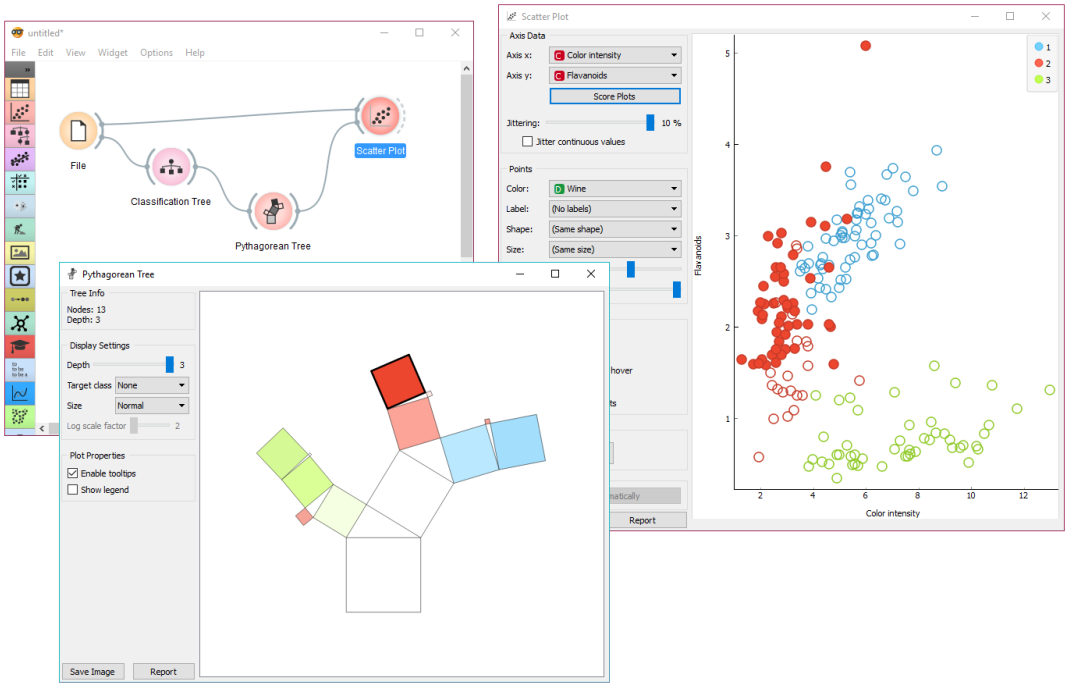


Introduction to Data Mining

Working notes for the hands-on course for PhD students at University of Ljubljana

These notes include Orange workflows and visualizations we will construct during the course.

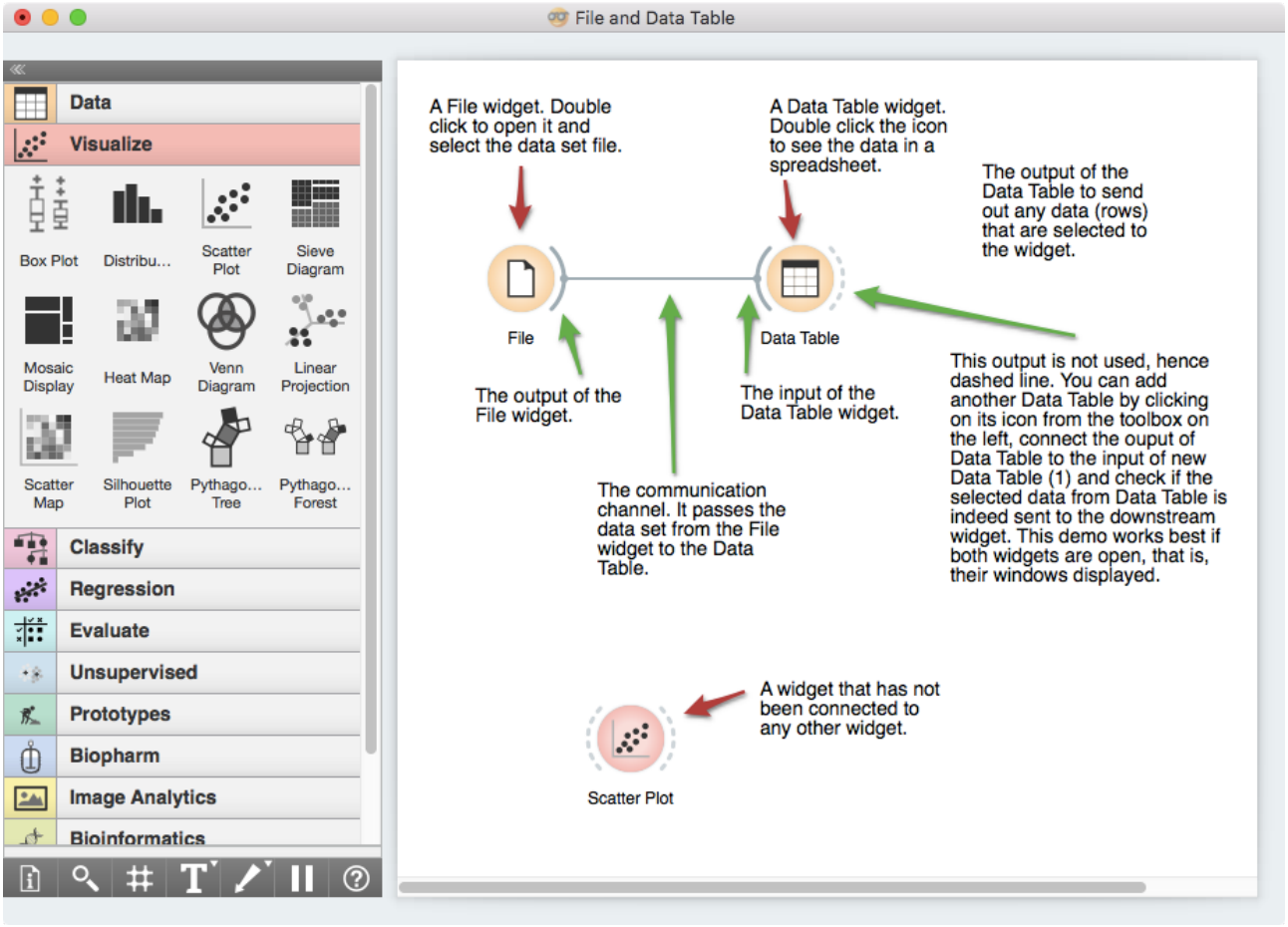
The working notes were prepared by Blaž Zupan and Janez Demšar with help from the members of the Bioinformatics Lab in Ljubljana that develop and maintain Orange.

Welcome to the course on Introduction to Data Mining! This course is designed for students and researchers of life sciences, engineering, and statistics. You will see how common data mining tasks can be accomplished without programming. We will use Orange to construct visual data mining workflows. Many similar data mining environments exist, but the lecturers prefer Orange for one simple reason—they are its authors.

If you haven't already installed Orange, please download the installation package from http://orange.biolab.si.

Lesson 1: Workflows in Orange

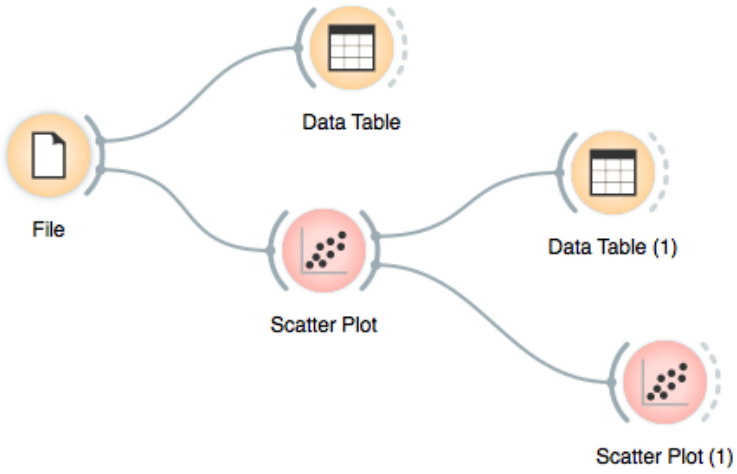
Orange workflows consist of components that read, process and visualize data. We call them "widgets." We place the widgets on a drawing board (the "canvas"). Widgets communicate by sending information along with a communication channel. An output from one widget is used as input to another.



A screenshot above shows a simple workflow with two connected widgets and one widget without connections. The outputs of a widget appear on the right, while the inputs appear on the left.

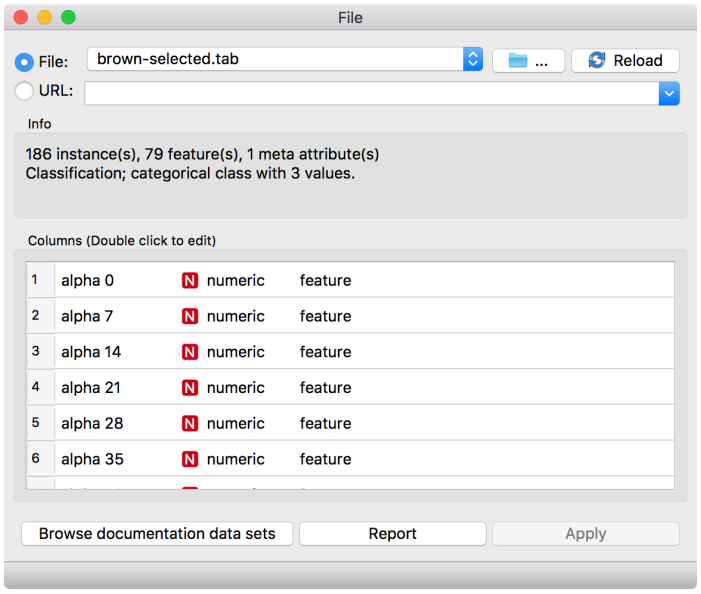
We construct workflows by dragging widgets onto the canvas and connecting them by drawing a line from the transmitting widget to the receiving widget. The widget's outputs are on the right and the inputs on the left. In the workflow above, the File widget sends data to the Data Table widget.

Start by constructing a workflow that consists of a File widget, two Scatter Plot widgets, and two Data Table widgets:



Workflow with a File widget that reads data from disk and sends it to the Scatter Plot and Data Table widget. The Data Table renders the data in a spreadsheet, while the Scatter Plot visualizes it. Selected data points from the Scatterplot are sent to two other widgets: Data Table (1) and Scatter Plot (1).

The File widget reads data from your local disk. Open the File Widget by double clicking its icon. Orange comes with several preloaded data sets. From these ("Browse documentation data sets..."), choose brown-selected.tab, a yeast gene expression data set.

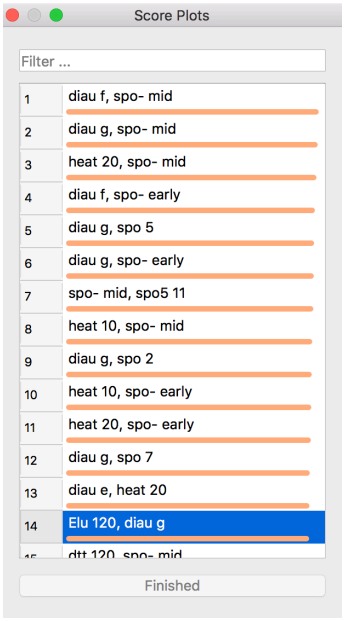


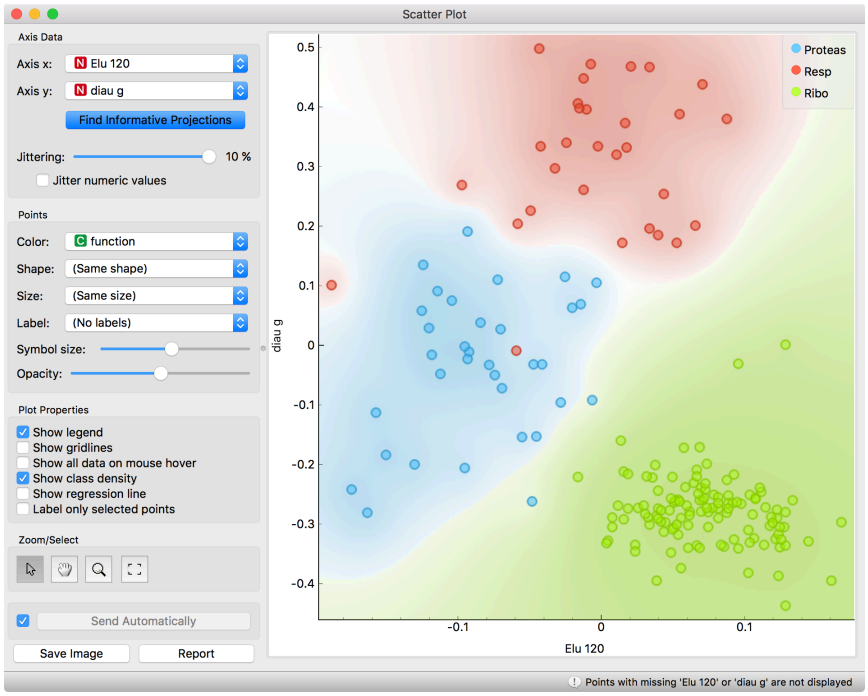
Orange workflows often start with a File widget. The brown-selected data set comprises 186 rows (genes) and 81 columns. Out of the 81 columns, 79 contain gene expressions of baker's yeast under various conditions, one column (marked as a "meta attribute") provides gene names, and one column contains the "class" value or gene function.

After you load the data, open the other widgets. In the Scatter Plot widget, select a few data points and watch as they appear in widget Data Table (1). Use a combination of two Scatter Plot widgets,

where the second scatter plot shows a detail from a smaller region selected in the first scatterplot.

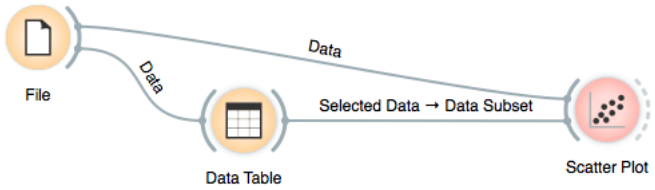
Following is more of a side note, but it won't hurt. Namely, the scatter plot for a pair of random features does not provide much information on gene function. Does this change with a different choice of feature pairs in the visualization? Rank projections (the button on the top left of the Scatter Plot widget) can help you find a good feature pair. How do you think this works? Could the suggested pairs of features be useful to a biologist?



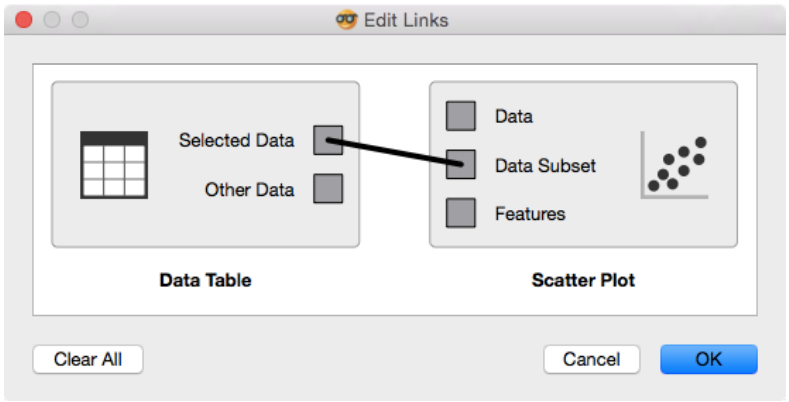


We can connect the output of the Data Table widget to the Scatter Plot widget to highlight the chosen data instances (rows) in the scatter plot.

In this workflow, we have turned on the option "Show channel names between widgets" in File→Preferences.

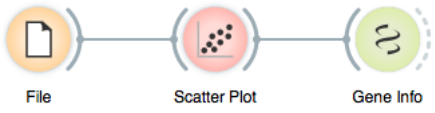


How does Orange distinguish between the primary data source and the data selection? It uses the first connected signal as the entire data set and the second one as its subset. To make changes or to check what is happening under the hood, double click on the line connecting the two widgets.



Orange comes with a basic set of widgets for data input, preprocessing, visualization and modeling. For other tasks, like text mining, network analysis, and bioinformatics, there are add-ons. Check them out by selecting "Add-ons…" from the options menu.

The rows in the data set we are exploring in this lesson are gene profiles. We can use the Gene Info widget from the Bioinformatics add-on to get more information on the genes we selected in any of the Orange widgets.

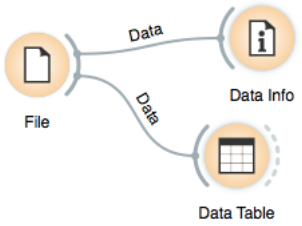


Lesson 2: Basic Data Exploration

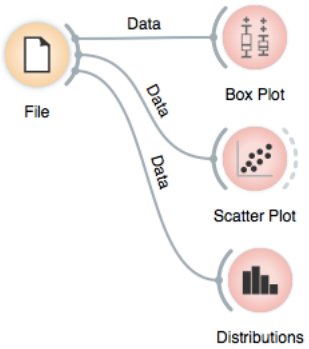
Let us consider another problem, this time from clinical medicine. We will dig for something interesting in the data and explore it a bit with various widgets. You will get to know Orange better and also learn about several interesting visualizations.

We will start with an empty canvas; to clean it from our previous lesson, use either File→New or select all the widgets and remove them (use the backspace/delete key, or Cmd-backspace if you are on Mac).

Now again, add the File widget and open another documentation data set: heart_disease. How does the data look?

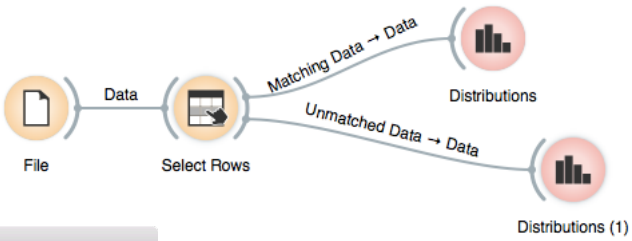


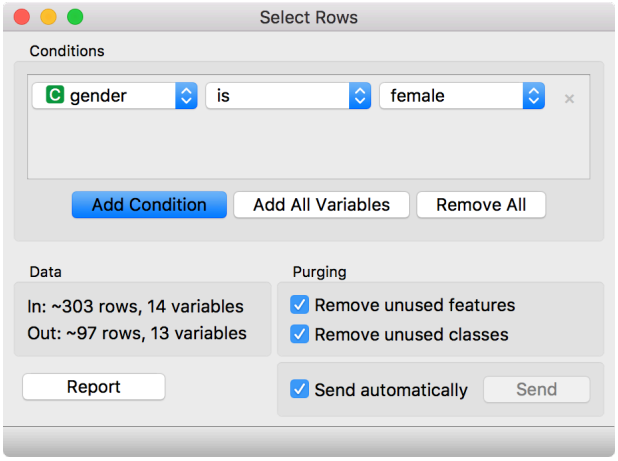
Let us check whether standard visualizations tell us anything interesting. (Hint: look for gender differences. These are always interesting and occasionally even real.)



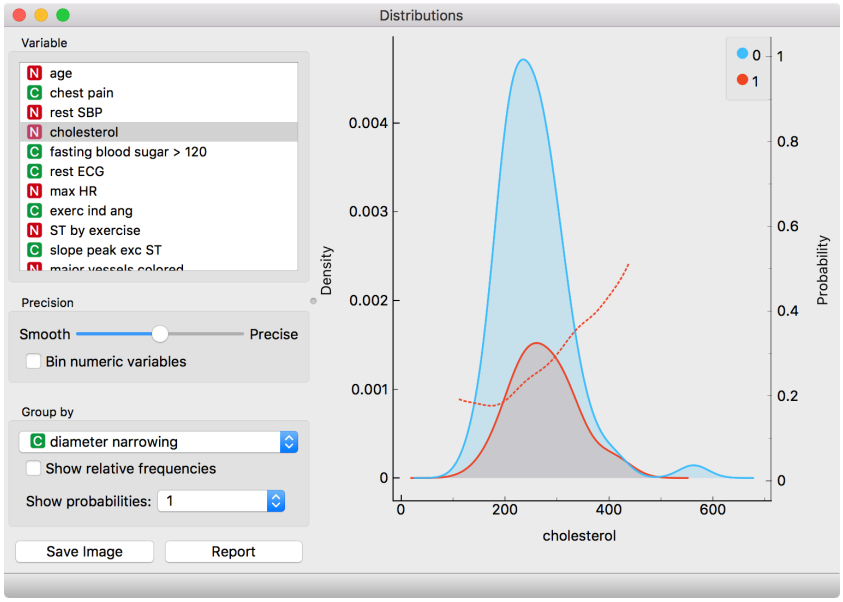
The two Distributions widgets get different data: the upper gets the selected rows, and the lower gets the others. Double-click the connection between the widgets to access setup dialog, as you've learned in the previous lesson.

Data can also be split by the value of features — in this case, gender — and analyze it separately.



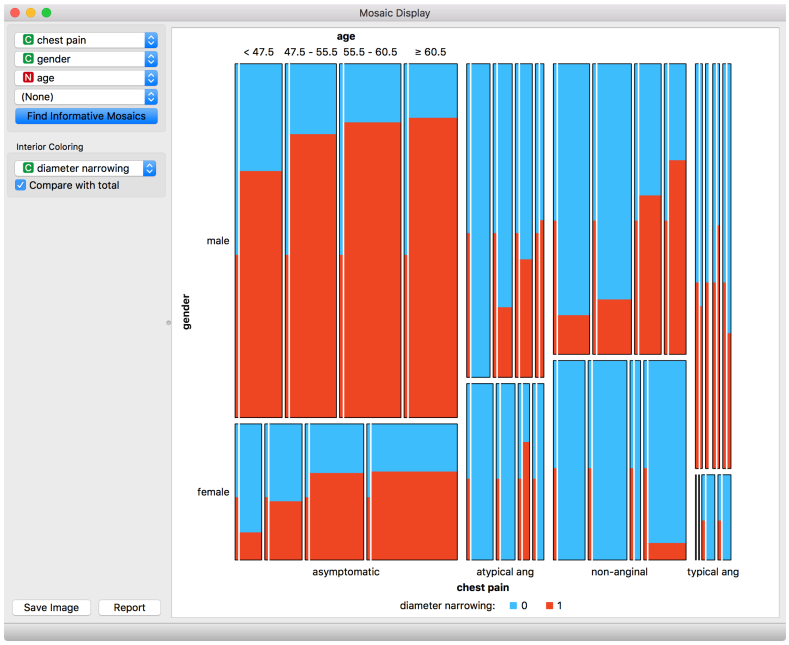


In the Select Rows widget, we choose the female patients. You can also add other conditions. Selection of data instances works well with visualization of data distribution. Try having at least two widgets open at the same time and explore the data.



There are two less known — but great — visualizations for observing interactions between features.

You can play with the widget by trying different combinations of 1-4 features.

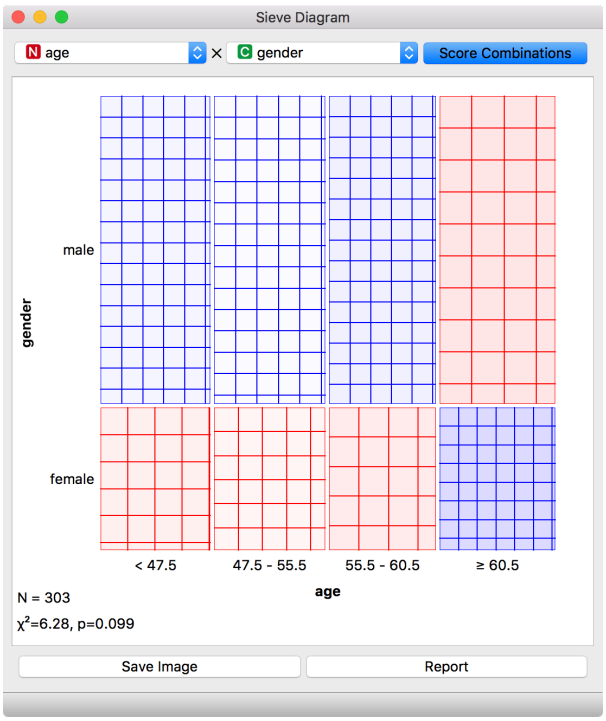


The mosaic display shows a rectangle split into columns with widths reflecting the prevalence of different types of chest pain. Each column is then further split vertically according to gender distributions within the column. The resulting rectangles are divided again horizontally according to age group sizes. Within the resulting bars, the red and blue areas represent the outcome distribution for each group and the tiny strip to the left of each shows the overall distribution.

What can we conclude from this diagram?

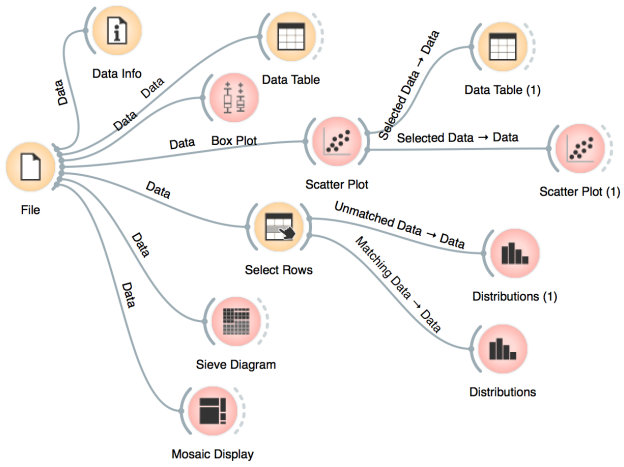
Another visualization, Sieve diagram, also splits a rectangle horizontally and vertically, but with independent cuts, so the areas correspond to the expected number of data instances assuming the observed variables are independent. For example, 1/4 of patients are older than 60, and 1/3 of patients are female, so the area of the bottom right rectangle is 1/12 of the total area. With roughly 300 patients, we would expect 1/12 × 300 = 25 older women in our data. There are 34. Sieve diagram shows the difference between the expected and the observed frequencies by the grid density and the color of the field.

See the Score Combinations button? Guess what it does? And how it scores the combinations? (Hint: there are some Greek letters at the bottom of the widget.)

