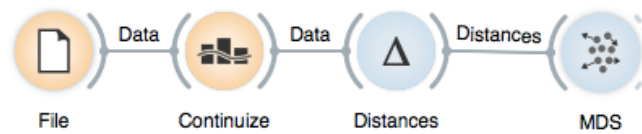
Does the map make any sense? Are similar animals together? Color the points by the types of animals and you should see.

The map of the US was accurate: one can put the points in a plane so that the distances correspond to actual distances between cities. For most data, this is usually impossible. What we get is a projection (a non-linear projection, if you care about mathematical finesses) of the data. You lose something, but you get a picture.

The MDS algorithm does not always find the optimal map. You may want to restart the MDS from random positions. Use the slider "Show similar pairs" to see whether the points that are placed together (or apart) actually belong together. In the above case, the honeybee belongs closer to the wasp, but could not fly there as in the process of optimization it bumped into the hostile region of flamingos and swans.

Lesson 23: Maps and Clusters

Remember the mixed cluster in the zoo data that contained invertebrates, reptiles, amphibian, and even a mammal. Was this a homogeneous cluster? Why the mammal there? And how far is this mammal to other mammals? And why is this cluster close to the cluster of mammals?
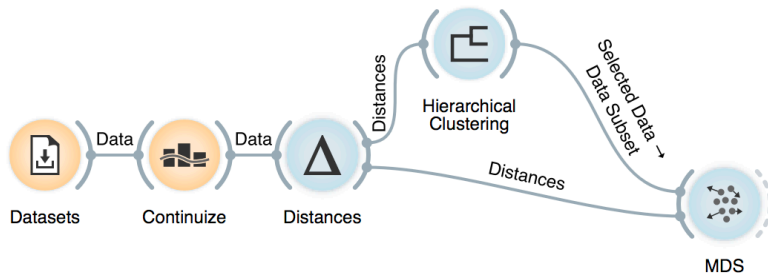
So many questions. But we can answer them all with a combination of clustering and multi-dimensional scaling. We would like to show any cluster that we selected from a dendrogram to be shown on the map of animals presented by MDS. And we would like to use cosine distances, so we need to take care of the composition of the workflow and proper connections between widgets.
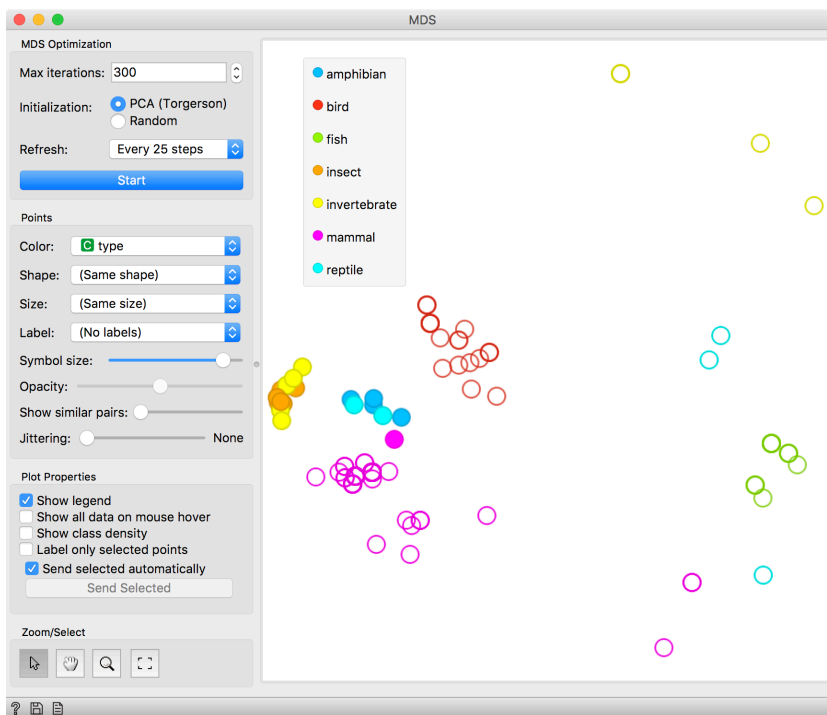
Clustering and two-dimensional embedding make a great combination for data exploration. Clustering finds the coherent groups, and embedding, such as MDS, reveals the relations between the clusters and positions the cluster on the data map. There are other dimensionality reduction and embedding techniques that we could use, but for smaller data sets, MDS is great because it tries to preserve the distances from the original data space.

