

BIOLAB AND COLLABORATORS

USING ORANGE

BIOLAB

Copyright © 2022 Biolab and Collaborators

PUBLISHED BY BIOLAB

TUFTE-LATEX.GOOGLECODE.COM

Licensed under the Apache License, Version 2.0 (the “License”); you may not use this file except in compliance with the License. You may obtain a copy of the License at <http://www.apache.org/licenses/LICENSE-2.0>. Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an “AS IS” BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

First printing, April 2022

Contents

<i>Workflows in Orange</i>	7
<i>Basic data exploration</i>	11
<i>Saving your work</i>	14
<i>Loading data sets</i>	15
<i>Assignment: Data Inspection</i>	17
<i>Classification</i>	18
<i>Classification Trees</i>	19
<i>Model Inspection</i>	22
<i>Naive Bayes</i>	23
<i>Classification Accuracy</i>	24
<i>Assignment: Decision Boundaries</i>	25
<i>How to Cheat</i>	26
<i>Random Forests</i>	29
<i>Cross-Validation</i>	30

<i>Assignment: Overfitting</i>	31
<i>A Few More Classifiers</i>	32
<i>Logistic Regression</i>	33
<i>Support Vector Machines</i>	35
<i>k-Nearest Neighbors</i>	36
<i>Assignment: Model Explanation</i>	37
<i>Linear Regression</i>	38
<i>Regularization</i>	41
<i>Hierarchical Clustering</i>	43
<i>Animal Kingdom</i>	45
<i>k-Means Clustering</i>	46
<i>Silhouettes</i>	49
<i>Principal Component Analysis</i>	52
<i>Mapping the Data</i>	57
<i>Assignment: Clustering</i>	60
<i>Assignment: Clustering vs. Classification</i>	61

<i>Image Embedding</i>	62
<i>Text Mining: Introduction</i>	65
<i>Preprocessing Text</i>	67
<i>Concordances</i>	70
<i>Bag of Words</i>	71
<i>Text Clustering</i>	72
<i>Text Classification</i>	75
<i>Logistic Regression</i>	76
<i>Model Evaluation</i>	77
<i>Word Enrichment</i>	78
<i>Predictions</i>	80
<i>Twitter</i>	81
<i>Sentiment Analysis</i>	83
<i>Networks</i>	85
<i>Network Analysis</i>	88
<i>Networks from Distances</i>	90

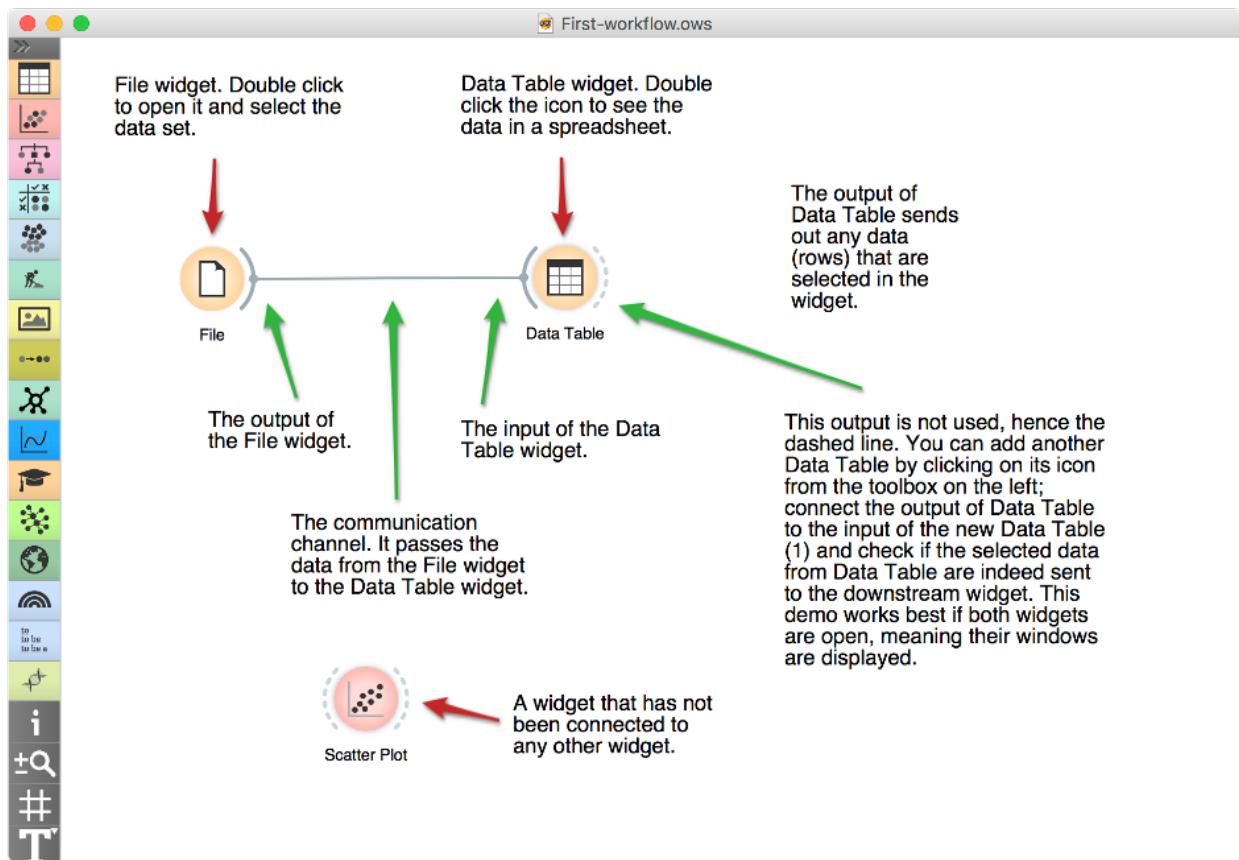
Network from Text 92

Bibliography 95

Index 96

Workflows in Orange

ORANGE WORKFLOWS consist of components that read, process, and visualize data. We refer to these components as “widgets”. We place the widgets on a drawing board called the “canvas” to design a workflow. Widgets communicate by sending information along their communication channel. Output from one widget can be used as input to another.

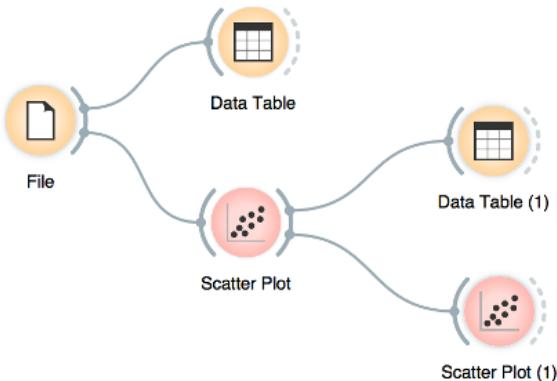


We construct workflows by dragging widgets onto the canvas and connecting them by drawing a line from the transmitting widget to the receiving widget. The widget’s outputs are on the right, and the inputs on the left. In the workflow above, the *File* widget sends data to the *Data Table* widget.

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.

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

A workflow with a File widget that reads the data from a 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. The plot's selected data points 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 pre-loaded 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.

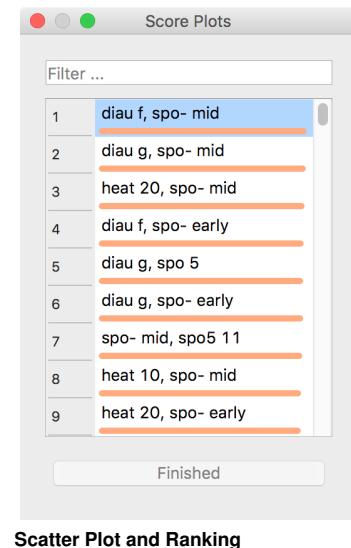
/6	Name	Type	Role	Values
77	diau d	N numeric	feature	
78	diau e	N numeric	feature	
79	diau f	N numeric	feature	
80	diau g	N numeric	feature	
81	function	C categorical	target	Proteas, Resp, Ribo
	gene	S text	meta	

After you load the data:

1. Open the other widgets.
2. Select a few data points in the *Scatter Plot* widget and watch as they appear in the *Data Table (1)*.

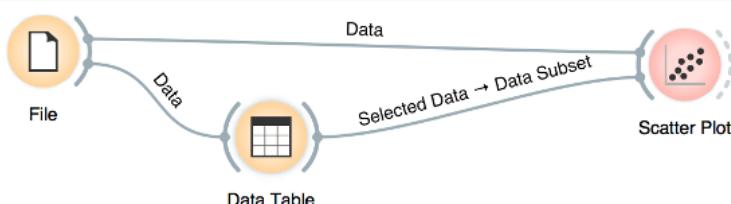
3. Use a combination of two *Scatter Plot* widgets, where the second scatter plot shows a detail from a smaller region selected in the first scatter plot.

The following is a side note, but it won't hurt. 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* at 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 helpful to a biologist?



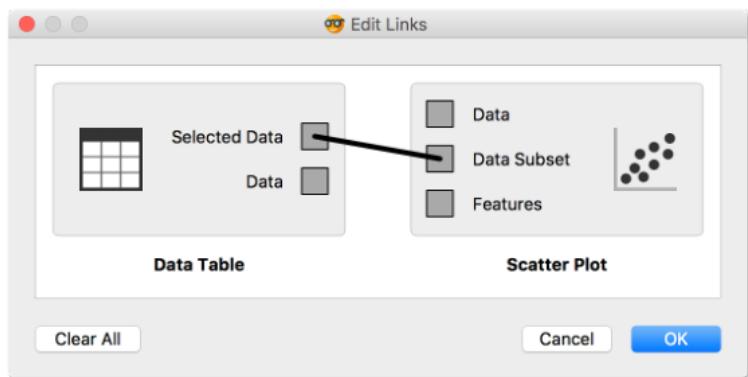
Scatter Plot and Ranking

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 switched 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.



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

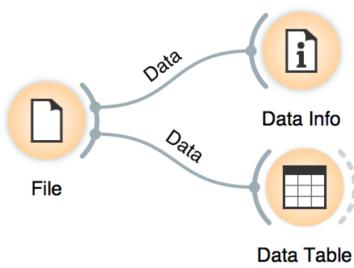


Basic data exploration

LET US CONSIDER ANOTHER PROBLEM, this time from clinical medicine. We will dig for something interesting in the data and explore it with visualization widgets. You will get to know Orangebetter, and also learn about several interesting visualizations.

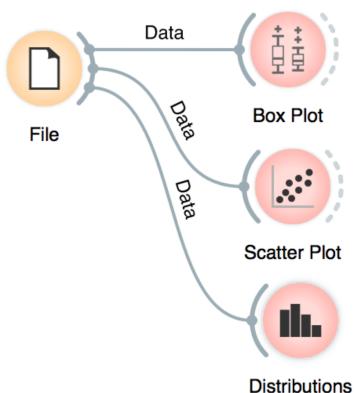
We will start with an empty canvas; to clean it from our previous lesson, use either File/New or select all the widgets and remove them (use the backspace/delete key).

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



A simple workflow to inspect the loaded dataset.

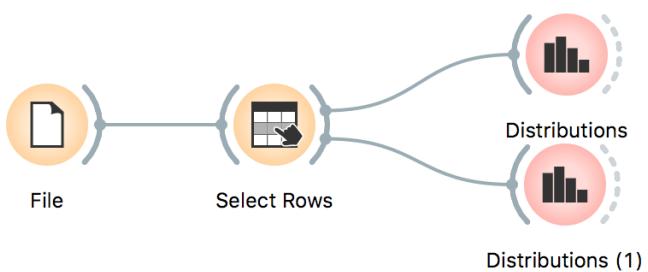
Let us check whether common visualizations tell us anything interesting. (Hint: look for gender differences. These are always interesting and occasionally even real.)



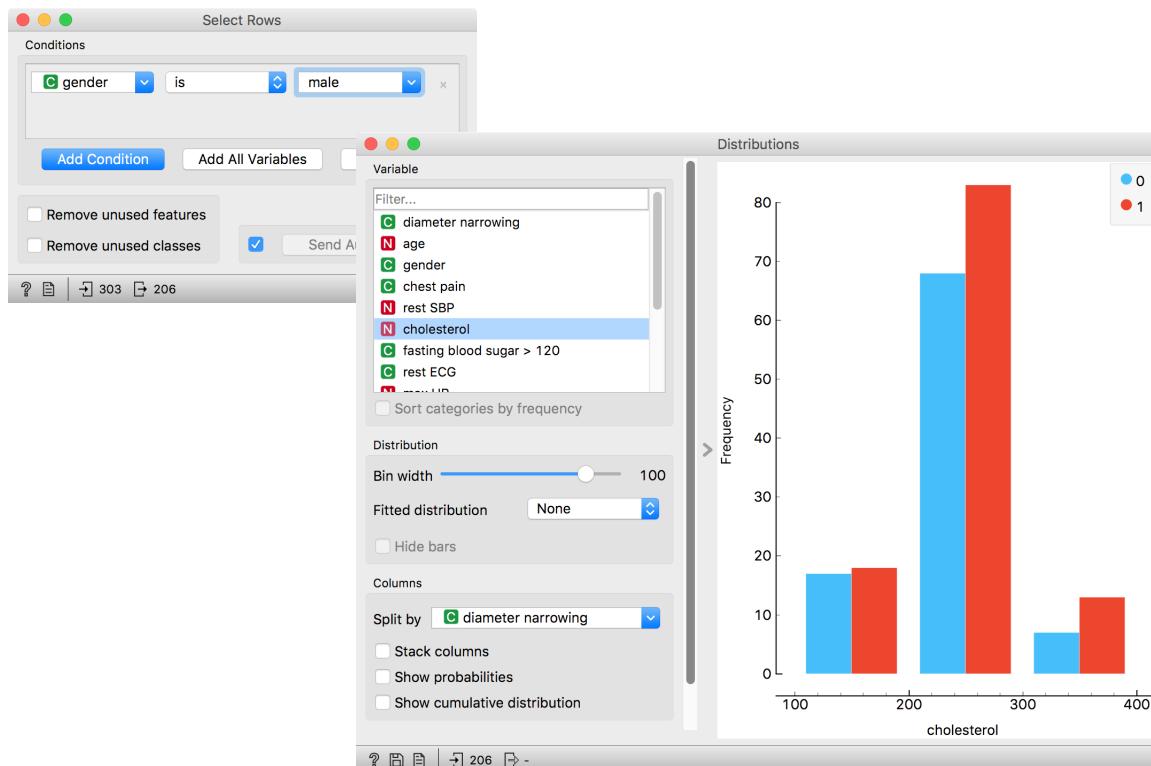
Quick check with common statistics and visualization widgets.

Data can also be split by the value of features, in this case the gender.

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



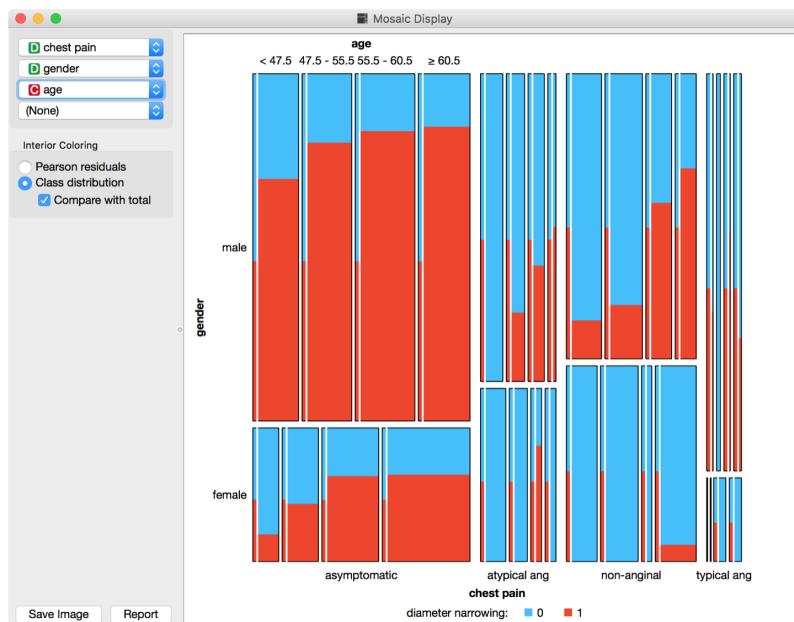
In the *Select Rows* widget, we select the female patients. You can also add other conditions. The selection of data instances provides a powerful combination with visualization of data distribution. Try having at least two widgets open simultaneously and explore the data.



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

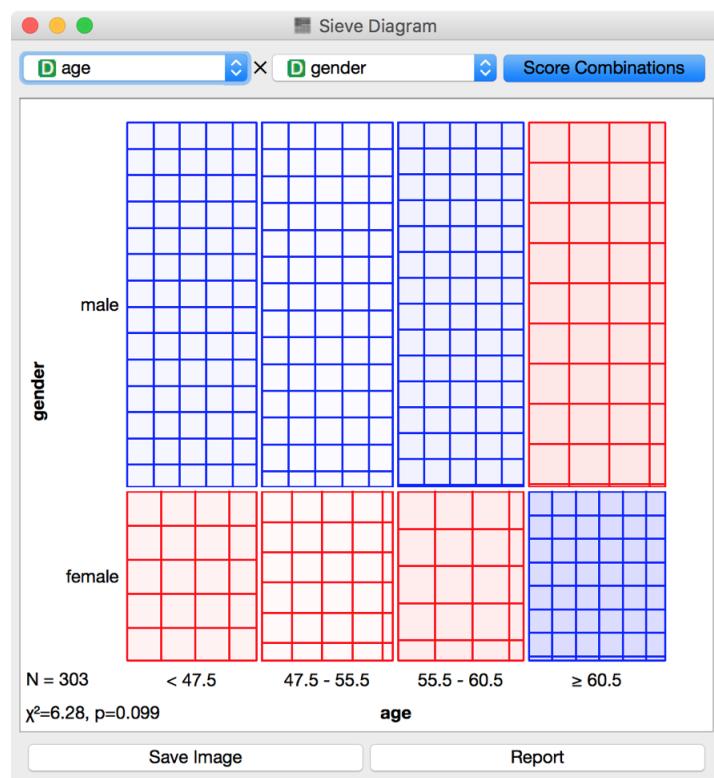
The mosaic display shows a rectangle split into columns with widths reflecting the prevalence of different types of chest pain. Each column is then further split vertically according to gender distributions within the column. The resulting rectangles are split again horizontally according to age group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

What can you read from this diagram?



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

Another visualization, the Sieve diagram also splits a rectangle horizontally and vertically, but with independent cuts, so the areas correspond to the expected number of data instances if the observed variables were independent. For instance, $1/4$ 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 $1/12 \times 300 = 25$ older women in our data. Instead, there are 34. The sieve diagram shows the difference between the expected and the observed frequencies by the grid density and the color of the field.

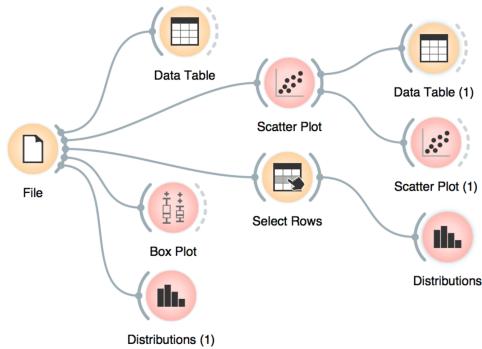


See the Score Combinations button? Try to guess what it does? And how does it score the combinations? Hint: there are some Greek letters at the bottom of the widget.

Saving your work

AT THE END OF A LESSON, your workflow may look like this:

A fairly complex workflow that you would want to share or reuse at a later time.



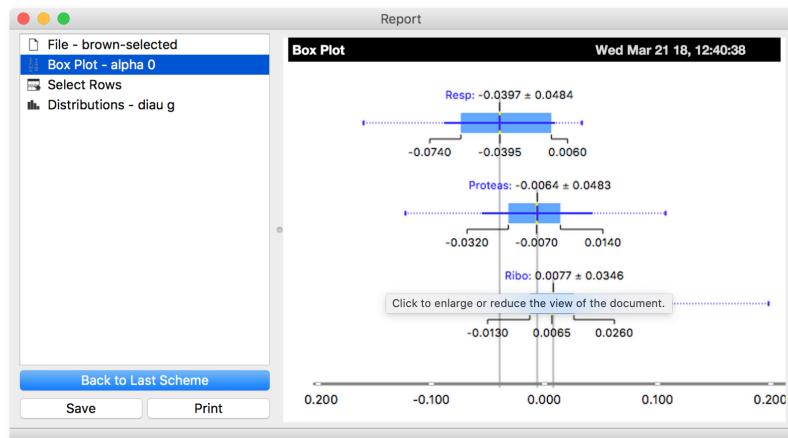
You can save this workflow using the File/Save menu and share it with your colleagues. Just don't forget to put the data files in the same directory as the file with the workflow.

Widgets also have a Report button in their bottom status bar, which you can use to keep a log of your analysis. When you find something interesting, just click it and the graph will be added 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 a section of the report window allows you to add a comment.



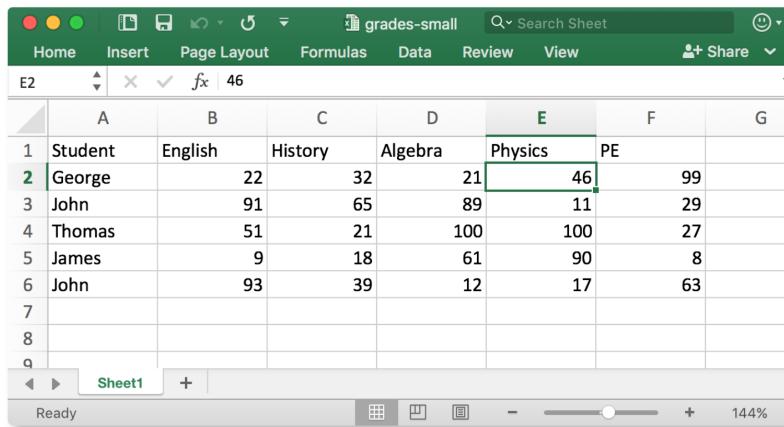
The report window and the additional text input box (bottom).



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

Loading data sets

THE DATA SETS WE HAVE WORKED WITH in the previous lesson come with the Orange installation. Orange can read data from many file formats which include tab and comma separated and Excel files. To see how this works, let's prepare a data set (with school subjects and grades) in Excel and save it on a local disk.