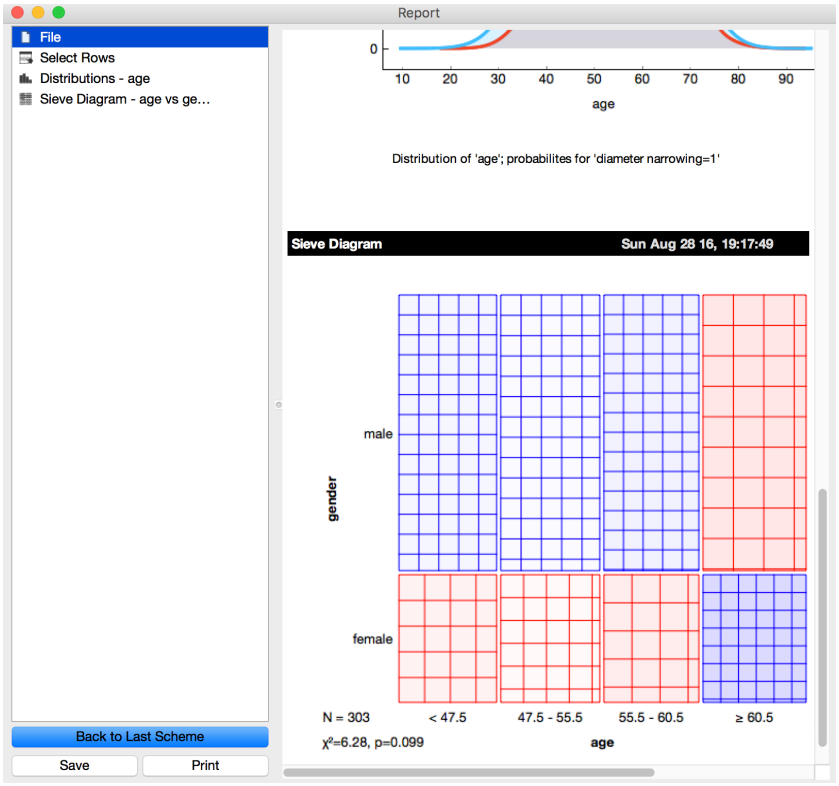


Lesson 3: Saving Your Work

If you followed the instructions so far — except for those about removing widgets — your workflow might look like this.



You can save it (File→Save) and share it with your colleagues. Just don't forget to put the data files in the same directory as the file with the workflow.



One more trick: Pressing Ctrl-C (or ⌘-C, on Mac) copies a visualization to the clipboard, so you can paste it to another application.

Widgets also have a Report button, which you can use to keep a log of your analysis. When you find something interesting, like an unexpected Sieve Diagram, just click Report to add the graph to your log. You can also add reports from the widgets on the path to this one, to make sure you don't forget anything relevant.

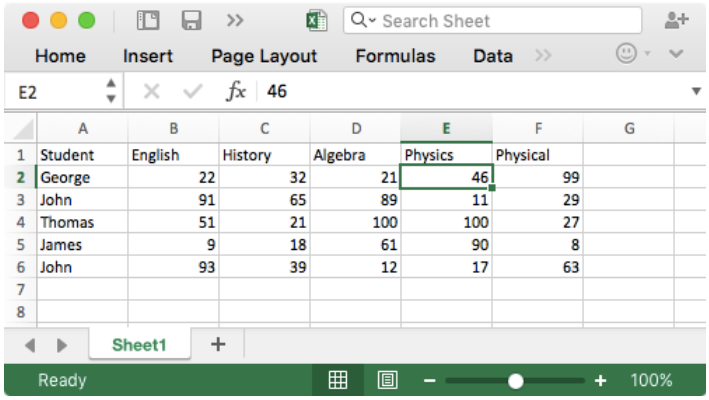
Clicking on the part of the report also allows you to add a comment.

Clicking on a part of the report also allows you to add a comment.

You can save the report as HTML or PDF, or to a file that includes all workflows that are related to the report items and which you can later open in Orange. In this way, you and your colleagues can reproduce your analysis results.

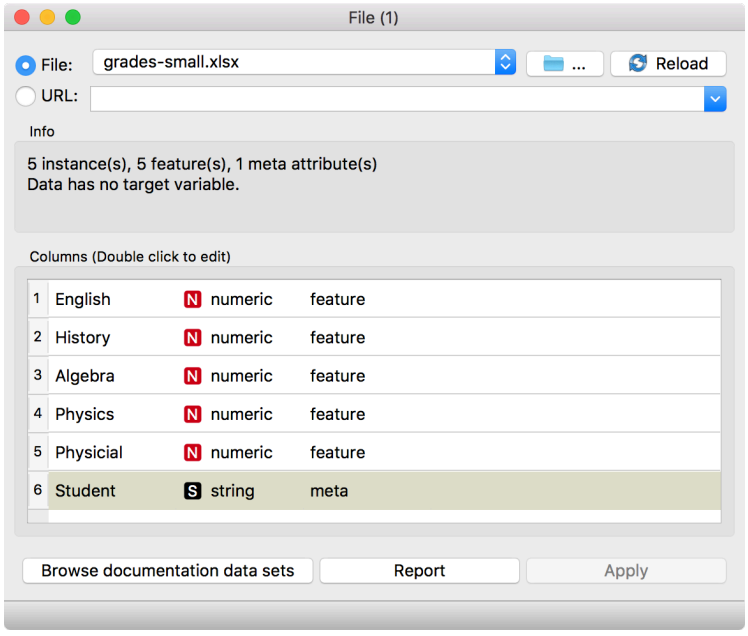
Lesson 4: Loading Your Own Data Set

The data sets we have worked with in previous lessons come with Orange installation. Orange can read data from spreadsheet file formats which include tab and comma separated and Excel files. Let us prepare a data set (with school subjects and grades) in Excel and save it on a local disk.



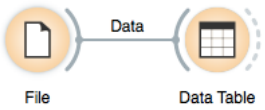
	A	B	C	D	E	F
1	Student	English	History	Algebra	Physics	Physical
2	George	22	32	21	46	99
3	John	91	65	89	11	29
4	Thomas	51	21	100	100	27
5	James	9	18	61	90	8
6	John	93	39	12	17	63
7						
8						

In Orange, we can use the File widget to load this data.

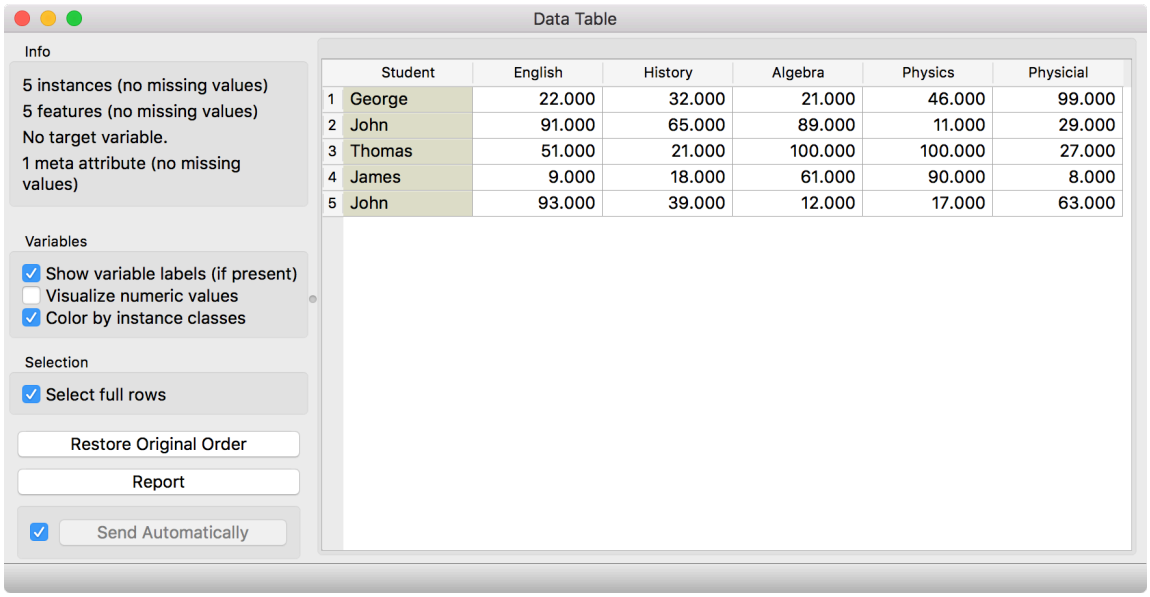


Looks ok. Orange has correctly guessed that student names are character strings and that this column in the data set is special, meant to provide additional information and not to be used for modeling (more about this in the coming lectures). All other columns are numeric features.

It is always good to check if Orange read the data correctly. We can connect our File widget with the Data Table widget,



and double click on the Data Table to see the data in the spreadsheet format.



	Student	English	History	Algebra	Physics	Physicial
1	George	22.000	32.000	21.000	46.000	99.000
2	John	91.000	65.000	89.000	11.000	29.000
3	Thomas	51.000	21.000	100.000	100.000	27.000
4	James	9.000	18.000	61.000	90.000	8.000
5	John	93.000	39.000	12.000	17.000	63.000

Nice, everything is here.

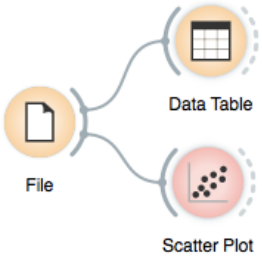
We can also use Google Sheets, a free online spreadsheet alternative. Then, instead of finding the file on the local disk, we would enter its URL address to the File widget's URL entry box.

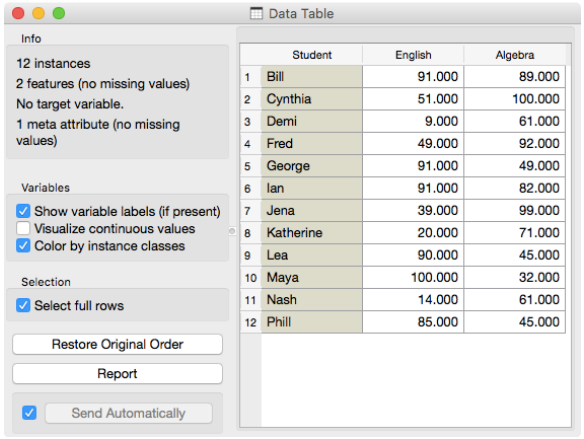
There is more to input data formatting and loading. We can define the type and kind of the data column, specify that the column is a web address of an image, and more. But enough for the first day. If you would like to dive deeper, check out the documentation page on Loading your Data, or a video on this subject.

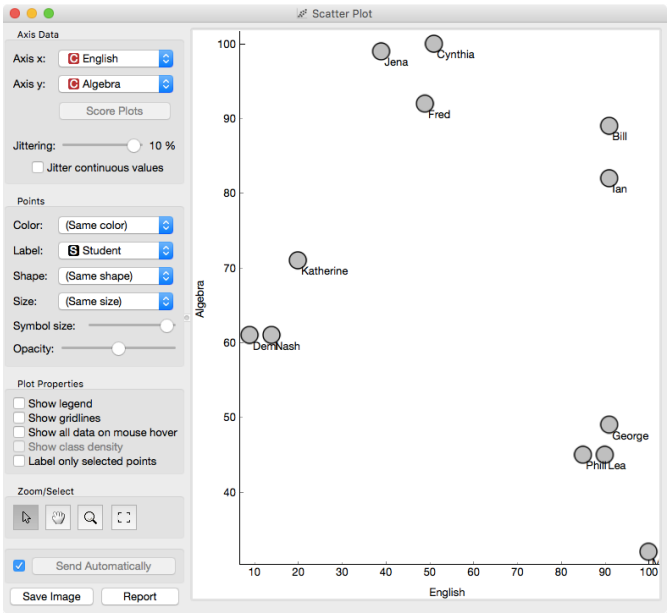
Lesson 5: Hierarchical Clustering

In the class, we will introduce clustering using a simple data set on students and their grades in English and Algebra. Load the data set from http://file.biolab.si/files/grades2.tab.

Say that we are interested in finding clusters in the data. That is, we would like to identify groups of data instances that are close together, similar to each other. Consider a simple, two-featured data set (see the side note) and plot it in the Scatter Plot. How many clusters do we have? What defines a cluster? Which data instances belong to the same cluster? What would a procedure for discovering clusters look like?







How do we measure the similarity between clusters if we only know the similarities between points? By default, Orange computes the average distance between all their pairs of data points; this is called average linkage. We could instead take the distance between the two closest points in each cluster (single linkage), or the two points that are furthest away (complete linkage).

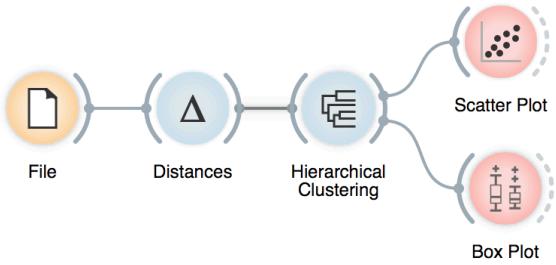
We need to start with a definition of "similar". One simple measure of similarity for such data is the Euclidean distance: square the differences across every dimension, some them and take the square root, just like in Pythagorean theorem. So, we would like to group data instances with small Euclidean distances.

Now we need to define a clustering algorithm. We will start with each data instance being in its own cluster. Next, we merge the clusters that are closest together - like the closest two points - into one cluster. Repeat. And repeat. And repeat. And repeat until you end up with a single cluster containing all points.

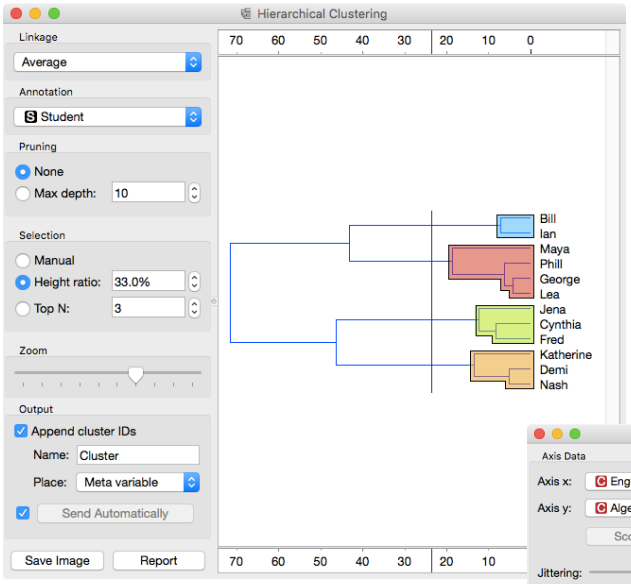
This procedure constructs a hierarchy of clusters, which explains why we call it hierarchical clustering. After it is done, we can

observe the entire hierarchy and decide which would be a good point to stop. With this we decide the actual number of clusters.

One possible way to observe the results of clustering on our small data set with grades is through the following workflow:



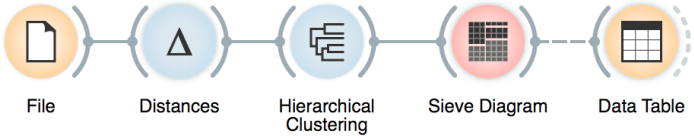
Let us see how this works. Load the data, compute the distances and cluster the data. In the Hierarchical clustering widget, cut hierarchy at a certain distance score and observe the corresponding clusters in the Scatter plot.



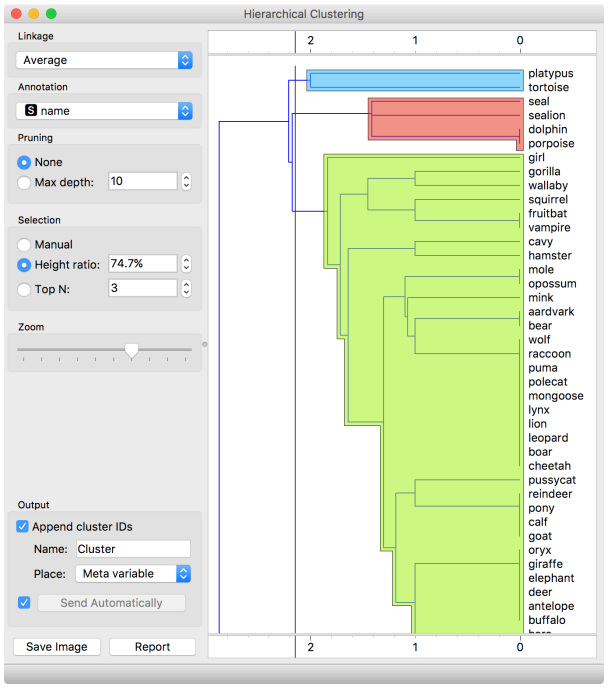
You can also observe the properties of the clusters - that is, the average grades in Algebra and English - in the box plot.

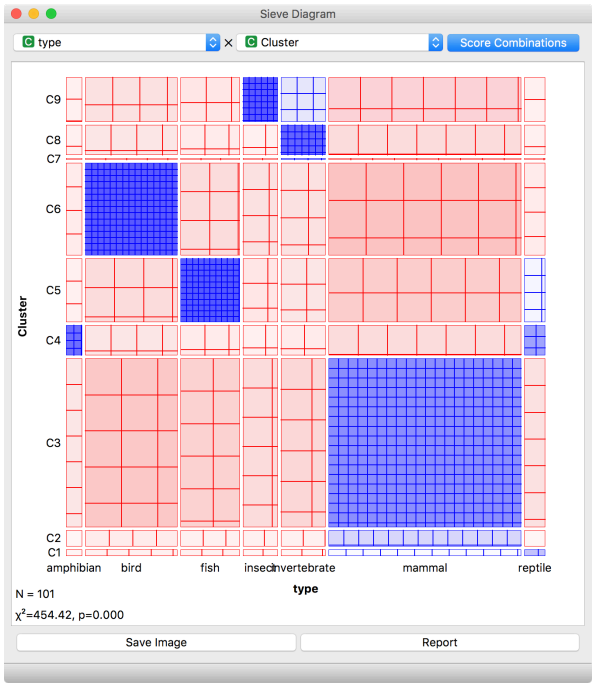
Lesson 6: Animal Kingdom

Your lecturers spent 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. Then our kids came, and the story repeated. Some things stay forever. 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.



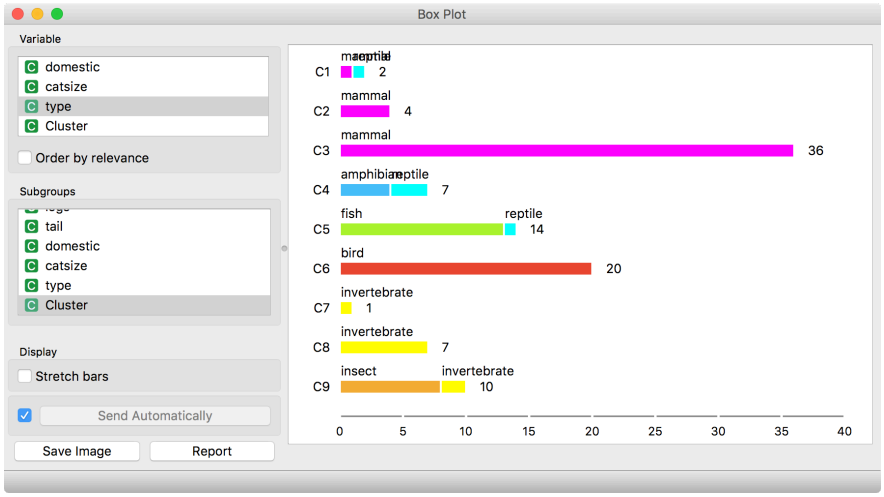
Here we use zoo data (from documentation data sets) with attributes that report on various features of animals (has hair, has feathers, lays eggs). We measure the distance and compute the clustering. Animals in this data set are annotated with type (mammal, insect, bird, and so on). It would be cool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



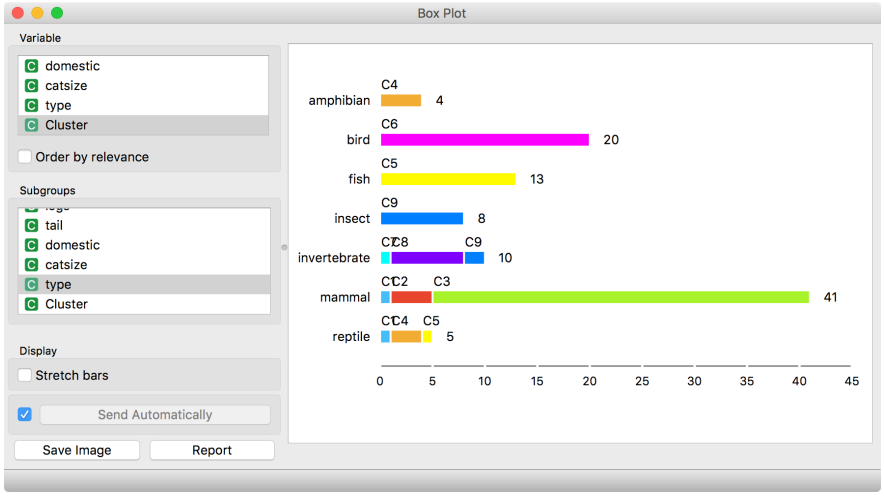


Looks great. Birds, say, are in cluster C6. Cluster C4 consists of amphibians and some reptiles. And so forth.

Checking this in the Box plot is even cooler. We can get a distribution of animal types in each cluster:



Or we can turn it around and see how different types of animals are spread across clusters.

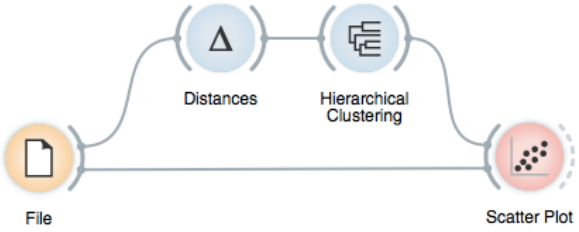


What is wrong with those mammals? Why can't they be in one single cluster? Two reasons. First, they represent 40 % of the data instances. Second, they include some weirdos. Click on the clusters in the box plot and discover who they are.

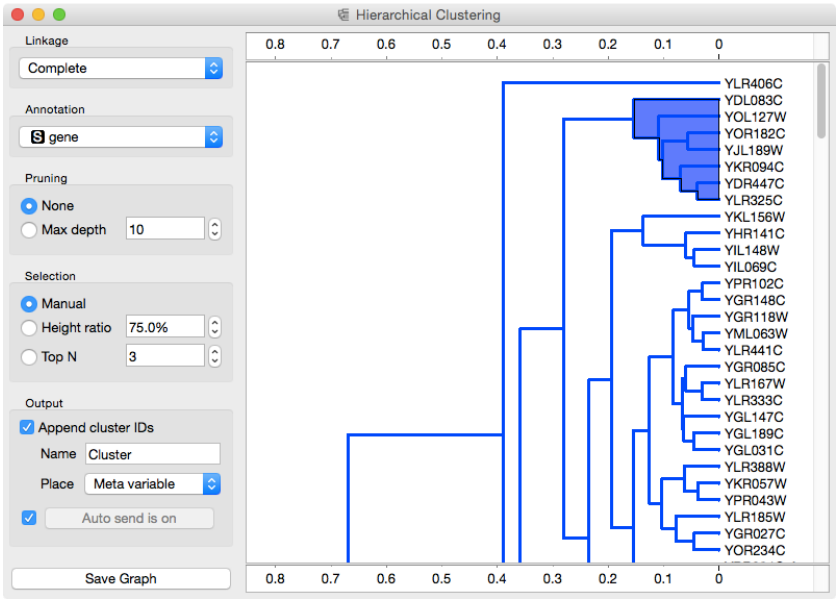
Lesson 7: Discovering clusters

Can we replicate this on some real data? Can clustering indeed be useful for defining meaningful subgroups?

Take brown-selected (from documentation data sets) connect the hierarchical clustering so the you can see a cluster as a subset in the scatterplot.



So far, we used the dendrogram to set a cut-off point. Now we will click on a branch in a dendrogram to select a subset of the data instances. By combining it with the Scatter Plot widget, we get a great tool for exploring the clusters. Try it with an appropriate pair of features to visualize (use Rank projections).



By using a scatter plot or other widgets, an expert can determine whether the clusters are meaningful.

For this data set, though, we can do something even better. The data already contains some predefined groups. Let us check how

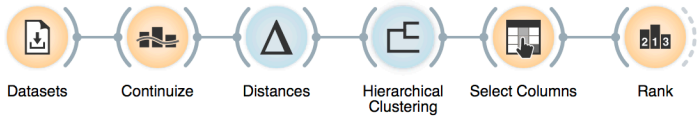
well the clusters match the classes - which we know, but clustering did not.

We will use the dendrogram to set a suitable threshold that splits the data into some three to five clusters. We can plot this data in a new scatter plot; we find a reasonable pair of attributes and then set the color of the points to represent the cluster they belong to. Do the clusters match the actual classes? The result is rather impressive if you keep two things in mind. First, the clustering algorithm did not actually know about the classes, it discovered them by itself. Second, it did not operate on the picture you see in the scatter plot and in which the clusters are quite pronounced, but in a 79-dimensional data space with possibly plenty of redundant features. Yet it identified the three groups of genes almost without mistakes.

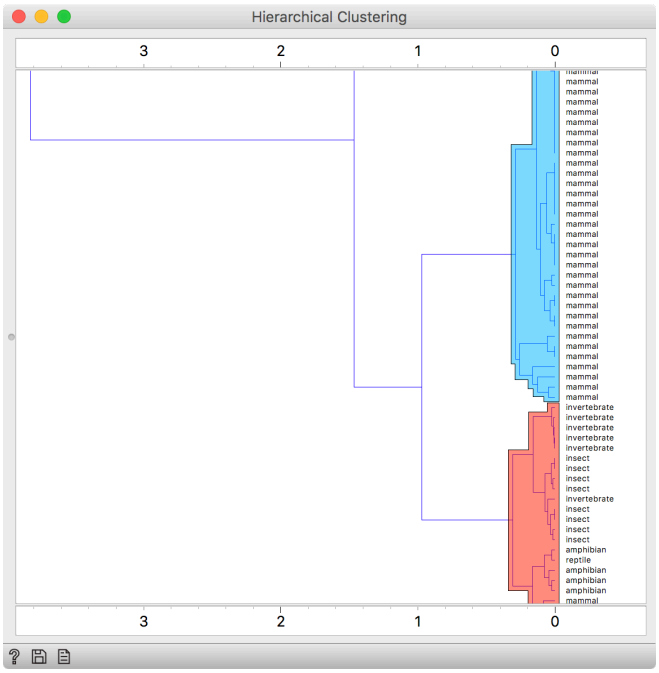
This lessons is not a recipe for what you should be doing in practice. If your data already contains groups labels, say gene group annotations, there is no need to discover them (again) by using clustering. In this case you should be interested in predictive models from previous lessons. If you do not have such a grouping but you suspect that the data contains distinct subgroups, run clustering. The sole purpose of this lesson was to demonstrate that clustering can indeed find a meaningful subgroups in the data; we pretend we did not know the groups, use the clustering to discover them, and checked how well the correspond to the actual groups.