Can you change the workflow to explore the position of individual clusters found by k-means?

Lesson 24: Principal Component Analysis

Which of the following three scatterplots (showing x vs. y, x vs. z and y vs. z) for the same three-dimensional data gives us the best picture about the actual layout of the data in space?
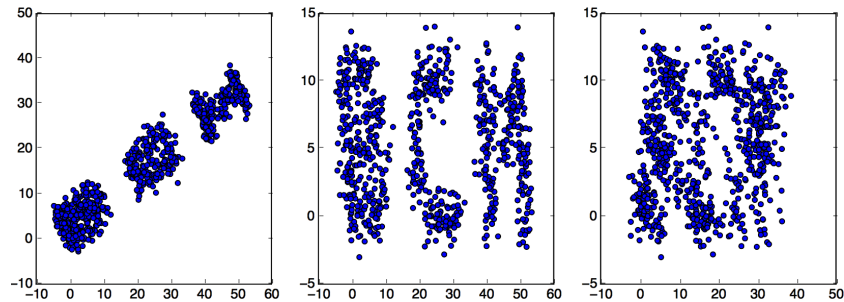
Yes, the first scatter plot looks very useful: it tells us that x and y are highly correlated and that we have three clusters of somewhat irregular shape. But remember: this data is three dimensional. What if we saw it from another, perhaps better perspective?
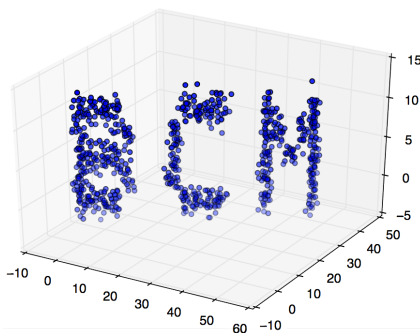
Let's make another experiment. Go to https://in-the-sky.org/ngc3d.php, disable Auto-rotate and Show labels and select Zoom to show Local Milky Way. Now let's rotate the picture of the galaxy to find the layout of the stars.

Think about what we've done. What are the properties of the best projection?

We want the data to be as spread out as possible. If we look from the direction parallel to the galactic plane, we see just a line. We lose one dimension, keeping only a single coordinate for each star. (This is unfortunately exactly the perspective we see on the night sky: most stars are in the bright band we call the milky way, and we only look at the outliers.) Among all possible projections, we attempt to find the one with the highest spread across the scatter plot. This projection may not be (and usually isn't) orthogonal to any of the axis; it may be projection to an arbitrary plane.

We again talk about two-dimensional projection only for the sake of illustration. Imagine that we have ten thousand dimensional data and we would like, for some reason, keep just ten features. Yes, we can rank the features and keep the most informative, but

what if these are correlated and tell us the same thing? Or what if our data does not have any target variable: with what should the "good features" be correlated? And what if the optimal projection is not aligned with the axes at all, so "good" features are combinations of the original ones?

We can do the same reasoning as above: we want to find a 10-dimensional (for the sake of examples) projection in which the data points spread as widely as possible.

How do we do this? Let's go back to our every day's three-dimensional world and think about how to find a two-dimensional projection.

Imagine you are observing a swarm of flies; your data are their exact coordinates in the room, so three numbers describe the position of each fly. Then you discover that your flies fly in a formation: they are (almost) on the same line. You could then describe the position of each fly with a single number that represents the fly's location along the line. Plus, you need to know where in the space the line lies. We call this line the first principal component. By using it, we reduce the three-dimensional space into a single dimension.

After some careful observation, you notice the flies are a bit spread in one other direction, so they do not fly along a line but along the band. Therefore, we need two numbers, one along the first and one along the — you guessed it — second principal component.

It turns out the flies are also spread in the third direction. Thus you need three numbers after all.

Or do you? It all depends on how they spread in the second and in the third direction. If the spread along the latter is relatively small in comparison with the first, you are okay with a single dimension. If not, you need two, but perhaps still not three.