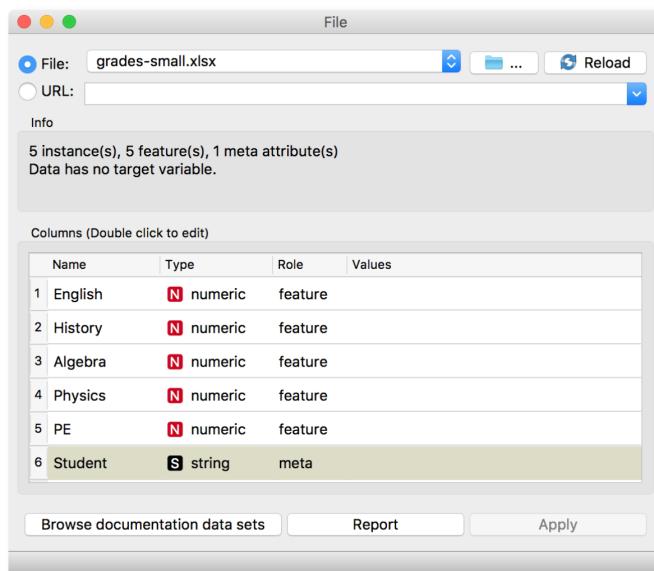


A screenshot of Microsoft Excel showing a data table titled "grades-small". The table has columns labeled A through G. Column A is "Student", B is "English", C is "History", D is "Algebra", E is "Physics", F is "PE", and G is empty. Row 1 contains the column headers. Rows 2 through 6 contain data for five students: George, John, Thomas, James, and John. The "Physics" column (E) shows values 46, 11, 100, 61, and 17 respectively. The "PE" column (F) shows values 99, 29, 27, 8, and 63 respectively. The "Student" column (A) shows values George, John, Thomas, James, and John.

	A	B	C	D	E	F	G
1	Student	English	History	Algebra	Physics	PE	
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							
9							

Make a spreadsheet in Excel with the numbers shown on the left. Of course, you can use any other editor, but remember to save your file in the *comma separated values (*.csv)* format.

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



The *File* widget allows you to select a local file or even paste a URL to a Google Spreadsheet. In the Info box, you will see a quick summary about the data you loaded. By double clicking the fields, you can also edit the types of entries and their role, that will be relevant for further processing.

Looks good! 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 any kind of modeling (more about this in the upcoming lectures). All other columns are numeric features.

It is always good to check if all the data was read correctly. Now, you can connect the *File* widget with the *Data Table* widget,

Construct a simple workflow shown on the right.



and double click on the Data Table to see the data in a spreadsheet format. Nice, everything is here.

	Student	English	History	Algebra	Physics	Physical	GPA
1	George	22.000	32.000	21.000	46.000	99.000	3.000
2	John	91.000	65.000	89.000	11.000	29.000	3.000
3	Thomas	51.000	21.000	100.000	100.000	27.000	3.000
4	James	9.000	18.000	61.000	90.000	8.000	2.000
5	John	93.000	39.000	12.000	17.000	63.000	1.000

The *Data Table* widget shows the loaded data set, you can select rows, which will appear on the output of the widget. It is also possible to do simple data visualizations. Explore the functionalities!

Instead of using Excel, we could 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 URL entry box.

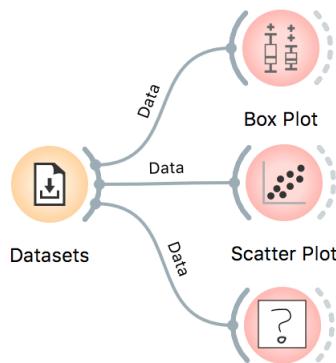
Orange's legacy native data format is a tab-delimited text file with three header rows. The first row lists the attribute names, the second row defines their type (continuous, discrete, time and string, or abbreviated c, d, t, and s), and the third row an optional role (class, meta, weight, or ignore).

There is more to input data formatting and loading. If you would really like to dive in for more, check out the documentation page on [Loading your Data](#), or a [video tutorial](#) on this subject.

Assignment: Data Inspection

UNDERSTANDING THE DATA IS CRUCIAL for any data science task. And the best way to achieve this is with visualizations. With the *Datasets* widget load *Employee Attrition* data set, which describes employees of a company with 32 features, such as age, job role, years at company, and so on. We also know, whether an employee resigned from the company (*Attrition* = Yes) or not (*Attrition* = No).

Workflow for the assignment.



Inspect the data to understand it better:

1. Use *Scatter Plot* and try *Find Informative Projections*. How does the top combination look like? Is it useful? Why (not)?
2. Use *Box Plot* and try *Order by relevance to subgroups*. Which attribute best splits the data by *Attrition*? Explain it.
3. Which widget would you use to observe relationship between two discrete variables in relation to *Attrition*? Use *Find informative projections* in that widget and explore the top result.

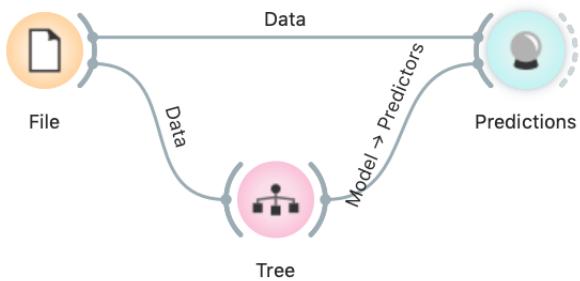
Classification

We call the variable we wish to predict a target variable, or an outcome or, in traditional machine learning terminology, a class. Hence we talk about classification, classifiers, classification trees...

We have seen the iris data before. We wanted to predict varieties based on measurements—but we actually did not make any predictions. We observed some potentially interesting relations between the features and the varieties, but have never constructed an actual model.

Let us create one now.

Something in this workflow is conceptually wrong. Can you guess what?



iris	sepal length	sepal width
Iris-setosa	5.1	3.8
Iris-setosa	4.6	3.2
Iris-setosa	5.3	3.7
Iris-setosa	5.0	3.3
Iris-versicolor	7.0	3.2
Iris-versicolor	6.4	3.2
Iris-versicolor	6.9	3.1
Iris-versicolor	5.5	2.3
Iris-versicolor	6.5	2.8

The data is fed into the *Tree* widget, which infers a classification model and gives it to the *Predictions* widget. Note that unlike in our past workflows, in which the communication between widgets included only the data, we here have a channel that carries a predictive model.

The *Predictions* widget also receives the data from the *File* widget. The 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 very important questions), what is a classification tree? And how does Orange create one? Is this algorithm something we should really use?

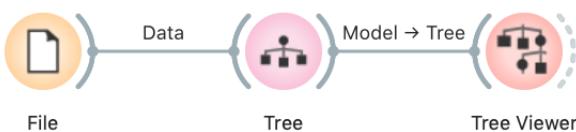
So many questions to answer!

Classification Trees

In the previous lesson, we used a classification tree, 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 (more on this later). So, what kind of model is a classification tree?

Let us load *iris* data set, build a tree (widget *Tree*) and visualize it in a *Tree Viewer*.

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.



Data Table

	iris	sepal length	sepal width	petal length	petal width
1	Iris-setosa	5.1	3.5	1.4	0.2
2	Iris-setosa	4.9	3.0	1.4	0.2
3	Iris-setosa	4.7	3.2	1.3	0.2
4	Iris-setosa	4.6	3.1	1.5	0.2
5	Iris-setosa	5.0	3.6	1.4	0.2
6	Iris-setosa	5.4	3.9	1.7	0.4
7	Iris-setosa	4.6	3.4	1.4	0.3
8	Iris-setosa	5.0	3.4	1.5	0.2
9	Iris-setosa	4.4	2.9	1.4	0.2
10	Iris-setosa	4.9	3.1	1.5	0.1
11	Iris-setosa	5.4	3.7	1.5	0.2
12	Iris-setosa	4.8	3.4	1.6	0.2
13	Iris-setosa	4.8	3.0	1.4	0.1
14	Iris-setosa	4.3	3.0	1.1	0.1
15	Iris-setosa	5.8	4.0	1.2	0.2
16	Iris-setosa	5.7	4.4	1.5	0.4
17	Iris-setosa	5.4	3.9	1.3	0.4

Info
150 instances (no missing values)
4 features (no missing values)
Discrete class with 3 values (no missing values)
No meta attributes

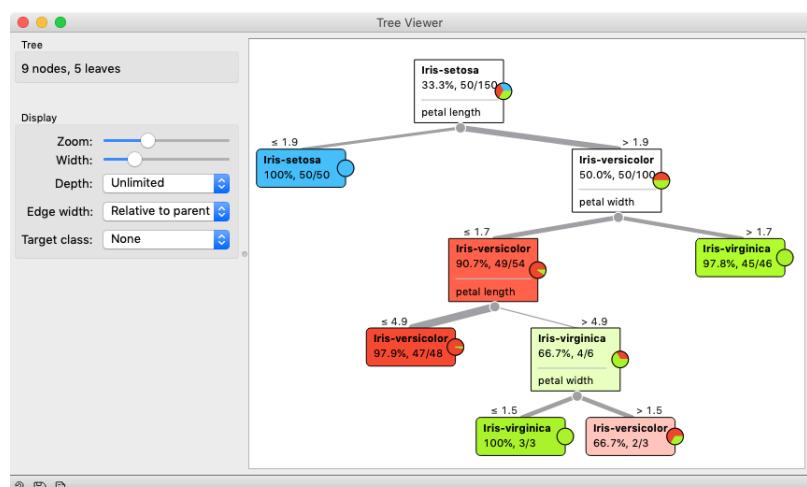
Variables
 Show variable labels (if present)
 Visualize numeric values
 Color by instance classes

Selection
 Select full rows

Restore Original Order
 Send Automatically

We read the tree from top to bottom. Looks like the column *petal length* best separates the iris variety *setosa* from the others, and in the next step, *petal width* then almost perfectly separates the remaining two varieties.

Trees place the most useful feature at the root. What would be the most useful feature? The feature that splits the data into two purest possible subsets. It then splits both subsets further, again by their most useful features, and keeps doing so until it reaches subsets in which all data belongs to the same class (leaf nodes in strong blue or red) or until it runs out of data instances to



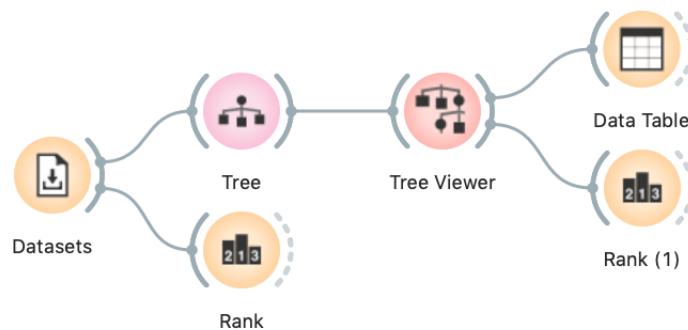
The *Rank* widget can be used on its own to show the best predicting features. Say, to figure out which genes are best predictors of the phenotype in some gene expression data set.

The *Datasets* widget is set to load the *Sailing* data set. To use the second *Rank*, select a node in the *Tree Viewer*.

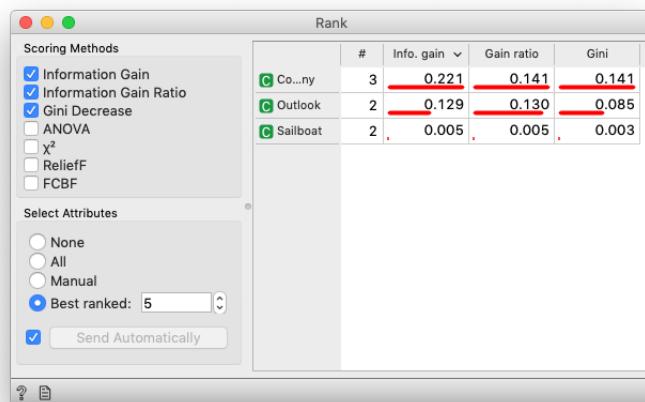
For the whole *Sailing* data set, *Company* is the most class-informative feature according to all measures shown.

split or out of useful features (the two leaf nodes in white).

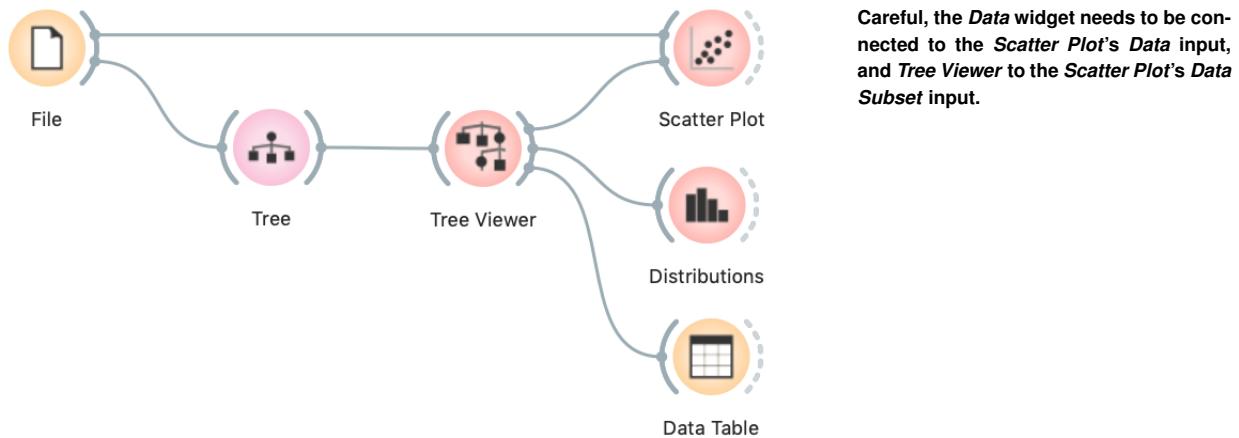
We still have not been very explicit about what we mean by "the most useful" feature. There are many ways to measure the quality of features, based on how well they distinguish between classes. We will illustrate the general idea with information gain. We can compute this measure in Orange using the *Rank* widget, which estimates the quality of data features and ranks them according to how informative they are about the class. We can either estimate the information gain from the whole data set, or compute it on data corresponding to an internal node of the classification tree in the *Tree Viewer*. In the following example we use the *Sailing* data set.



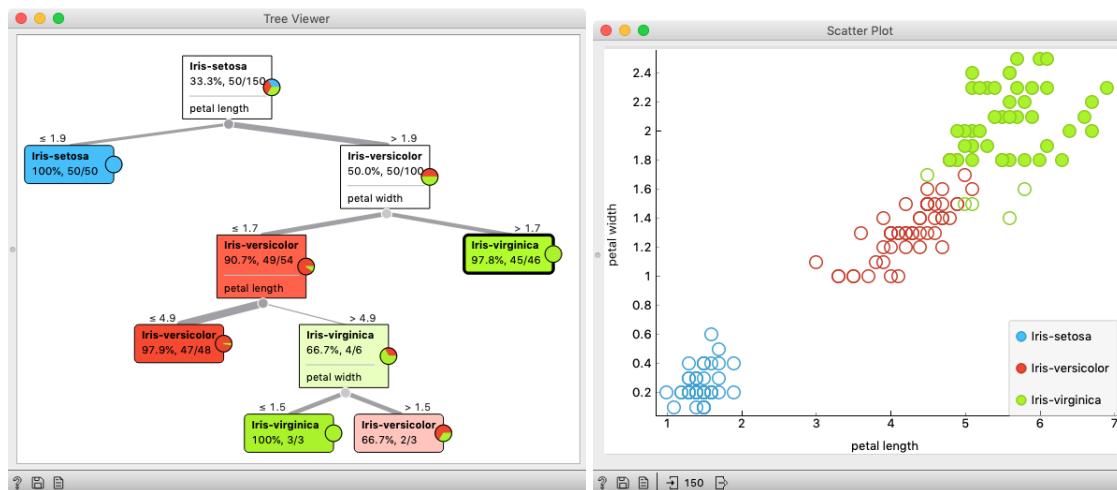
Besides the information gain, *Rank* displays several other measures (including Gain Ratio and Gini), which are often quite in agreement and were invented to better handle discrete features with many different values.



Here is an interesting combination of a *Tree Viewer* and a *Scatter Plot*. This time, use the *Iris* data set. In the *Scatter Plot*, we first find the best visualization of this data set, that is, the one that best separates the instances from different classes. Then we connect the *Tree Viewer* to the *Scatter Plot*. Data instances (particular irises) from the selected node in the *Tree Viewer* are shown in the *Scatter Plot*.



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

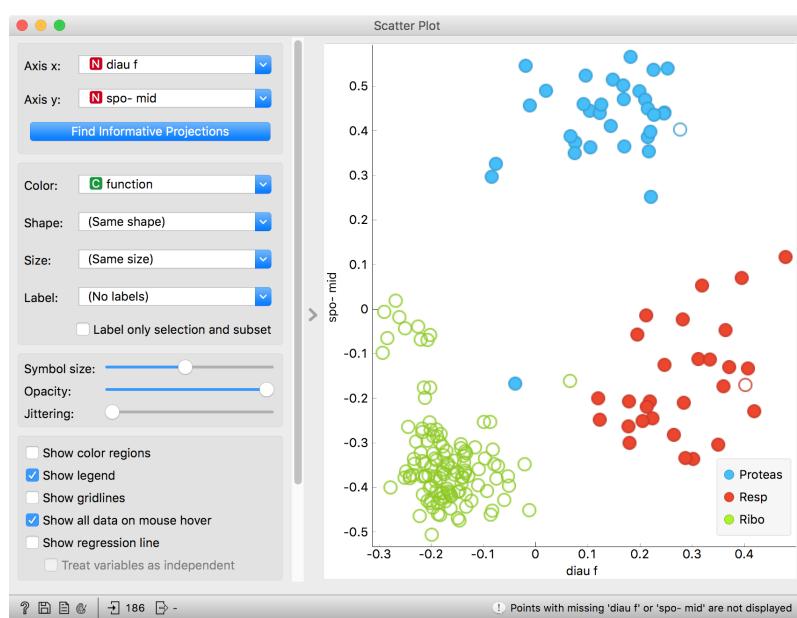
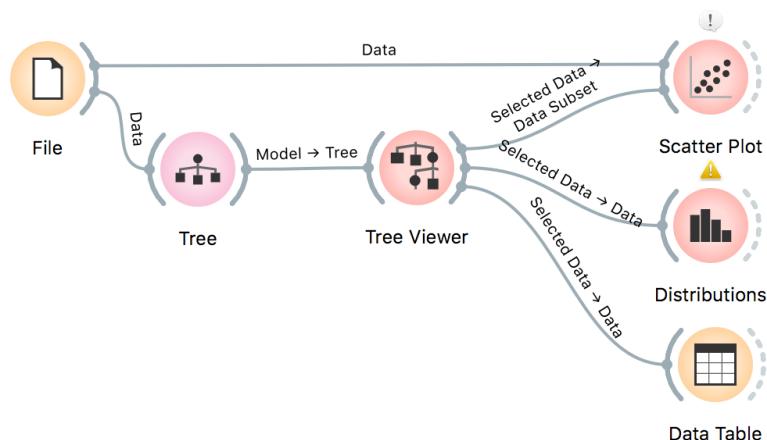


Wherever possible, visualizations in Orange are designed to support selection and passing of the data that applies to it. Finding interesting data subsets and analyzing their commonalities is a central part of explorative data analysis, a data analysis approach favored by the data visualization guru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

In the *Tree Viewer* we selected the right-most node. All data instances coming to the selected node are highlighted in *Scatter Plot*.

Model Inspection

Here's another interesting combination of widgets: *Tree Viewer* and *Scatter Plot*. In Scatter Plot, find the best visualization of this data set, that is, the one that best separates instances from different classes. Then connect Tree Viewer to Scatter Plot. Selecting any node of the tree will output the corresponding data subset, which will be shown in the scatter plot.



Just for fun, we have included a few other widgets in this workflow. The Tree Viewer selects data instances by inferring rules from the data itself and optimizing to obtain purer data subsets.

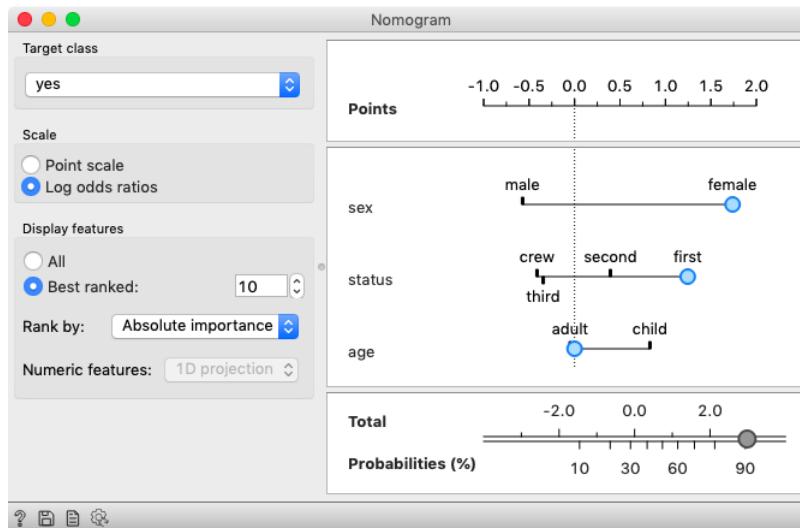
Naive Bayes

Naive Bayes is also a classification method. To see how naive Bayes works, we will use a data set on passengers' survival in the Titanic disaster of 1912. The *Titanic* data set describes 2201 passengers, with their tickets (first, second, thirds class or crew), age and gender.



We inspect naive Bayes models with the *Nomogram* widget. There, we see a scale 'Points' and scales for each feature. Below we can see probabilities. Note the 'Target class' in upper left corner. If it is set to 'yes', the widget will show the probability that a passenger survived.

The nomogram shows that gender was the most important feature for survival. If we move the blue dot to 'female', the survival probability increases to 73%. Furthermore, if that woman also travelled in the first class, she survived with probability of 90%. The bottom scales show the conversion from feature contributions to probability.



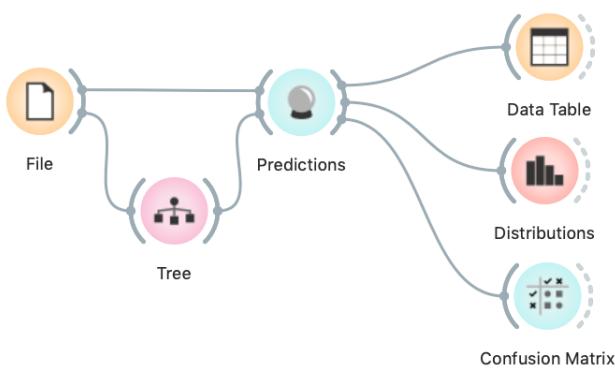
Naive Bayes assumes class-wise independent features. For a data set where features would actually be independent, which rarely happens in practice, the naive Bayes would be the ideal classifier.