Lesson 8: Cluster interpretation

Once we have inferred the clusters, we would like to know what are the distinguishing features. For the zoo data set, we could, for instance, mark two clusters, and then ask for the features that distinguish among these. Having data marked with cluster identifiers takes us back to classification, and we can use any of visualization, model inference, or feature ranking techniques we have introduced there. Here, we will show how to use ranking to infer what features characterize the group of mammals when compared to a close cluster of other species.

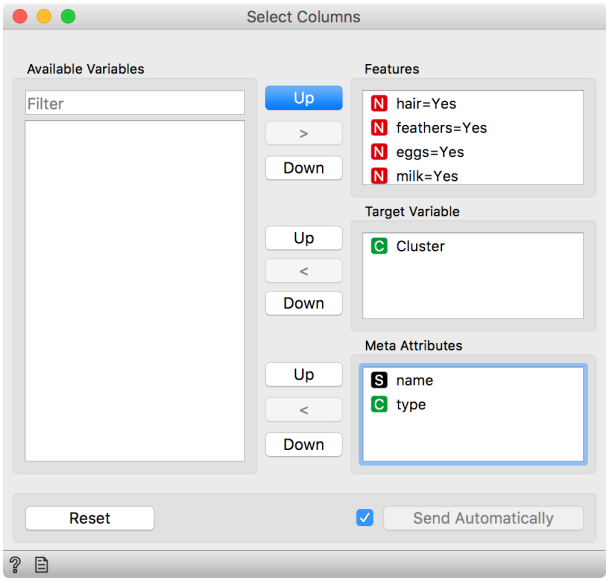


We load zoo data from Datasets, continuize the categorical features (this will be changed in Orange soon, Distances should automatically perform continuization), estimate the distances, and feed everything in Hierarchical Clustering. So far, nothing new.

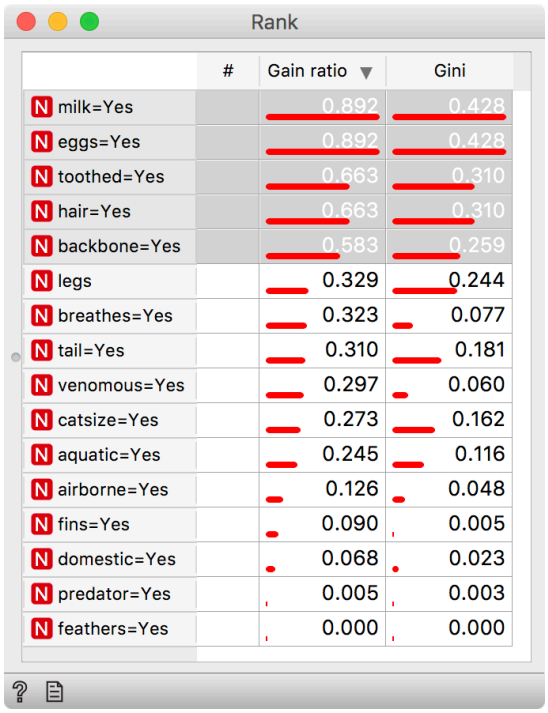


Use modifier keys (command) to select different branches of the dendrogram and mark them as separate clusters. Done correctly, Hierarchical Clustering will mark the branches with different colors.

In the hierarchical clustering, we choose two clusters. Note that Hierarchical Clustering adds cluster identifier as a meta feature; to make the data ready for classification-specific tasks, we need to promote cluster identifier into target variable (a class) by reassigning the feature types in Select Columns.



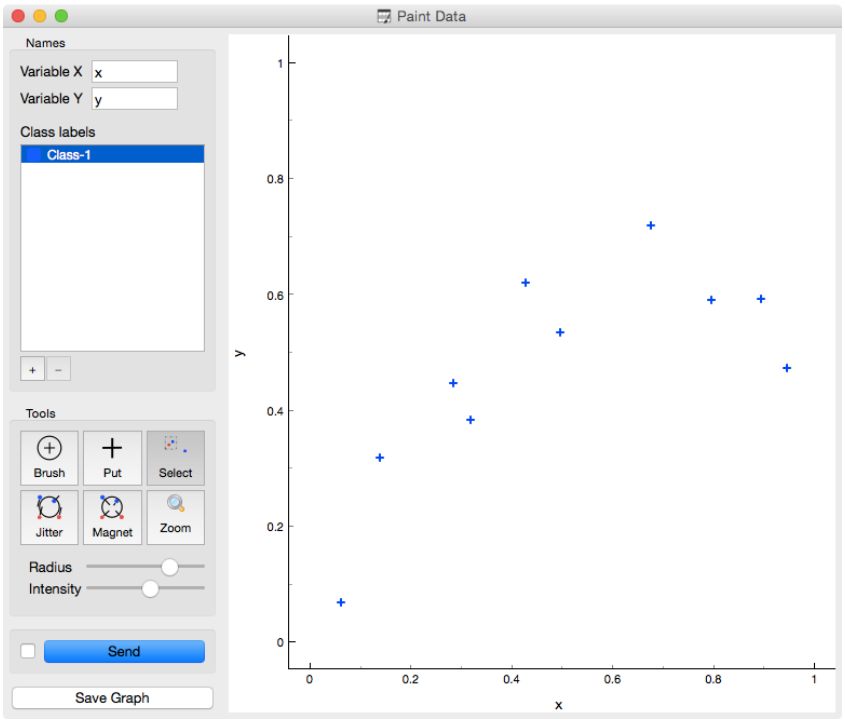
The data is now ready for classification-based analysis. Here, we used a rank widget.

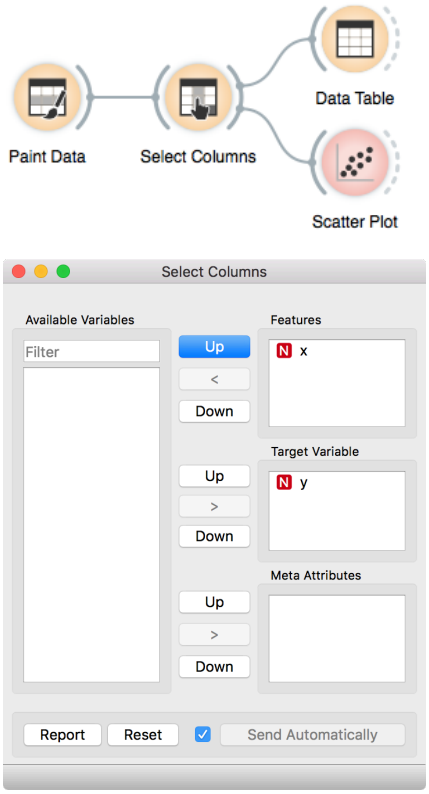


Lesson 9: Linear Regression

In the Paint Data widget, remove the Class-2 label from the list. If you have accidentally left it while painting, don't despair. The class variable will appear in the Select Columns widget, but you can "remove" it by dragging it into the Available Variables list.

For a start, let us construct a very simple data set. It will contain a just one continuous input feature (let's call it x) and a continuous class (y). We will use Paint Data, and then reassign one of the features to be a class by using Select Column and moving the feature y from the list of "Features" to a field with a target variable. It is always good to check the results, so we are including Data Table and Scatter Plot in the workflow at this stage. We will be modest this time and only paint 10 points and will use Put instead of the Brush tool.

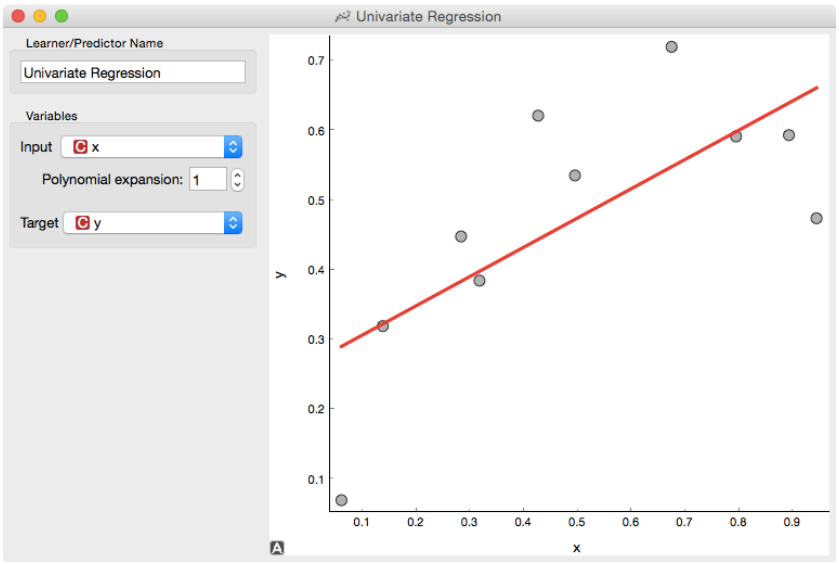




We would like to build a model that predicts the value of class y from the feature x. Say that we would like our model to be linear, to mathematically express it as $h(x)=\theta_0+\theta_1 x$. Oh, this is the equation of a line. So we would like to draw a line through our data points. The θ_0 is then an intercept, and θ_1 is a slope. But there are many different lines we could draw. Which one is the best one? Which one is the one that is a best fit to our data?

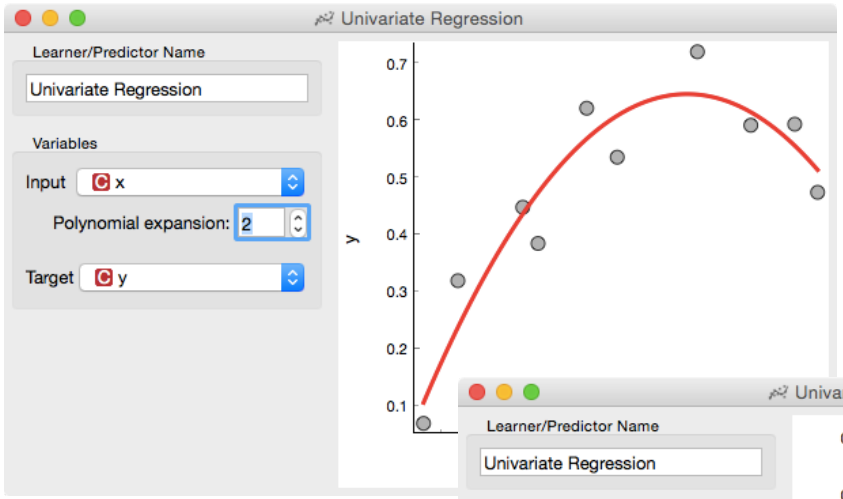
The questions above require us to define what is a good fit. Say, this could be the error the fitted model (the line) makes when it predicts the value of y for a given data point (value of x). The prediction is h(x), so the error is h(x) - y. We should treat the negative and positive errors equally, plus, let us agree, we would prefer punishing larger errors more severely than smaller ones. Therefore, it is perfectly ok if we square the errors for each data point and then sum them up. We got our objective function! Turns out that there is only one line that minimizes this function. The procedure that finds it is called linear regression. For cases where we have only one input feature, Orange has a special widget in the educational add-on called Polynomial Regression.

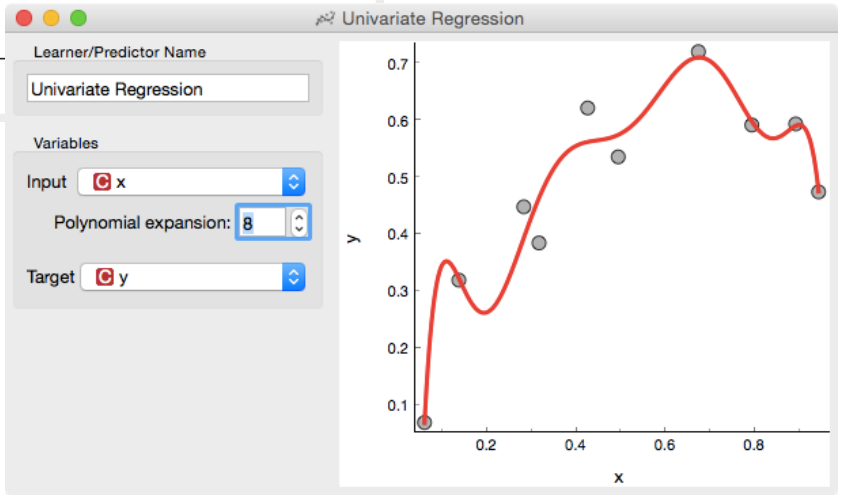
Do not worry about the strange name of the widget Polynomial Regression, we will get there in a moment.



Looks ok. Except that these data points do not appear exactly on the line. We could say that the linear model is perhaps too simple for our data sets. Here is a trick: besides column x, the widget Univariate Regression can add columns x^2, x^3... x^n to our data set. The number n is a degree of polynomial expansion the widget performs. Try setting this number to higher values, say to two, and then three, and then, say, to nine. With the degree of three, we are then fitting the data to a linear function h(x) = $\theta_0 + \theta_1 x + \theta_1 x^2 + \theta_1 x^3$.

The trick we have just performed (adding the higher order features to the data table and then performing linear regression) is called Polynomial Regression. Hence the name of the widget. We get something reasonable with polynomials of degree two or three, but then the results get really wild. With higher degree polynomials, we totally overfit our data.





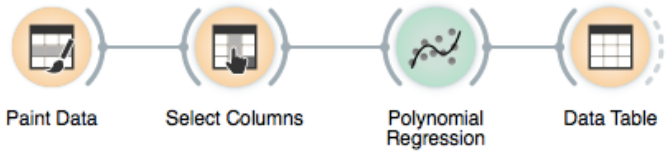
It is quite surprising to see that linear regression model can result in fitting non-linear (univariate) functions. That is, functions with curves, such as those on the figures. How is this possible? Notice though that the model is actually a hyperplane (a flat surface) in the space of many features (columns) that are powers of x. So for the degree 2, $\mathbf{h}(x)=\theta_0+\theta_1 x+\theta_1 x^2$ is a (flat) hyperplane. The visualization gets curvy only once we plot $\mathbf{h}(x)$ as a function of x.

Overfitting is related to the complexity of the model. In polynomial regression, the models are defined through parameters θ. The more parameters, the more complex is the model.

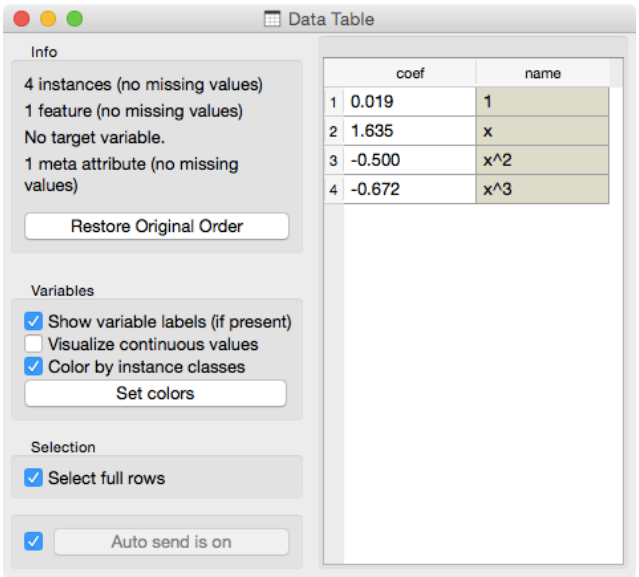
Obviously, the simplest model has just one parameter (an intercept), ordinary linear regression has two (an intercept and a slope), and polynomial regression models have as many parameters as is the degree of the polynomial. It is easier to overfit with a more complex model, as this can adjust to the data better. But is the overfitted model really discovering the true data patterns? Which of the two models depicted in the figures above would you trust more?

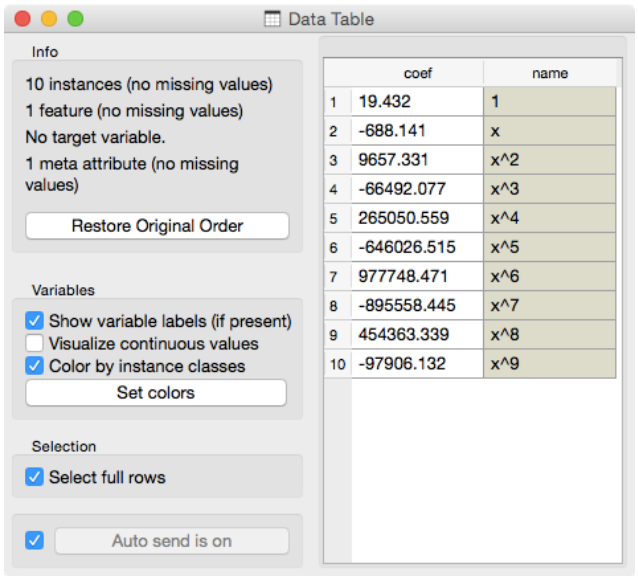
Lesson 10: Regularization

There has to be some cure for the overfitting. Something that helps us control it. To find it, let's check what the values of the parameters θ under different degrees of polynomials actually are



With smaller degree polynomials values of θ stay small, but then as the degree goes up, the numbers get really large.



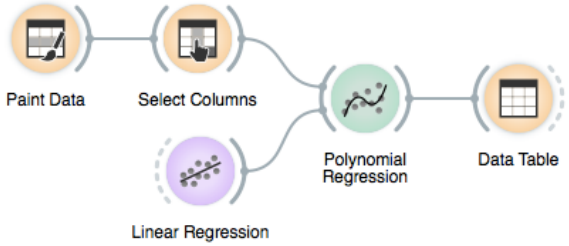


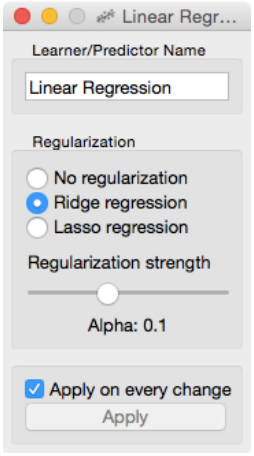
More complex models can fit the training data better. The fitted curve can wiggle sharply. The derivatives of such functions are high, and so need to be the coefficients θ. If only we could force the linear regression to infer models with a small value of coefficients. Oh, but we can. Remember, we have started with the optimization function the linear regression minimizes, the sum of squared errors. We could simply add to this a sum of all θ squared. And ask the linear regression to minimize both terms. Perhaps we should weigh the part with θ squared, say, we some coefficient λ, just to control the level of regularization.

Which inference of linear model would overfit more, the one with high λ or the one with low λ? What should the value of λ be to cancel regularization? What if the value of λ is really high, say 1000?

Internally, if no learner is present on its input, the Polynomial Regression widget would use just its ordinary, non-regularized linear regression.

Here we go: we just reinvented regularization, a procedure that helps machine learning models not to overfit the training data. To observe the effects of the regularization, we can give Polynomial Regression our own learner, which supports these kind of settings.





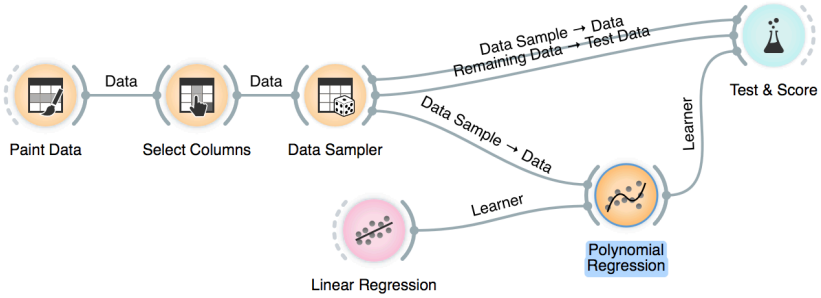
The Linear Regression widget provides two types of regularization. Ridge regression is the one we have talked about and minimizes the sum of squared coefficients θ. Lasso regression minimizes the sum of absolute value of coefficients. Although the difference may seem negligible, the consequences are that lasso regression may result in a large proportion of coefficients θ being zero, in this way performing feature subset selection.

Now for the test. Increase the degree of polynomial to the max. Use Ridge Regression. Does the inferred model overfit the data? How does degree of overfitting depend on regularization strength?

Lesson 11: Regularization and Accuracy on Test Set

Overfitting hurts. Overfit models fit the training data well, but can perform miserably on new data. Let us observe this effect in regression. We will use hand-painted data set, split it into the training (50%) and test (50%) data set, polynomially expand the training data set to enable overfitting, build a model on it, and test the model on both the (seen) training data and the (unseen) held-out data:

Paint about 20 to 30 data instances. Use attribute y as target variable in Select Columns. Split the data 50:50 in Data Sampler. Cycle between test on train or test data in Test & Score. Use ridge regression to build linear regression model.



Now we can vary the regularization strength in Linear Regression and observe the accuracy in Test & Score. For accuracy scoring, we will use RMSE, root mean squared error, which is computed by observing the error for each data point, squaring it, averaging this across all the data instances, and taking a square root. And we will also make use of coefficient of determination, denoted R^2 or r^2, the proportion of the variance in the dependent variable that is predictable from the independent variable(s).

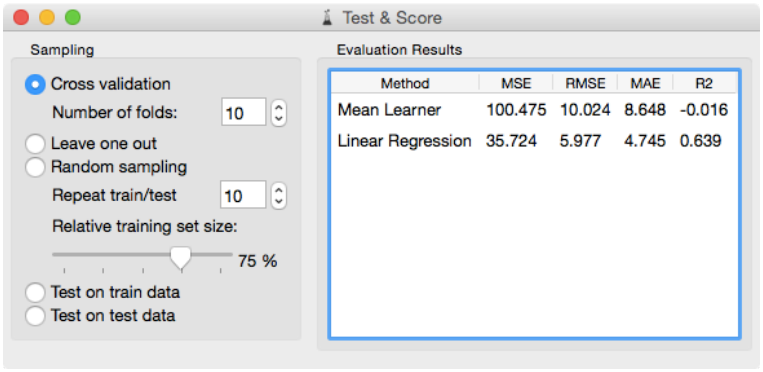
Orange is currently not equipped with parameter fitting and we need to find the optimal level of regularization manually. At this stage, it suffices to say that parameters must be found on the training data set without touching the test data.

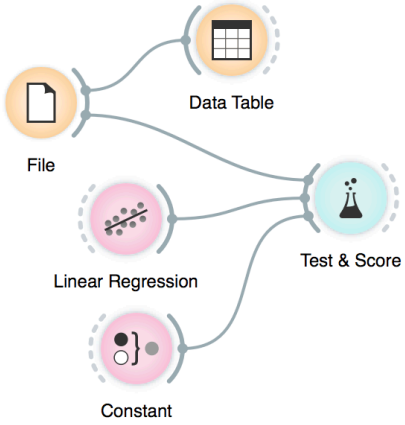
The core of this lesson is to compare the error on the training and test set while varying the level of regularization. Remember, regularization controls overfitting - the more we regularize, the less tightly we fit the model to the training data. So for the training set, we expect the error to drop with less regularization and more overfitting, and to increase with more regularization and less fitting. No surprises expected there. But how does this play out on the test set? Which sides minimizes the test-set error? Or is the optimal level of regularization somewhere in between? How do we estimate this level of regularization from the training data alone?

Lesson 12: Prediction of Tissue Age from Level of Methylation

Download the methylation data set from http://file.biolab.si/files/methylation.tab. Predictions of age from methylation profile were investigated by Horvath (2013) Genome Biology 14:R115.

Enough painting. Now for the real data. We will use a data set that includes human tissues from subjects at different age. The tissues were profiled by measurements of DNA methylation, a mechanism for cells to regulate the gene expression. Methylation of DNA is scarce when we are young, and gets more abundant as we age. We have prepared a data set where the degree of methylation was expressed per each gene. Let us test if we can predict the age from the methylation profile - and if we can do this better than by just predicting the average age of subjects in the training set.





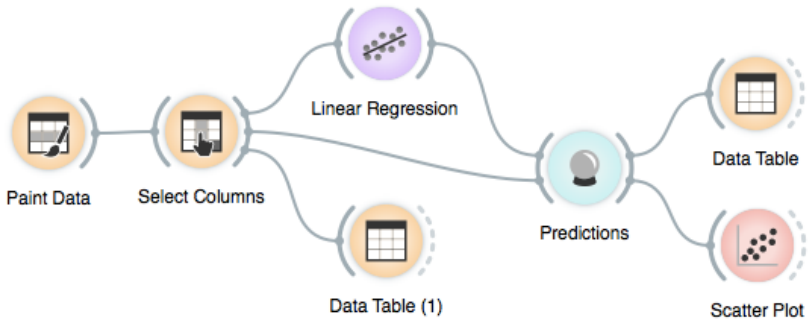
Using other learners, like random forests, takes a while on this data set. But you may try to sample the features, obtain a smaller data set, and try various regression learners.

This workflow looks familiar and is similar to those for classification problems. The Test & Score widget reports on statistics we have not seen before. MAE, for one, is the mean average error. Just like for classification, we have used cross-validation, so MAE was computed only on the test data instances and averaged across 10 runs of cross validation. The results indicate that our modeling technique misses the age by about 5 years, which is a much better result than predicting by the mean age in the training set.

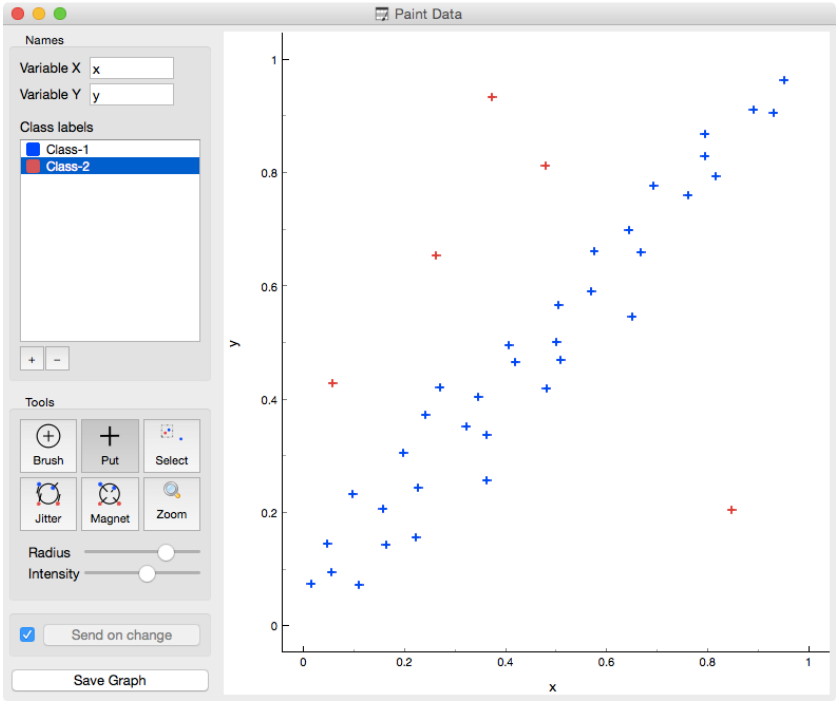
Lesson 13: Evaluating Regression

The last lessons quickly introduced scoring for regression, and important measures such as RMSE and MAE. In classification, a nice addition to find misclassified data instances was the confusion matrix. But the confusion matrix could only be applied to discrete classes. Before Orange gets some similar for regression, one way to find misclassified data instances is through scatter plot!