Let's step back a bit: why would one who carefully measured expressions of ten thousand genes want to throw most data away and reduce it to a dozen dimensions? The data, in general, may not and does not have as many dimensions as there are features. Say you have an experiment in which you spill different amounts of two chemicals over colonies of amoebas and then measure the expressions of 10.000 genes. Instead of flies in a three-dimensional

space, you now profile colonies in a 10,000-dimensional space, the coordinates corresponding to gene expressions. Yet if expressions of genes depend only on the concentrations of these two chemicals, you can compute all 10,000 numbers from just two. Your data is then just two-dimensional.

A technique that does this is called Principle Components Analysis, or PCA. The corresponding widget is simple: it receives the data and outputs the transformed data.
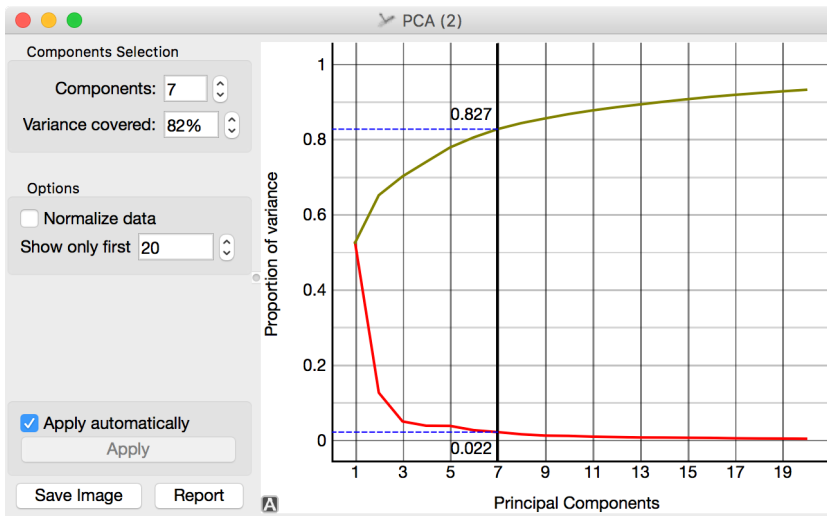
The widget allows you to select the number of components and helps you by showing how much information (technically: explained variance) you retain with respect to the number of components (brownish line) and the amount of information (explained variance) in each component.

The PCA on the left shows the scree diagram for brown-selected data. Set like this, the widget replaces the 80 features with just seven - and still keeping 82.7% of information. (Note: disable "Normalize data" checkbox to get the same picture.) Let us see a scatter plot for the first two components.
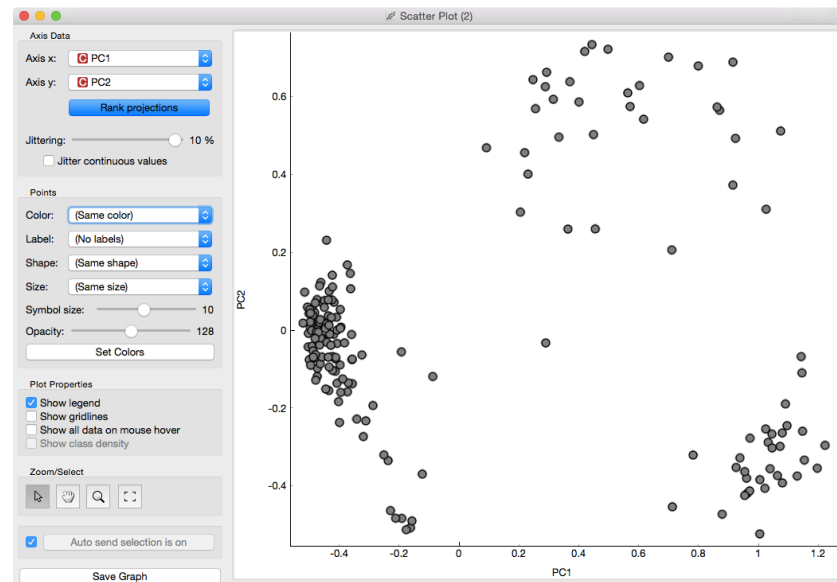
The axes, PC1 and PC2, do not correspond to particular features in the original data, but to their linear combination. What we are looking at is a projection onto the plane, defined by the first two components. When you consider only two components, you can imagine that PCA put a hyperplane into multidimensional space and projecting all data into it.