According to the probability theory individual contributions should be multiplied. Nomograms get around this by working in a log-space: a sum in the log-space is equivalent to multiplication in the original space. Therefore nomograms sum contributions (in the log-space) of all feature values and then convert them back to probability.

Classification Accuracy

$$\text{accuracy} = \frac{\#\{\text{correct}\}}{\#\{\text{all}\}}$$

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.

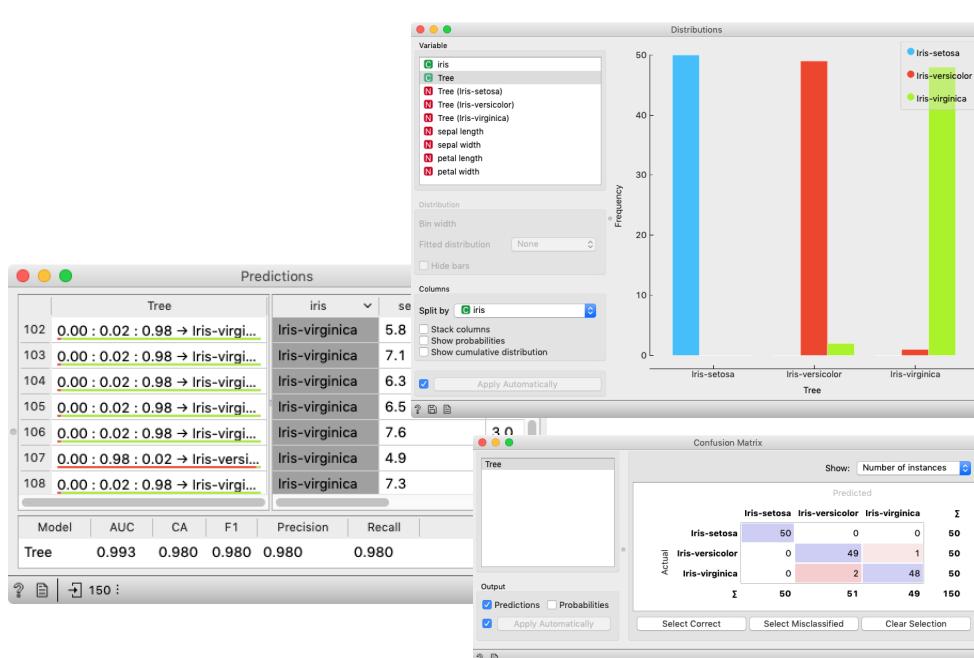


Let us try this schema with the *iris* data set. The *Predictions* widget outputs a data table augmented with a column that includes predictions. In the *Data Table* widget, we can sort the data by any of these two columns, and manually select data instances where the values of these two features are 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.

For precise statistics of correctly and incorrectly classified examples open the *Confusion Matrix* widget.

Let us try this schema with the *iris* 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.

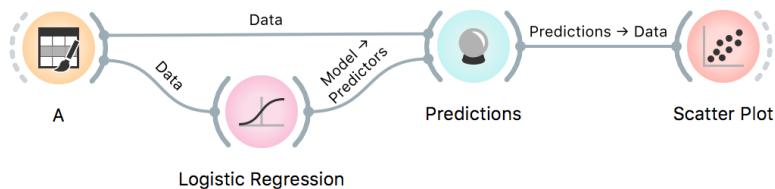
For precise statistics of correctly and incorrectly classified examples open the *Confusion Matrix* widget.



The *Confusion Matrix* shows 3 incorrectly classified examples, which makes the accuracy $(150 - 3)/150 = 98\%$.

Assignment: Decision Boundaries

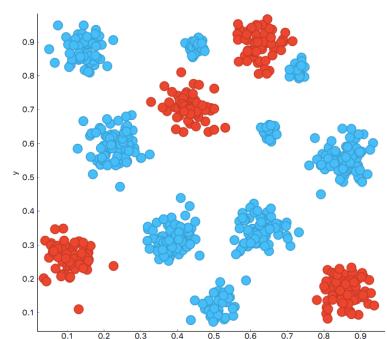
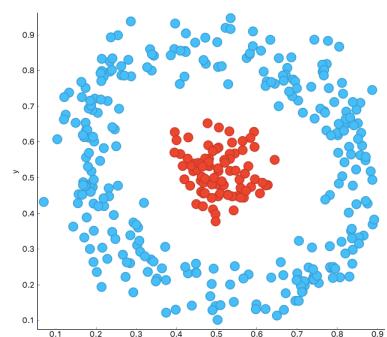
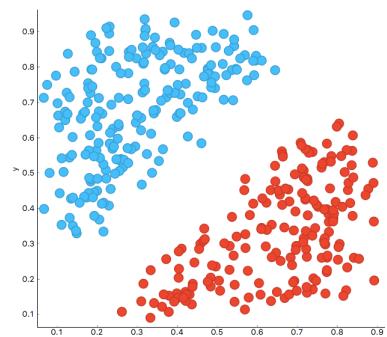
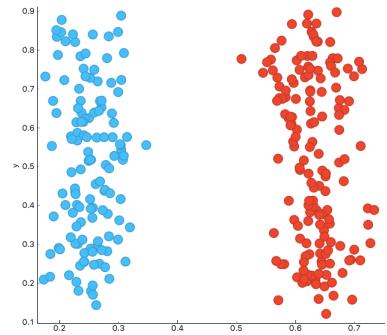
CLASSIFIERS COME IN ALL SHAPES AND SIZES. What we mean by that is that each has its own way of learning from the data, its own strengths and weaknesses. Knowing how classifiers work is crucial for selecting the right algorithm for your task.



Try *Tree*, *Logistic Regression*, *SVM*, *Random Forest*, and *kNN* classifiers:

1. Which classifiers work well with data set A? Which with B, C, and D?
2. Which classifier is struggling the most? Which one the least? Why?
3. Look at the Tree with *Tree Viewer* for data set C. What do you notice?
4. In the above example, you can separate classes with a single stroke of a pen. Now limit Tree depth by setting *Limit maximal tree depth to 2*, which replicates drawing a single line to separate the classes. What do you notice? What happens, when you increase the depth of the tree?

You can try painting the data yourself or download it from [here](#).



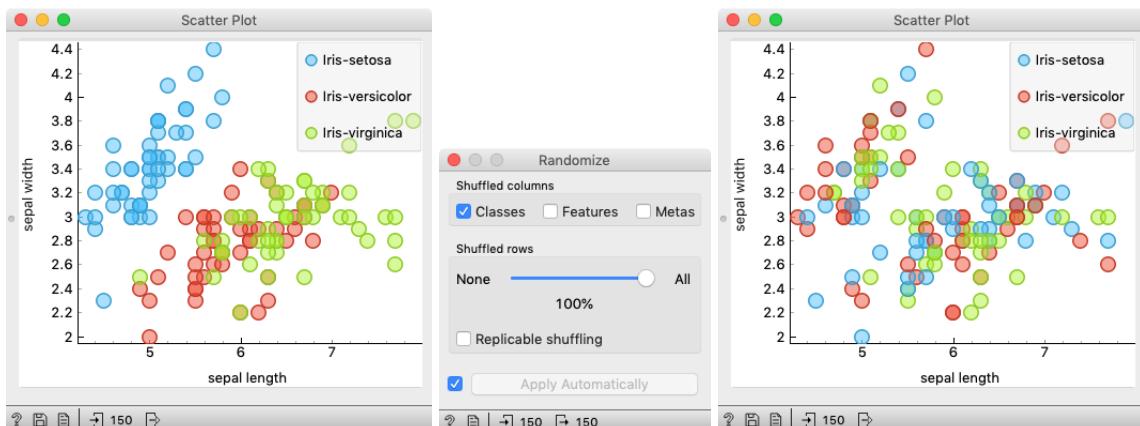
This lesson has a strange title and it is not obvious why it was chosen. Maybe you, the reader, should tell us what this lesson has to do with cheating.



How to Cheat

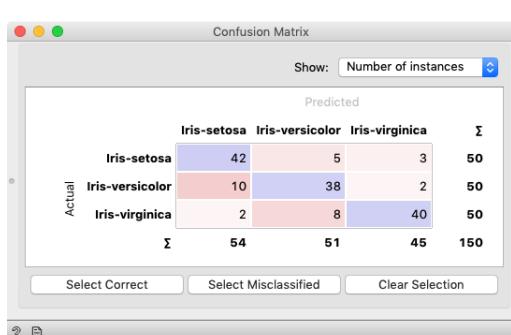
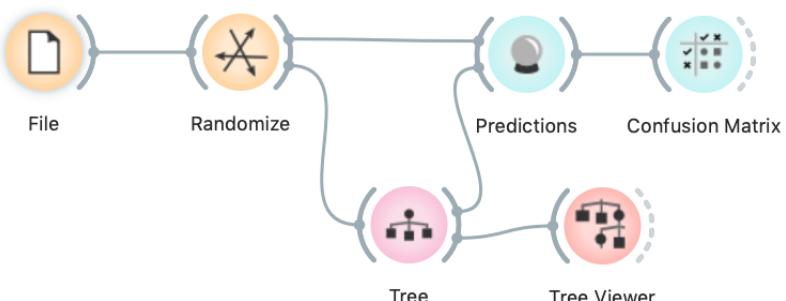
At this stage, the classification tree looks very good. There's only one data point where it makes a mistake. Can we mess up the data set so bad that the trees will ultimately fail? Like, remove any existing correlation between features and the class?

We can! There's the *Randomize* widget with class shuffling. Check out the chaos it creates in the *Scatter Plot* visualization where there were nice clusters before randomization!



Left: scatter plot of the *Iris* data set before randomization; right: scatter plot after shuffling 100% of rows.

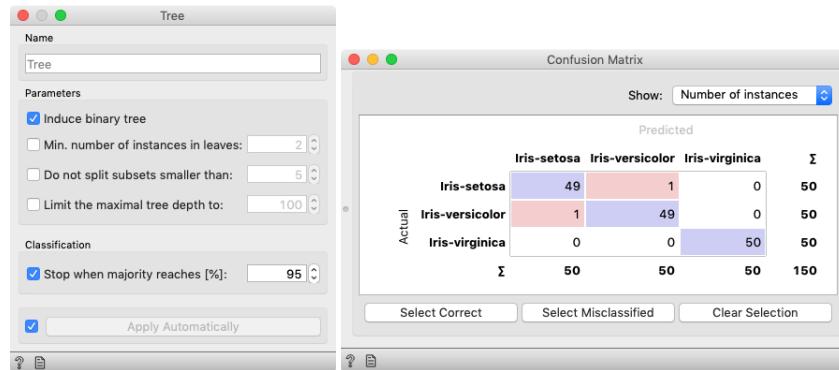
Fine. There can be no classifier that can model this mess, right? Let's make sure.



And the result? Here is a screenshot of the *Confusion Matrix*.

Most unusual. Despite shuffling all the classes, which destroyed any connection between features and the class variable, about 80% of predictions were still correct.

Can we further improve accuracy on the shuffled data? Let us try to change some properties of the induced trees: in the *Tree* widget, disable all early stopping criteria.



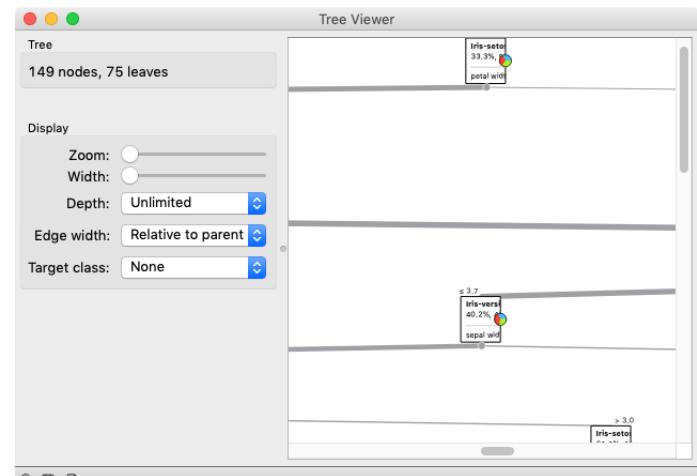
After we disable 2–4 check box in the *Tree* widget, our classifier starts behaving almost perfectly.

Wow, almost no mistakes now. How is this possible? On a class-randomized data set?

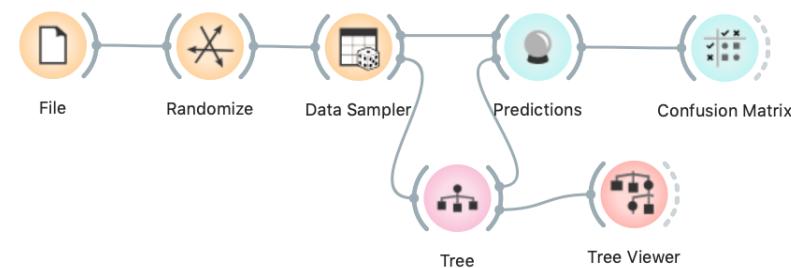
To find the answer to this riddle, open the *Tree Viewer* and check out the tree. How many nodes does it have? Are there many data instances in the leaf nodes?

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.

Ha, if this is so, if a classifier remembers everything from a data set but without discovering any general patterns, it should perform miserably on any new data set. 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.



In the build tree, there are 75 leaves. Remember, there are only 150 rows in the *Iris* data set.



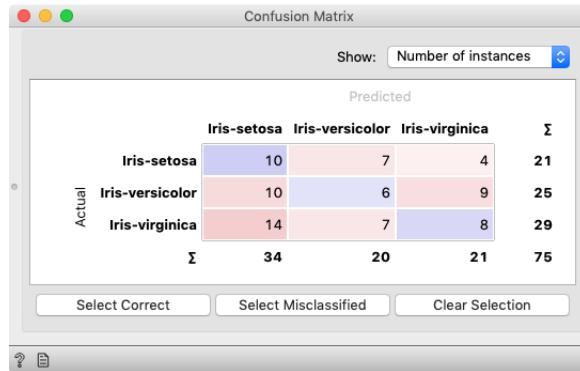
Connect the *Data Sampler* widget carefully. The *Data Sampler* splits the data to a sample and out-of-sample (so called remaining data). The sample was given to the *Tree* widget, while the remaining data was handed to the *Predictions* widget. Set the *Data Sampler* so that the size of these two data sets is about equal.

Let's check how the *Confusion Matrix* looks after testing the classifier on the test data.

The first two classes are a complete fail. The predictions for ribosomal genes are a bit better, but still with lots of mistakes. On the

class-randomized training data our classifier fails miserably. Finally, just as we would expect.

Confusion matrix if we estimate accuracy on a data set that was not used in learning.

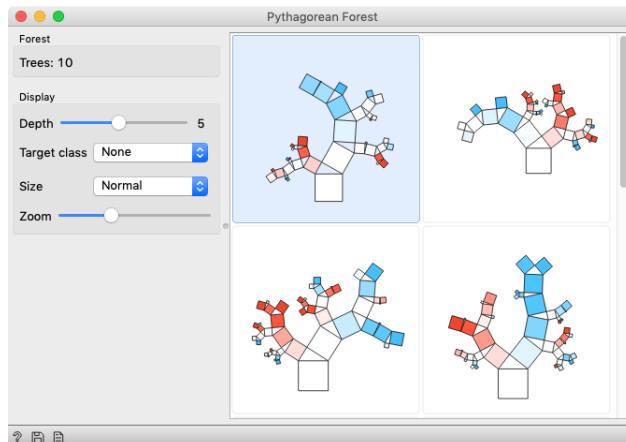
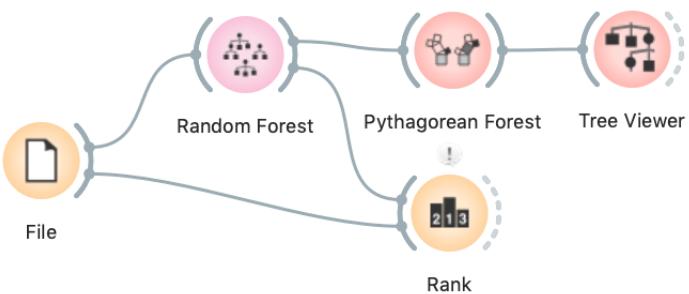


We have just learned that we need to train the classifiers on the training set and then test it on a separate test set to really measure performance of a classification technique. With this test, we can distinguish between those classifiers that just memorize the training data and those that actually learn a general model.

Learning is not only memorizing. 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 estimate 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.

Random Forests

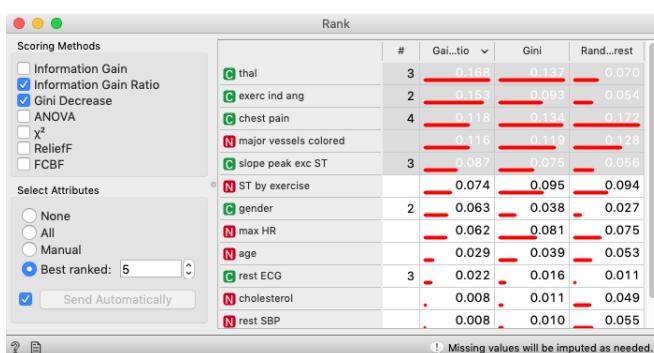
Random forests, a modeling technique introduced in 2001, is still one of the best performing classification and regression techniques. Instead of building a tree by always choosing the one feature that seems to separate best at that time, it builds many trees in slightly random ways. Therefore the induced trees are different. For the final prediction the trees vote for the best class.



The *Pythagorean Forest* widget shows us how random the trees are. If we select a tree, we can observe it in a *Tree Viewer*.

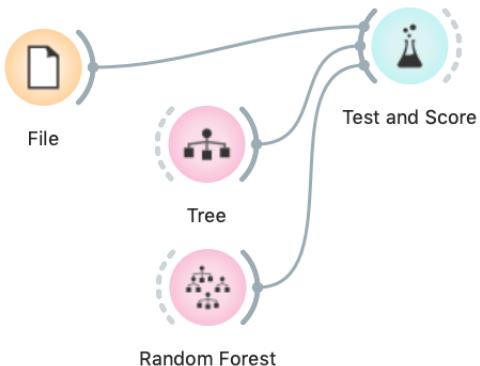
There are two sources of randomness: (1) training data is sampled with replacement, and (2) the best feature for a split is chosen among a subset of randomly chosen features.

Which features are the most important? The creators of random forests also defined a procedure for computing feature importances from random forests. In Orange, you can use it with the *Rank* widget.



Feature importance according to two univariate measures (gain ratio and gini index) and random forests. Random forests also consider combinations of features when evaluating their importance.

Cross-Validation



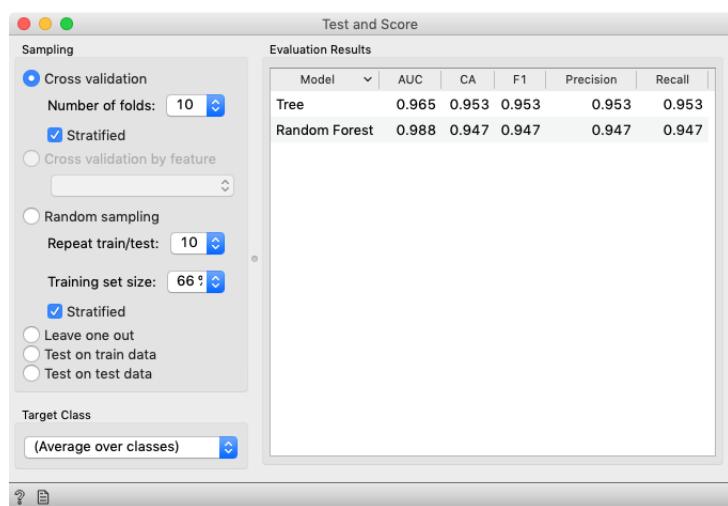
For geeks: a learner is an object that, given the data, outputs a classifier. Just what *Test and Score* needs.

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.

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 in the *Test and Score* widget.

Note that in each iteration, *Test and Score* will pick a 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 the 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 and Score* needs a learner on its input.

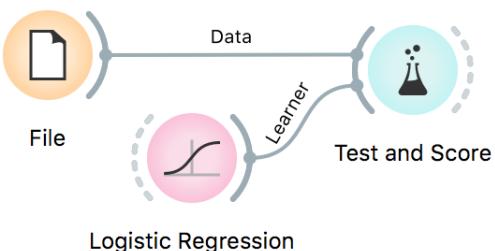
This is another way to use the *Tree* widget. In the workflows from the previous lessons we have used another of its outputs, called *Model*; its construction required data. This time, no data is needed for *Tree*, because all that we need from it is a *Learner*.



In the *Test and Score* widget, the second column, CA, stands for classification accuracy, and this is what we really care for now.

Assignment: Overfitting

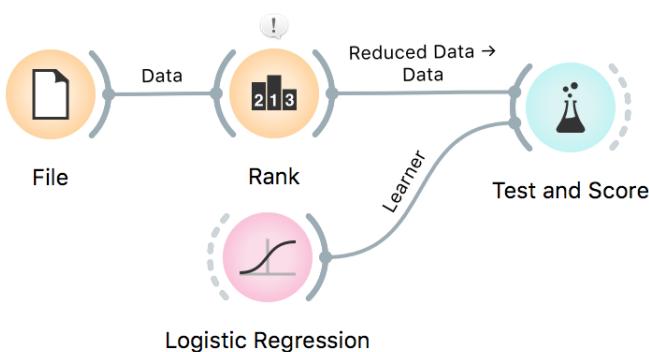
OVERFITTING IS SOMETHING WE TRY TO AVOID AT ALL TIMES. But overfitting comes in many shapes and sizes. For this exercise we will use a *blood-loneliness* data set with the File widget. This data set relates gene expressions in blood with a measure of loneliness obtained from a sample of elderly persons. Let's try to model loneliness with logistic regression and see how well the model performs.



To load the blood loneliness data set copy and paste the below URL to the URL field of the File widget.

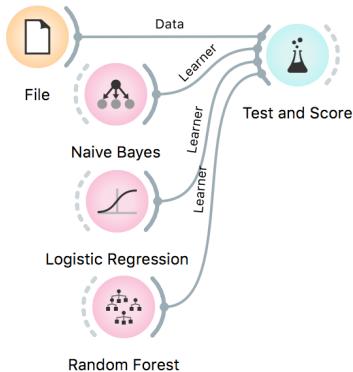
<http://file.biolab.si/datasets/blood-loneliness-GDS3898.tab>

1. Train the Logistic Regression model on the data and observe its performance. What is the result?
2. We have many features in our data. What if we select only the most relevant ones, the genes that actually matter? Use Rank to select the top 20 best performing features.