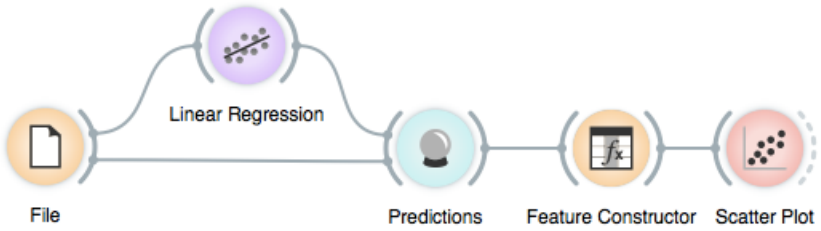
This workflow visualizes the predictions that were performed on the training data. How would you change the widget to use a separate test set? Hint: The Sample widget can help.

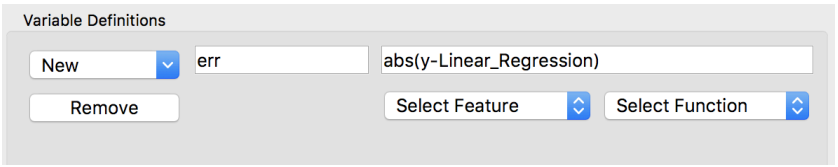


We can play around with this workflow by painting the data such that the regression would perform well on blue data point and fail on the red outliers. In the scatter plot we can check if the difference between the predicted and true class was indeed what we have expected.



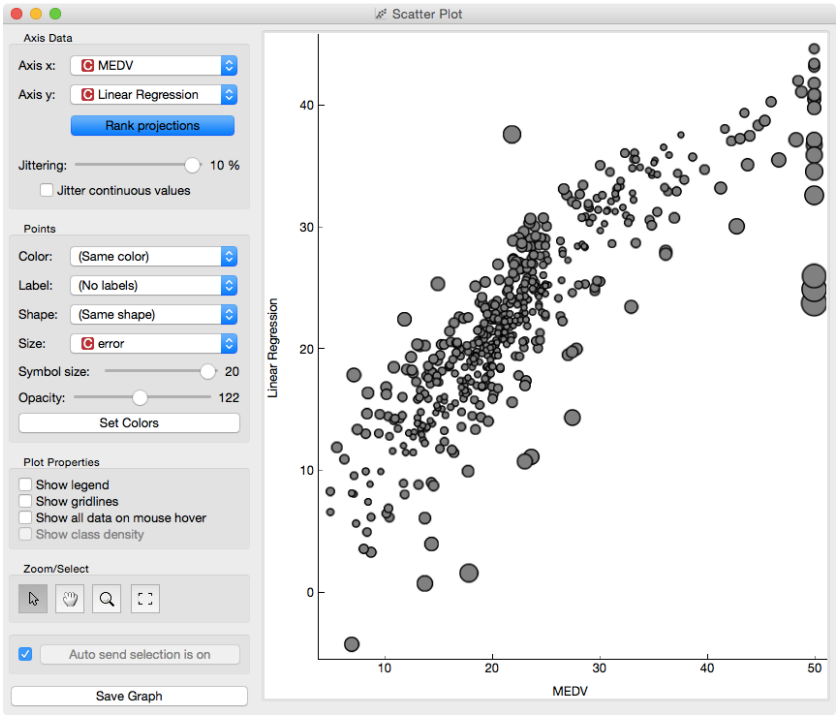
A similar workflow would work for any data set. Take, for instance, the housing data set (from Orange distribution). Say, just like above, we would like to plot the relation between true and predicted continuous class, but would like to add information on the absolute error the predictor makes. Where is the error coming from? We need a new column. The Feature Constructor widget (albeit being a bit geekish) comes to the rescue.





In the Scatter Plot widget, we can now select the data where the predictor erred substantially and explore the results further.

We could, in principle, also mine the errors to see if we can identify data instances for which this was high. But then, if this is so, we could have improved predictions at such regions. Like, construct predictors that predict the error. This is weird. Could we then also construct a predictor, that predicts the error of the predictor that predicts the error? Strangely enough, such ideas have recently led to something called Gradient Boosted Trees, which are nowadays among the best regressors (and are coming to Orange soon).



Lesson 14: Feature Scoring and Selection

For this lesson, load the data from imports-85.tab using the File widget and Browse documentation data sets.

Linear regression infers a model that estimate the class, a real-valued feature, as a sum of products of input features and their weights. Consider the data on prices of imported cars in 1985.

Data Table (1)

	height	curb-weight	engine-type	num-of-cylinders	engine-size	fuel-system	bore	stroke
1	48.800	2548.000	dohc	four	130.000	mpfi	3.470	2.680
2	48.800	2548.000	dohc	four	130.000	mpfi	3.470	2.680
3	52.400	2823.000	ohcv	six	152.000	mpfi	2.680	3.470
4	54.300	2337.000	ohc	four	109.000	mpfi	3.190	3.400
5	54.300	2824.000	ohc	five	136.000	mpfi	3.190	3.400
6	53.100	2507.000	ohc	five	136.000	mpfi	3.190	3.400
7	55.700	2844.000	ohc	five	136.000	mpfi	3.190	3.400
8	55.700	2954.000	ohc	five	136.000	mpfi	3.190	3.400
9	55.900	3086.000	ohc	five	131.000	mpfi	3.130	3.400
10	52.000	3053.000	ohc	five	131.000	mpfi	3.130	3.400
11	54.300	2395.000	ohc	four	108.000	mpfi	3.500	2.800
12	54.300	2395.000	ohc	four	108.000	mpfi	3.500	2.800
13	54.300	2710.000	ohc	six	164.000	mpfi	3.310	3.190
14	54.300	2765.000	ohc	six	164.000	mpfi	3.310	3.190
15	55.700	3055.000	ohc	six	164.000	mpfi	3.310	3.190
16	55.700	3230.000	ohc	six	209.000	mpfi	3.620	3.390
17	53.700	3380.000	ohc	six	209.000	mpfi	3.620	3.390
18	56.300	3505.000	ohc	six	209.000	mpfi	3.620	3.390

Inspecting this data set in a Data Table, it shows that some features, like fuel-system, engine-type and many others, are discrete. Linear regression only works with numbers. In Orange, linear regression will automatically convert all discrete values to numbers, most often using several features to represent a single discrete features. We also do this conversion manually by using Continuize widget.

Continuize

Categorical Features
- Target or first value as base
- Most frequent value as base
- One attribute per value
- Ignore multinomial attributes
- Remove categorical attributes
- Treat as ordinal
- Divide by number of values

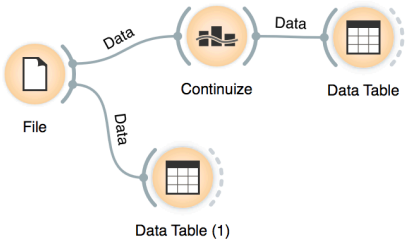
Numeric Features
- Leave them as they are
- Normalize by span
- Normalize by standard deviation

Categorical Outcomes
- Leave it as it is
- Treat as ordinal
- Divide by number of values
- One class per value

Value Range
- From -1 to 1
- From 0 to 1

Report

Apply Automatically

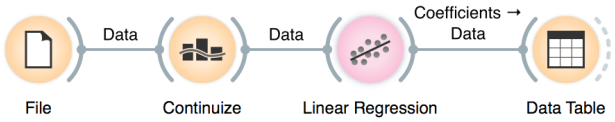


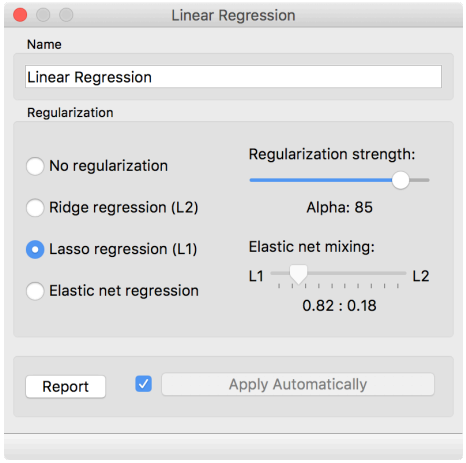
Before we continue, you should check what Continuize actually does and how it converts the nominal features into real-valued features. The table below should provide sufficient illustration.

Data Table

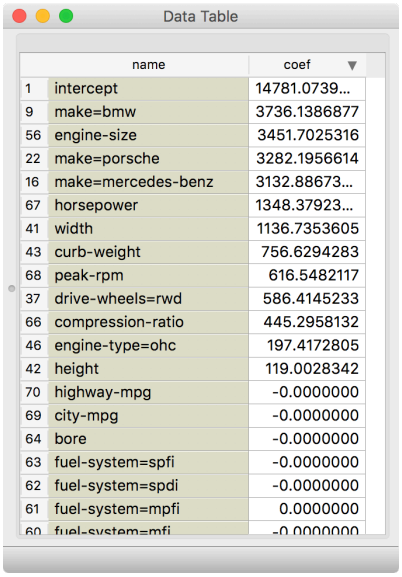
	symboling=3	ormalized-losses	make=audi	make=bmw	make=chevrolet	make=dodge
1	1.000	?	0.000	0.000	0.000	0.000
2	1.000	?	0.000	0.000	0.000	0.000
3	0.000	?	0.000	0.000	0.000	0.000
4	0.000	1.189	1.000	0.000	0.000	0.000
5	0.000	1.189	1.000	0.000	0.000	0.000
6	0.000	?	1.000	0.000	0.000	0.000
7	0.000	1.019	1.000	0.000	0.000	0.000
8	0.000	?	1.000	0.000	0.000	0.000
9	0.000	1.019	1.000	0.000	0.000	0.000
10	0.000	?	1.000	0.000	0.000	0.000
11	0.000	1.981	0.000	1.000	0.000	0.000
12	0.000	1.981	0.000	1.000	0.000	0.000
13	0.000	1.868	0.000	1.000	0.000	0.000
14	0.000	1.868	0.000	1.000	0.000	0.000
15	0.000	?	0.000	1.000	0.000	0.000
16	0.000	?	0.000	1.000	0.000	0.000
17	0.000	?	0.000	1.000	0.000	0.000
18	0.000	?	0.000	1.000	0.000	0.000
19	0.000	-0.028	0.000	0.000	1.000	0.000
20	0.000	-0.679	0.000	0.000	1.000	0.000

Now to the core of this lesson. Our workflow reads the data, coninuizes it such that we also normalize all the features to bring them the to equal scale, then we load the data into Linear Regression widget and check out the feature coefficients in the Data Table.





In Linear Regression, we will use L1 regularization. Compared to L2 regularization, which aims to minimize the sum of squared weights, L1 regularization is more rough and minimizes the sum of absolute values of the weights. The result of this "roughness" is that many of the feature will get zero weights.



But this may be also exactly what we want. We want to select only the most important features, and want to see how the model that uses only a smaller subset of features actually behaves. Also, this smaller set of features is ranked. Engine size is a huge factor in pricing of our cars, and so is the make, where Porsche, Mercedes and BMW cost more than other cars (ok, no news here).

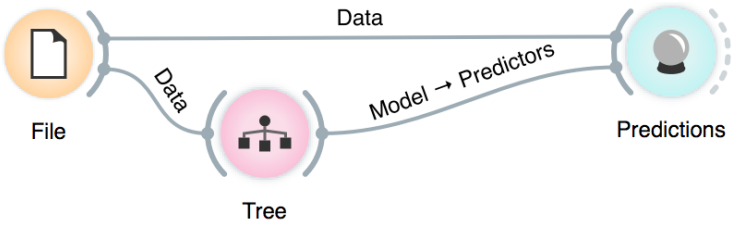
We should notice that the number of features with non-zero weights varies with regularization strength. Stronger regularization would result in fewer features with non-zero weights.

Lesson 11: Classification

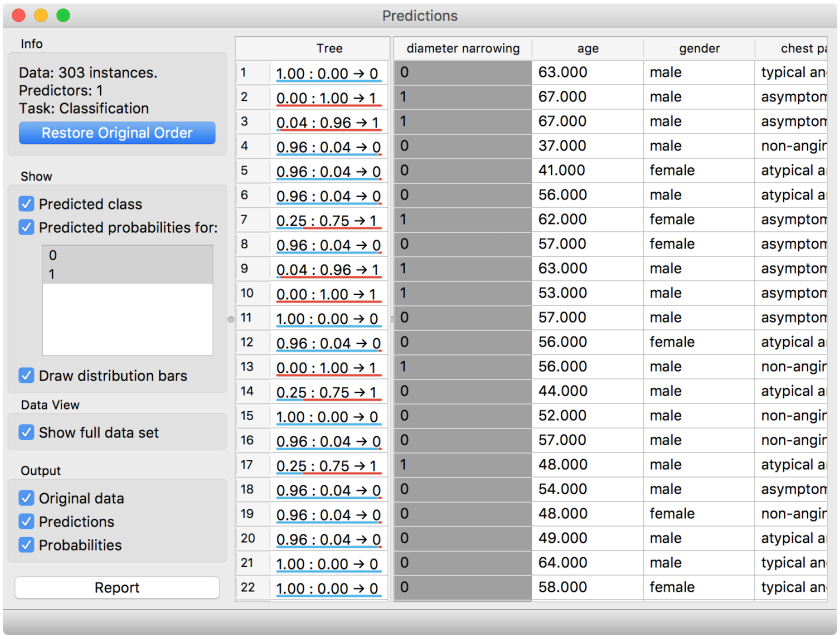
Heart disease data was presented in notes, but we have not gone through it in the class. While you are encouraged to explore it, you can also use the workflows from this lesson on, say, iris data set.

In one of the previous lessons, we explored the heart disease data. We wanted to predict which persons have clogged arteries — but we did not make any predictions. Let's try it now.

This won't do: the widget Predictions shows the data, but no makes no predictions. It can't. For this, it needs a model. Like this.



The data is fed into the Tree widget, which uses it to infer a predictive model. The Predictions widget now gets the data from the File widget and also a predictive model from the Tree widget. This is something new: in our past workflows, widgets passed only data to each other, but here we have a channel that carries a model.



	Tree	diameter narrowing	age	gender	chest pa
1	1.00 : 0.00 → 0	0	63.000	male	typical an
2	0.00 : 1.00 → 1	1	67.000	male	asympton
3	0.04 : 0.96 → 1	1	67.000	male	asympton
4	0.96 : 0.04 → 0	0	37.000	male	non-angir
5	0.96 : 0.04 → 0	0	41.000	female	atypical a
6	0.96 : 0.04 → 0	0	56.000	male	atypical a
7	0.25 : 0.75 → 1	1	62.000	female	asympton
8	0.96 : 0.04 → 0	0	57.000	female	asympton
9	0.04 : 0.96 → 1	1	63.000	male	asympton
10	0.00 : 1.00 → 1	1	53.000	male	asympton
11	1.00 : 0.00 → 0	0	57.000	male	asympton
12	0.96 : 0.04 → 0	0	56.000	female	atypical a
13	0.00 : 1.00 → 1	1	56.000	male	non-angir
14	0.25 : 0.75 → 1	0	44.000	male	atypical a
15	1.00 : 0.00 → 0	0	52.000	male	non-angir
16	0.96 : 0.04 → 0	0	57.000	male	non-angir
17	0.25 : 0.75 → 1	1	48.000	male	atypical a
18	0.96 : 0.04 → 0	0	54.000	male	asympton
19	0.96 : 0.04 → 0	0	48.000	female	non-angir
20	0.96 : 0.04 → 0	0	49.000	male	atypical a
21	1.00 : 0.00 → 0	0	64.000	male	typical an
22	1.00 : 0.00 → 0	0	58.000	female	typical an

The Predictions widget uses the model to make predictions about the data and shows them in the table.

How correct are these predictions? Do we have a good model? How can we tell?

But (and even before answering these critical questions), what is a tree? How does it look like? How does Orange create one? Is this algorithm something we should use? So many questions to answer today!

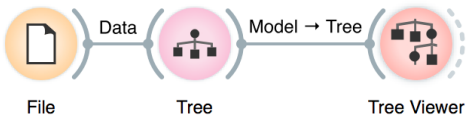
Lesson 15: Classification

We learned how to predict numeric values, like tissue age. What if the value we need to predict is categorical, like "yes" or "no", or "red", "blue", "green" or "white"? Such target variables are often usually called classes, so predicting class values is called classification, and models are referred to as classifiers. Fitting of such models was traditionally in domain of machine learning.

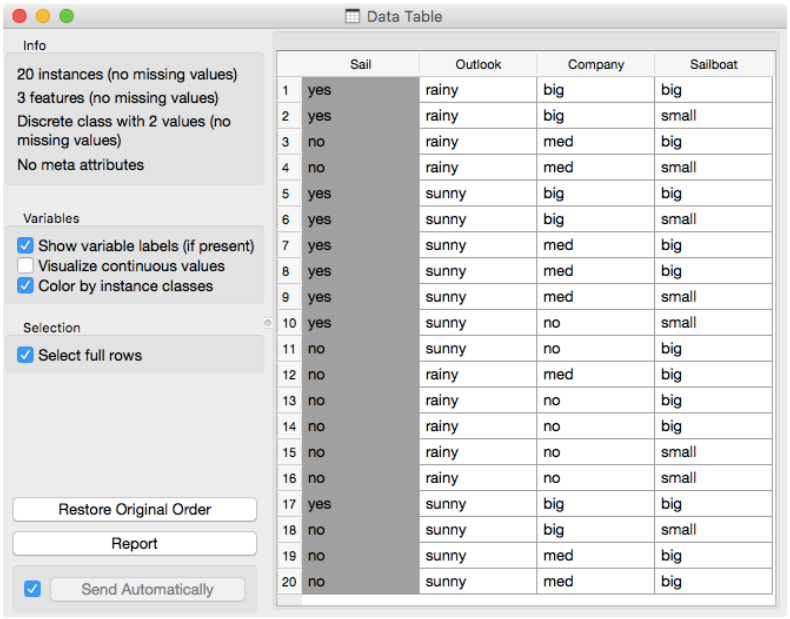
Classification tree is one of the oldest, but still popular, machine learning methods. We like it since the method is easy to explain and gives rise to random forests, one of the most accurate machine learning techniques. So, what kind of model is a classification tree?

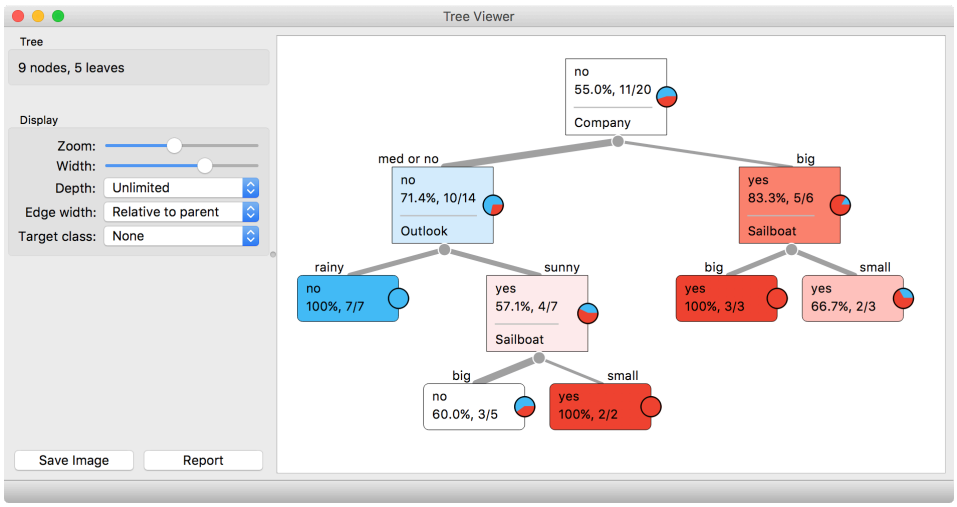
Let us load a data set from http://file.biolab.si/datasets/sailing.tab that records the conditions under which a skipper went sailing, build a tree with a Tree widget and visualize it in the Tree Viewer.

The data set we will use is stored on a server. Copy the web address and paste it into URL entry box in the File widget. An alternative way to access this data is to use the Data Sets widget that is currently available in the Prototypes add-on.



Here's a warning: this sailing data is small. Therefore, any relations inferred from the classification tree on this page are unreliable. What should the size of the data set be to acquire stronger conclusions?





We read the tree from top to bottom. It looks like this skipper is a social person; as soon as there's company, the probability of her sailing increases. When joined by a smaller group of individuals, there is no

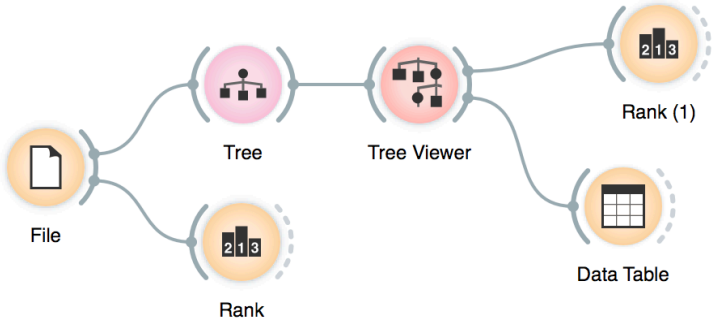
sailing if there is rain. (Thunderstorms? Too dangerous?) When she has a smaller company, but the boat at her disposal is big, there is no sailing either.

Classification trees were hugely popular in the early years of machine learning, when they were first independently proposed by the engineer Ross Quinlan (C4.5) and a group of statisticians (CART), including the father of random forests Leo Brieman.

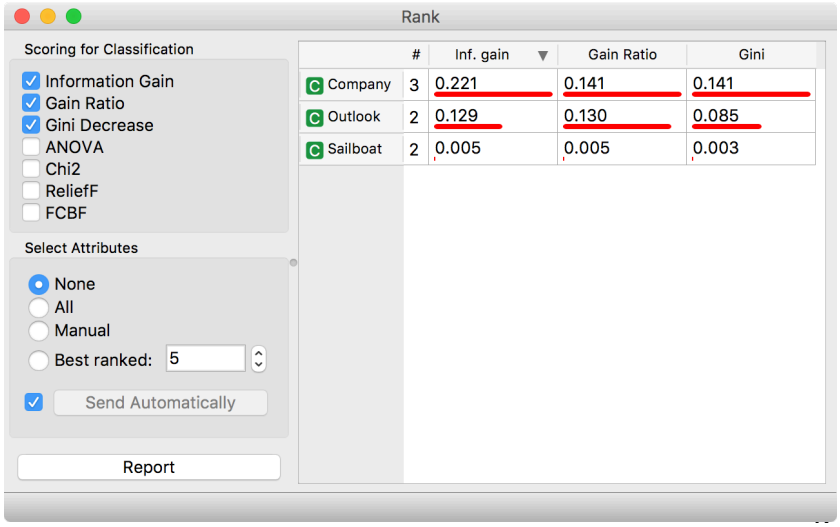
Trees place the most useful feature at the root. What would be the most useful feature? It is the feature that splits the data into two purest possible subsets. These are then split further, again by the most informative features. This process of breaking up the data subsets to smaller ones repeats until we reach subsets where all data belongs to the same class. These subsets are represented by leaf nodes in strong blue or red. The process of data splitting can also terminate when it runs out of data instances or out of useful features (the two leaf nodes in white).

The Rank widget could be used on its own. Say, to figure out which genes are best predictors of the phenotype in some gene

We still have not been very explicit about what we mean by "the most useful" feature. There are many ways to measure this. We can compute some such scores in Orange using the Rank widget, which estimates the quality of data features and ranks them according to how much information they carry. We can compute the scores from the whole data set or from data corresponding to some node of the classification tree in the Tree Viewer.



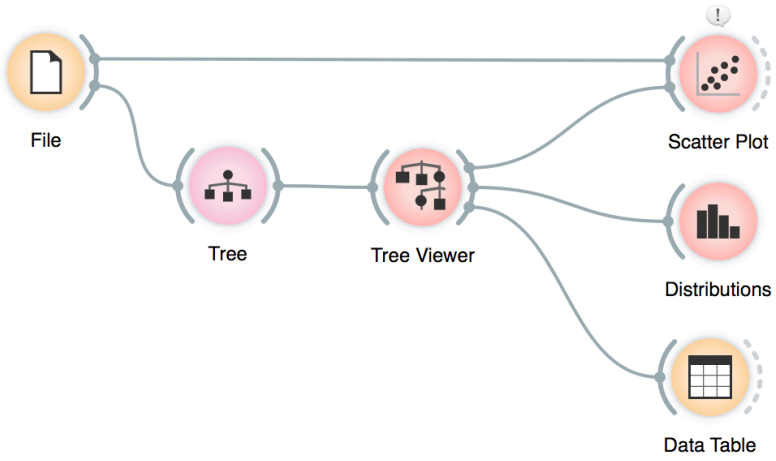
In this class, we will not dive into definitions. If you are interested, there's a good explanation of information gain on stackoverflow.com.



Lesson 13: Model Inspection

Here's another interesting combination of widgets: the classification tree viewer and the scatterplot. This time, consider the famous Iris data set (comes with Orange). In the Scatter Plot, find the best visualization of this data set, that is, the one that best separates the instances from different classes. Then connect the Tree Viewer to the Scatterplot. Selecting any node of the tree will output the corresponding data subset, which will be shown in the scatter plot.

Wherever possible, visualizations in Orange are designed to support selection and passing of the data that applies to it. Finding interesting data subsets and analyzing their commonalities is a central part of explorative data analysis, a data analysis approach favored by the data visualization guru Edward Tufte.



Just for fun, we have included a few other widgets in this workflow. In a way, the Tree Viewer widget behaves like the Select Rows widget, except that the rules used to filter the data are inferred from the data itself and optimized to obtain purer data subsets.