Note that this is an unsupervised method: it does not care about the class. The classes in the projection may be be well separated or not. Let's add some colors to the points and see how lucky we are this time.
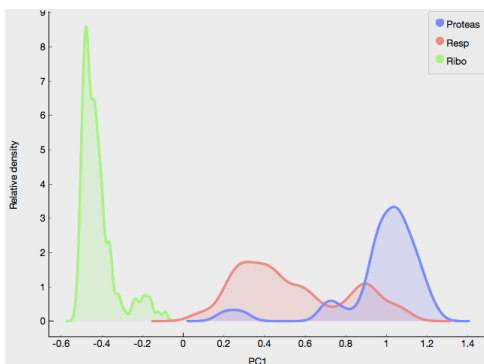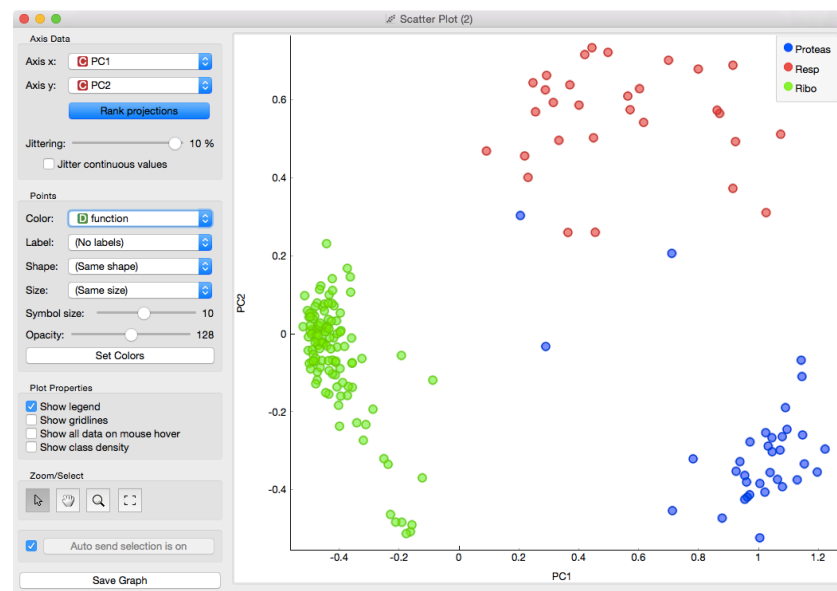
The data separated so well that these two dimensions alone may suffice for building a good classifier. No, wait, it gets even better. The data classes are separated well even along the first component. So we should be able to build a classifier from a single feature!
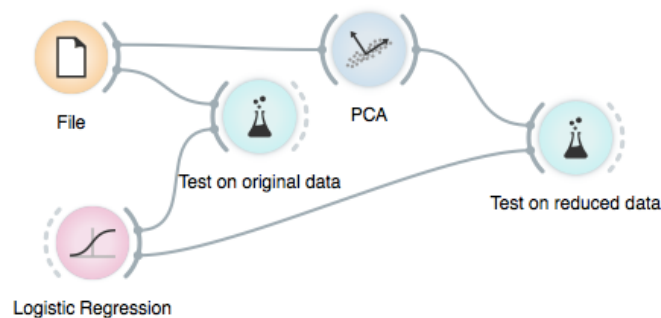
In the above schema we use the ordinary Test & Score widget, but renamed it to "Test on original data" for better understanding of the workflow.

On the original data, Logistic regression gets 98% AUC and classification accuracy. If we select just single component in PCA, we already get a 93%, and if we take two, we get the same result as on the original data.

PCA is thus useful for multiple purposes. It can simplify our data by combining the existing features to a much smaller number of features without losing much data. The directions of these features may tell us something about the data. Finally, it can find us good two-dimensional projections that we can observe in scatter plots.

Lesson 25: Image Embedding

Every data set so far came in the matrix (tabular) form: objects (say, tissue samples, students, flowers) were described by row vectors representing a number of features. Not all the data is like this; think about collections of text articles, nucleotide sequences, voice recordings or images. It would be great if we could represent them in the same matrix format we have used so far. We would turn collections of, say, images, into matrices and explore them with the familiar prediction or clustering techniques.