3. How are the results now? What happened? Is there a better way of performing feature selection?

A Few More Classifiers



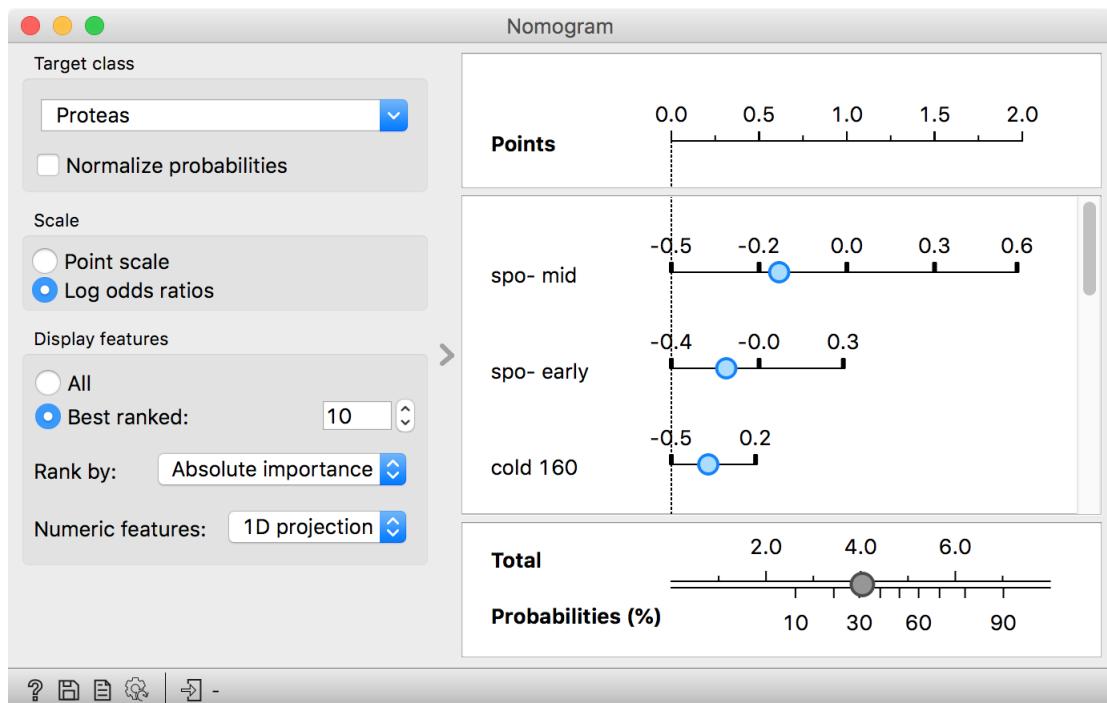
Logistic regression is a classification, not a regression method. It is called regression because it is methodologically similar to linear regression.



We have ended the previous lesson with cross-validation and classification trees. 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.

Another popular classifier is logistic regression. In this model, each variable has its weight or importance. Logistic regression computes weights of each variable during the training phase. For prediction, it simply multiplies the weight of the variable with its value, computes the total sum and log transforms it into probability.

We can use Nomogram to observe the importance of variables in a model and their weights. The variables in the plot are ranked and the length of the line corresponds to their importance in the model.



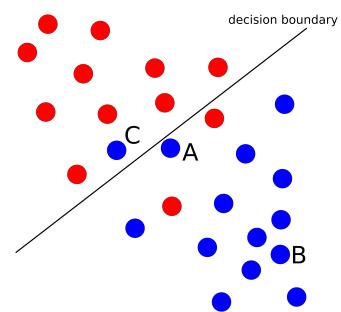
Logistic Regression

Logistic regression is one of the best-known classifiers. The model returns the probability of a class variable, based on input features. First, it computes probabilities with a one-versus-all approach, meaning that for a multiclass problem, it will take one target value and treat all the rest as "other", effectively transforming the problem to binary classification.

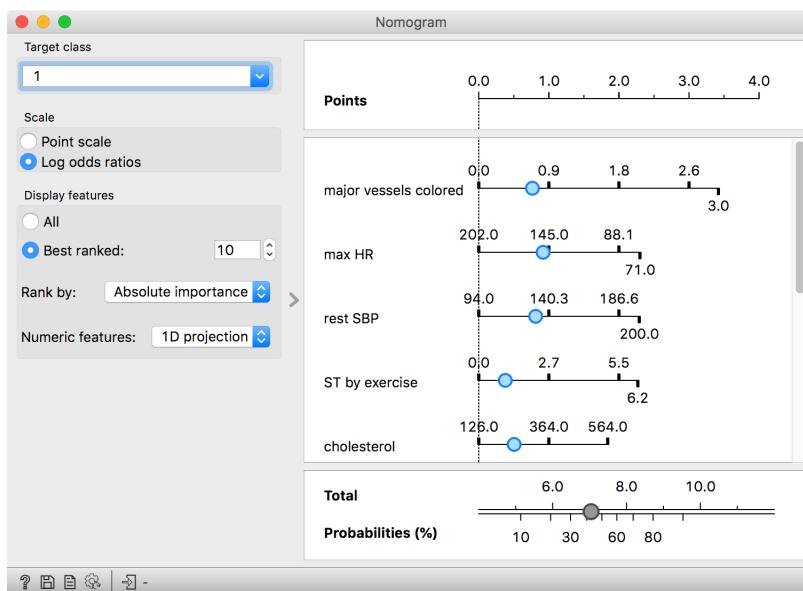
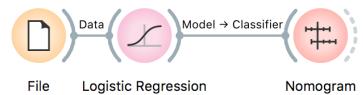
Second, it tries to find an optimal plane that separates instances with the target value from the rest. Then it uses logistic function to transform the distance to the plane into probabilities. The further away from the plane an instance will be, the higher the probability it belongs to the class on that side of the plane. The closer it is to the decision boundary (the plane), the more uncertain the prediction becomes (i.e. it gets close to 0.5).

Logistic regression tries to find such a plane that all points from one class are as far away from the boundary (in the correct direction) as possible.

A great thing about *Logistic Regression* is that we can interpret it with a *Nomogram*. Nomogram shows the importance of variables for the model. The higher the variable is in the list, the greater its importance. Also, the longer the line, the greater the importance. The line corresponds to the coefficient of the variable, which is then mapped to the probability. You can drag the blue point on the line left or right, decreasing or increasing the probability of the target class. This will show you how different values affect the outcome of the model.



Can you guess what would the probability for belonging to the blue class be for A, B, and C?



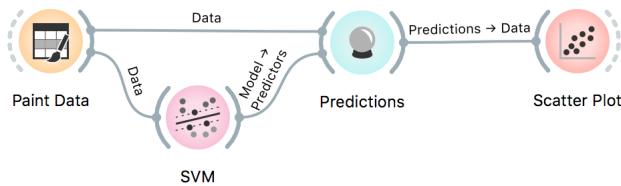
Another characteristic of logistic regression is that it observes all variables at once and takes the correlation into account. If some variables are correlated, their importance will be spread among them.

A not so great thing about logistic regression is that it operates with planes, meaning that the model won't work when the data cannot be separated in such a way. Can you think of such a data set?

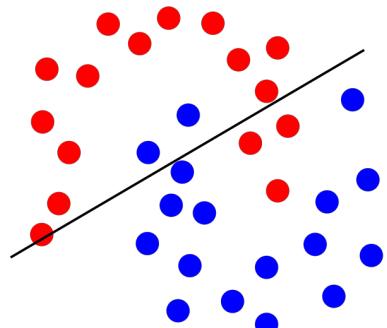
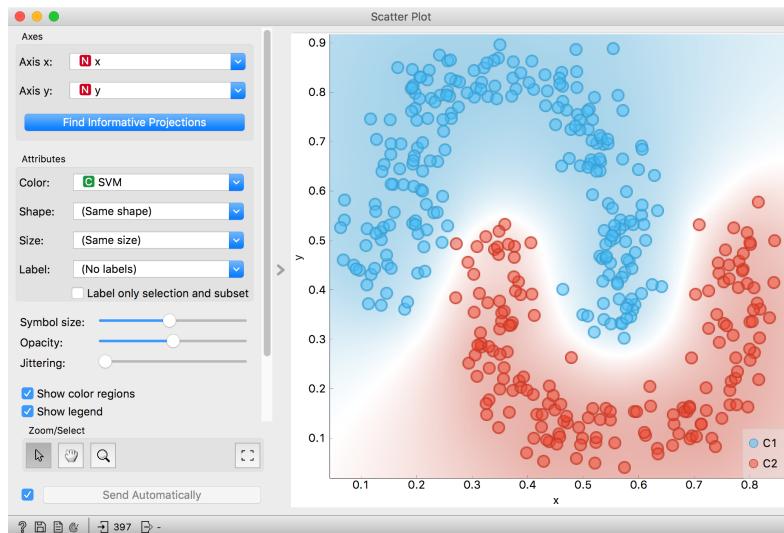
Support Vector Machines

Support vector machines (SVM) are another example of linear classifiers, similar to logistic or linear regression. However, SVM can overcome splitting the data by a plane by using the so-called *kernel trick*. This means the hyperplane (decision boundary) can be transformed to a higher-dimensional space, which can fit the data nicely. In such a way, SVM becomes a non-linear classifier and can fit more complex data sets.

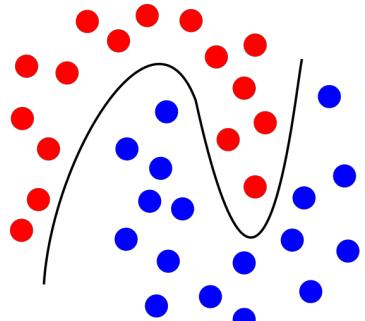
The magic of SVM (and other methods that can use kernels, and are thus called kernel methods) is that they will implicitly find a transformation into a (usually infinite-dimensional) space, in which the distances between objects are such as prescribed by the kernel, and draw a hyperplane in this space.



Abstract talking aside, SVM with different kernels can split the data not by ordinary hyperplanes, but with more complex curves. The complexity of the curve is decided by the kernel type and by the arguments given to the algorithm, like the degree and coefficients, and the penalty for misclassifications.

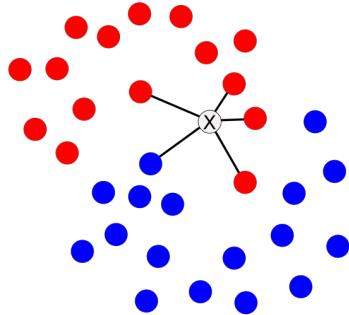


Decision boundary of a linear regression classifier.



Decision boundary of a support vector machine classifier with an RBF kernel.

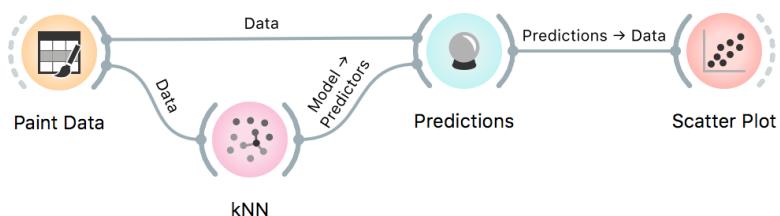
k-Nearest Neighbors



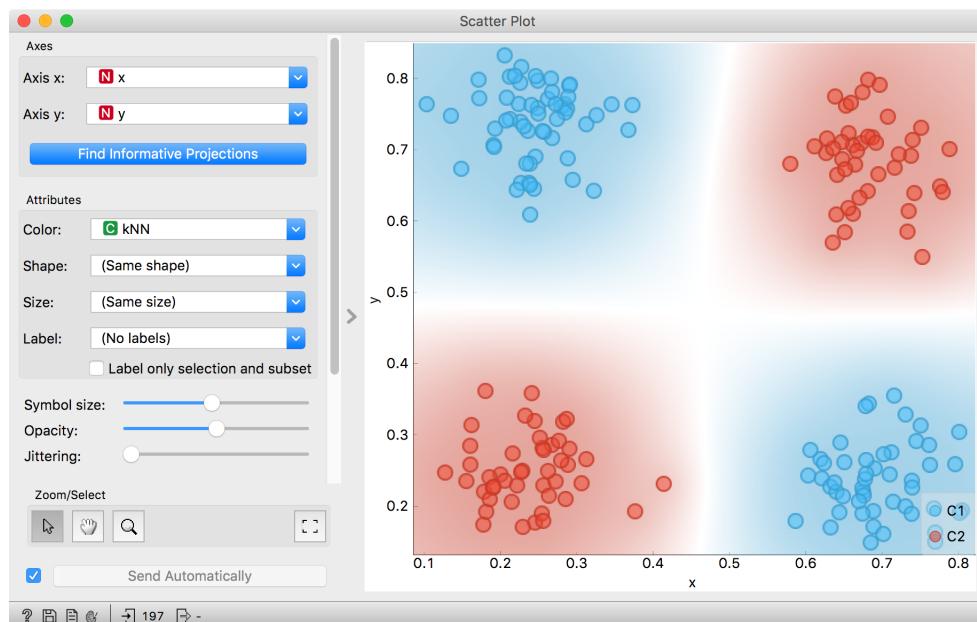
kNN classifier looks at k nearest neighbors, say 5, of instance X . 4 neighbors belong to the red class and 1 to the blue class. X will thus be classified as red with 80% probability.

The idea of k -nearest neighbors is simple - find k instances that are the most similar to each data instance. We make the prediction or estimate probabilities based on the classes of these k instances. For classification, the final label is the majority label of k nearest instances. For regression, the final value is the average value of k nearest instances.

Unlike most other algorithms, kNN does not construct a model but just stores the data. This kind of learning is called *lazy learning*.



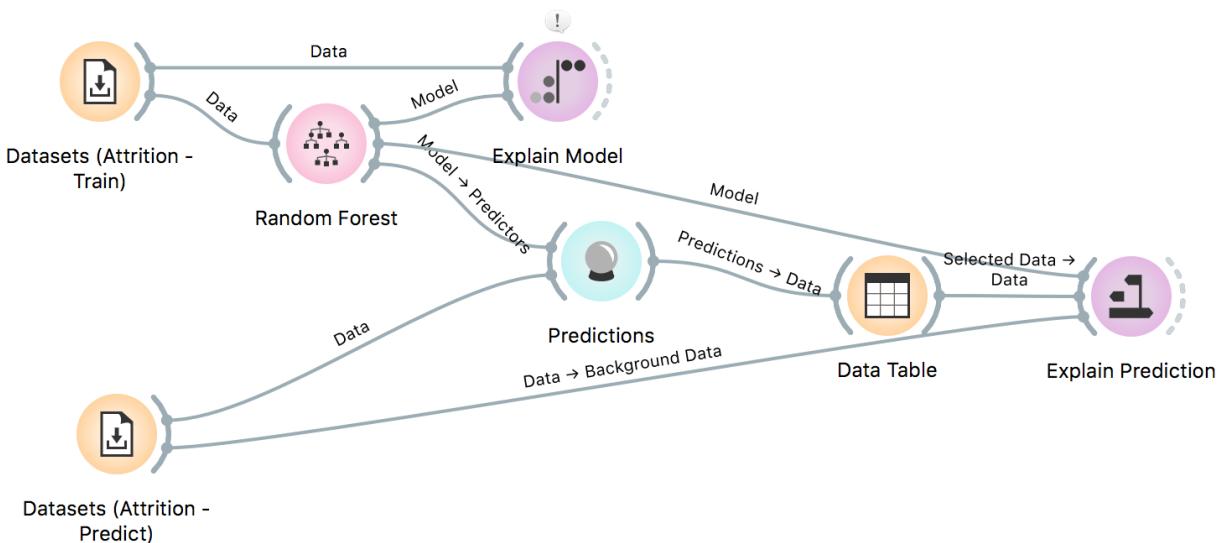
The advantage of kNN algorithm is that it can successfully model the data, where classes are not linearly separable. It can also be re-trained quickly, because new data instances effect model only locally. However, the first training is can be slow for large data sets, as the model has to estimate k distances for data instance.



Assignment: Model Explanation

UNDERSTANDING THE MODEL IS ESSENTIAL FOR DECISION-MAKING.
We will be using *Attrition - Train* data from the *Datasets* widget. We already know the data and its properties. But for reaching any kind of decisions regarding employee attrition, it is important not only to evaluate the models, but to understand them - what they do, which features are important, and in what way.

For this assignment, we will be using Explain add-on, which you can install in Options – Add-ons.

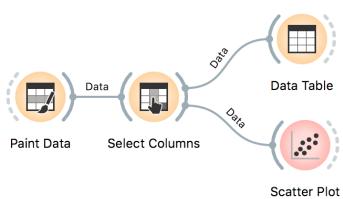


Use *Random Forest* to train a model, then answer the following questions:

1. Looking at *Explain Model*, which are the top three features for the random forest model?
2. Change the target class to *Yes*. What happens?
3. Two of the top three features are the same. Why?
4. In *Data Table*, select the employee for whom you wish to explain the prediction. Say, we go with John. Will John likely stay with the company or resign? Look at *Predictions* for an answer.
5. In *Explain Predictions*, explain *why* John is leaving or staying.

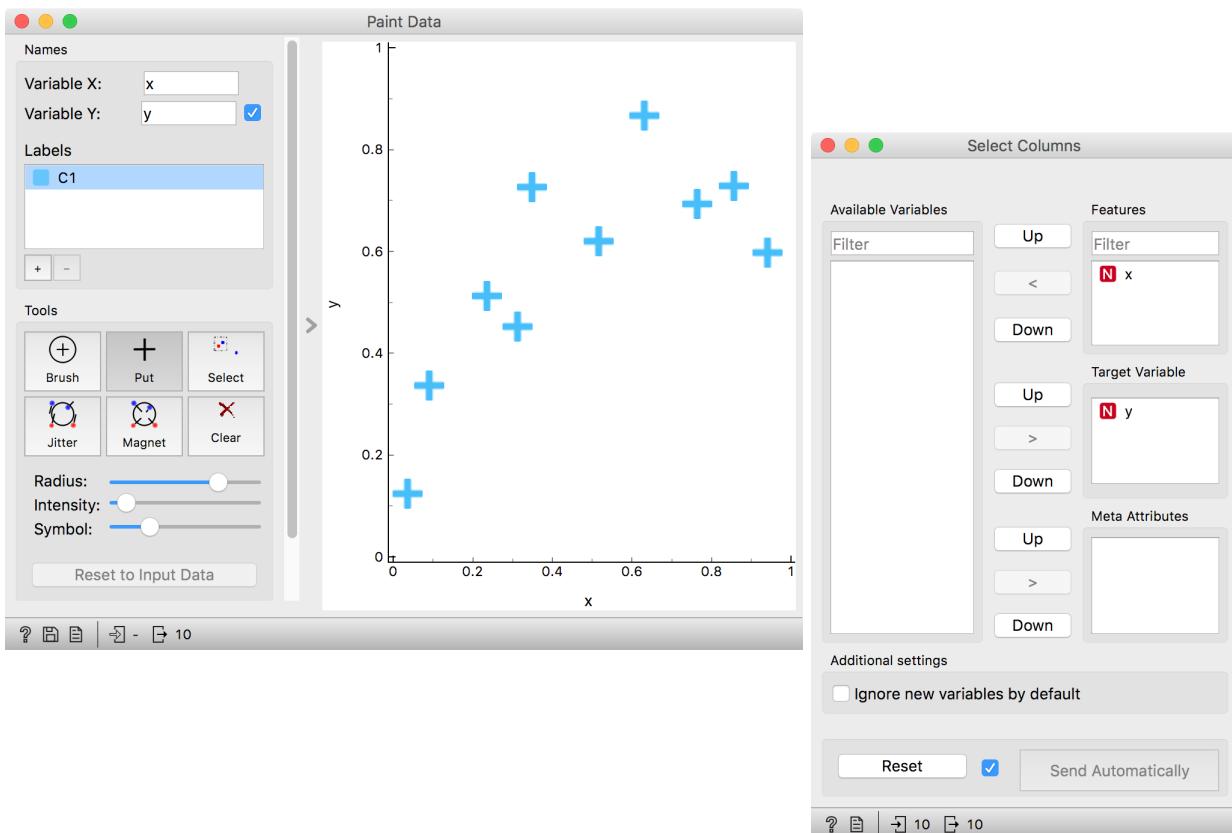
Linear Regression

In the *Paint Data* widget, remove the C2 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 just one continuous input feature (let's call it x) and a continuous class (let's call it y). We will use *Paint Data*, and then reassign one of the features to be a class using *Select Columns* and moving the feature y from "Features" to "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 use Put instead of the Brush tool.

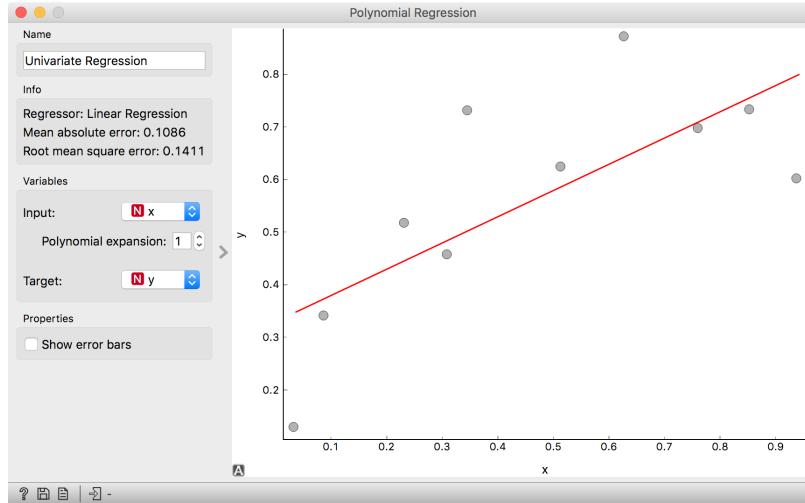
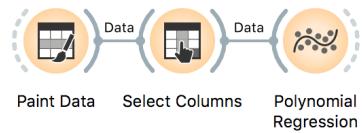
We want to build a model that predicts the value of the target variable y from the feature x . Say that we would like our model to be linear, to mathematically express it as $h(x) = \theta_0 + \theta_1x$. Oh, this is the equation of a line. So we would like to draw a line through our data points. The 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? Which one is the one that fits our data the most? Are they the same?