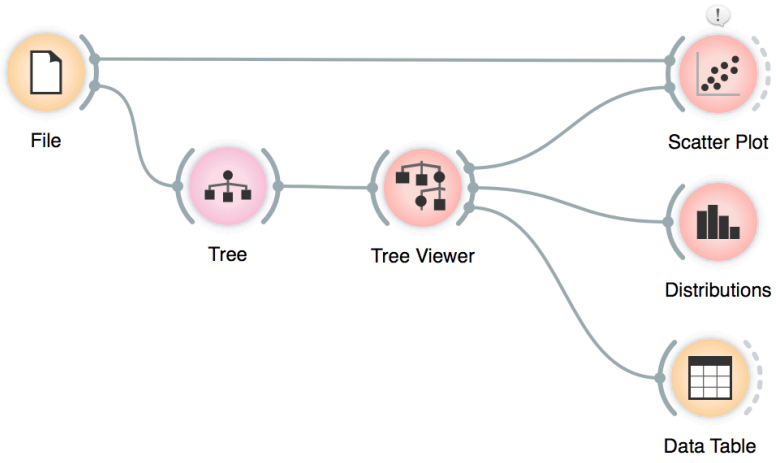
Lesson 16: Model Inspection

As Arthur Weasley used to say, you should never trust a thing if you don't know where it keeps its brains. For predictive models, it is always great to see them and understand how they make decisions. This may let us learn about patterns they spotted in the data, interpret predictions they make, it may help us improve the models, or collect better data.

Some models can be explored in this way and some can't. Trees are obviously of the former kind: the first this we ever did with the tree was showing the entire model. Now we shall explore the model on some data.

Let us go back to the Brown-selected data set, which we have already encountered in the first lesson. Feed the data to Tree and Tree Viewer. But then also add a Scatter plot, give it the data from the file and the also the data from Tree Viewer. Selecting any node of the tree will output the corresponding data subset, which will be shown in the scatter plot. Which two variables will we choose in the Scatter plot to be able to observe how the tree works?

Wherever possible, visualizations in Orange are designed to support selection and passing of the data that applies to it. Finding interesting data subsets and analyzing their commonalities is a central part of explorative data analysis, a data analysis approach favored by the data visualization guru Edward Tufte.

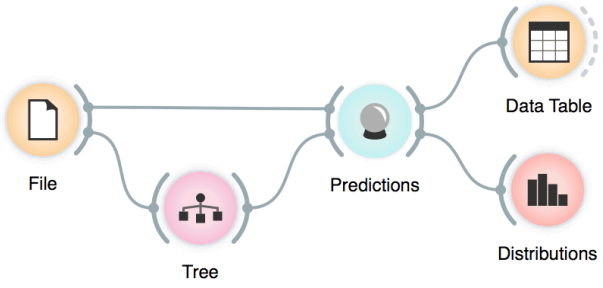


We can also use Distributions widget to see the distribution of classes in each node. In the case of the Brown data, this is pretty boring, though.

Lesson 17: Classification Accuracy

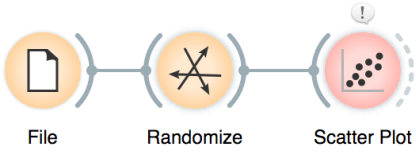
Now that we know what classification trees are, the next question is what is the quality of their predictions. For beginning, we need to define what we mean by quality. In classification, the simplest measure of quality is classification accuracy expressed as the proportion of data instances for which the classifier correctly guessed the value of the class. Let's see if we can estimate, or at least get a feeling for, classification accuracy with the widgets we already know.

Measuring of accuracy is such an important concept that it would require its widget. But wait a while, there's educational value in reusing the widgets we already know.



Let us try this schema with the brown-selected data set. The Predictions widget outputs a data table augmented with a column that includes predictions. In the Data Table widget, we can sort the data by any of these two columns, and manually select data instances where the values of these two features are different (this would not work on big data). Roughly, visually estimating the accuracy of predictions is straightforward in the Distribution widget, if we set the features in view appropriately.

This lesson has a strange title and it is not obvious why it was chosen. Maybe you, the reader, should tell us what does this lesson have to do with cheating.

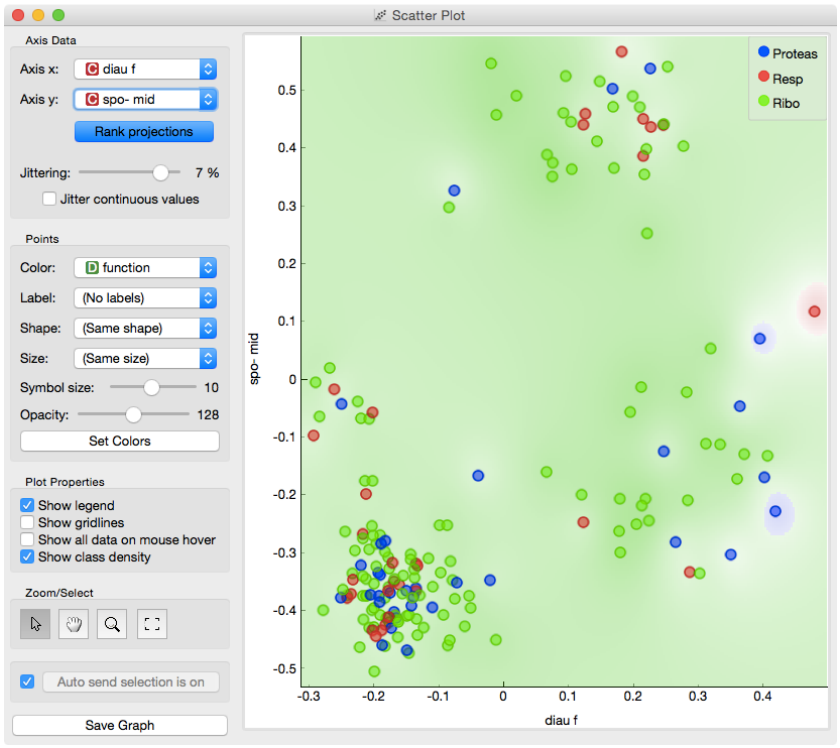


Randomize widget shuffles the column in the data table. It can shuffle the class column, columns with data features or columns with meta information. Shuffling the class column breaks any relation between features and the class, keeping the data points (genes profiles) intact.

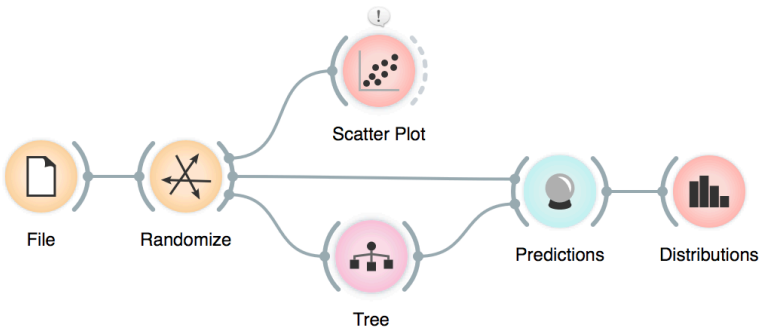
Why is the background in this scatter plot so green, and only green? Why have the other colors disappeared after the class randomization?

Lesson 18: How to Cheat

At this stage, the classification tree looks very good. There's only one data point where it makes a mistake. Can we mess up the data set so bad that the trees will ultimately fail? Like, remove any existing correlation between gene expression profiles and class? We can! There's the Randomize widget that can shuffle the class column. Check out the chaos it creates in the Scatter Plot visualization where there were nice clusters before randomization!

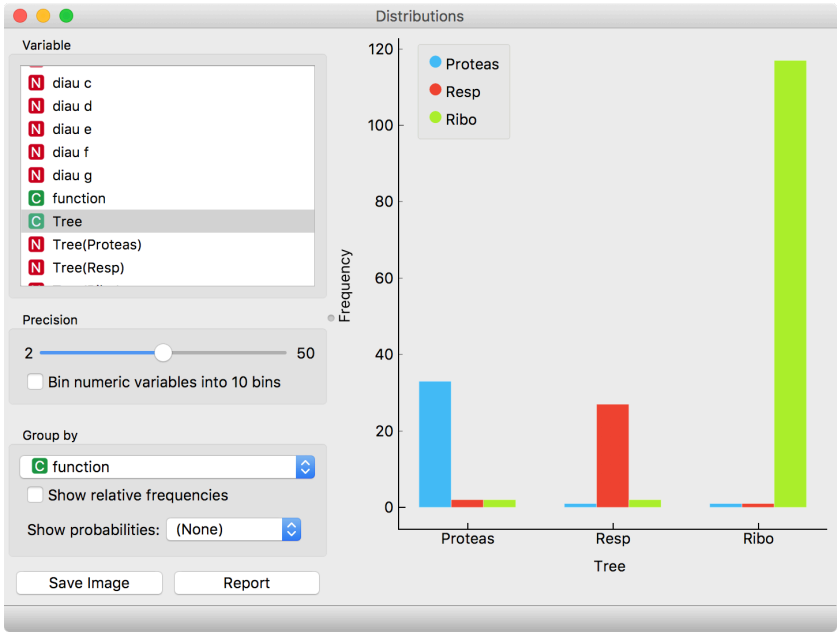


Fine. There can be no classifier that can model this mess, right? Let us test this. We will build classification tree and check its performance on the messed-up data set.



And the result? Here is a screenshot of the Distributions:

At this stage, it may be worthwhile checking how do the trees look. Try comparing the tree inferred from original and shuffled data!



Most unusual. Almost no mistakes. How is this possible? On a class-randomized data set?

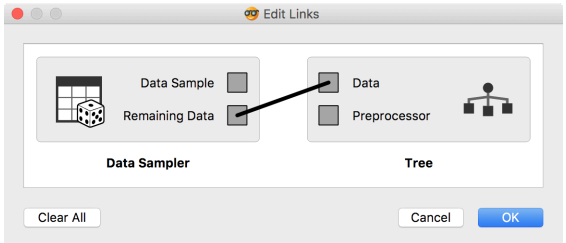
To find the answer to this riddle, open the Tree Viewer and check out the tree. How many nodes does it have? Are there many data instances in the leaf nodes?

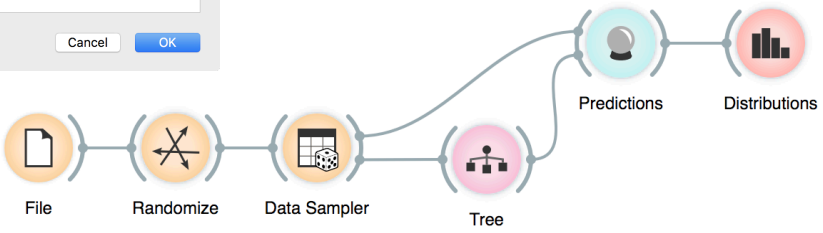
It looks like the tree just memorized every data instance from the data set. No wonder the predictions were right. The tree makes no sense, and it is complex because it simply remembered everything.

This should be a bit of déjà vu. Is not this the same as regression modelling with high degree polynomials?

If a classifier remembers everything from a data set but without discovering any general patterns, it should perform miserably on any new data set, right? Let us check this out. We will split our data set into two sets, training and testing, train the classification tree on the training data set and then estimate its accuracy on the test data set.

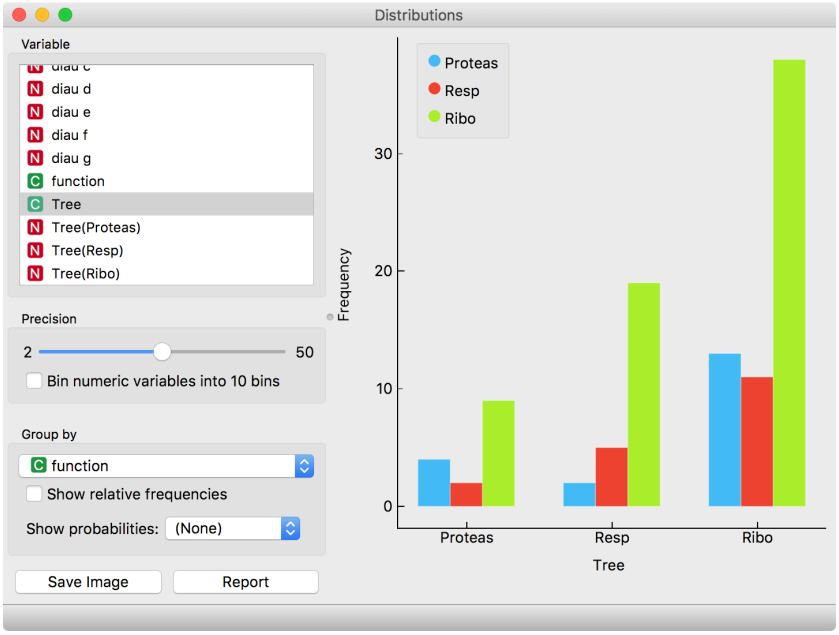
The signals from the Data Sampler widget have not been named in our workflow to save space. The Data Sampler splits the data to a sample and out-of-sample (so called remaining data). The sample was given to the Tree widget, while the remaining data was handed to the Predictions widget. Set the Data Sampler so that the size of these two data sets is about equal.





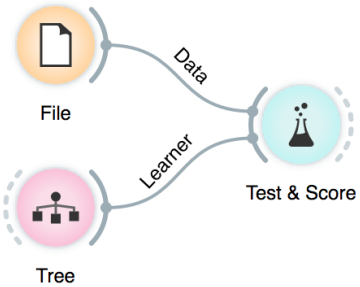
Let's check how the Distributions widget looks after testing the classifier on the test data.

Turns out that for every class value the majority of data instances has been predicted to the ribosomal class (green). Why? Green again (like green from the Scatter Plot of the messed-up data)? Here is a hint: use the Box Plot widget to answer this question.



The first two classes are a complete fail. Predictions for ribosomal genes are a bit better, but still with lots of mistakes. On class-randomized training data, our classifier fails miserably. Finally, this is just as we would expect.

To test the performance (accuracy) of the classification technique, we have just learned that we need to train the classifiers on the training set and then test it on a separate test set. With this test, we can distinguish between those classifiers that just memorize the training data and those that learn a useful model.



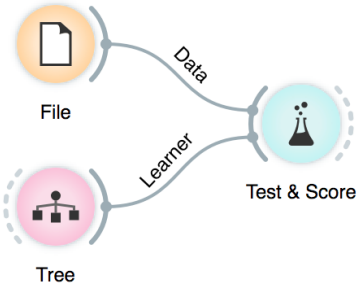
Learning is not only remembering. Rather, it is discovering patterns that govern the data and apply to new data as well. To estimate the accuracy of a classifier, we, therefore, need a separate test set. This assessment should not depend on just one division of the input data set to training and test set (here's a place for cheating as well). Instead, we need to repeat the process of estimation several times, each time on a different train/test set and report on the average score.

Testing classification models is thus the same as testing regression models, just with a different score. All other techniques we have seen before, such as cross-validation, apply here, too.

Lesson 16: Cross-Validation

Estimating the accuracy may depend on a particular split of the data set. To increase robustness, we can repeat the measurement several times, each time choosing a different subset of the data for training. One such method is cross-validation. It is available in Orange through the Test & Score widget.

Note that in each iteration, Test & Score will pick part of the data for training, learn the predictive model on this data using some machine learning method, and then test the accuracy of the resulting model on the remaining, test data set. For this, the widget will need on its input a data set from which it will sample data for training and testing, and a learning method which it will use on the training data set to construct a predictive model. In Orange, the learning method is simply called a learner. Hence, Test & Score needs a learner on its input. A typical workflow with this widget is as follows.

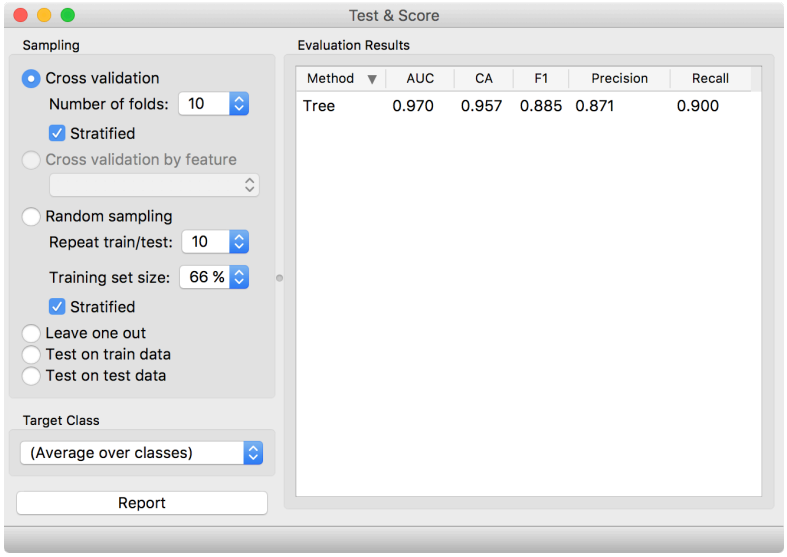


For geeks: a learner is an object that, given the data, outputs a classifier. Just what Test & Score needs.

This is another way to use the Tree widget. In the workflows from the previous lessons we have used another of its outputs, called Model: its construction required the data. This time, no data is needed for Tree, because all that we need from it a learner.

Here we show Test & Score widget looks like. CA stands for classification accuracy, and this is what we really care for for now. We will talk about other measures, like AUC, later.

Cross validation splits the data sets into, say, 10 different non-overlapping subsets we call folds. In each iteration, one fold will be used for testing, while the data from all other folds will be used for training. In this way, each data instance will be used for testing exactly once.

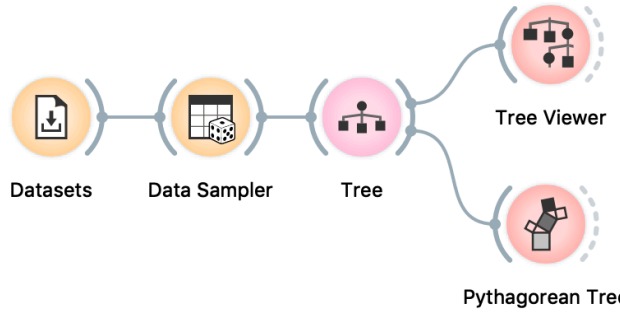


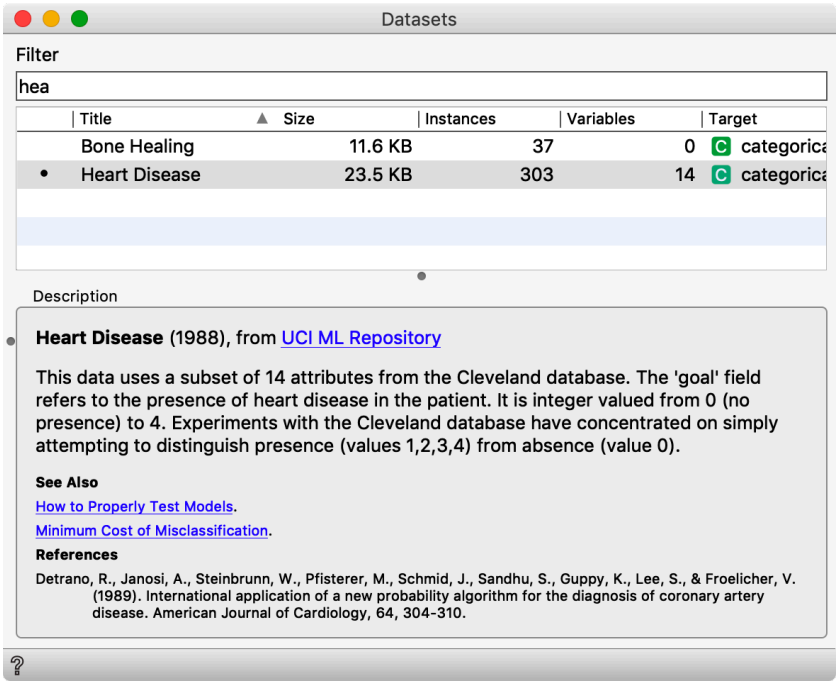
Lesson 17: Trees are not Stable

Classification trees may be sensitive to any changes in the input data. With a minor change in the data, the structure of the classification tree can drastically change. Data in biomedicine may include measurement or observation noise. Beautiful property of the trees is that they are interpretable; that is, we can "read" the model and explain it. But with the instability of the trees, if this is indeed the case, the interpretability does not make much sense.

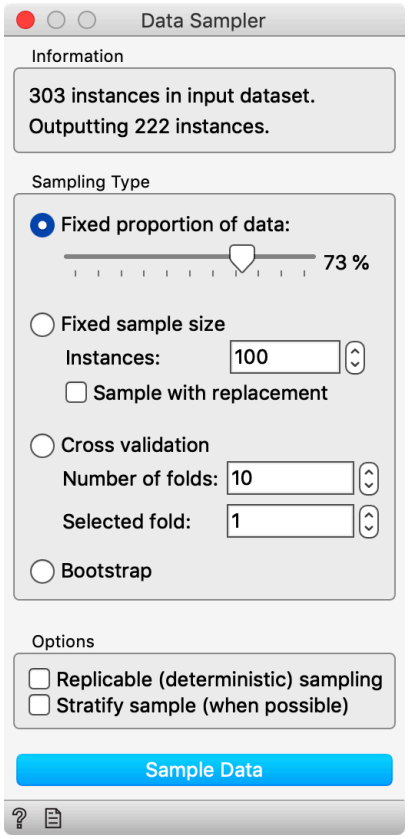
Let us observe the stability of the classification trees on an example. We will consider the data on the hearth disease with 14 attributes and a binary class variable that reports on the presence of the disease.

The trees inferred from heart disease dataset are large and do not fit well in Tree Viewer's visualization. Orange has a widget Pythagorean Tree with an alternative, more compact display.

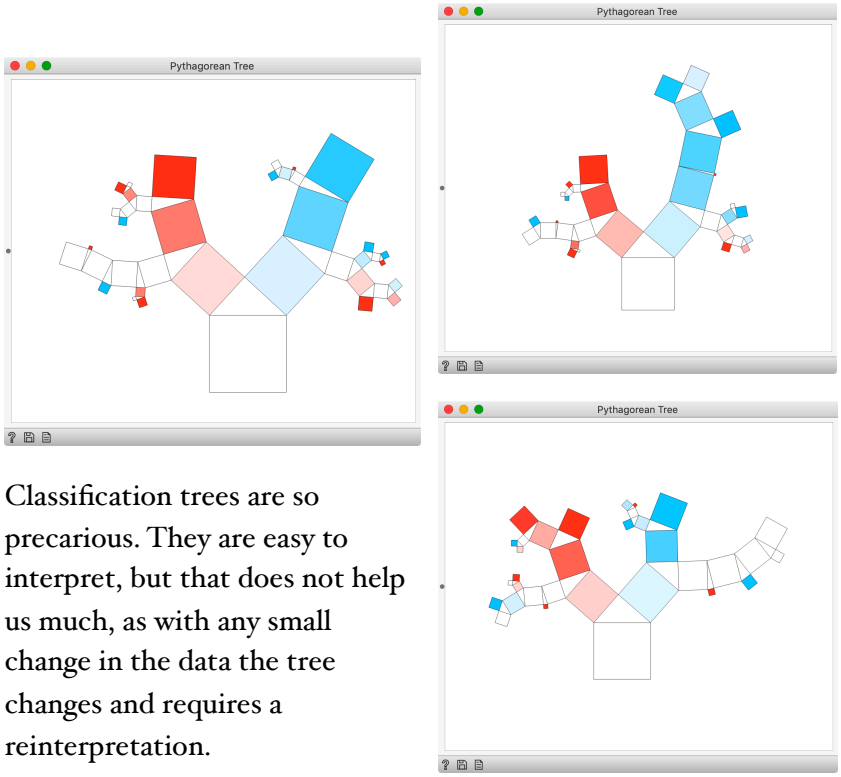




In the workflow, we sample 70% of the data, develop a classification tree, and visualize it. Keeping both Data Sampler and

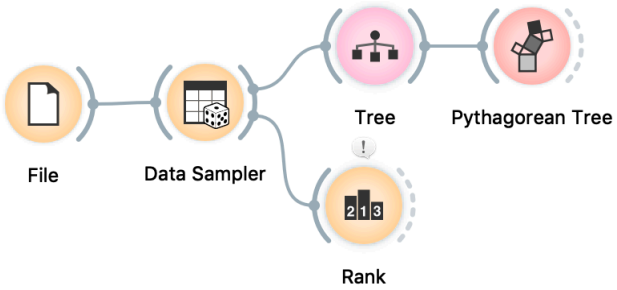


Pythagorean Tree widgets on the top, we can now create a new sample and see the changes in the tree structure. They are substantial, and it looks like that with every new sample, we obtain an entirely different tree.



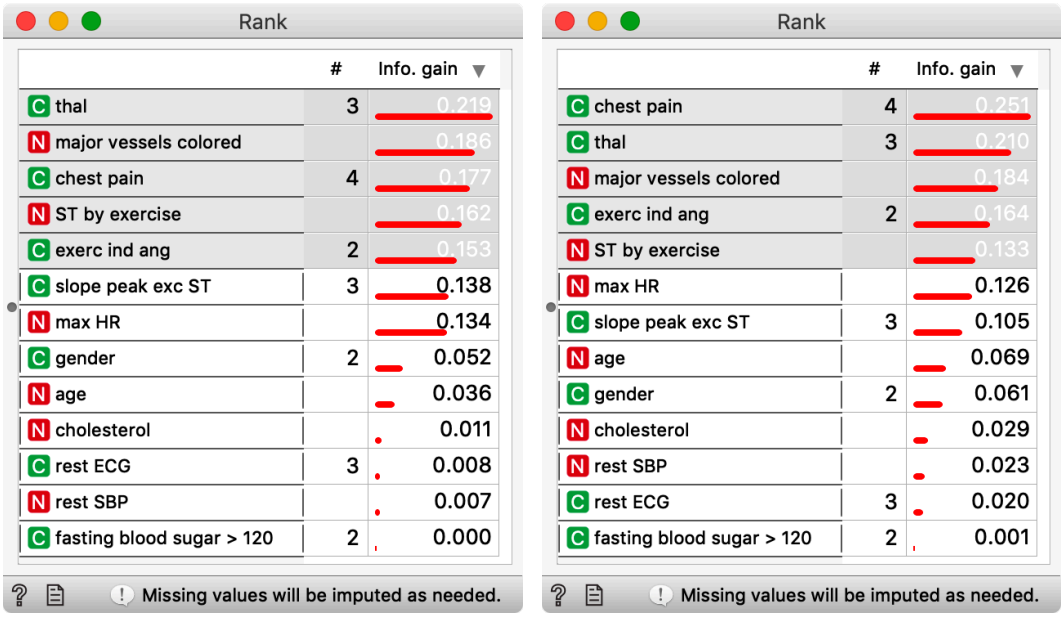
Classification trees are so precarious. They are easy to interpret, but that does not help us much, as with any small change in the data the tree changes and requires a reinterpretation.