This depiction of deep learning network was borrowed from http://www.amax.com/blog/?
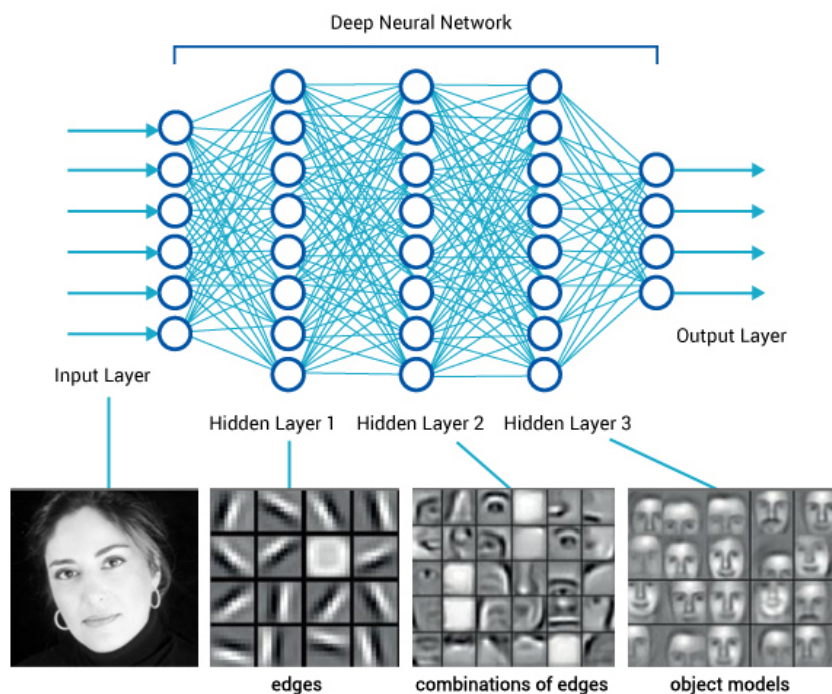
Until very recently, finding useful representation of complex objects such as images was a real pain. Now, technology called deep learning is used to develop models that transform complex objects to vectors of numbers. Consider images. When we, humans, see an image, our neural networks go from pixels, to spots, to patches, and to some higher order representations like squares, triangles, frames, all the way to representation of complex objects. Artificial neural networks used for deep learning emulate these through layers of computational units (essentially, logistic regression models and some other stuff we will ignore here). If we put an image to an input of such a network and collect the outputs from the higher levels, we get vectors containing an abstraction of the image. This is called embedding.

Deep learning requires a lot of data (thousands, possibly millions of data instances) and processing power to prepare the network. We will use one which is already prepared. Even so, embedding takes time, so Orange doesn't do it locally but uses a server invoked through the ImageNet Embedding widget.
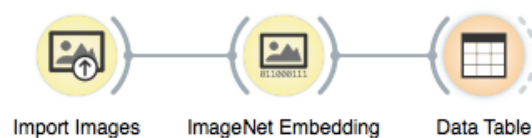
Image embedding describes the images with a set of 2048 features appended to the table with meta features of images.

	mage name	image image	size	width	height	n0	n1	n2	n3	n4	n5	n6
1	calf	/Users/bla...	45538	191	152	0.181	0.212	0.041	0.016	0.180	0.071	0.24
2	cat	/Users/bla...	22193	105	137	0.055	0.156	0.649	0.000	0.156	0.136	0.28
3	chick	/Users/bla...	14891	85	92	0.127	0.032	0.097	0.015	0.169	0.080	0.10
4	cow	/Users/bla...	62159	210	189	0.475	0.130	0.048	0.082	0.130	0.599	0.23
5	dog	/Users/bla...	28745	129	125	0.049	0.187	0.181	0.111	0.188	0.516	0.69
6	duck	/Users/bla...	39583	158	172	0.131	0.037	0.073	0.040	0.162	0.221	0.18
7	duckling	/Users/bla...	17109	99	119	0.068	0.050	0.033	0.055	0.184	0.189	0.12
8	foal	/Users/bla...	39210	147	177	0.061	0.252	0.040	0.155	0.481	0.348	0.17
9	goat	/Users/bla...	53039	221	179	0.265	0.124	0.017	0.019	0.176	0.110	0.27
10	goose	/Users/bla...	34442	141	202	0.355	0.246	0.159	0.000	0.422	0.374	0.10
11	hen	/Users/bla...	41716	134	168	0.389	0.062	0.037	0.083	0.429	0.218	0.11
12	horse	/Users/bla...	69109	285	195	0.280	0.229	0.084	0.095	0.387	0.295	0.20
13	kid	/Users/bla...	36290	170	160	0.131	0.140	0.024	0.067	0.130	0.030	0.19
14	lamb	/Users/bla...	35520	123	168	0.358	0.034	0.189	0.055	0.331	0.162	0.40
15	ox	/Users/bla...	56401	191	189	0.520	0.003	0.096	0.106	0.139	0.235	0.28