The question above requires us to define what a good fit is. 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

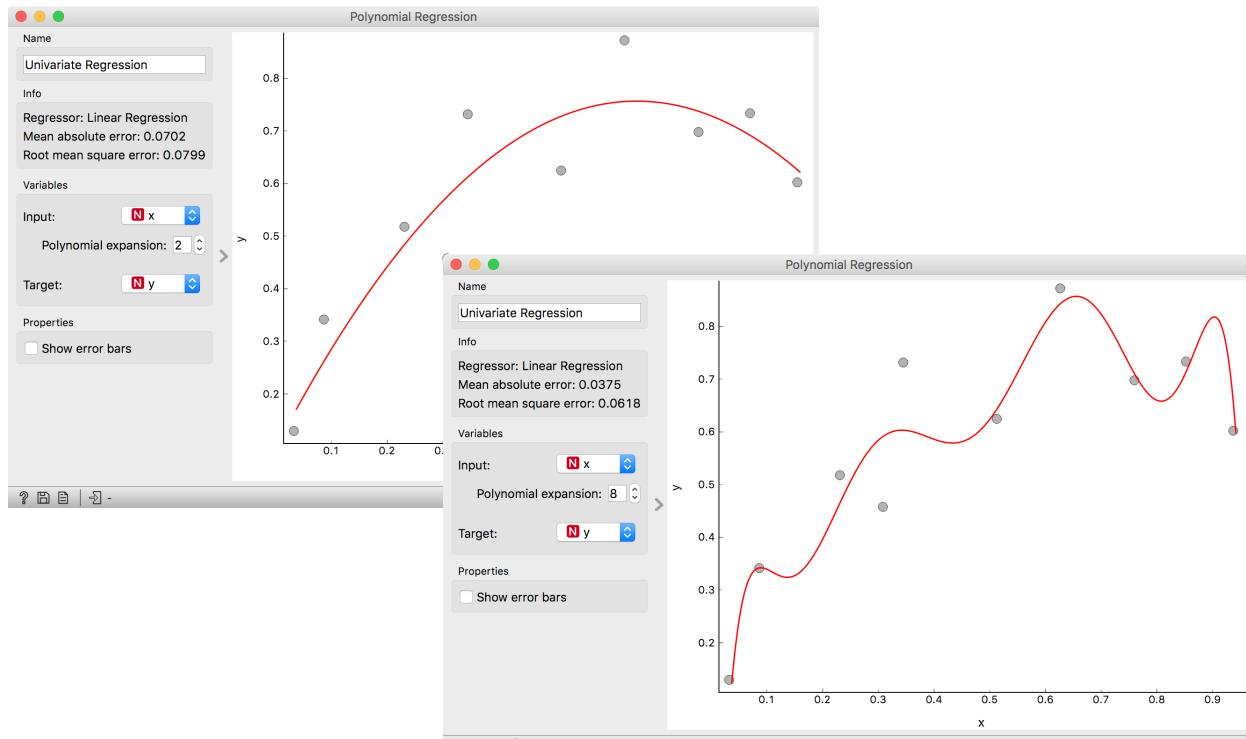
positive errors equally, plus – let us agree – we would prefer punishing larger errors more severely than smaller ones. Therefore, we should square the errors for each data point and sum them up. We got our objective function! It turns out that there is only one line that minimizes this function. The 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 *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 set. Here is a trick: besides the column x , the widget *Polynomial Regression* can add columns x^2, x^3, \dots, 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 2, and then 3, and then, say, to 8. With the degree of 3, we are then fitting the data to a linear function $h(x) = \theta_0 + \theta_1 x + \theta_2 x^2 + \theta_3 x^3$.

The trick we have just performed is polynomial regression, adding higher-order features to the data table and then performing linear regression. Hence the name of the widget. We get something reasonable with polynomials of degree 2 or 3, but then the results get wild. With higher degree polynomials, we overfit our data.

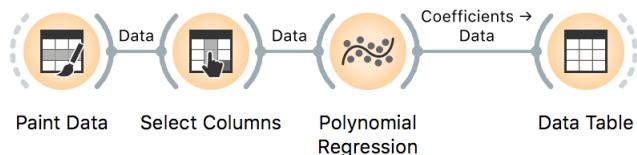


It is quite surprising to see that the linear regression model can fit non-linear (univariate) functions. It can fit the data with curves, such as those on the figures. How is this possible? Notice, though, that the model is a hyperplane (a flat surface) in the space of many features (columns) that are the powers of x . So for the degree 2, $h(x) = \theta_0 + \theta_1x + \theta_2x^2$ is a (flat) hyperplane. The visualization gets curvy only once we plot $h(x)$ as a function of x .

Overfitting is related to the complexity of the model. In polynomial regression, the parameters θ define the model. With the increased number of parameters, the model complexity increases. 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 the polynomial degree. It is easier to overfit the data with a more complex model, as it can better adjust to the data. But is the overfitted model discovering the true data patterns? Which of the two models depicted in the figures above would you trust more?

Regularization

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



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

	name	coef
1	1	0.106121
2	x	1.90152
3	x^2	-1.21305
4	x^3	-0.244903

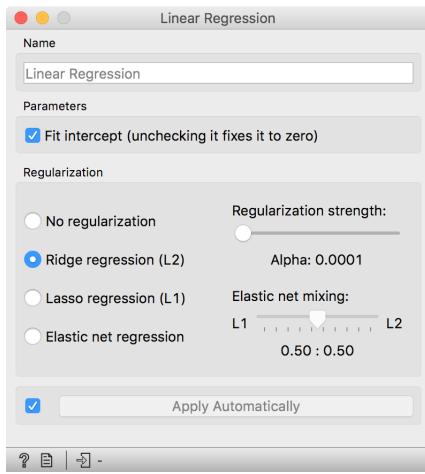
	name	coef
1	1	-0.787028
2	x	40.3077
3	x^2	-553.499
4	x^3	3756.01
5	x^4	-13830.3
6	x^5	29051.4
7	x^6	-34730.1
8	x^7	21961.7
9	x^8	-5696.56

More complex models can fit the training data better. The fitted curve can wiggle sharply. The derivatives of such functions are high, so the coefficients θ need be. 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 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, with some coefficient λ , to control the level of regularization.

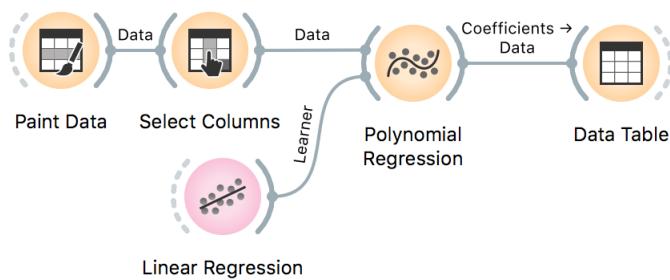
Here we go: we just reinvented regularization, which helps machine learning models not overfit the training data. To observe the effects of regularization, we can give *Polynomial Regression* to our linear regression learner, which supports these settings.

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

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



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 the 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.

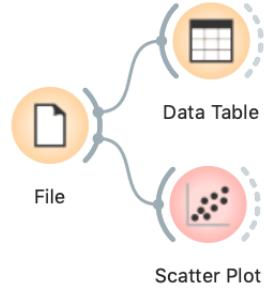


Now for the test. Increase the degree of polynomial to the max. Use Ridge Regression. Does the inferred model overfit the data? How does the degree of overfitting depend on regularization strength?

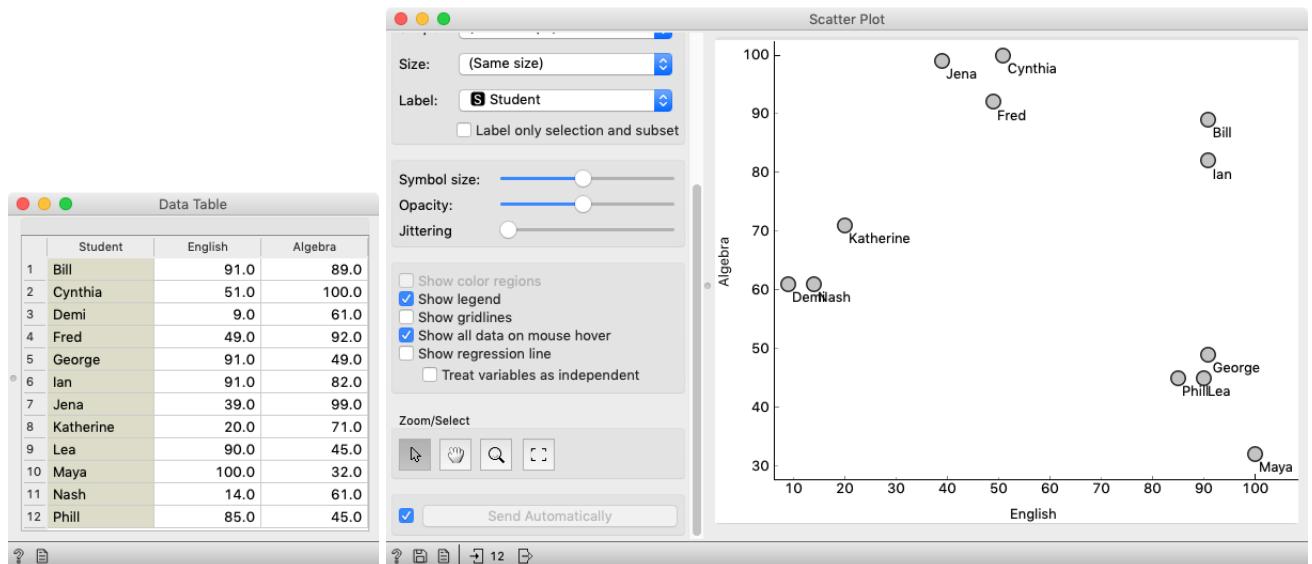
Hierarchical Clustering

WE ARE INTERESTED IN FINDING CLUSTERS IN OUR DATA. We want to identify groups of data instances 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 should belong to the same cluster? How does the clustering algorithm work?

First, we need to define what we mean by "similar". We will assume that all our data instances are described (profiled) with continuous features. One simple measure of similarity is the Euclidean distance. So, we would like to group data instances with small Euclidean distances.

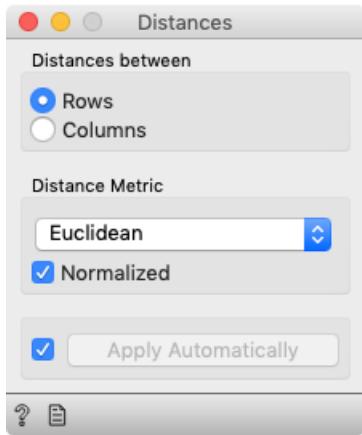


We will introduce clustering with a simple data set on students and their grades in English and Algebra. Load the data set from <http://file.biolab.si/text/grades.tab>.

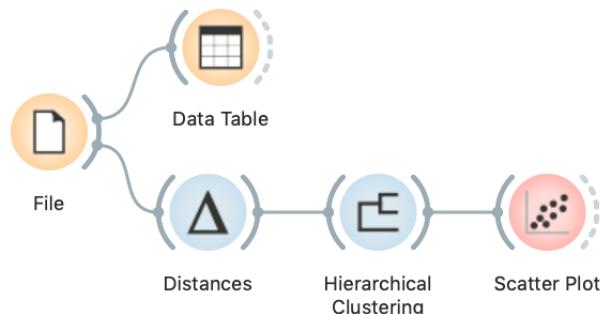


Next, we need to define a clustering algorithm. Say that we start with each data instance being its cluster, and then, at each step, we join the closest clusters. We estimate the distance between the clusters with the average distance between all their pairs of data points. This algorithm is called hierarchical clustering.

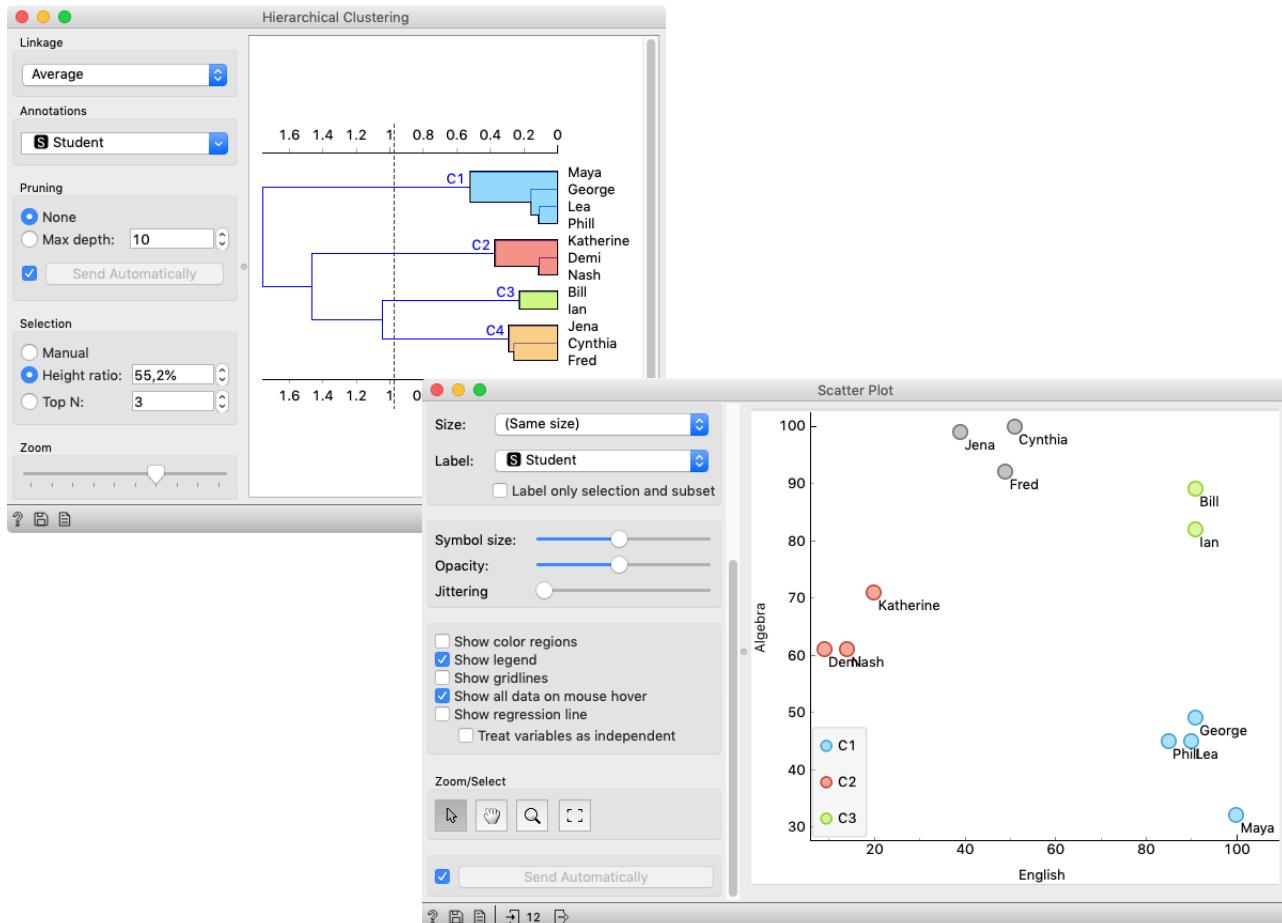
There are different ways to measure the similarity between clusters. The estimate we have described is called average linkage. We could also estimate the distance through the two closest points in each group (single linkage) or through the two points that are furthest away (complete linkage).



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



It couldn't be simpler. Load the data, measure the distances, use them in hierarchical clustering, and visualize the results in a scatter plot. The *Hierarchical Clustering* widget allows us to cut the hierarchy at a specific distance score and output the corresponding clusters:



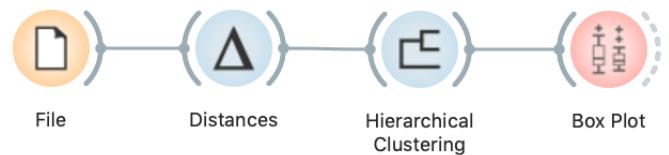
Animal Kingdom

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

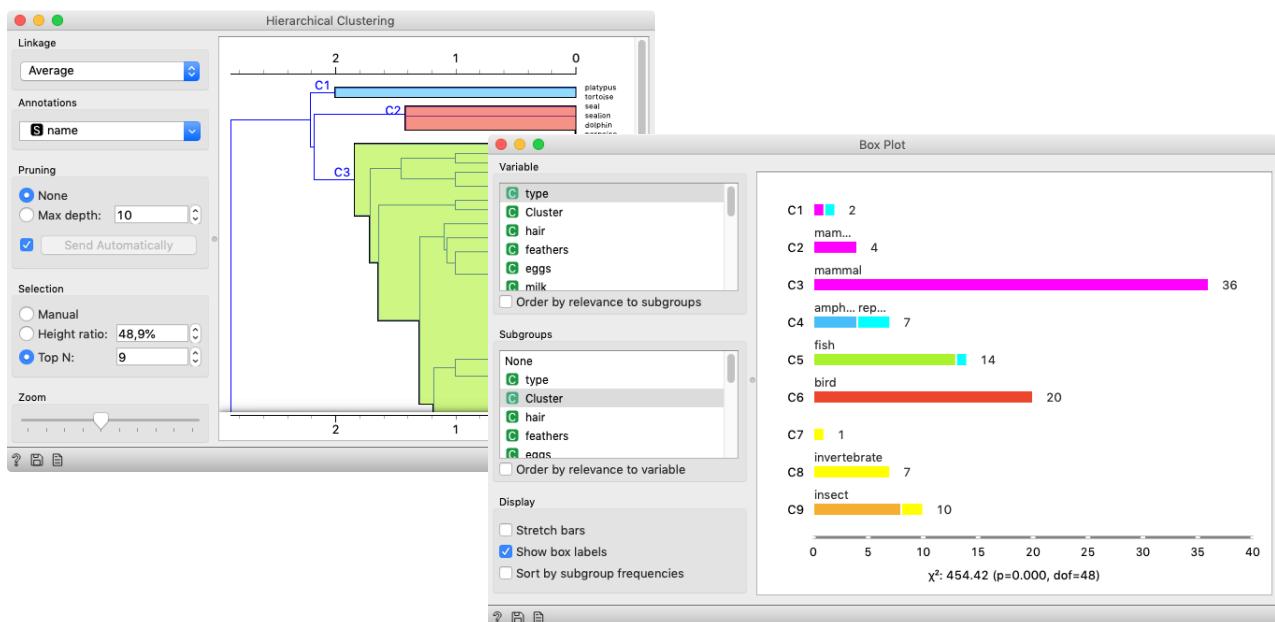
Funny stuff was we never understood the order in which the cards were laid out in the album. We later learned about taxonomy, but being more inclined to engineering we never mastered learning it in our biology classes. Luckily, there's data mining and the idea that taxonomy simply stems from measuring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

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

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



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



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

k-Means Clustering

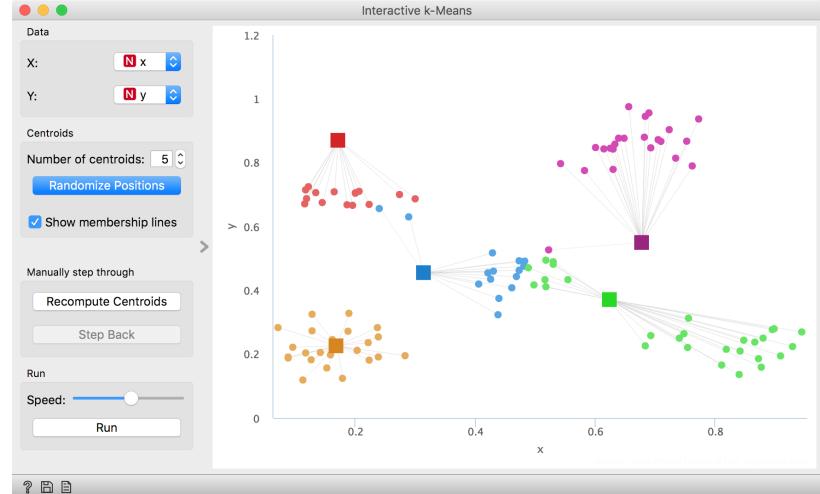
HIERARCHICAL CLUSTERING IS NOT SUITABLE FOR LARGER DATA SETS due to the prohibitive size of the distance matrix: with 30 thousand objects, the distance matrix already has almost one billion elements. An alternative approach that avoids using the distance matrix is k-means clustering.

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

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

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

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



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

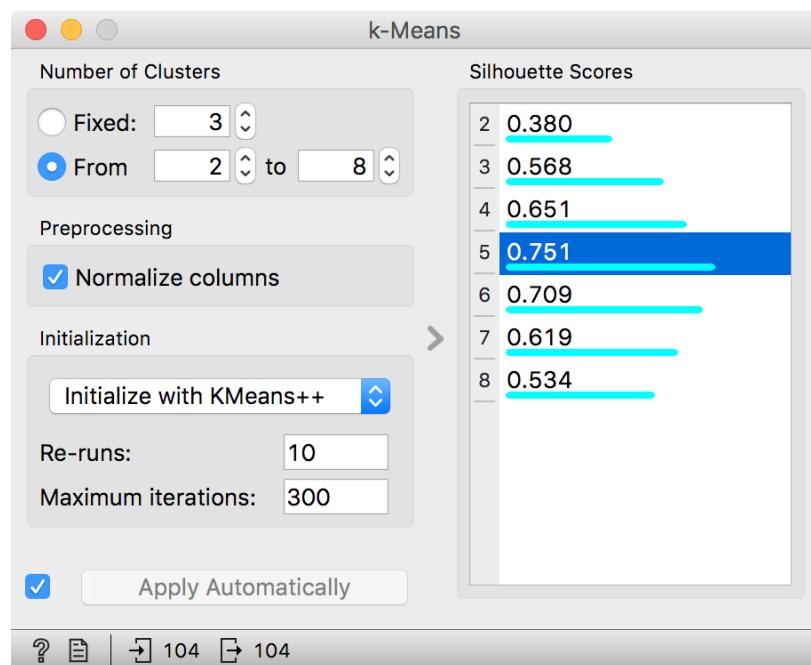
How do we set the initial number of clusters? That's simple: we choose the number that gives the optimal clustering.