Instability of the trees emerges as, when developing the tree, at each internal node the algorithm chooses the most informative feature for the split of the node's training data. If there are several features with comparable correlation with the class, tiny changes in the data can change the ranking of the features. Changes in the feature order on the top reflects the change in the choice of the feature that will be selected to split the data at the root.



To explore how the changes in the data impact the ranking of the features in the tree's top node, that is, for the entire data set, we can use the Rank widget. Even for the whole dataset, when the set dataset is still reasonably large, different features win for the different data sample. When the feature selected for the root of the tree changes, the structure of the entire tree would change.

The instabilities of feature ranks are even more pronounced in the lower layers of the tree, where the datasets pertinent to each node becomes smaller.

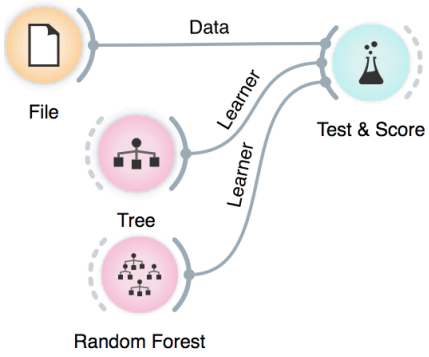


Feature ranking on entire sample of heart disease data set shows that the order of the features depends on the sample. Features most correlated with the class always appear on the top of the list, but their order changes. The tree induction algorithm would always pick the top-ranked feature for the split and the sole change in the root of the tree would result in entirely different trees.

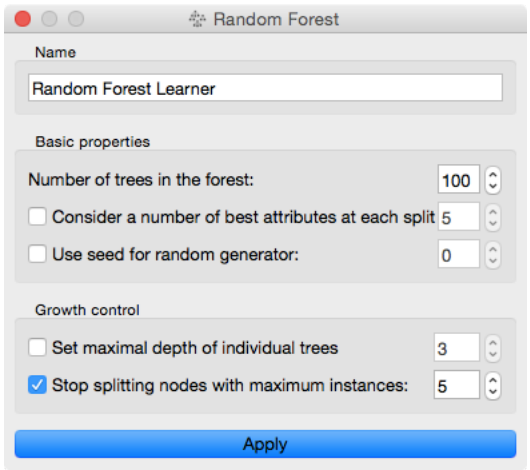
Lesson 19: A Few More Classifiers

We have so far played just with classification trees. Surely this is not the only classification model there is, right?

There are many other, much more accurate classifiers. A particularly interesting one is Random Forest, which averages across predictions of hundreds of classification trees. It uses two tricks to construct different classification trees. First, it infers each tree from a sample of the training data set (with replacement). Second, instead of choosing the most informative feature for each split, it randomly selects from a subset of most informative features. In this way, it randomizes the tree inference process. Think of each tree shedding light on the data from a different perspective. Just like in the wisdom of the crowd, an ensemble of trees (called a forest) usually performs better than a single tree.

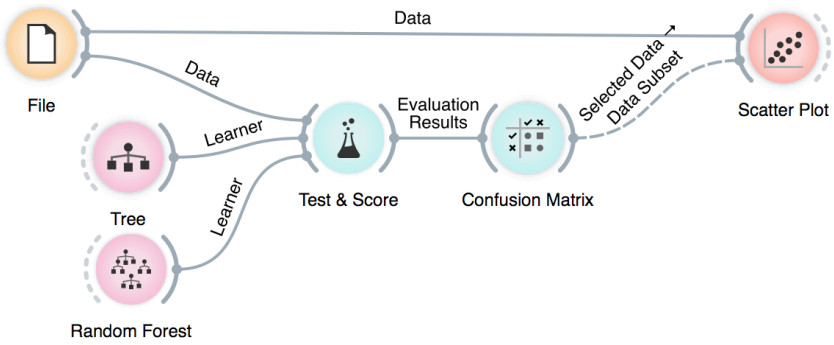


Let us see if this is really so. We give two learners to the Test Learners widget and check if cross-validated classification accuracy is indeed higher for random forest. Choose different classification data sets for this comparison, starting with those we already know (hearth disease, iris, brown selected).

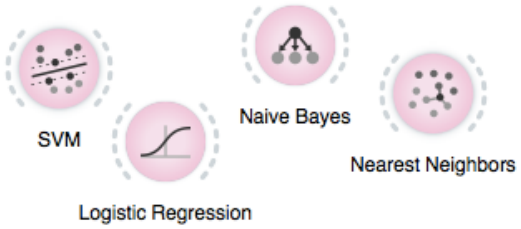


It may be interesting to compare where different classification methods make mistakes. We can use Confusion Matrix for this purpose, and then pass the signal from this widget to the Scatter Plot.

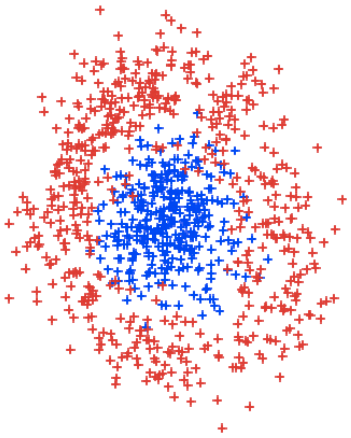
What kind of object is sent from the Test & Score widget to the Confusion Matrix widget? So far, we have used widgets that send data, or even learners. But what could the Test & Score widget communicate to other widgets?



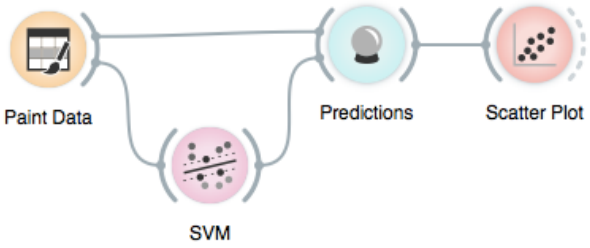
There are other classifiers we can try. We will briefly mention a few more, we won't dive to deep into how they work (we could spend a semester on this!).

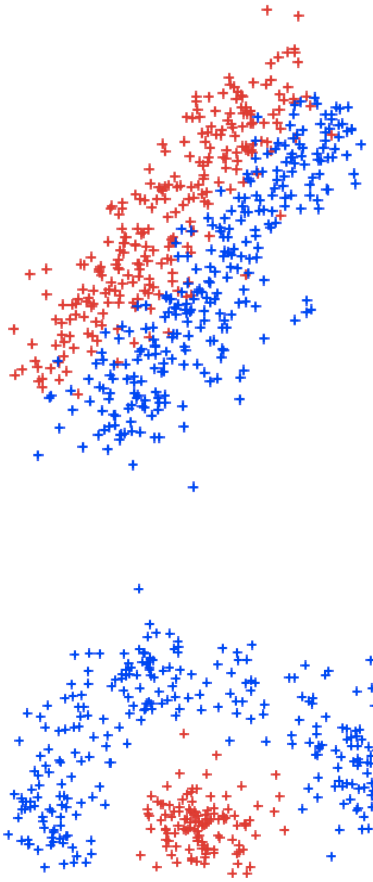


It would be nice if we could, at least on the intuitive level, understand the differences between all these methods and their variants (every method has some parameters). Remember, the classification tree finds hyperplanes orthogonal to the axis; those hyperplanes split the data space to regions with different class probabilities. The tree's decision boundaries are flat. Nearest neighbors classifies the data instance according to the few neighboring data instances in the training set. Decision boundaries with this approach could be very complex. Logistic regression infers just one hyperplane (decision boundary) in an arbitrary direction. This is similar to support vector machines with linear kernel, but then again, the kernels with SVM can be changed, resulting in more complex decision boundaries.

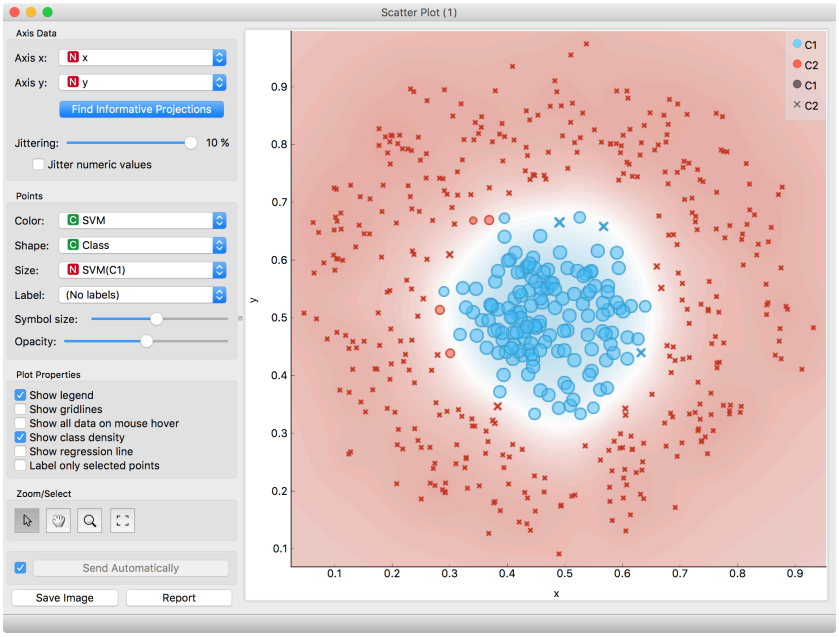


Ok, we have to admit: the above paragraph reads almost like gibberish. We would need a workflow where we could actually see the decision boundaries. And perhaps invent the data sets to test the classifiers. Best in 2D. Maybe, for a start, we could just paint the data. Time to stop writing this long passage of text, end the suspense, and construct a workflow that does this all.





Be creative when painting the data! Also, instead of SVM, use different classifiers. Also, try changing the parameters of the classifiers. Like, limit the depth of the decision tree to 2, or 3, 4. Or switch from SVM with linear kernel to the radial basis function. Appropriately set up the scatter plot to observe the changes.

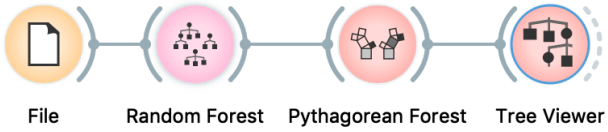


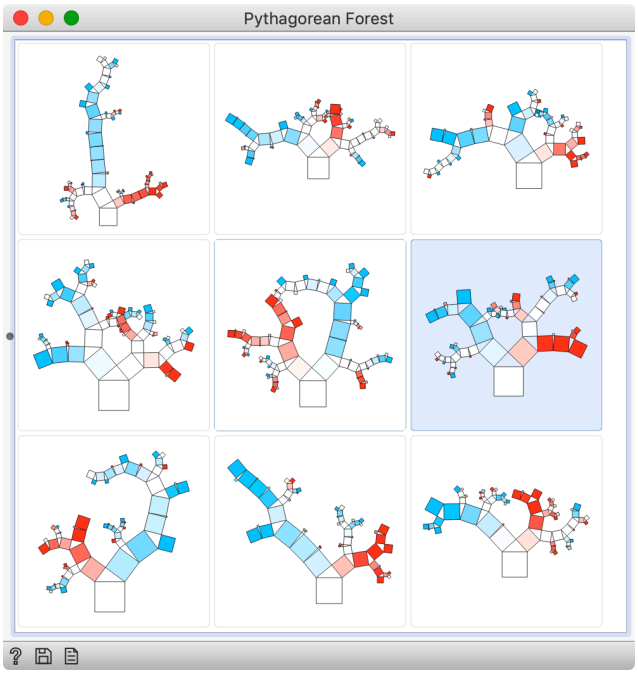
Lesson 18: Random Forests

With instability of trees, the interpretive advantage of the tree is gone. But constructing different, or better, entirely different trees from the same data set could have an advantage. Remember the TV show Who Wants to Be a Millionaire? In the show, the contestant has to answer consecutive multiple-choice questions of increasing difficulty. There is a lifeline called Ask the Audience, where the contestant asks the audience to answer the question and obtains the help by seeing the distribution of the answers. Most often, the answer that received the most votes from the audience is the right one. Such "wisdom of the crowd" approach, where the collective opinion of a group of individuals rather than that of a single expert, is today employed by social information sites. And is used in machine learning. Each of the tree, inferred from the sample of the training data, could be considered an individual. A collection of the tree would vote for the class, and a machine learning technique called random forests would then predict the class value that received the majority of votes.

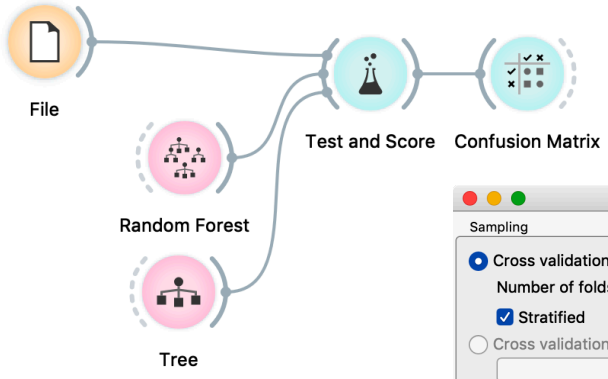
Random forest develop trees from so-called bootstrap sample, a sample of the training data that is of the same size but draws randomly from the training set with replication. To increase the diversity of the trees, the features on which the tree nodes split the data are randomly drawn from the top-ranked features.

Random forest consist of a set of trees that can be in visualized in Orange using Pythagorean Forest widget. To observe the details of the tree, a selected tree from the forest can be sent to the Tree Viewer or to Pythagorean Tree for further inspection. Visualizing the trees in the forest serves only to assure us that the inferred trees are indeed different. Else, random forest are treated as black boxes.

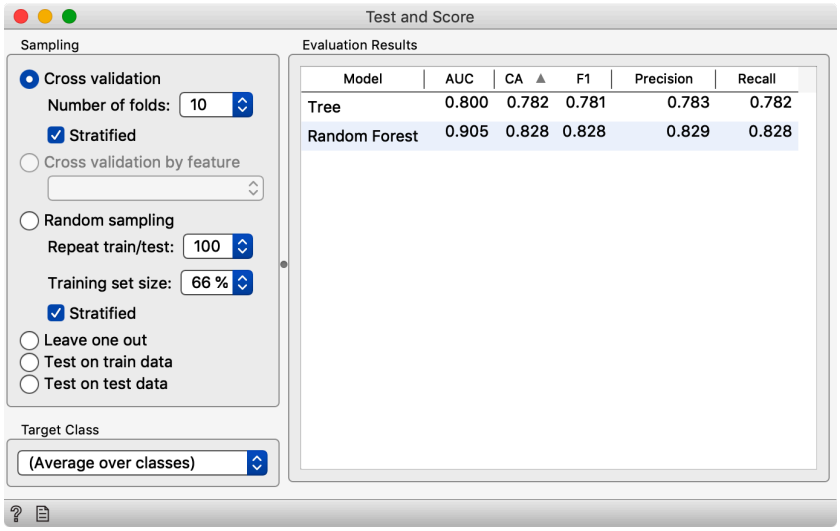




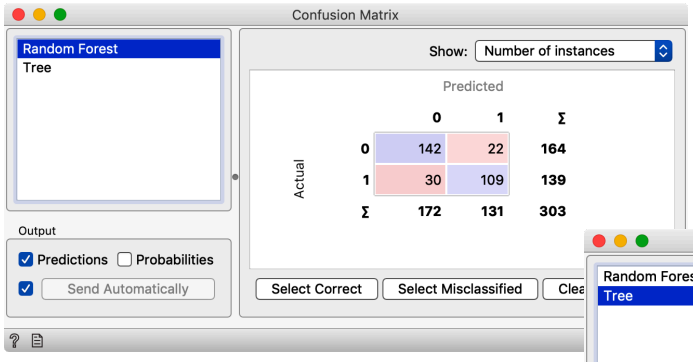
Random forests are impossible to interpret. Their only gain is in accuracy. And this could be substantial. Let us measure this through cross-validation on a heart disease data.

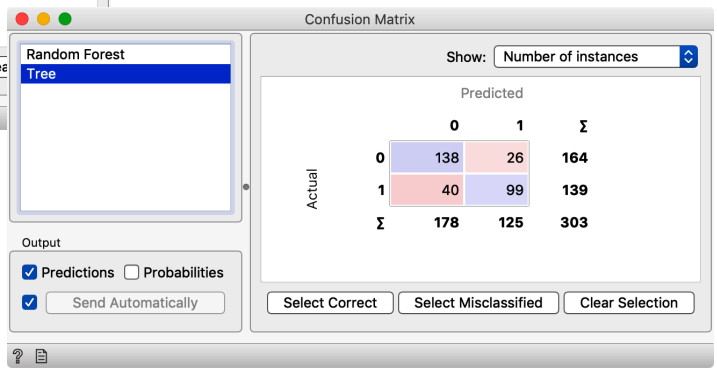


A core parameter of the random forest machine learning method is the number of trees. We have set this parameter to 100 in our experiment. Most often, a few hundred trees in the forest are sufficient for optimized accuracy, and increasing the number of trees above 500 does not yield any further gains.



The gain of the forest over the individual tree is substantial. Not just in classification accuracy (CA), but also in every other statistics that we will present in detail in one of the following lessons.

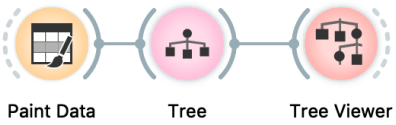


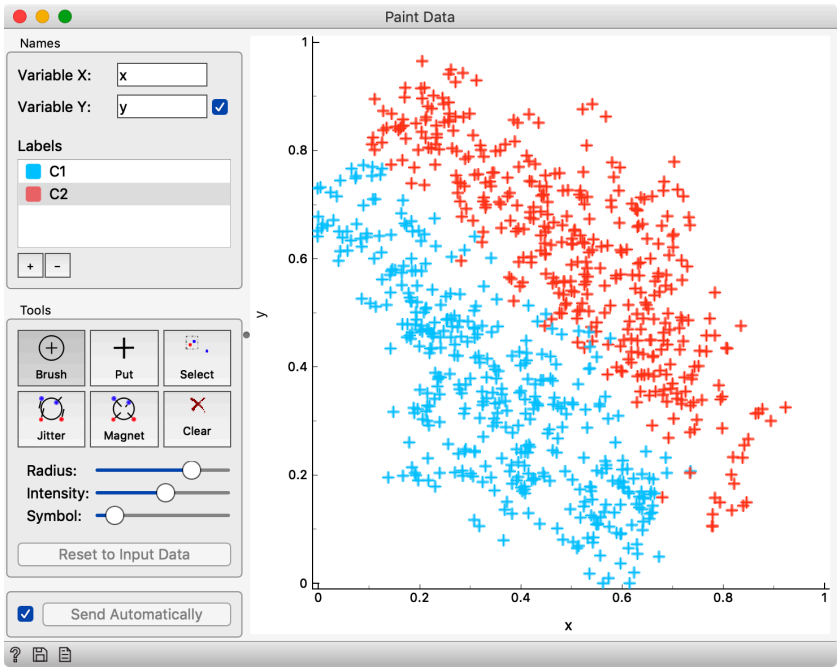


With classifications, it is always wise to check the confusion matrix. Here, we see that out 303 data instances, the trees misclassified 66, and random forests misclassified 52, with a most significant reduction of misclassification taking place for the subjects with disease.

Lesson 19: Logistic Regression

Trees "cut" the feature space according to the value of a single feature, that is, they discover "boxes" in the feature space with a prevailing class value. The method struggles with more complex shapes of decision boundaries. We can paint the data set where the classification tree would struggle. For instance, for the painted data set as depicted below, the inferred tree is quite complicated and contains 22 internal nodes and 11 laves.

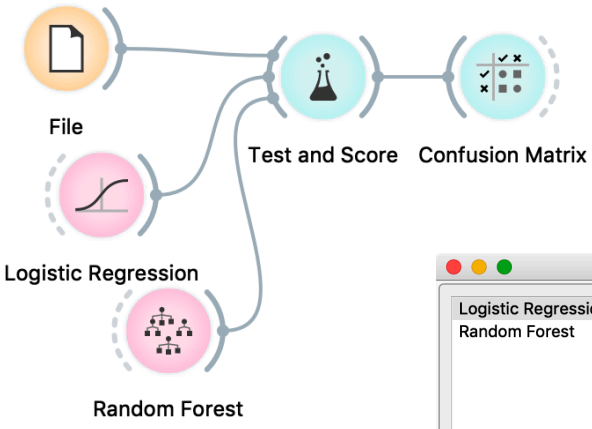


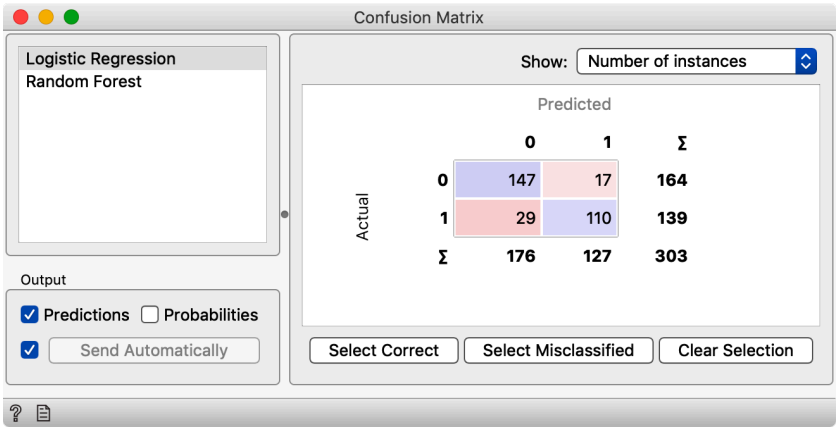


Possibly the best model for the data shown above is a line that splits blue and red points. We could also assign the class probabilities according to this separation line: the further away from the line a data point, the more likely it belongs to one or the other class. The data points on the separation line are those for which we cannot decide on a class, and where the class probabilities are equal, that is. Notice that a line could separate the two classes in two-dimensional feature space, whereas for three or higher dimensional spaces we need a plane or a hyperplane.

The model that follows our reasoning above, and from the data infers the planes that separate the two classes is called logistic regression. The "language" of this model is linear as the planes are flat. Surprisingly, though, this model behaves rather well and is often comparable to the random forest. On the heart disease data set, the logistic regression even slightly beats the forest, as

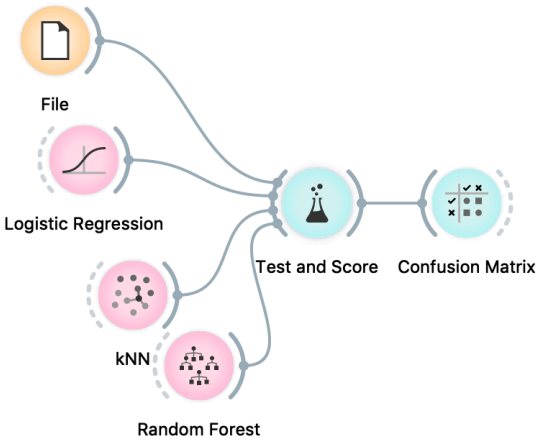
demonstrated in the now-familiar workflow with *Test & Score* widget.





There are, of course, cases where logistic regression fails miserably, and it is not hard to, for instance, paint a data set where the performance of this model would score much lower than random forests. Because of the simplicity and robustness that comes with it, logistic regression is an often-used technique. Logistic regression is also a core building block in neural networks, and we will address this briefly towards the end of this course.

Lesson 20: *k*-Nearest Neighbors

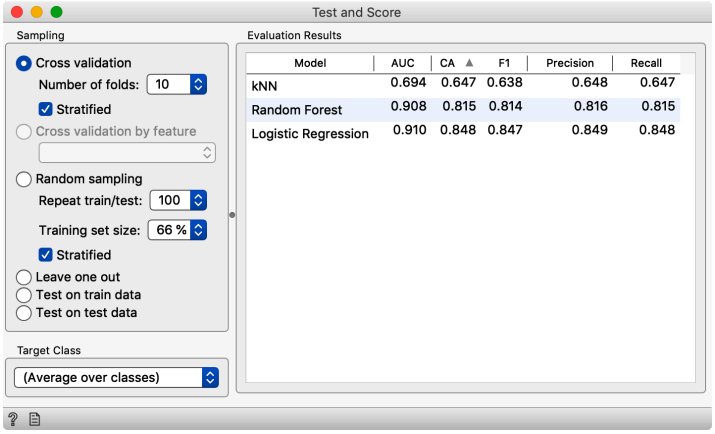


Here is one more technique before we finish with methods for classification. It is perhaps the most intuitive technique of them all, but surprisingly imprecise. We mention it here solely because of the concept, and not as a method you should use when developing a predictive model.

The method is called *k*-nearest neighbors, and it does what the name says. It classifies the new data instance according to its k nearest neighbors from the training set. For example, in the space with two continuous features, the data instance marked with A will be classified as blue, and instance B as red. We have used 3 for the value of *k*, that is, for every new instance, we found five closest data instances in the training set.

The parameter for k-nearest neighbors method is *k*, which is by default set to 10, and the distance function, where the default is Euclidean distance. In general, *k*-NN could use any other approach to distance scoring.

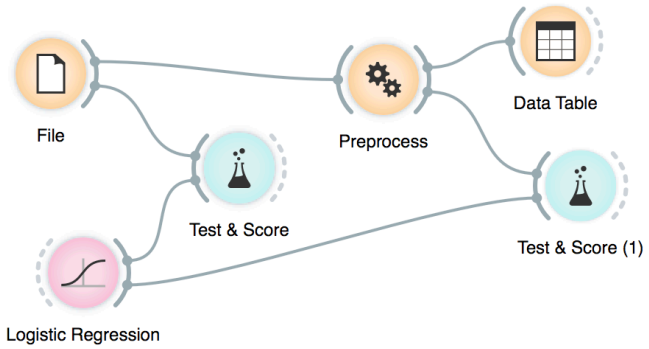
The performance of the nearest neighbor classification on heart disease data set is not stellar. Nearest neighbors perform substantially worse than logistic regression. In cases, where data would reside in two-dimensions, *k*-NN would do something similar to what we would be doing manually. Thus, this approach is at least conceptually interesting.



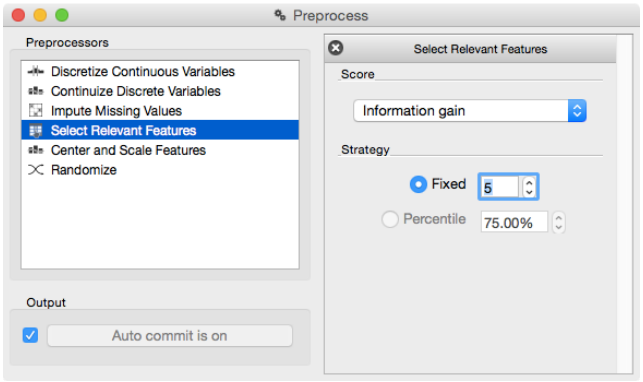
Lesson 21: More Cheating

Gene expression data set we will use was borrowed from Gene Expression Omnibus. There is a special widget in Orange bioinformatics add on that we could use to fetch this and similar data sets. Instead, we will here rely on GEO data set gds360 available at http://file.biolab.si/datasets/gds360.pkl.