Info
19 instances
2048 features (no missing values)
No target variable.
5 meta attributes (no missing values)

Variables
☑ Show variable labels (if present)
☐ Visualize continuous values
☑ Color by instance classes

Selection
☑ Select full rows

Restore Original Order
Report

☑ Send Automatically

We have no idea what these features are, except that they represent some higher-abstraction concepts in the deep neural network (ok, this is not very helpful in terms of interpretation). Yet, we have just described images with vectors that we can compare and measure their similarities and distances. Distances? Right, we could do clustering. Let's cluster the images of animals and see what happens.

Import Images → ImageNet Embedding → Distances → Hierarchical Clustering → Image Viewer

To recap: in the workflow about we have loaded the images from the local disk, turned them into numbers, computed the distance matrix containing distances between all pairs of images, used the distances for hierarchical clustering, and displayed the images that correspond to the selected branch of the dendrogram in the image viewer. We used cosine similarity to assess the distances (simply because of the dendrogram looked better than with the Euclidean distance).

Even the lecturers of this course were surprised at the result. Beautiful!
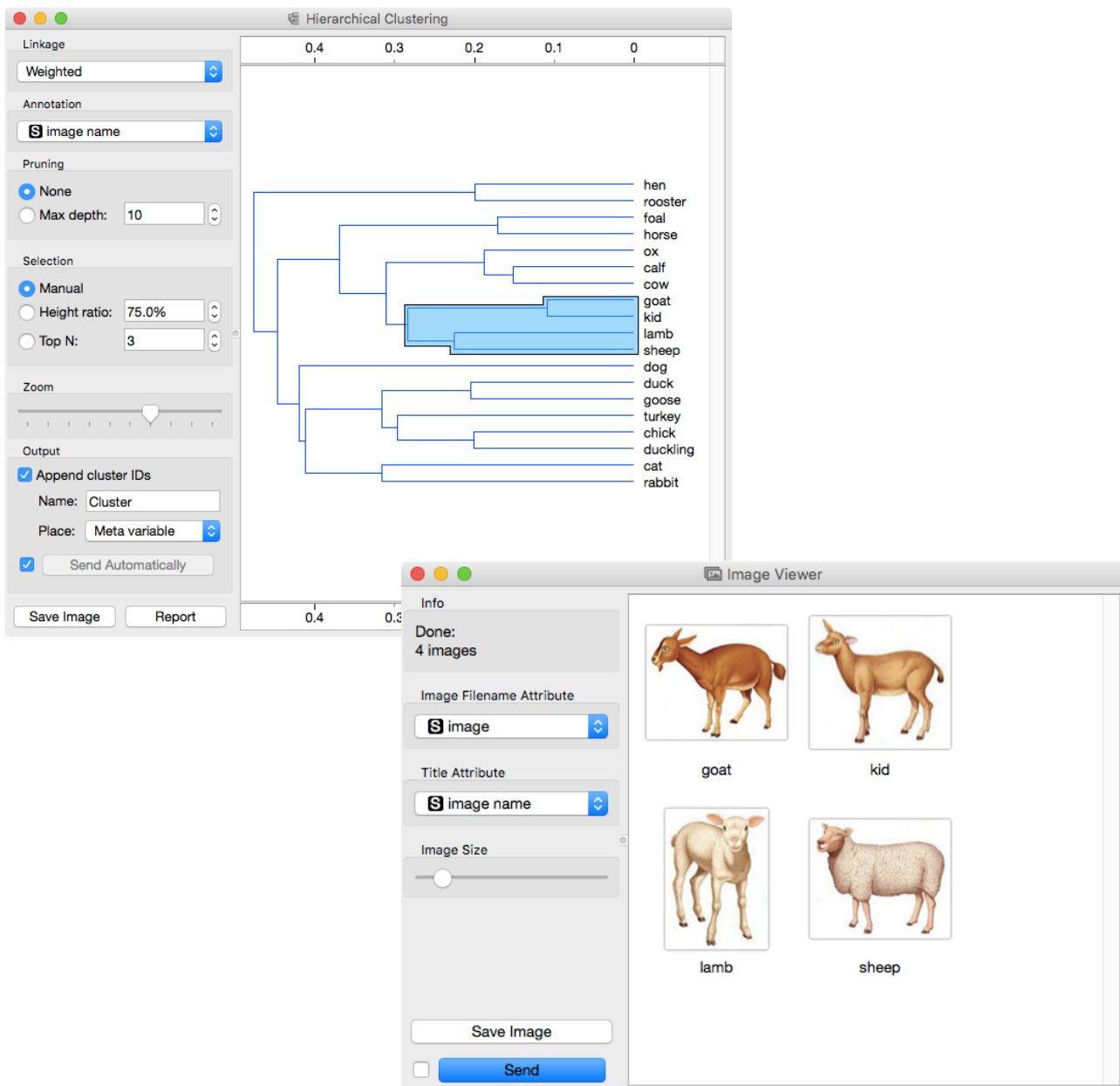
Lesson 26: Images and Classification