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

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

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

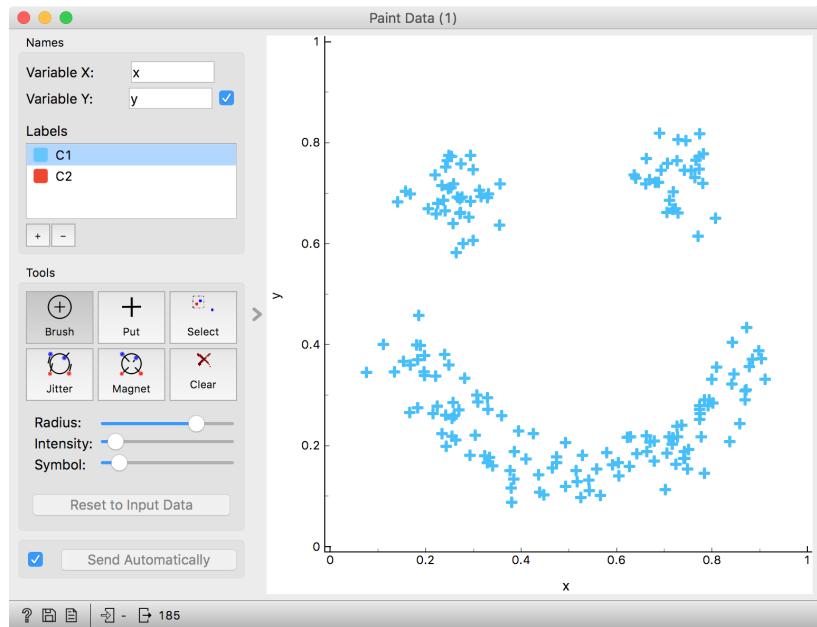
For this, we abandon our educational toy and connect Paint Data to the widget k-Means. We tell it to find the optimal number of clusters between 2 and 8, as scored by the Silhouette.



Works like a charm.

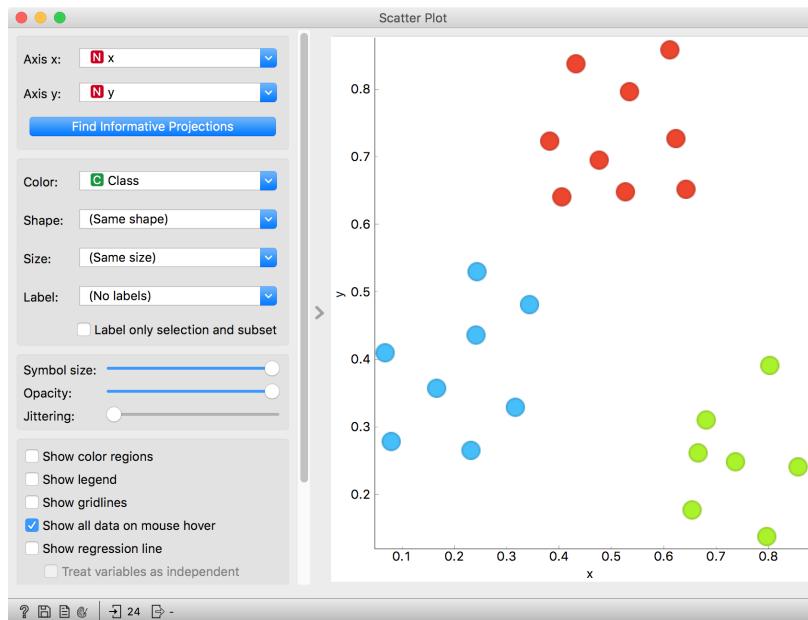
Except that it often doesn't. First, the result of k-means clustering depends on the initial selection of centers. With unfortunate selection, it may get stuck in a local optimum. We solve this by re-running the clustering multiple times from random positions and using the best result. Second, the silhouette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

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

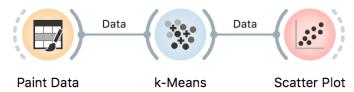


Silhouettes

CONSIDER A TWO-FEATURE DATA SET which we have painted in the *Paint Data* widget. We send it to the k-means clustering, tell it to find three clusters, and display the clustering in the scatter plot.



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

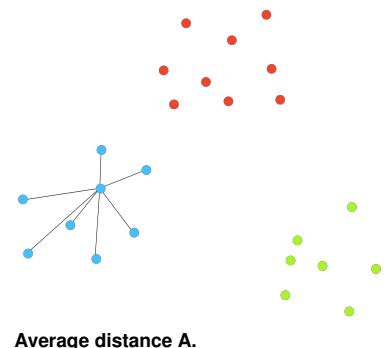


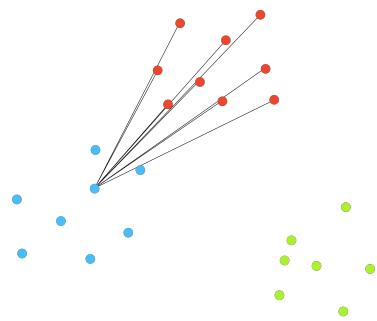
The data points in the green cluster are well separated from those in the other two. Not so for the blue and red points, where several points are on the border between the clusters. We would like to quantify the degree of how well a data point belongs to the cluster to which it is assigned.

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

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

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



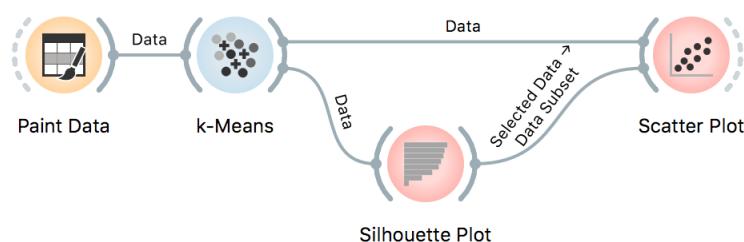
**Average distance B.**

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

The larger the B, the better.

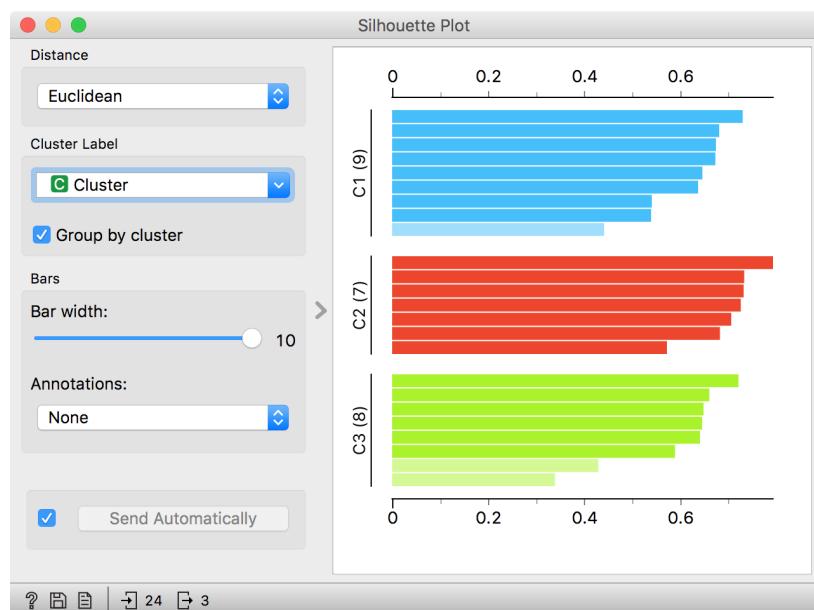
The point is well rooted within its own cluster if the distance to the points from the neighboring cluster (B) is much larger than the distance to the points from its own cluster (A), hence we compute B-A. We normalize it by dividing it with the larger of these two numbers, $S = (B - A) / \max(A, B)$.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

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



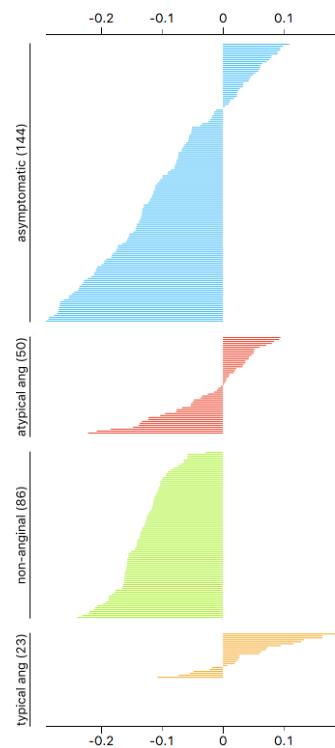
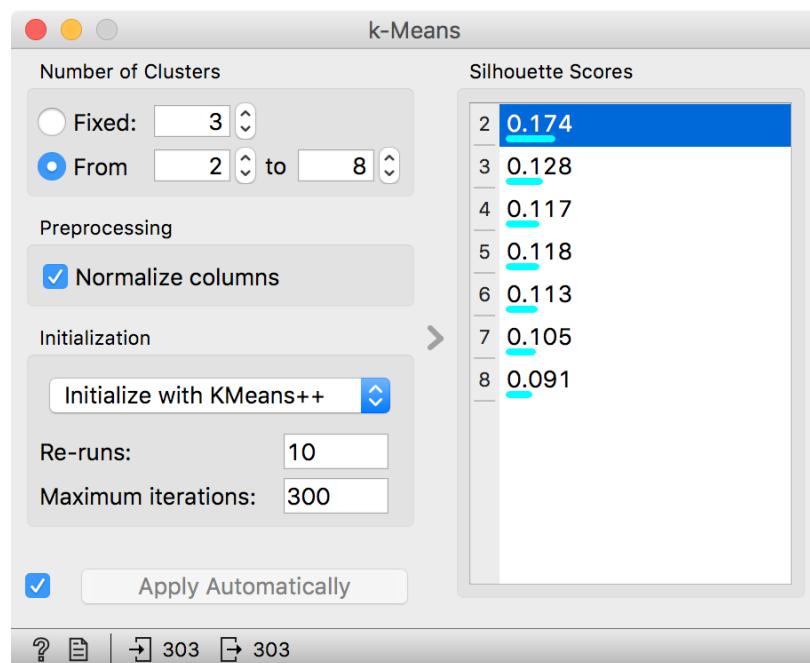
We selected three data instances with the worst silhouette scores. Can you guess where they lie in the scatter plot?

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



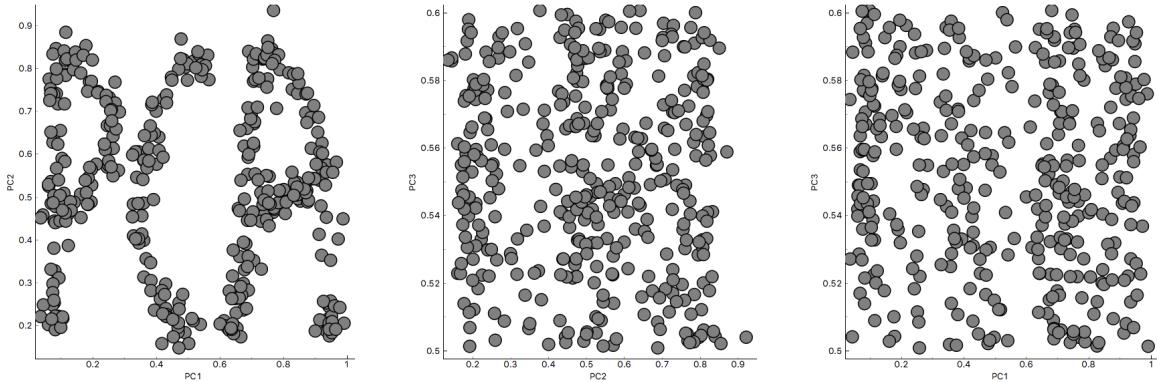
The total quality of clustering - the silhouette of the clustering - is the average silhouette across all points. When the *k-Means* widget searches for the optimal number of clusters, it tries a different number of clusters and displays the corresponding silhouette scores. Ah, one more thing: Silhouette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

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



Principal Component Analysis

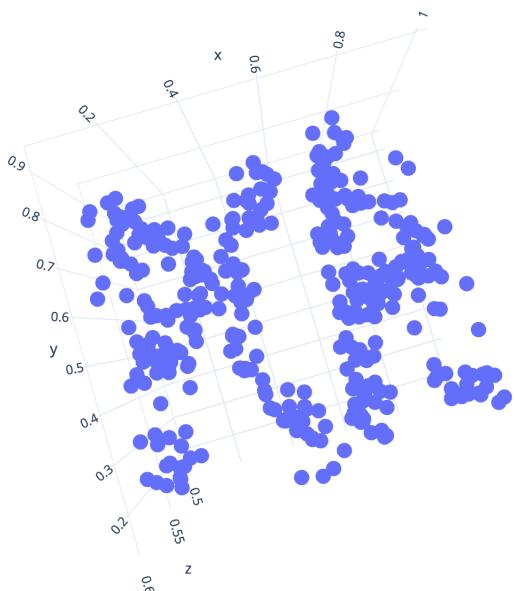
Which of the following three scatter plots (showing x vs. y, x vs. z and y vs. z) for the same three-dimensional data gives us the best picture about the actual layout of the data in space?



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 is 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, essentially keeping just 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 see 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 axis; it may be a 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 are as spread as possible.

How do we do this? Let's go back to our everyday'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 the position of each fly is described by three numbers. Then you discover that your flies actually 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 position 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.

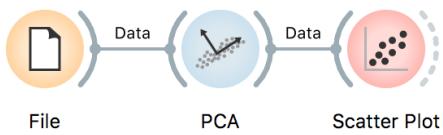
After some careful observation, you notice the flies are a bit spread in one other direction, so they do not fly along a line but along a band. Therefore, we need two numbers, one along the first and one along the — you guessed it — second principal component.

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

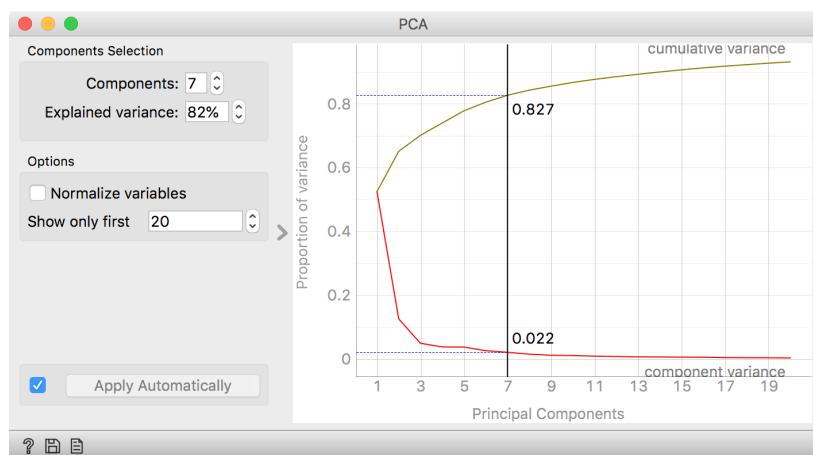
Or do you? It all depends on how spread they are in the second and in the third direction. If the spread along the second is relatively small in comparison with the first, you are fine with a single dimension. If not, you need two, but perhaps still not three.

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.

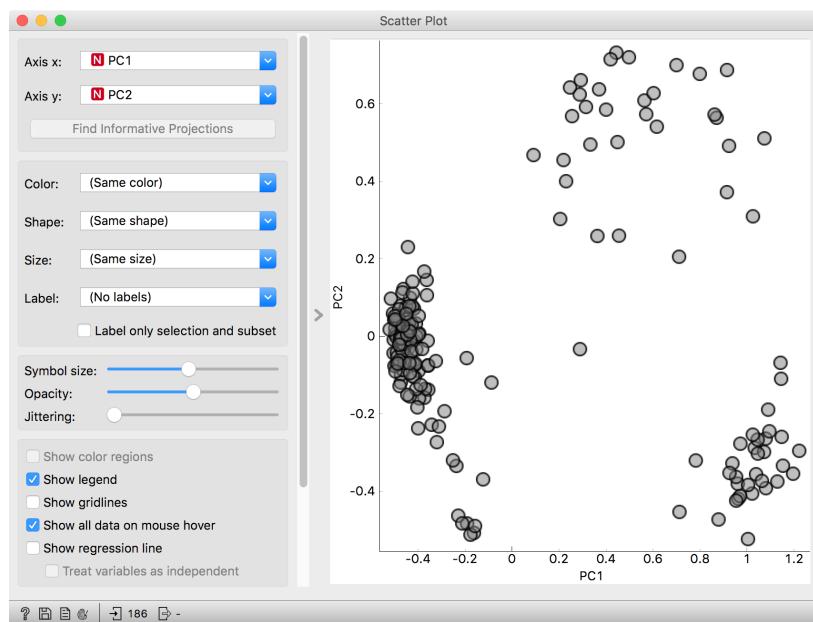
A technique that does this is called Principle Components Analysis, or *PCA*. The corresponding widget is simple: it receives the data and outputs the transformed data.



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.

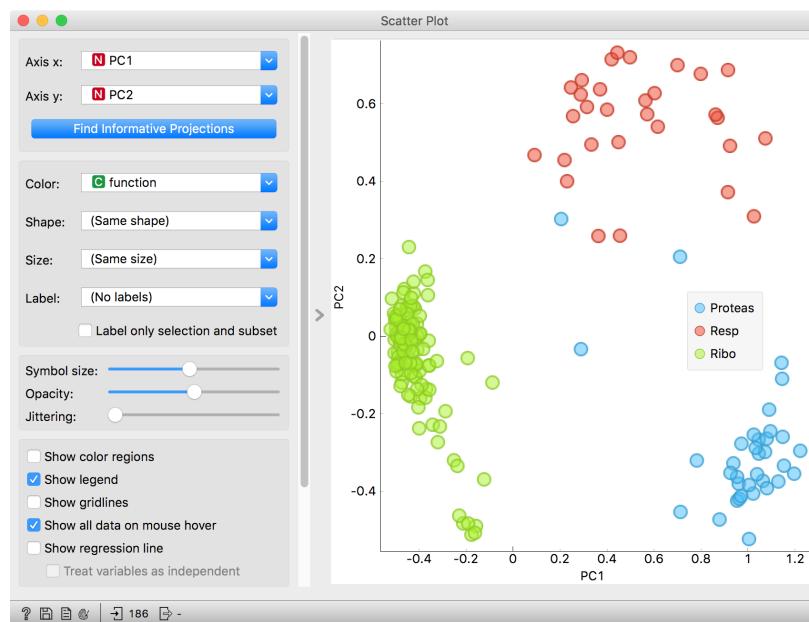


The PCA on the left shows the scree diagram for brown-selected data. Set like this, the widget replaces the 80 features with just seven - and still keeping 82.7% of information. (Note: disable "Normalize data" checkbox to get the same picture.) Let us see a scatter plot for the first two components.

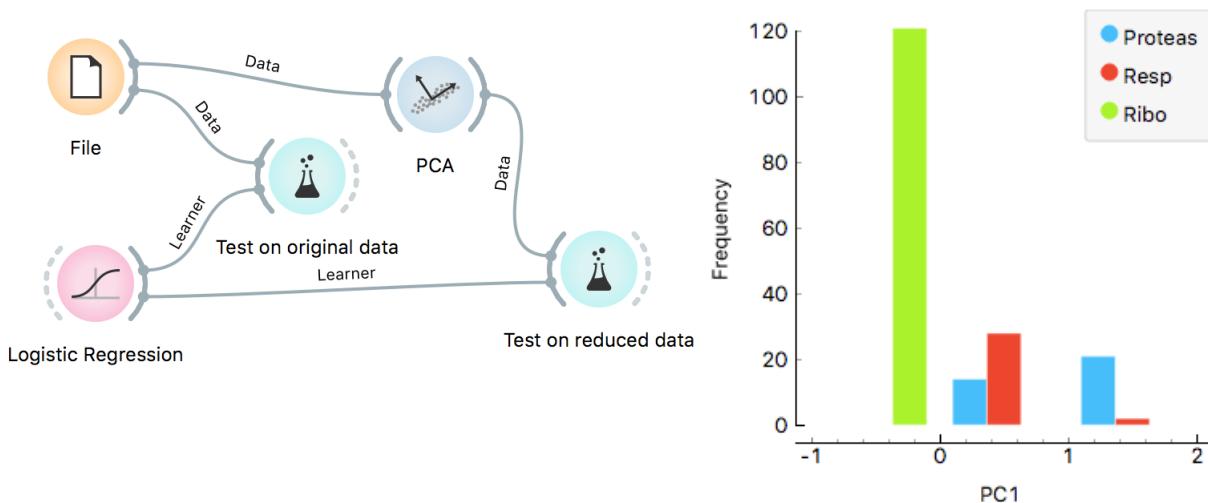


The axes, PC1 and PC2, do not correspond to particular features in the original data, but to their linear combination. What we are looking at is a projection onto the plane, defined by the first two components. When you consider only two components, you can imagine that PCA puts a hyperplane into multidimensional space and projects 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 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 uses the ordinary Test and Score widget, but renamed it to "Test on original data" for better understanding of the workflow.

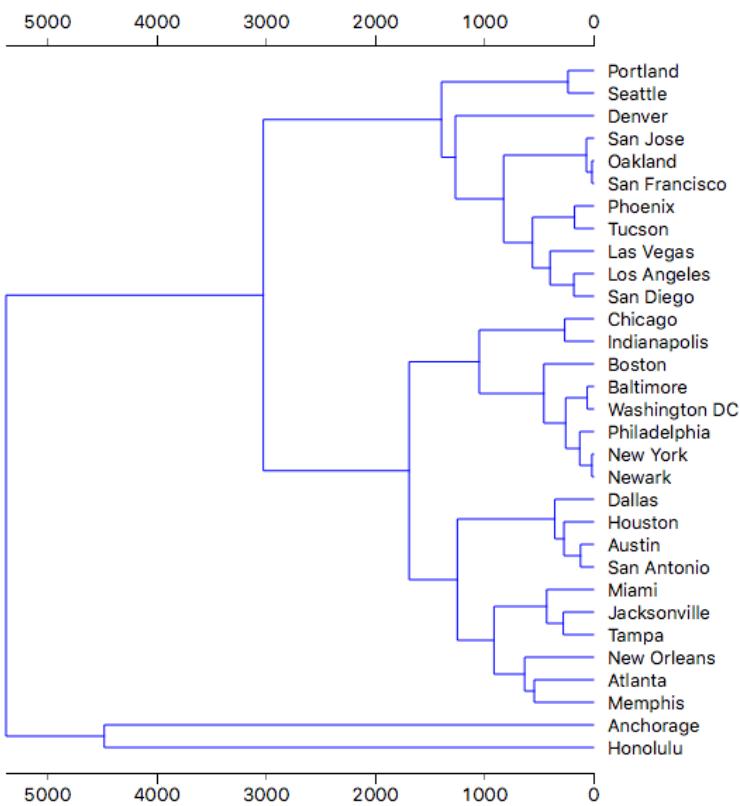
On the original data, logistic regression gets 98% AUC and classification accuracy. If we select just a single component in PCA, we already get a 93%, and if we take two, we get the same result as on the original data.

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 information. 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.

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 the 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.

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 leftovers; 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.