Consider a typical gene expression data set where we have samples in rows and genes expressions in columns. These data sets are usually fat: there are many more genes than samples. Fat data sets are almost typical for systems biology. When samples are labeled with phenotype and our task is phenotype classification, many features (genes) will be irrelevant and most often only a few will be highly correlated with class. So why not simply first select a set of most informative features, and then do the whole analysis? At least cross-validation will then work much faster, as the model inference algorithms will deal with much smaller data tables. Cool. What a nice trick! Let's try it out in the following workflow.



The workflow above uses the data preprocessing widget, which we have configured to select 5 most informative features.



Observe the classification accuracy obtained on the original data set, and on the data set with five best selected features. What is happening? Why?

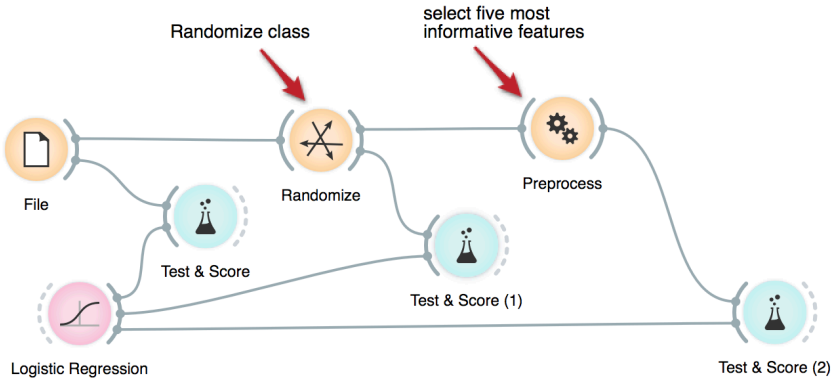
Lesson 22: Cheating Works Even on Randomized Data

We can push the example from our previous lesson to the extreme. We will randomize the classification data. That is, we will take the column with the class values and randomly permute it. We will use the Randomize widget to do this.

Later, we will do classification on this data set. We expect really low classification accuracy on randomized data set. Then, we will select five features that are most associated with the class. Even though we have randomly permuted the classes, there have to be some features that are weakly correlated with the class. Simply because we have tens of thousands of features, and we have only a few samples. There are enough features that some of them correlate with class simply by chance. Finally, we will score a random forest on a randomized data set with selected features.

Compare the scores reported by cross-validation on different data sets in this pipeline. Why is the accuracy in the final one rather high? Would adding more "most informative features" improve or degrade the cross-validated performance on a randomized data set?

Instead of selecting five most informative features, you can reduce this number even further. Say, to two most informative features. What happens? Why does accuracy raise after this change?



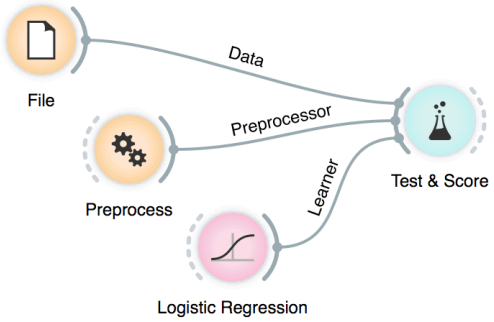
Lesson 23: How to Correctly Perform Test and Score

The writing on the right looks straightforward. But actually one needs to be extremely careful not to succumb to overfitting when reporting results of cross-validation tests. The literature on systems biology is polluted with reporting on overly optimistic results, and high impact factors provide no guarantee that studies were carried out correctly (in fact, due to a lack of reviewers from the field of machine learning, mistakes likely stay overlooked).

Simon et al. (2003) provides a great read on this topic. He found that many of the early papers in gene expression analysis reported high accuracy simply due to overfitting.

To put it simply: never, in any way, transform the data prior to cross-validation. Any transformation should happen within cross-validation loop, first on the training set, and then, if required, on a test set. In a relaxed form: it's ok to transform the data, but the transformation should be done independently on the train and the test set and the transformation on the test set should in no way use the information about the class value. Data imputation could be an example of such operation, but again it should be carried out separately for the train and test set and should not consider classes.

But how do we then correctly apply preprocessing in Orange? The idea of reducing the number of features prior to inferring a predictive model may be still appealing, now that we know we can use it on training data sets (leaving the test set alone). Following are two workflows that do this correctly.

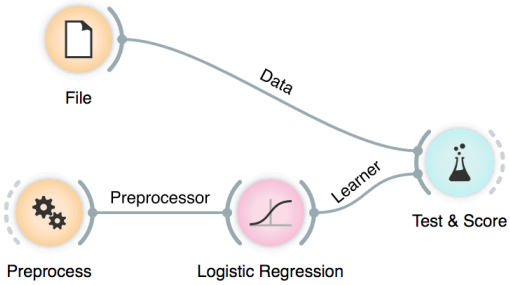


In this first workflow, we gave the Test & Score widget a preprocessor (feature selection was used in this example). The Test & Score widget uses it correctly only on the training sets. This type of workflow is preferred if we would like to test the effect of preprocessing on a number of different learning algorithms.

The Preprocess widget does not necessary require a data set on its input. An alternative use of this widget is to output a method for data preprocessing, which we can then pass to either a learning method or to a widget for cross validation.

This is not the first time we have used a widget that instead of a data passes forward a computation method. All the learners, like Random Forest, do so. A learner could get data on its input and pass a classifier to its output, or simple pass an instance of itself, that is, pass a learning algorithm to whichever widget could use it. For instance, to the Test & Score widget.

Alternatively, we can include a preprocessor in a learning method. The preprocessor is now called on the training data set just before this learner performs inference of the predictive model.



Can you extend this workflow to such an extent that you can test both a learner with preprocessing by feature subset selection and the same learner without this preprocessing? How does the number of selected features affect the cross-validated accuracies? Does the success of this particular combination of machine learning technique depend on the input data set? Does it work better for some machine learning algorithms? Try its performance on k-nearest neighbors learner (warning: use small data sets, this classifier could be very slow).

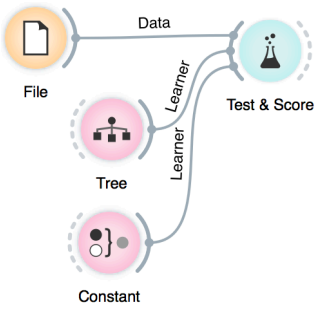
Somehow, in a shy way, we have also introduced a technique for feature selection, and pointed to its possible utility for classification problems. Feature subset selection, or FSS in short, was and still is, to some extent, an important topic in machine learning. Modern classification algorithms, though, perform it implicitly, and can deal with a large number of features without the help of external procedures for their advanced selection. Random forest is one such technique.

Lesson 20: Model Scoring

In multiple choice exams, you are graded according to the number of correct answers. The same goes for classifiers: the more correct predictions they make, the better they are. Nothing could make more sense. Right?

Maybe not. Dr. Smith is a specialist of a type and his diagnosis is correct in 98% of the cases. Would you consider visiting him if you have some symptoms related to his speciality?

Not necessarily. His specialty, in fact, are rare diseases (2 out of 100 of his patients have it) and, being lazy, he always dismisses everybody as healthy. His predictions are worthless — although extremely accurate. Classification accuracy is not an absolute measure, which can be judged out of context. At the very least, it has to be compared with the frequency of the majority class, which is, in case of rare diseases, quite ... major.



For instance, on GEO data set GDS 4182, the classification tree achieves 78% accuracy on cross validation, which may be reasonably good. Let us compare this with the Constant model, which implements Dr. Smith's strategy by always predicting the majority. It gets 83%. Classification trees are not so good after all, are they?

On the other hand, their accuracy on GDS 3713 is 57%, which seems rather good in comparison with the 50% achieved by predicting the majority.

What do other columns represent? Keep reading!

Method	AUC	CA	F1	Precision	Recall
Classification Tree	0.573	0.570	0.585	0.571	0.600
Majority	0.500	0.506	0.672	0.506	1.000

The problem with classification accuracy goes deeper, though.

Classifiers usually make predictions based on probabilities they compute. If a data instance belongs to class A with a probability of 80% and to B with a probability of 20%, it is classified as A. This makes sense, right?

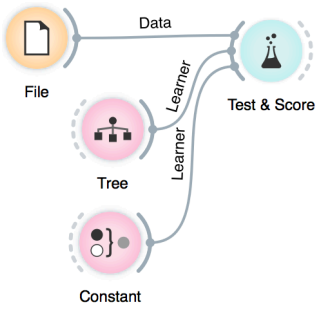
Maybe not, again. Say you fall down the stairs and your leg hurts. You open Orange, enter some data into your favorite model and

Lesson 24: Model Scoring

In multiple choice exams, you are graded according to the number of correct answers. The same goes for classifiers: the more correct predictions they make, the better they are. Nothing could make more sense. Right?

Maybe not. Dr. Smith is a specialist of a type and his diagnosis is correct in 98% of the cases. Would you consider visiting him if you have some symptoms related to his speciality?

Not necessarily. His specialty, in fact, are rare diseases (2 out of 100 of his patients have it) and, being lazy, he always dismisses everybody as healthy. His predictions are worthless — although extremely accurate. Classification accuracy is not an absolute measure, which can be judged out of context. At the very least, it has to be compared with the frequency of the majority class, which is, in case of rare diseases, quite ... major.

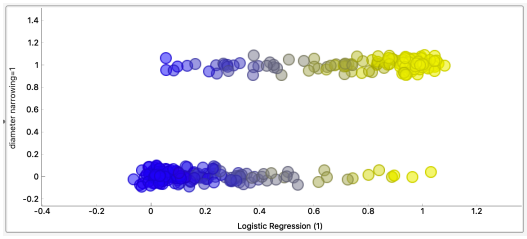


For instance, on GEO data set GDS 4182, the classification tree achieves 79% accuracy on cross validation, which may be reasonably good. Let us compare this with the Constant model, which implements Dr. Smith's strategy by always predicting the majority. It gets 83%. Classification trees are not so good after all, are they?

On the other hand, their accuracy on GDS 3713 is 72%, which seems rather good in comparison with the 50% achieved by predicting the majority.

What do other columns represent? Keep reading!

Model	AUC	CA	F1	Precision	Recall
Tree	0.703	0.722	0.720	0.726	0.722
Constant	0.488	0.506	0.340	0.256	0.506



Classes versus probabilities estimated by logistic regression. Can you replicate this image?

The problem with classification accuracy goes deeper, though.

Classifiers usually make predictions based on probabilities they compute. If a data instance belongs to class A with a probability of 80% and to B with a probability of 20%, it is classified as A. This makes sense, right?

Maybe not, again. Say you fall down the stairs and your leg hurts. You open Orange, enter some data into your favorite model

compute a 20% of having your leg broken. So you assume your leg is not broken and you take an aspirin. Or perhaps not?

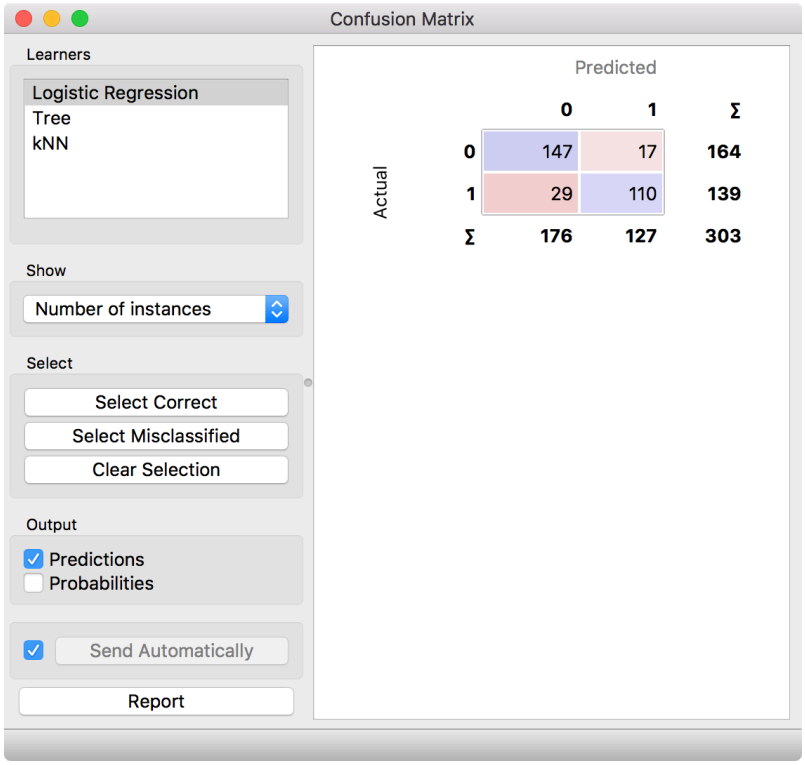
What if the chance of a broken leg was just 10%? 5%? 0.1%?

Say we decide that any leg with a 1% chance of being broken will be classified as broken. What will this do to our classification threshold? It is going to decrease badly — but we apparently do not care. What do we do care about then? What kind of "accuracy" is important?

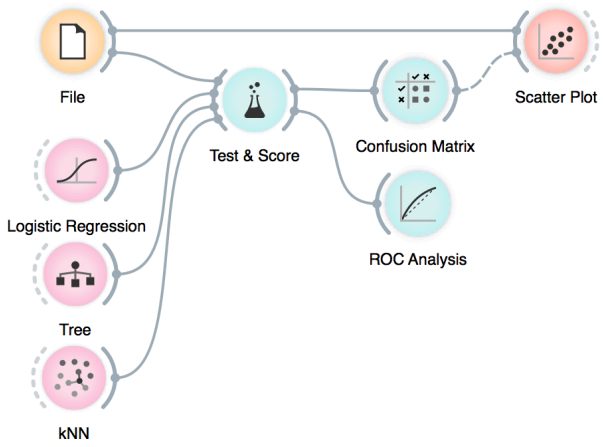
Not all mistakes are equal. We can summarize them in the Confusion Matrix. Here is one for logistic regression on the heart disease data.

These numbers in the Confusion Matrix have names. An instance can be classified as positive or negative; imagine this as being positive or negative when being tested for some medical condition. This classification can be true or false. So there are four options, *true positive* (TP), *false positive* (FP), *true negative* (TN) and *false negative* (FN).

Identify them in the table!



Use the output from Confusion Matrix as a subset for Scatter plot to explore the data instances that were misclassified in a certain way.



Logistic regression correctly classifies 147 healthy persons and 110 of the sick, the numbers on the diagonal. Classification accuracy is then 257 out of 303, which is 85%.

17 healthy people were unnecessarily scared. The opposite error is worse: the heart problems of 29 persons went undetected. We need to distinguish between these two kinds of mistakes.

We are interested in the probability that a person who has some problem will be correctly diagnosed. There were 139 such cases, and 110 were discovered. The proportion is 110 / 139 = 0.79. This measure is called *sensitivity* or *recall* or *true positive rate (TPR)*.

If you were interested only in sensitivity, though, here's Dr. Smith's associate partner — wanting to be on the safe side, she considers everybody ill, so she has a perfect sensitivity of 1.0.

To counterbalance the sensitivity, we compute the opposite: what is the proportion of correctly classified *negative* instances? 147 out of 164, that is, 90%. This is called *specificity* or *true negative rate*.

If you are interested in a complete list, see the Wikipedia page on Receiver operating characteristic, https://en.wikipedia.org/wiki/Receiver_operating_characteristic

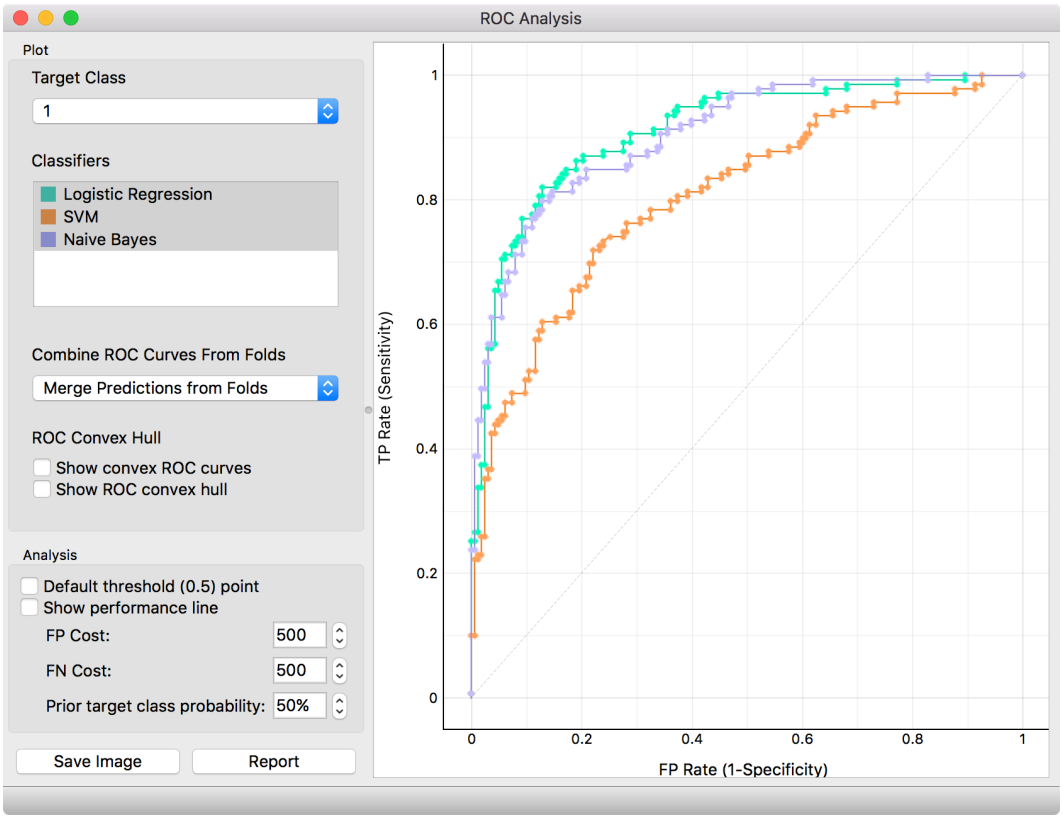
So, if you're classified as OK, you have a 90% chance of actually being OK? No, it's the other way around: 90% is the chance of being classified as OK, if you are OK. (Think about it, it's not as complicated as it sounds). If you're interested in your chance of being OK if the classifier tells you so, you look for the *negative predictive value*. Then there's also *precision*, the probability of being positive if you're classified as such. And the *fall-out* and *negative likelihood ratio* and ... a whole list of other indistinguishable fancy names, each useful for some purpose.

Lesson 16: Choosing the Decision Threshold

The common property of scores from the previous lesson is that they depend on the threshold we choose for classifying an instance as positive. By adjusting it, we can balance between them and find, say, the threshold that gives us the required sensitivity at an acceptable specificity. We can even assign costs (monetary or not) to different kinds of mistakes and find the threshold with the minimal expected cost.

A useful tool for this is the Receiver-Operating Characteristic curve. Don't mind the meaning of the name, just call it the ROC curve.

Here are the curves for logistic regression, SVM with linear kernels and naive Bayesian classifier on the same ROC plot.



The curves show how the sensitivity (y-axis) and specificity (x-axis, but from right to left) change with different thresholds.

Sounds complicated? If it helps: perhaps you remember the term *parametric curve* from some of your math classes. ROC is a parametric curve where x and y (the sensitivity and 1 - specificity) are a function of the same parameter, the decision threshold.

There exists, for instance, a threshold for logistic regression (the green curve) that gives us 0.65 sensitivity at 0.95 specificity (the curve shows 1 - specificity). Or 0.9 sensitivity with a specificity of 0.8. Or a sensitivity of (almost) 1 with a specificity of somewhere around 0.3.

The optimal point would be at top left. The diagonal represents the behavior of a random guessing classifier.

Which of the three classifiers is the best now? It depends on the specificity and sensitivity we want; at some points we prefer logistic regression and at some points the naive bayesian classifier. SVM doesn't cut it, anywhere.

There is a popular score derived from the ROC curve, called Area under curve, AUC. It measures, well, the area under the curve. This curve. If the curve goes straight up and then right, the area is 1; such an optimal AUC is not reached in practice. If the classifier guesses at random, the curve follows the diagonal and AUC is 0.5. Anything below that is equivalent to guessing + bad luck.

AUC has a kind of absolute scale. As a rule of a thumb: 0.6 is bad, 0.7 is bearable, 0.8 is publishable and 0.9 is suspicious.

ROC curves and AUC are fascinating tools. To learn more, read T. Fawcett: ROC Graphs: Notes and Practical Considerations for Researchers

AUC also has a nice probabilistic interpretation. Say that we are given two data instances and we are told that one is positive and the other is negative. We use the classifier to estimate the probabilities of being positive for each instance, and decide that the one with the highest probability is positive. It turns out that the probability that such a decision is correct equals the AUC of this classifier. Hence, AUC measures how well the classifier discriminates between the positive and negative instances.

From another perspective: if we use a classifier to rank data instances, then AUC of 1 signifies a perfect ranking, an AUC of 0.5 a random ranking and an AUC of 0 a perfect reversed ranking.

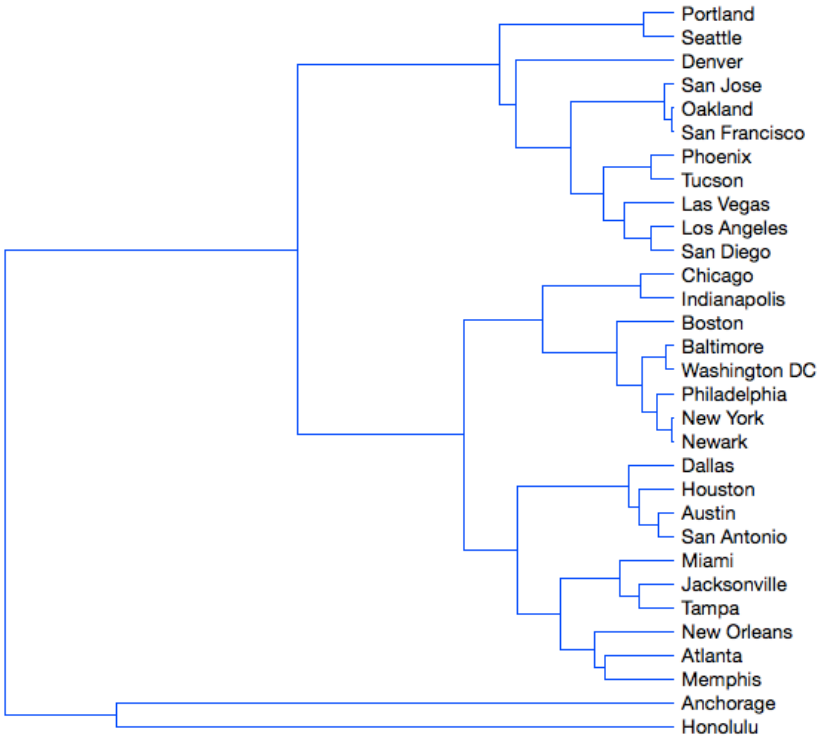
Lesson 18: Mapping the Data

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.

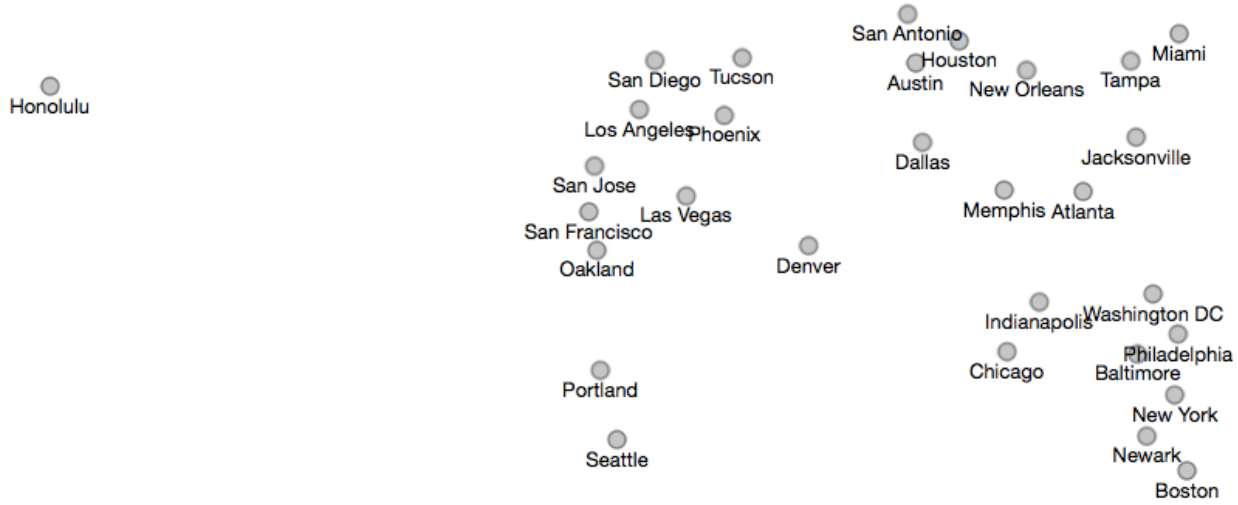
The real problem is New Orleans and San Antonio: New Orleans is close to Atlanta and Memphis, Miami is close to Jacksonville and

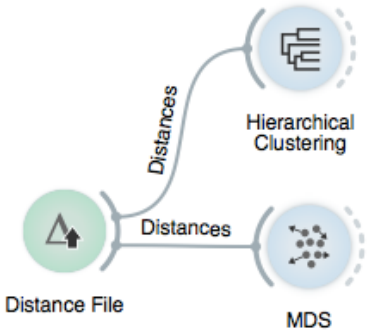
Tampa. And these two clusters are suddenly more similar to each other than to some distant cities in Texas.

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.

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

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.