In this lesson, we are using images of yeast protein localization (http://file.biolab.si/files/yeast-localization-small.zip) in the classification setup. But this same data set could be explored in clustering as well. The workflow would be the same as the one from previous lesson. Try it out! Do Italian cities cluster next to American or are

We can use image data for classification. For that, we need to associate every image with the class label. The easiest way to do this is by storing images of different classes in different folders. Take, for instance, images of yeast protein localization. Screenshot of the file names shows we have stored them on the disk.

Localization sites (cytoplasm, endosome, endoplasmic reticulum) will now become class labels for the images. We are just a step away from testing if logistic regression can classify images to their corresponding protein localization sites. The data set is small: you may use leave-one-out for evaluation in Test & Score widget instead of cross validation.

At about 0.9 the AUC score is quite high, and we can check where the mistakes are made and visualize these in an Image Viewer.
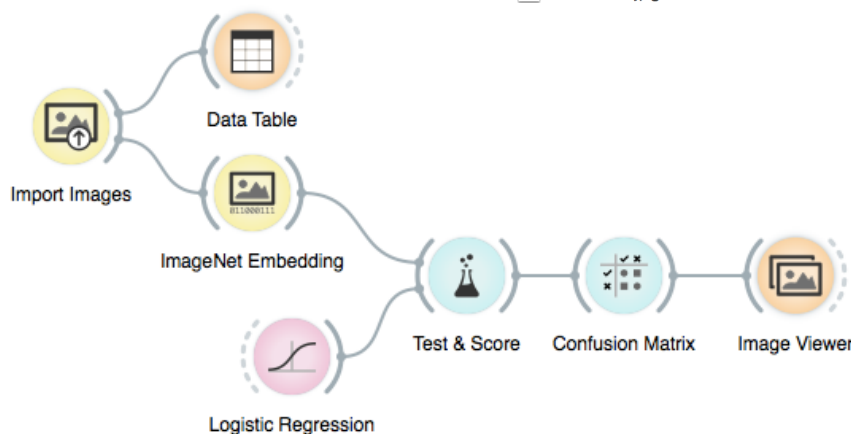
For the End

The course on introduction to data mining at the University of Ljubljana ends here. We covered quite some mileage, and we hope we have taught you some essential procedures that should be on the stack of every data scientist. The goal was not to turn you into one but to get you familiar with some basic techniques, tools, and concepts. Data science is a vast field, and it takes years of study and practice to master it. You may never become a data scientist, but as an expert in biomedicine, it should now be more comfortable to talk and collaborate with statisticians and computer scientists. And for those who want to go ahead with data science, well, you now know where to start.