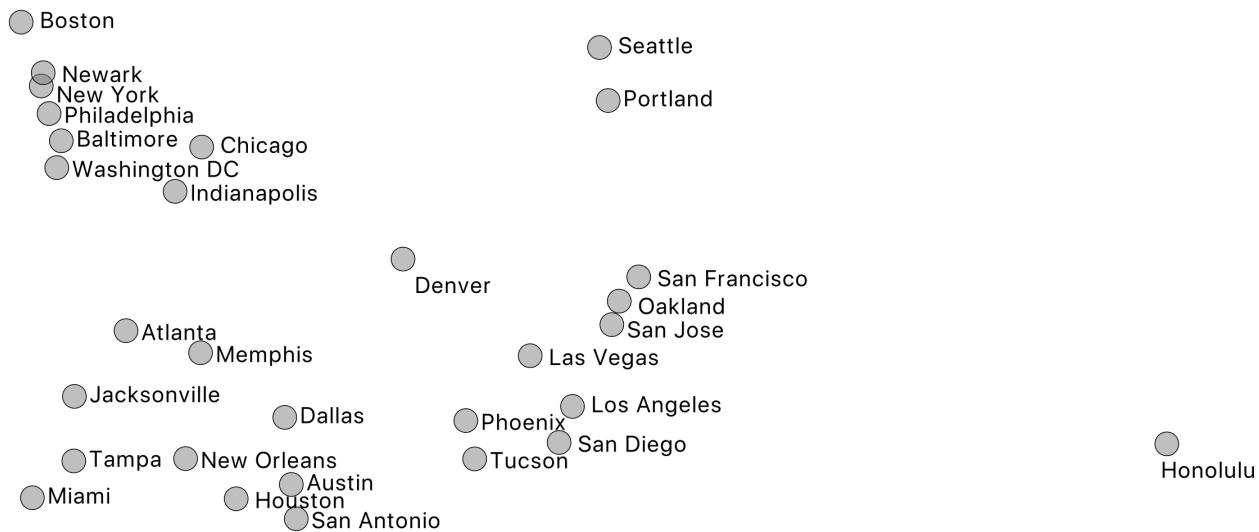
The real problem is New Orleans and San Antonio: New Orleans is close to Atlanta and Memphis, Miami is close to Jacksonville and

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.

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

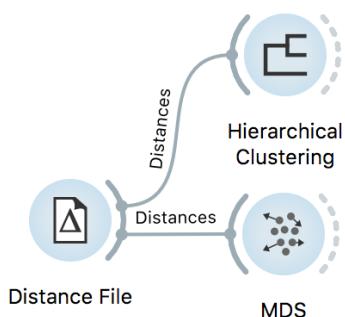
In general, two points from different clusters may be more similar to each other than to some points from their corresponding clusters.

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 a paper with the 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.

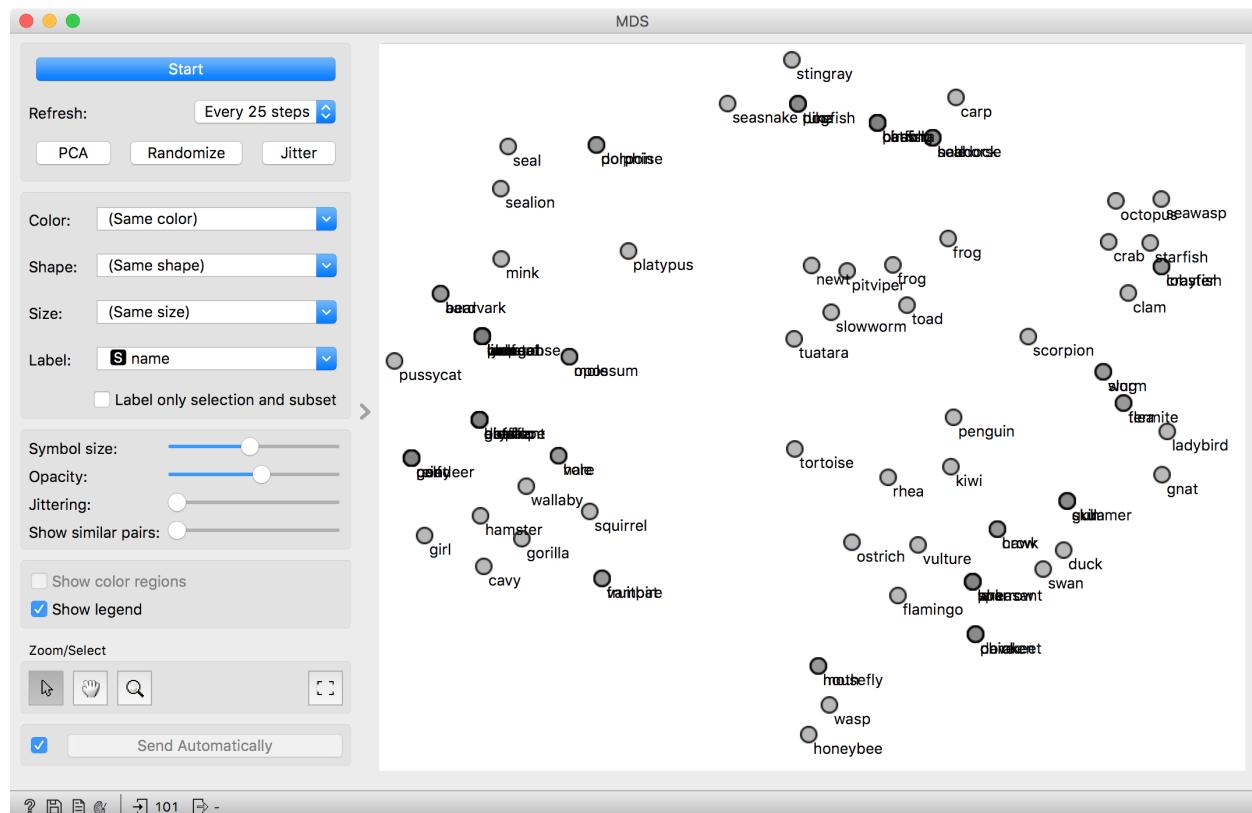


We have not constructed this map manually, of course. We used a widget called *MDS*, which stands for Multidimensional scaling.

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.



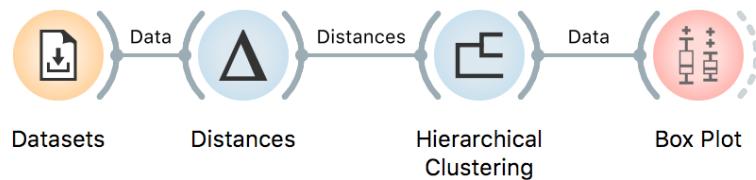
The map of the US was accurate: one can put the points in a plane so that the distances correspond to actual distances between cities. For most data, this is usually impossible. What we get is a projection (a non-linear projection, if you care about mathematical finesse) of the data. You lose something,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

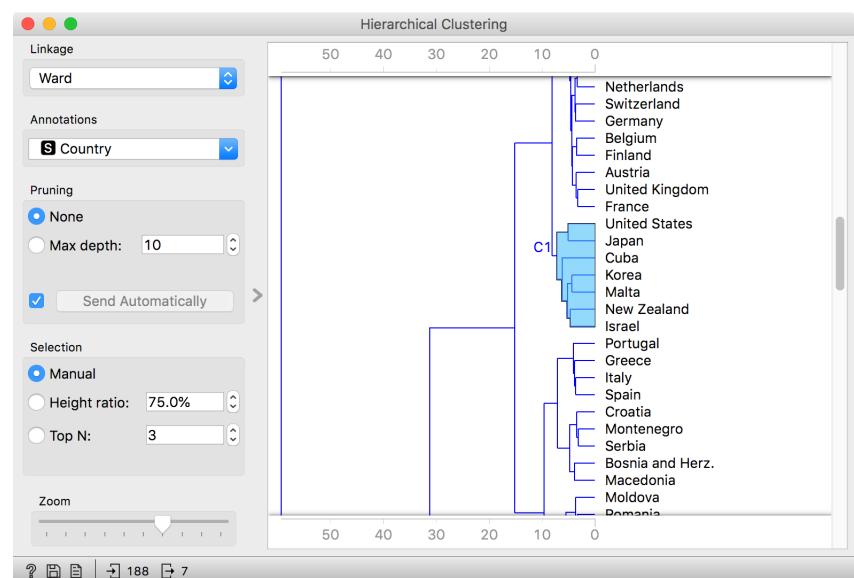
Assignment: Clustering

CLUSTERING HELPS TO DISCOVER GROUPS OF DATA, for example similar countries based on certain socio-economic features. For this task, we will use HDI data set from the *Datasets* widget. The data reports the Human Development Index for the year 2016 for 188 countries. While HDI has its limitations, the data offers an interesting exercise in clustering.

Workflow for the assignment.



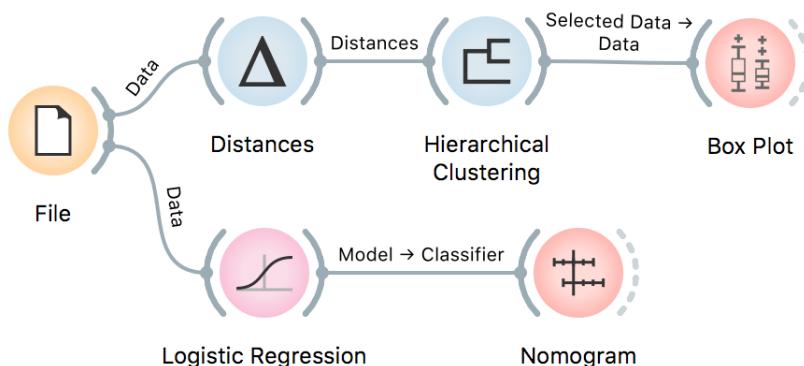
1. Try Euclidean and cosine distance with Ward linkage. Which one works better? Why?
2. How many groups did you discover? What number would make sense?
3. Explain the final clusters. What defines each of them?
4. Use Euclidean distance and Ward linkage. Can you explain why is Cuba clustered together with South Korea and the United States? Use *Box Plot* and the Data output from Hierarchical Clustering to answer this question.



Assignment:

Clustering vs. Classification

CLUSTERING AND CLASSIFICATION ARE FUNDAMENTALLY DIFFERENT TASKS. The former tries to find similar data instances and puts them into groups or clusters. The latter looks for the patterns in the data and infers correlations between the features and the target variable. Clustering is a part of unsupervised learning and doesn't require a target variable. Classification is a part of supervised learning and requires a target variable. A target variable is the thing we want to predict. With clustering, we are not predicting anything, but trying to find groups of similar data instances.



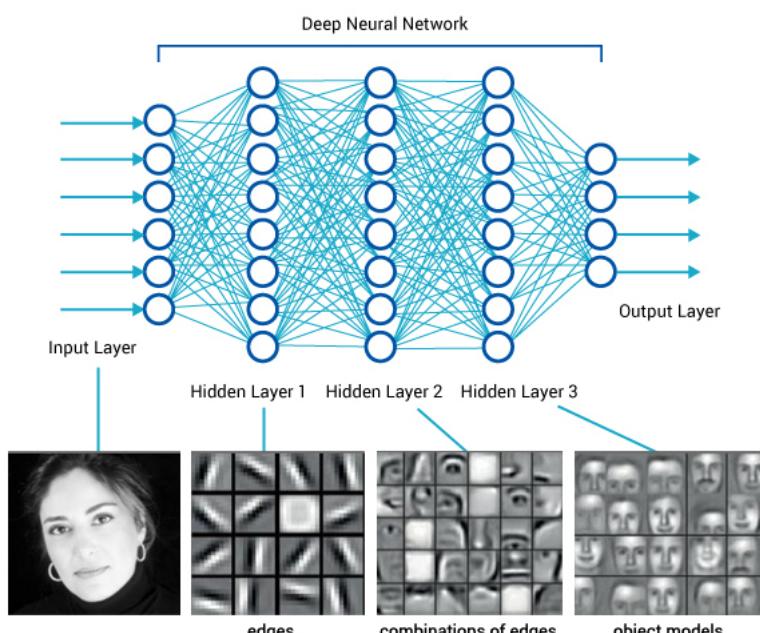
For this task, use *heart-disease* data and build a simple hierarchical clustering workflow (with Euclidean distance and Ward linkage) and a logistic regression model and inspect it in a nomogram. In clustering, cut at two clusters. Refer to the workflow above and answer the following questions:

1. Explore the groups with *Box Plot*. What is the characteristic of cluster 1 (C_1)? What is the characteristic of cluster 2 (C_2)?
2. Which are the top three attributes distinguishing between clusters? Write them down.
3. In *Nomogram*, which are the three most important attributes for the model?
4. Are the attributes from clustering the same as those from logistic regression? Why (not)?
5. *Major vessels colored* is the most important attribute for logistic regression. How well does it split between the clusters?

Image Embedding

This depiction of deep learning network was borrowed from <http://www.amax.com/blog/?p=804>

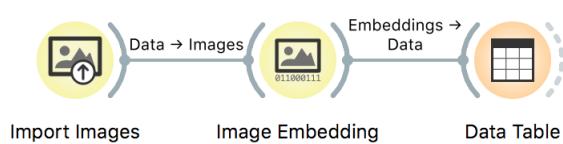
Every data set so far came in the matrix (tabular) form: objects (say, tissue samples, students, flowers) were described by row vectors representing a number of features. Not all the data is like this; think about collections of text articles, nucleotide sequences, voice recordings or images. It would 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.



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 *Image Embedding* widget.

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

The screenshot shows a Data Table window with the following details:

- Info:** 19 instances (no missing data), 2048 features, No target variable, 5 meta attributes.
- Variables:**
 - Show variable labels (if present)
 - Visualize numeric values
 - Color by instance classes
- Selection:** Select full rows
- Buttons:** Restore Original Order, Send Automatically.

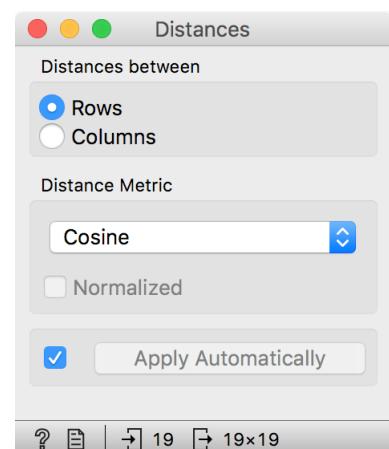
The main table has columns: hidden origin type, image name, image, size, width, height, n0 True, n1 True, n2 True, n3 True. The data includes rows for duck, dog, horse, rabbit, ox, turkey, sheep, cow, calf, hen, foal, cat, and goose, each with its corresponding image file path and numerical features.

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.

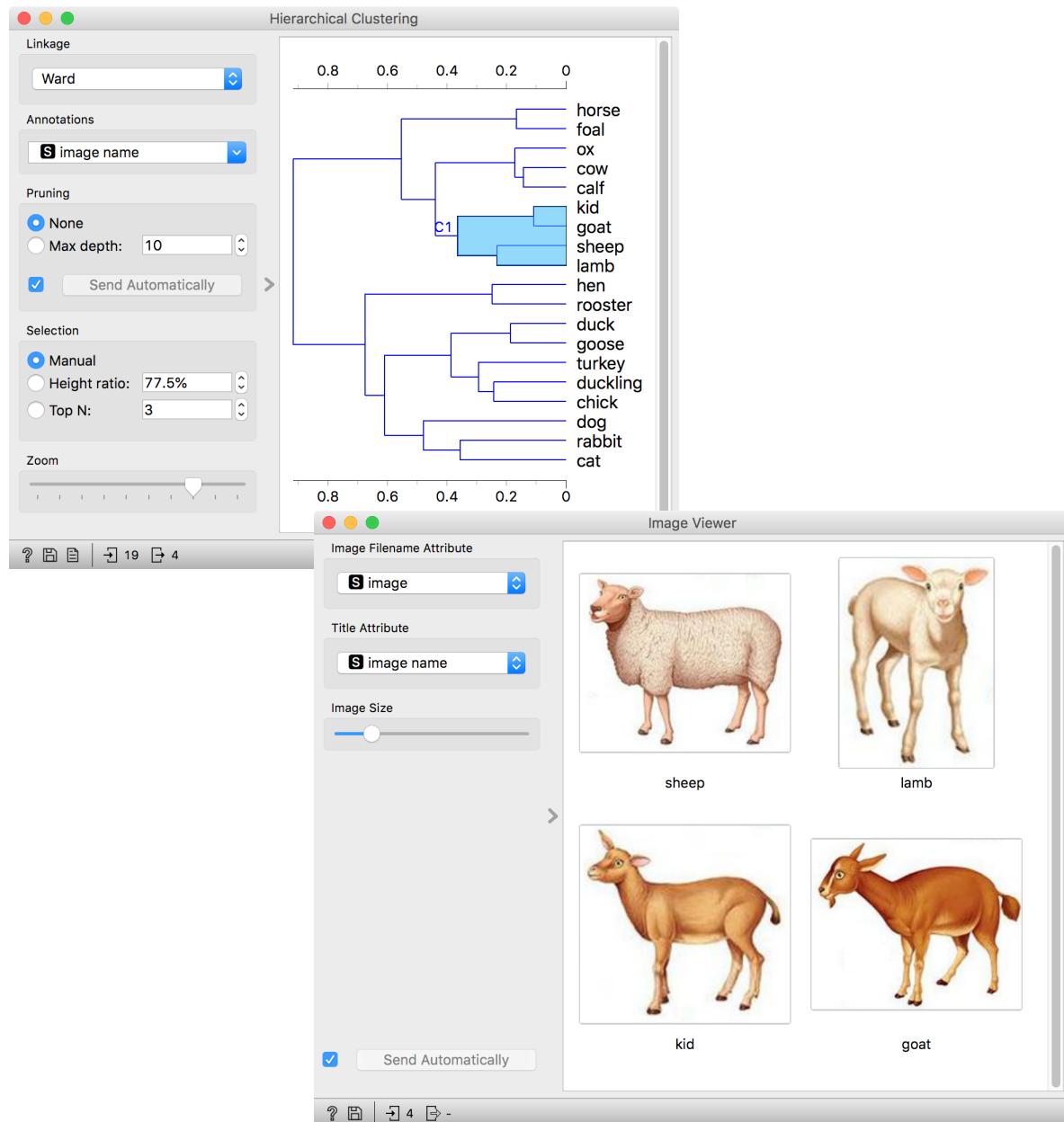
Images are available at <http://file.biolab.si/images/domestic-animals.zip>



To recap: in the workflow above 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 the dendrogram looked better than with the Euclidean distance).



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

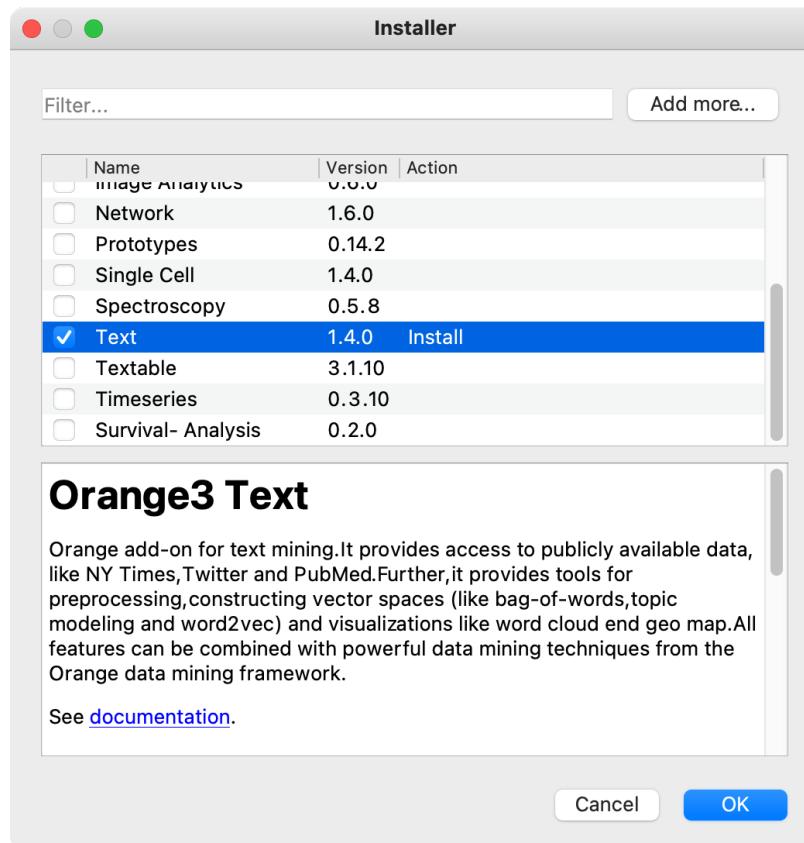


Text Mining: Introduction

TEXT MINING IS A COMPUTATIONAL ANALYSIS OF TEXTS. It uses statistics, natural language processing, and machine learning to extract information from documents.

Installing the Text add-on

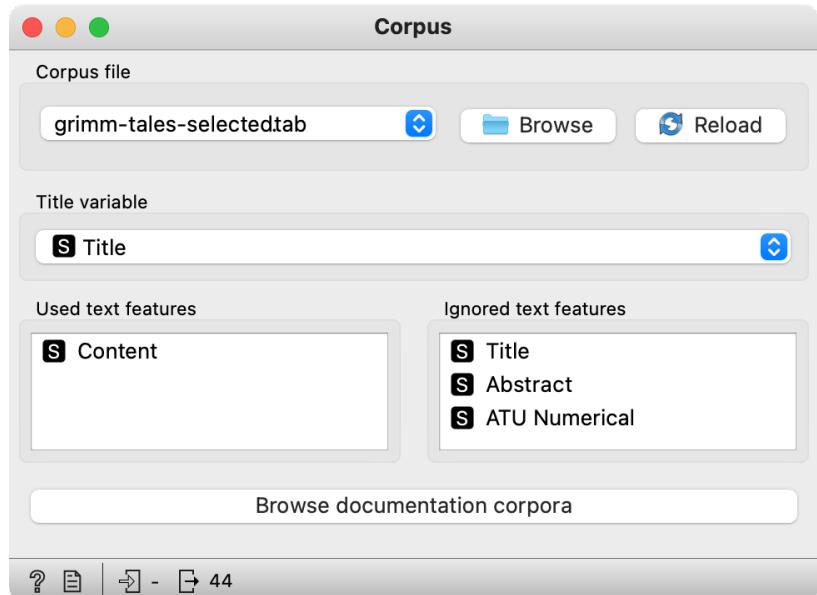
You will need the *Text add-on*, which introduces components for text preprocessing, analytics, visualization and deep-learning-based embeddings to Orange. To install Text add-on, go to Options → Add-ons and select Text from the list. You will have to restart Orange for Text widgets to appear.



A new pane with widgets from the Text add-on will appear on the left side of the canvas.