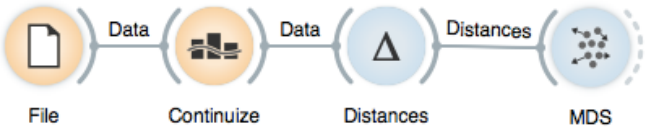


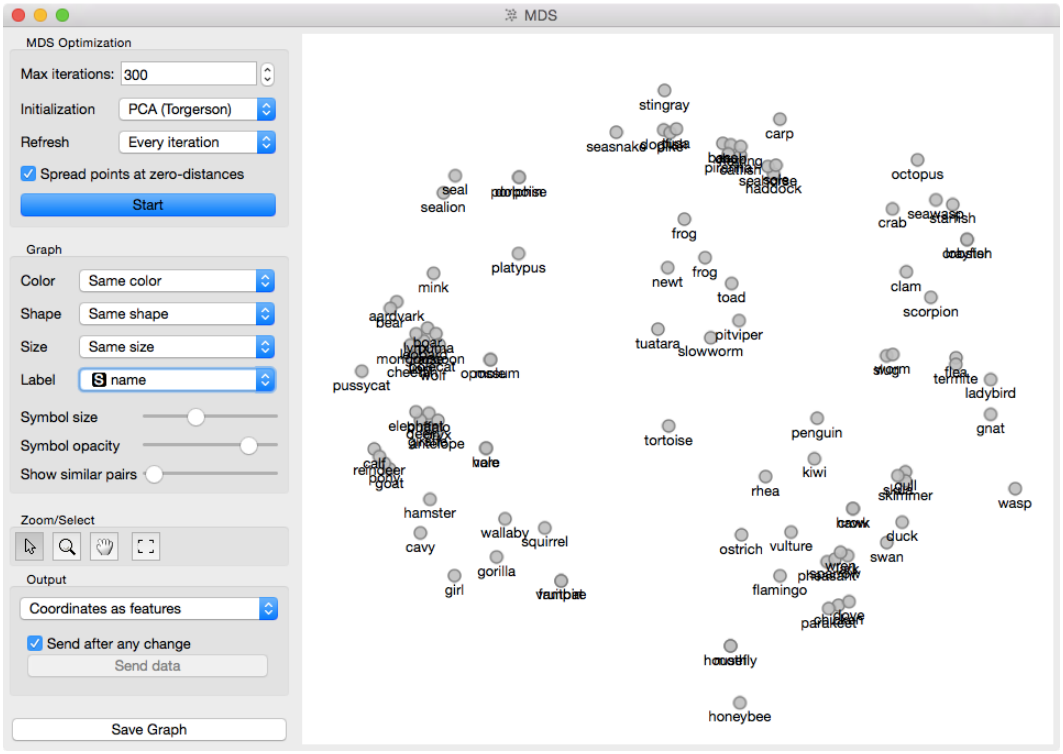


We have not constructed this map manually, of course. We used a widget called MDS, which stands for Multidimensional scaling.

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?





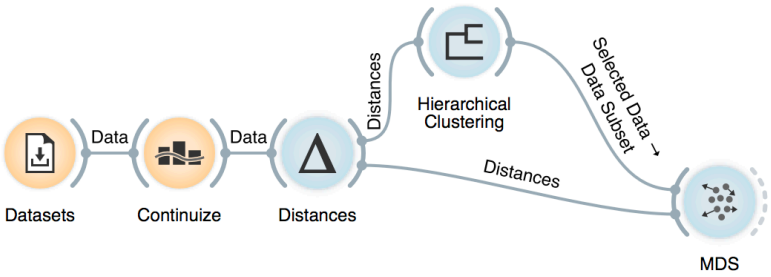
Does the map make any sense? Are similar animals together? Color the points by the types of animals and you should see.

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.

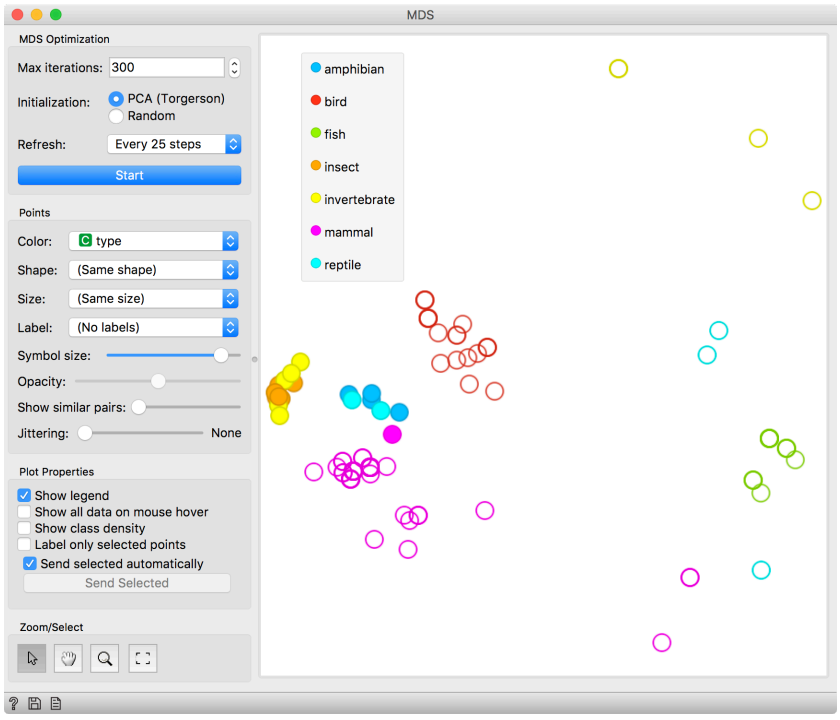
The MDS algorithm does not always find the optimal map. You may want to restart the MDS from random positions. Use the slider "Show similar pairs" to see whether the points that are placed together (or apart) actually belong together. In the above case, the honeybee belongs closer to the wasp, but could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

Lesson 19: 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.

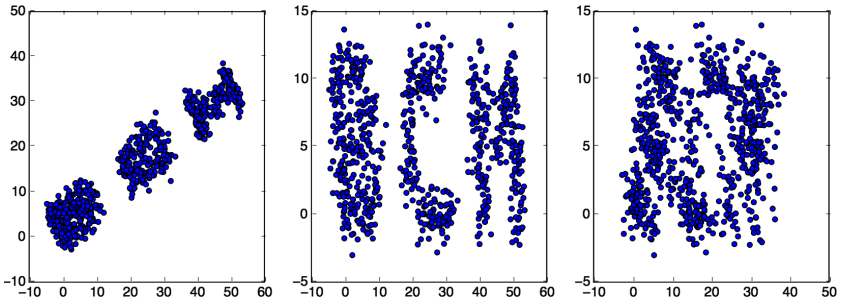


Clustering and two-dimensional embedding make a great combination for data exploration. Clustering finds the coherent groups, and embedding, such as MDS, reveals the relations between the clusters and positions the cluster on the data map. There are other dimensionality reduction and embedding techniques that we could use, but for smaller data sets, MDS is great because it tries to preserve the distances from the original data space.

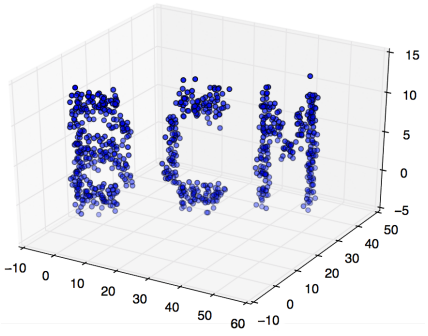
Can you change the workflow to explore the position of individual clusters found by k-means?

Lesson 20: 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.

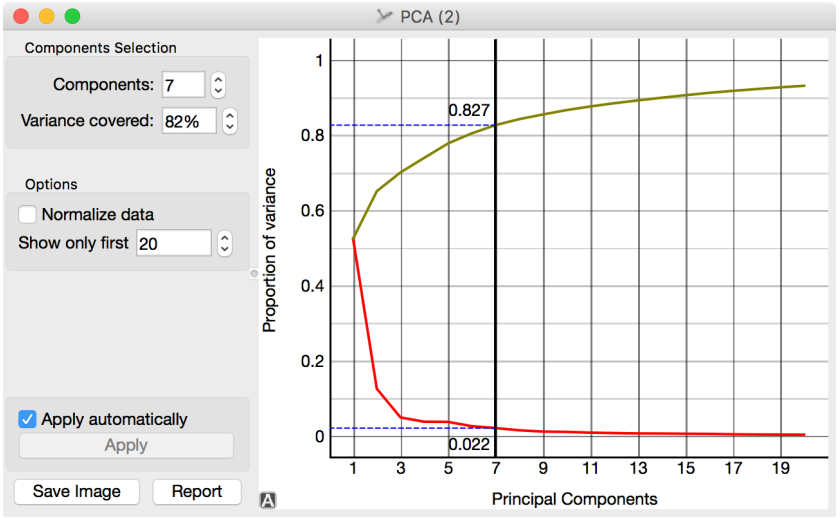
It turns out the flies are also spread in the third direction. Thus you need three numbers after all.

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

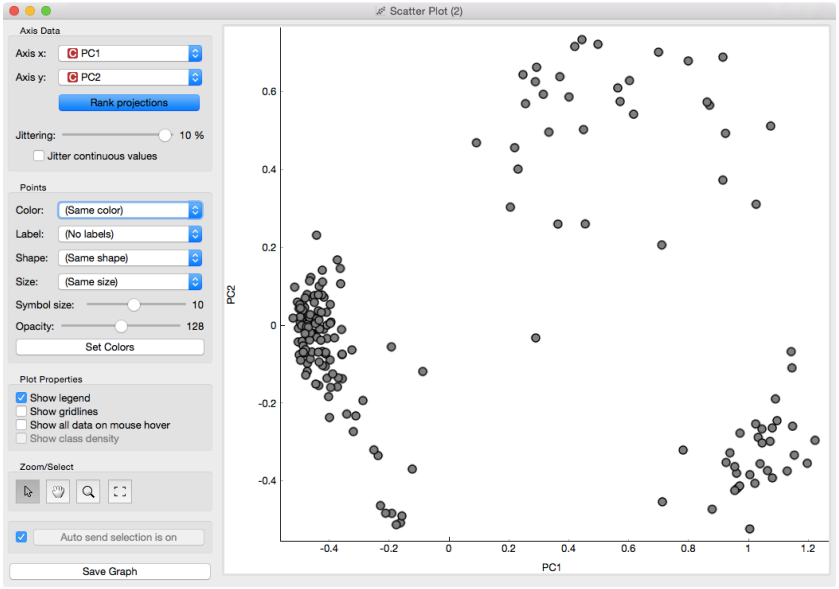
space, you now profile colonies in a 10,000-dimensional space, the coordinates corresponding to gene expressions. Yet if expressions of genes depend only on the concentrations of these two chemicals, you can compute all 10,000 numbers from just two. Your data is then just two-dimensional.

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.



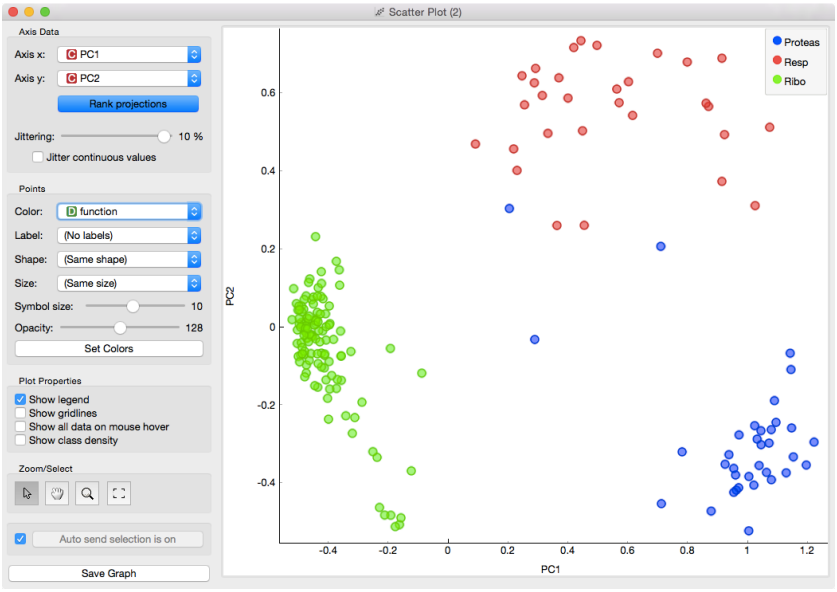
The widget allows you to select the number of components and helps you by showing how much information (technically: explained variance) you retain with respect to the number of components (brownish line) and the amount of information (explained variance) in each component.

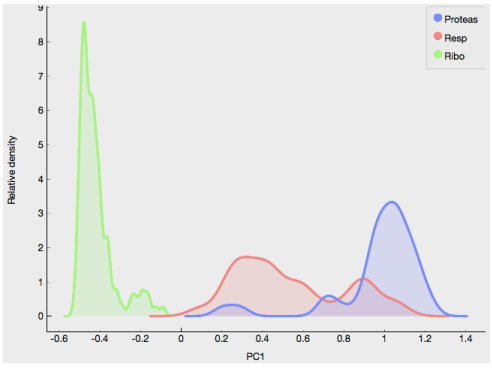
The PCA on the left shows the scree diagram for brown-selected data. Set like this, the widget replaces the 80 features with just seven - and still keeping 82.7% of information. (Note: disable "Normalize data" checkbox to get the same picture.) Let us see a scatter plot for the first two components.



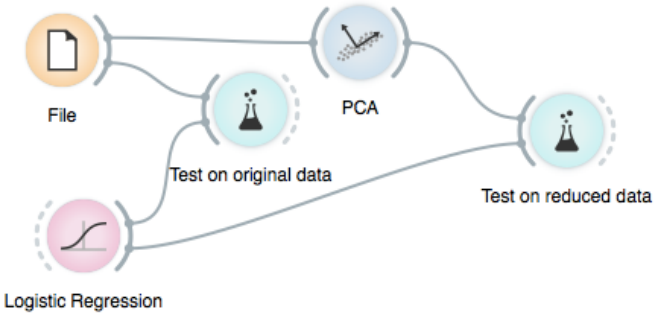
The axes, PC1 and PC2, do not correspond to particular features in the original data, but to their linear combination. What we are looking at is a projection onto the plane, defined by the first two components. When you consider only two components, you can imagine that PCA put a hyperplane into multidimensional 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.

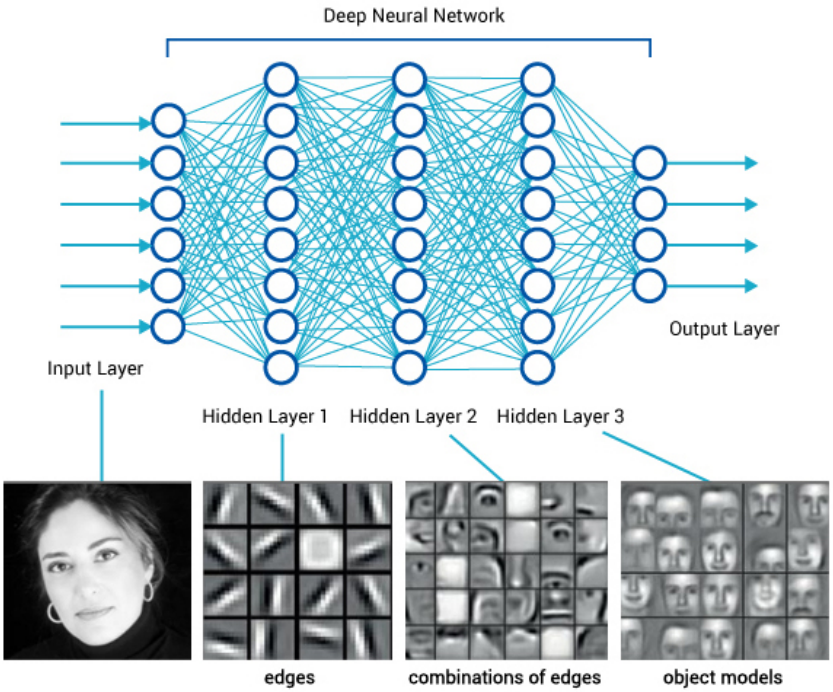
On the original data, Logistic regression gets 98% AUC and classification accuracy. If we select just single component in PCA, we already get a 93%, and if we take two, we get the same result as on the original data.

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 21: 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.

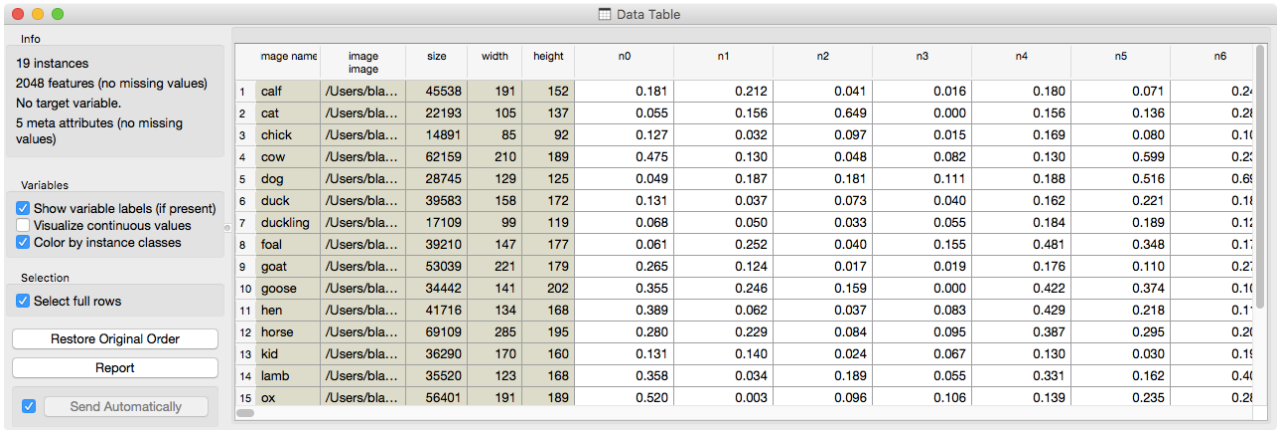
This depiction of deep learning network was borrowed from http://www.amax.com/blog/?



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.

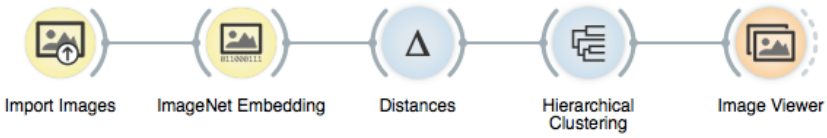
Deep learning requires a lot of data (thousands, possibly millions of data instances) and processing power to prepare the network. We will use one which is already prepared. Even so, embedding takes time, so Orange doesn't do it locally but uses a server invoked through the ImageNet Embedding widget.

Image embedding describes the images with a set of 2048 features appended to the table with meta features of images.



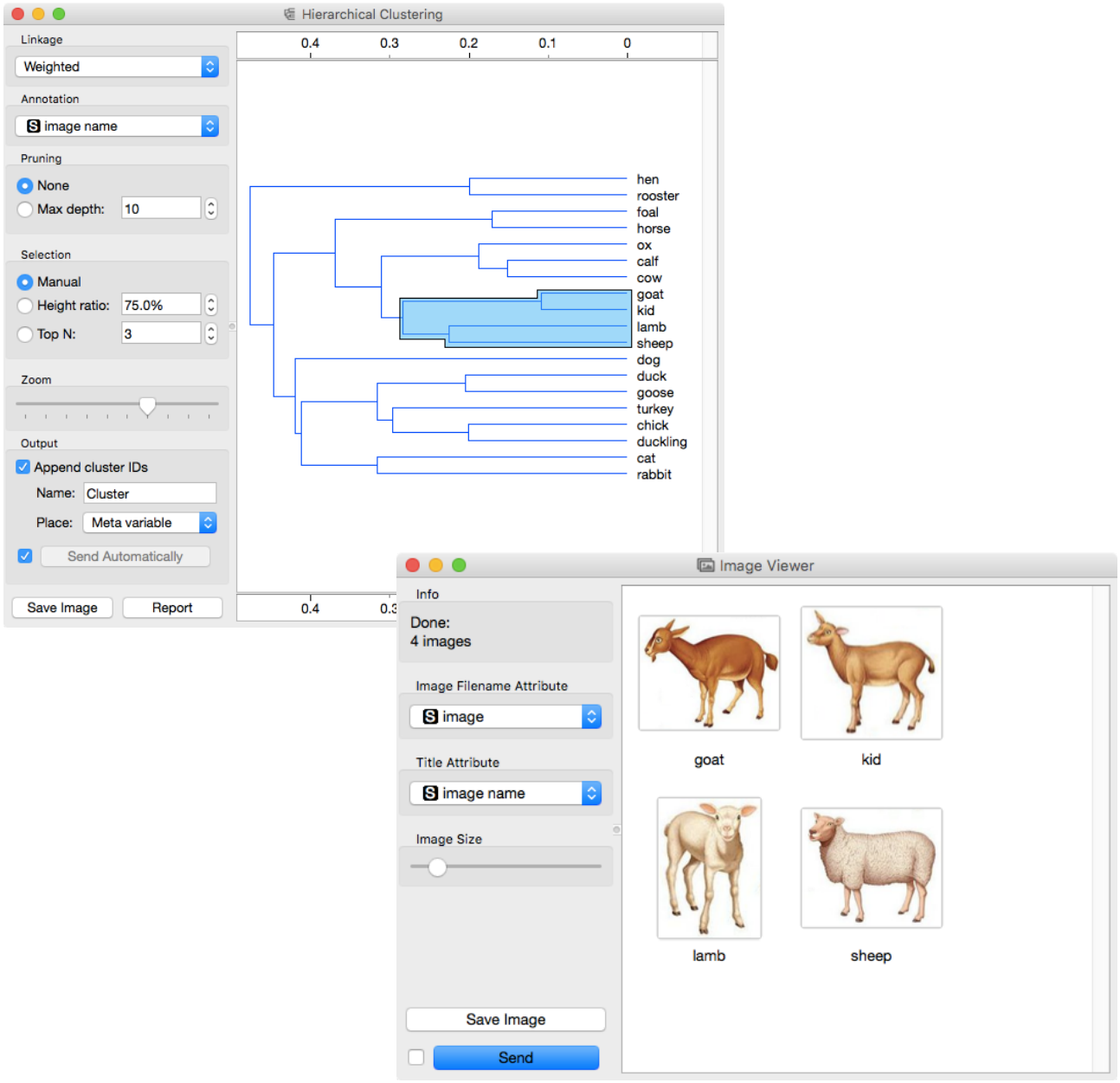
	mage name	image image	size	width	height	n0	n1	n2	n3	n4	n5	n6
1	calf	/Users/bla...	45538	191	152	0.181	0.212	0.041	0.016	0.180	0.071	0.2
2	cat	/Users/bla...	22193	105	137	0.055	0.156	0.649	0.000	0.156	0.136	0.2
3	chick	/Users/bla...	14891	85	92	0.127	0.032	0.097	0.015	0.169	0.080	0.1
4	cow	/Users/bla...	62159	210	189	0.475	0.130	0.048	0.082	0.130	0.599	0.2
5	dog	/Users/bla...	28745	129	125	0.049	0.187	0.181	0.111	0.188	0.516	0.6
6	duck	/Users/bla...	39583	158	172	0.131	0.037	0.073	0.040	0.162	0.221	0.1
7	duckling	/Users/bla...	17109	99	119	0.068	0.050	0.033	0.055	0.184	0.189	0.1
8	foal	/Users/bla...	39210	147	177	0.061	0.252	0.040	0.155	0.481	0.348	0.1
9	goat	/Users/bla...	53039	221	179	0.265	0.124	0.017	0.019	0.176	0.110	0.2
10	goose	/Users/bla...	34442	141	202	0.355	0.246	0.159	0.000	0.422	0.374	0.1
11	hen	/Users/bla...	41716	134	168	0.389	0.062	0.037	0.083	0.429	0.218	0.1
12	horse	/Users/bla...	69109	285	195	0.280	0.229	0.084	0.095	0.387	0.295	0.2
13	kid	/Users/bla...	36290	170	160	0.131	0.140	0.024	0.067	0.130	0.030	0.1
14	lamb	/Users/bla...	35520	123	168	0.358	0.034	0.189	0.055	0.331	0.162	0.4
15	ox	/Users/bla...	56401	191	189	0.520	0.003	0.096	0.106	0.139	0.235	0.2

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.



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).

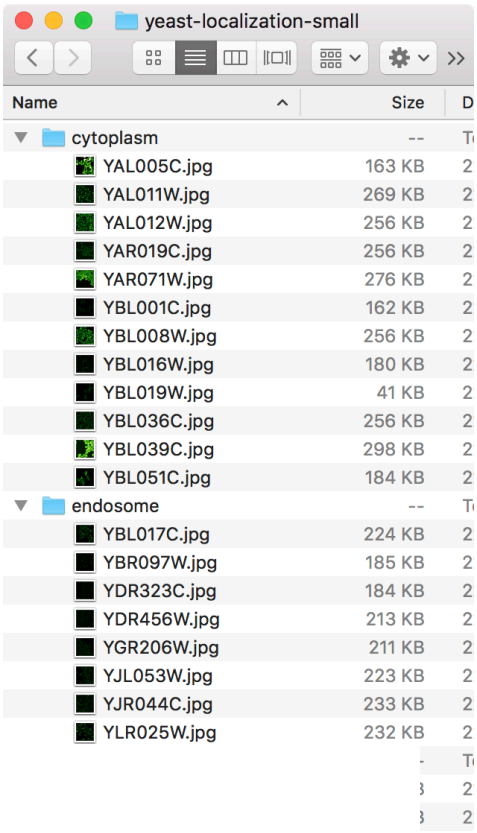
Even the lecturers of this course were surprised at the result. Beautiful!



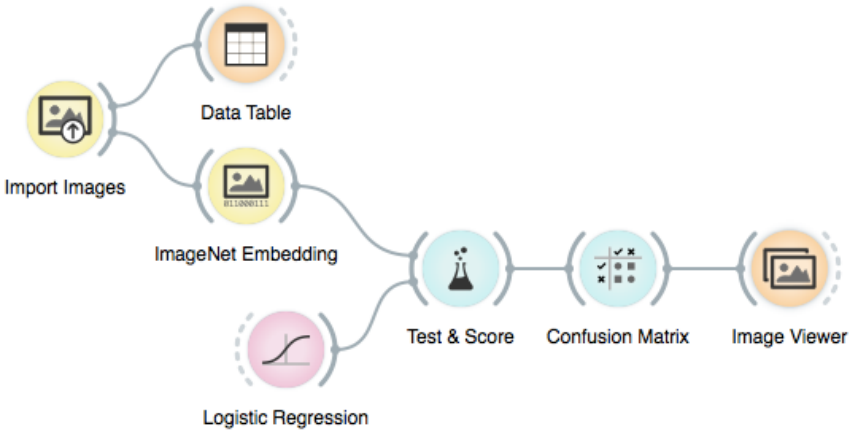
Lesson 22: Images and Classification

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 University of Ljubljana and its installment in 2018 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 scientists. 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.