Loading corpora

A collection of text documents is called a corpus. Widget for loading corpora is called *Corpus*. We will use the *grimm-tales-selected* corpus, which you can select from the drop-down menu in the widget. The corpus contains 44 folk tales, collected by the Grimm brothers.



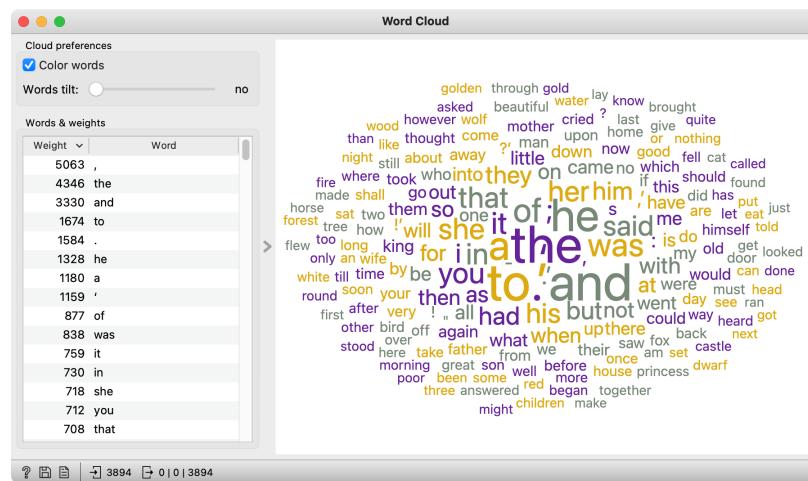
The widget is very similar to the *File* widget with one important distinction. Orange needs to know which attribute contains the content of the documents. The Grimm corpus has the attribute *Content*, which is already placed in the *Used text features* section. Alternatively, drag one or more attributes you would like to use for text mining to the box on the left.



Corpus

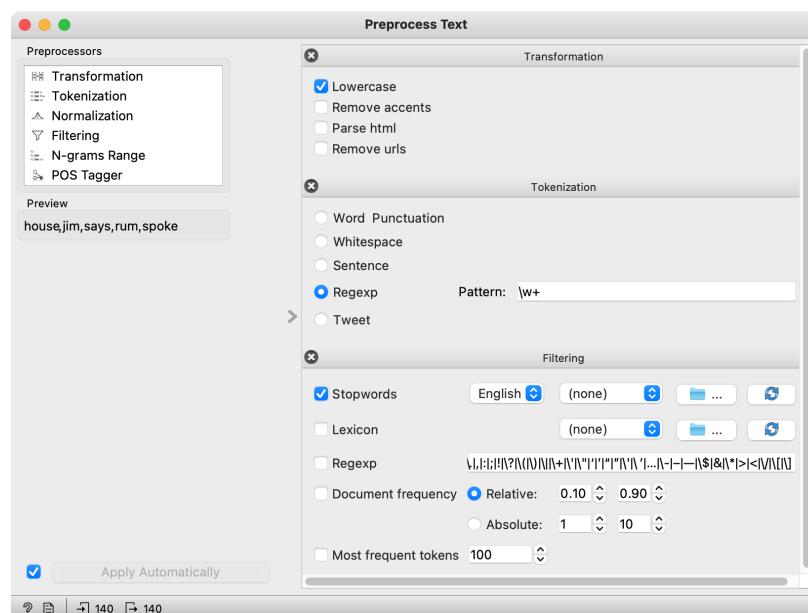
Preprocessing Text

Connect *Word Cloud* to *Corpus*. The visualization shows words, with their size corresponding to their frequency in the corpus. In this case, *Word Cloud* simply displays all the words and symbols found in the text. But this is often not what we want. We want to extract only meaningful units, such as semantically rich words. This is why we need text preprocessing.



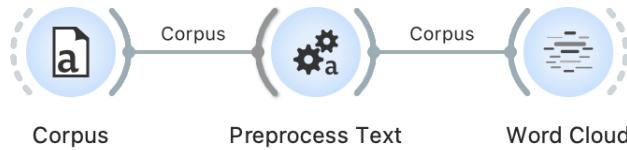
But the word cloud we are looking at is a mess! We got a bunch of semantic junk in our visualization. Is there a way to clean this up?

Of course. One of the most important tasks in text mining is a correct preprocessing. We can achieve this with the *Preprocess Text* widget.



Preprocessing is executed sequentially. We start by lowercasing

the text. This means words will be treated the same regardless of whether they appear at the beginning or middle of the sentence. However, words such as "apple" (a fruit) and "Apple" (a company) will also be treated the same, which is not always desirable.



We see the results of our preprocessing in the Word Cloud. Two of the most frequent words are "would" and "could". If we decide these two words are not important for our analysis, it would be good to omit them. We can do this with custom filtering. filtered out some stopwords. But perhaps filtering out generic stopwords is not enough for our analysis.

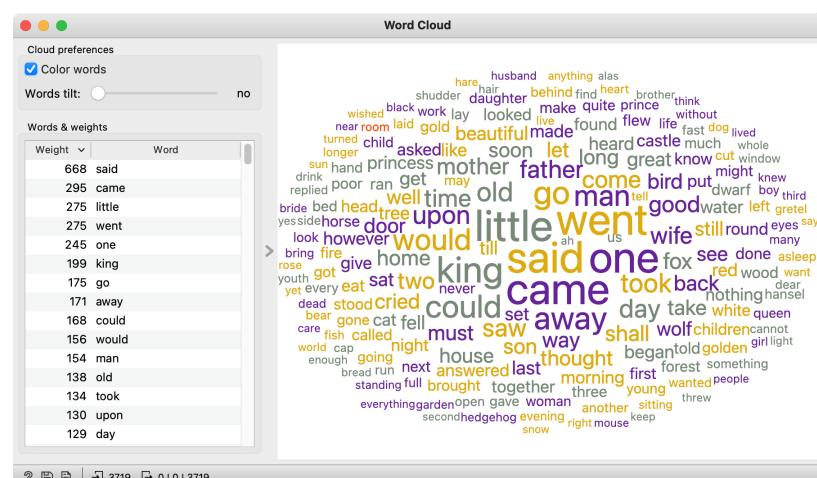
Next, the text is split into *tokens*, which are the core units of analysis. They are usually words, but they can also be sentences, bigrams, and so on. The default option, *Regexp* keeps only words, omitting the punctuation.

We also removed redundant words.

As we saw in the word cloud above, the most frequent words in English texts are "the", "and", "of", and so on. While these words are important for syntax, they do not carry any meaning, so they are often omitted from the analysis.

To sum up, we transformed all words to lowercase, treated each word as a token (and omit punctuation), and removed the stopwords (such as "in", "and", and "the"). This preprocessing outputs the following tokens:

"This is a sample sentence." → "sample", "sentence"

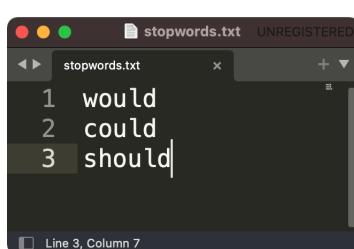


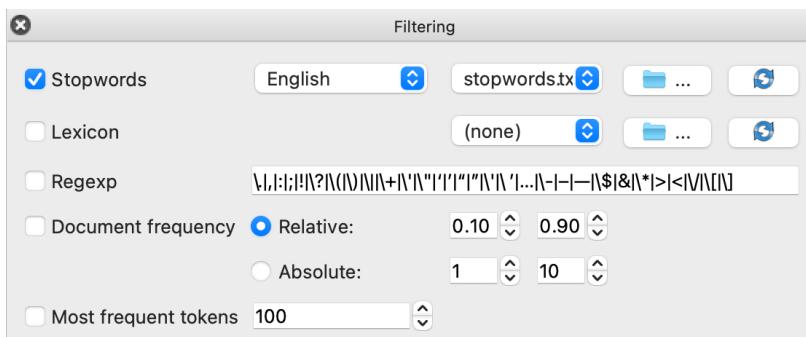
A good plain text editor is Sublime, but you can easily work with Notepad++.

We can always load our own custom stopword list. Open a plain text editor and create a custom list of stopwords. Write each new word on its own line and save the file.

Load the list of custom stopwords in the right-hand dropdown of the Filtering section.

Another preprocessing technique is to filter out words that are too rare and too frequent. Rare words are normally found in only a few documents and frequent words are likely stopwords or very general words. To retain only those words that truly represent the corpus and may distinguish between corpus documents, we use Document frequency filter with Relative frequencies. If we set the values to 0.1 and 0.9, we will retain only those words that appear in more than 10% of the documents and in fewer than 90%.





Preprocessing is really the key to a successful text analysis. We have only mentioned a few techniques, but you can experiment on your own with the following ones:

1. normalization transforms all words into lemmas or stems (for example *sons* to *son*)
2. n-grams are tokens of larger size, bigrams (a pair of consecutive words) and trigrams (word triplets)
3. POS tagging tags each token with a corresponding part-of-speech tag (*sons* → noun, plural, tag = NNS)

Title	Content
A Tale About the Boy ...	certain_JJ father_RB two_CD sons_NNS
Brier Rose	elder_VBP smart_JJ sensible_JJ
Cat and Mouse in ...	everything_NN younger_JJR stupid_JJ
Cinderella	neither_CC learn_JJ understand_NN
Hansel and Gretel	anything_NN people_NNS saw_VBD
Herr Korbes	said_VBD fellow_JJ give_VBP father_RBR
Jorinda and Jorindel	trouble_NN anything_NN done_VBN
Little Red Riding Hood	always_RB elder_RBR forced_JJ father_NN
Mother Holle	bade_NN fetch_NN anything_NN late_JJ
Old Sultan	night_NN time_NN way_NN led_VBD
Pack of Scoundrels	churchyard_NN dismal_I.I place_NN

Concordances

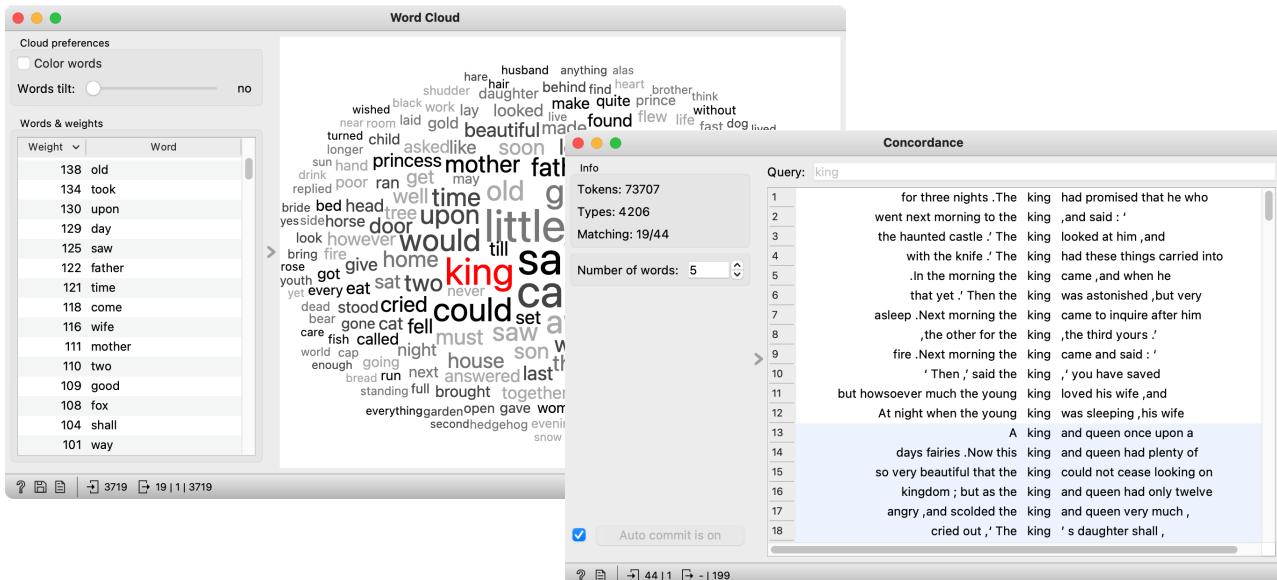
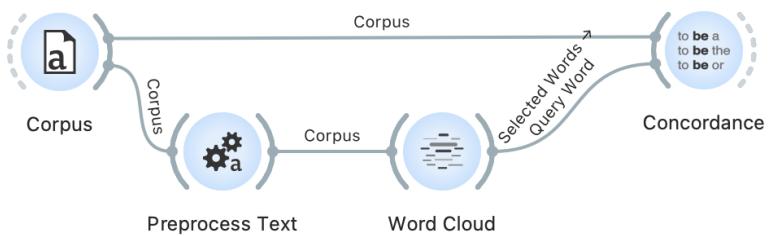
Visualizations in Orange are designed to support selection and passing of the data that applies to it. Finding interesting data subsets and analyzing their similarities is a central part of exploratory data analysis.

To inspect the documents containing a particular word, select the documents in Concordance and pass them to Corpus Viewer for a deeper analysis.

We have prepared our corpus and now it is time to visualize it. We have already seen some of the preprocessing results in a word cloud. But we still don't know much about the use of a specific word in a text. Since we lowercased the text, there might be some conflation. For example 'oh' could be a lowercase version of OH (the chemical compound of hydroxide), a simple exclamation 'Oh!' or an abbreviation for the state of Ohio.

To check the context of a particular word we can use Concordance widget. Concordance shows us the text around our word.

Connect *Concordance* to *Corpus* to pass the text to the widget. To browse the word, type it in the query line at the top or provide it with the *Word Cloud*. Here we have selected the word 'king' in the Word Cloud and observed the context in Concordance.



Bag of Words

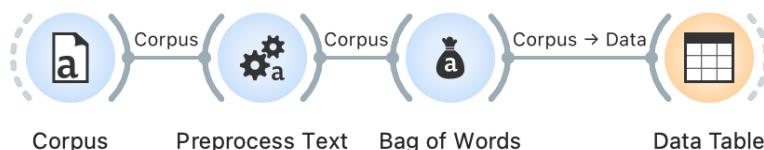
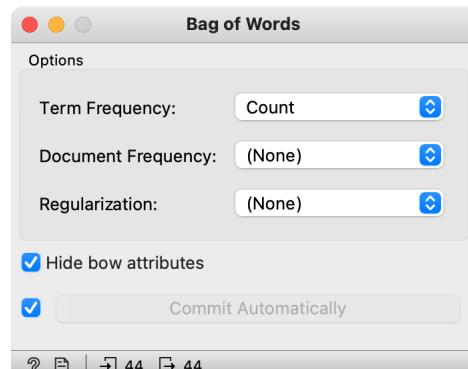
EVERY MACHINE LEARNING NEEDS NUMBERS, BUT ALL WE HAVE IS TEXT. A simple way to convert documents into numeric vectors is to ... well, count the words in each text.

Document	this	is	an	example	another	apple
This is an example	1	1	1	1	0	0
Another example	0	0	0	1	1	0
This is another apple	1	1	0	0	1	1

Bag of Words creates a table with words in columns and documents in rows. Values are word frequencies in each document.

We can simply count the words (TF or term frequency) or weigh the words according to how often they appear in the documents (IDF or inverse document frequency). Using TF-IDF, common words will have a low value as they appear across most documents, while significant words will have a high value because they appear frequently in a small number of documents.

Pass the data through a *Bag of Words* widget and then again to a *Data Table*. We get a new column that contains word counts for each document. Now that we have numbers, we can finally perform some magic!



Screenshot of the KNIME interface showing the Data Table window.

The Data Table window displays a table with the following columns:

bow-feature	ATU Topic	...	Title
hidden			True
include			
skip-normalization			
title			
1	Tales of Magic	able=1,accord=1,actually=1,afraid=1,ago=1,ah=7,a...	A Tale About the Boy ...
2	Tales of Magic	ale=3,alone=1,also=2,altogether=1,amiss=1,angry...	Brier Rose
3	Animal Tales	absence=1,acquaintance=1,advice=1,agreed=1,ala...	Cat and Mouse in Part...
4	Tales of Magic	_my_=1,afterwards=1,almost=1,also=1,altogether...	Cinderella
5	Tales of Magic	able=1,across=3,add=1,afar=1,afterwards=2,ah=2,...	Hansel and Gretel
6	Animal Tales	aboard=2,backward=1,along=1,answered=2,arrive...	Herr Korbes
7	Tales of Magic	_jug=1,alas=2,almost=1,alone=1,already=1,always...	Jorinda and Jorindel
8	Tales of Magic	able=1,act=1,afraid=1,afterwards=1,aged=1,ah=1,...	Little Red Riding Hood
9	Tales of Magic	according=1,afraid=2,ago=2,agreed=1,alas=2,alth...	Mother Holle
10	Animal Tales	accordingly=1,advice=1,afterwards=1,air=2,along...	Old Sultan

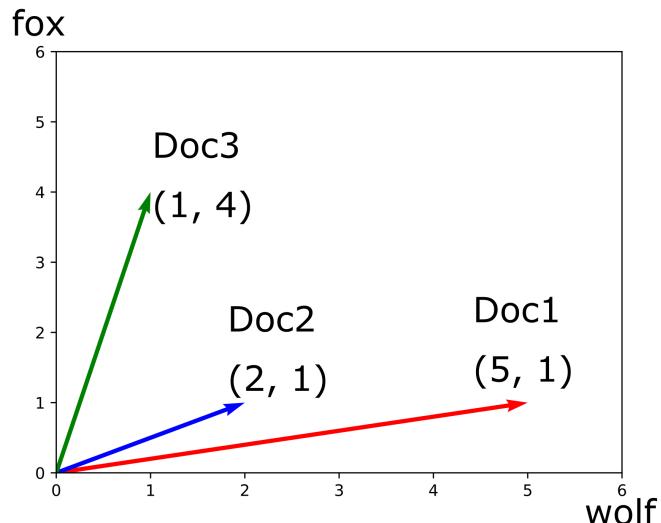
Text Clustering

A COMMON TASK IN TEXT MINING IS FINDING INTERESTING GROUPS OF SIMILAR DOCUMENTS. That is, we would like to identify documents that are similar to each other.

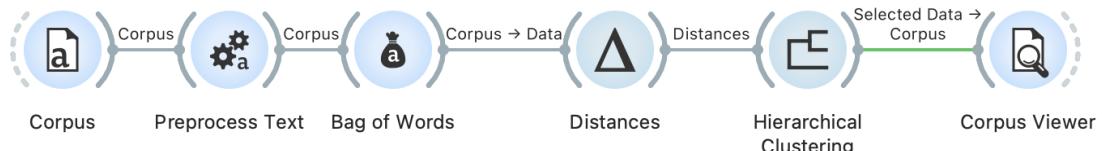
We've already seen a simple example of hierarchical clustering on a 2-dimensional data set. But clustering text needs some adaptations. We have been working previously with Euclidean distances, which are analogous to the distances we can measure with a ruler.

But for complex objects with many dimensions, Euclidean distances simply don't work so well anymore. Instead, one would use cosine distance. Word counts from BoW are vectors, each pointing in a direction defined by text content. Cosine distance is the angle between these vectors.

The plot shows vectors of three documents. Vectors correspond to the number of times the words "fox" and "wolf" appear in the text. By Euclidean distances, the two most similar documents would be Doc2 and Doc3, despite Doc2 mentioning wolves more frequently than foxes. If we take the angle between these vectors instead, Doc2 is the most similar (closest) to Doc1, which makes sense since they both talk predominantly about wolves. This is why we use the cosine distance.



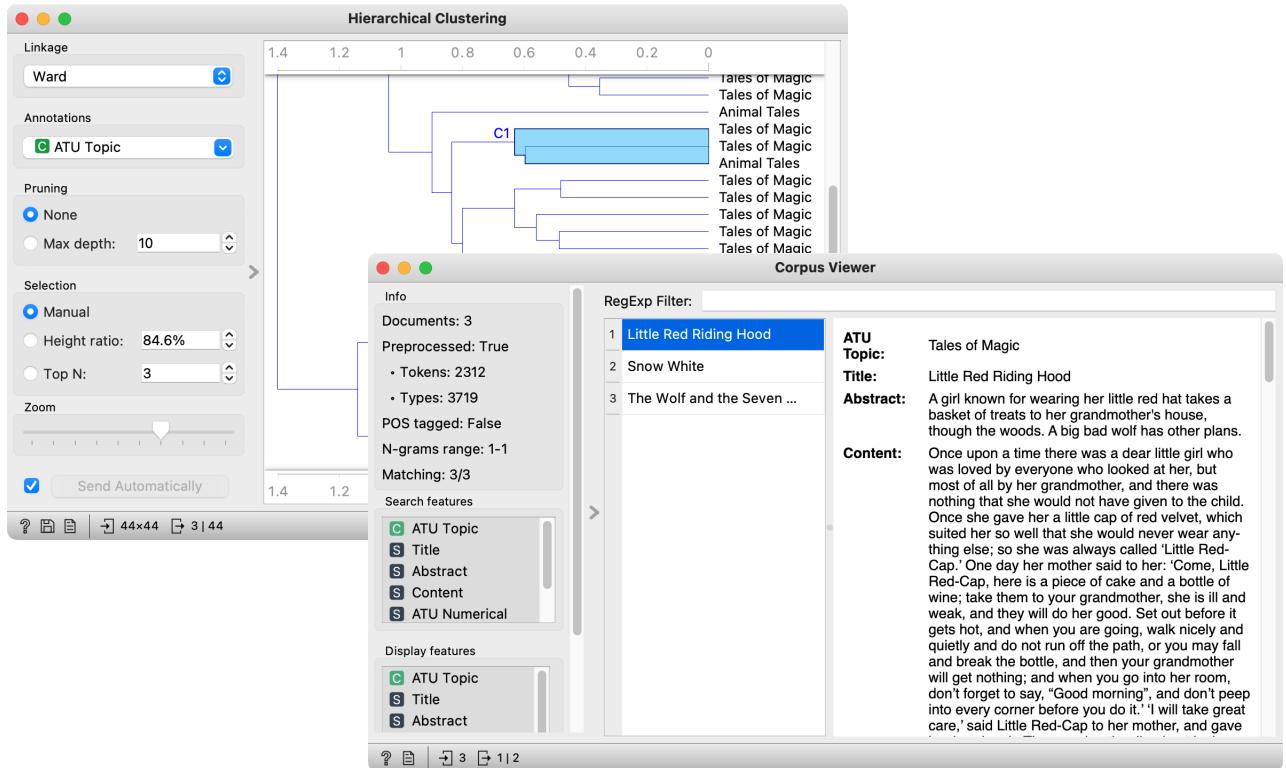
Now, let us go back to our Grimm's Tales and construct the following workflow:



You can try the same workflow on a different corpus, say bookexcerpt.tab, which contains excerpts from adult and children's books.

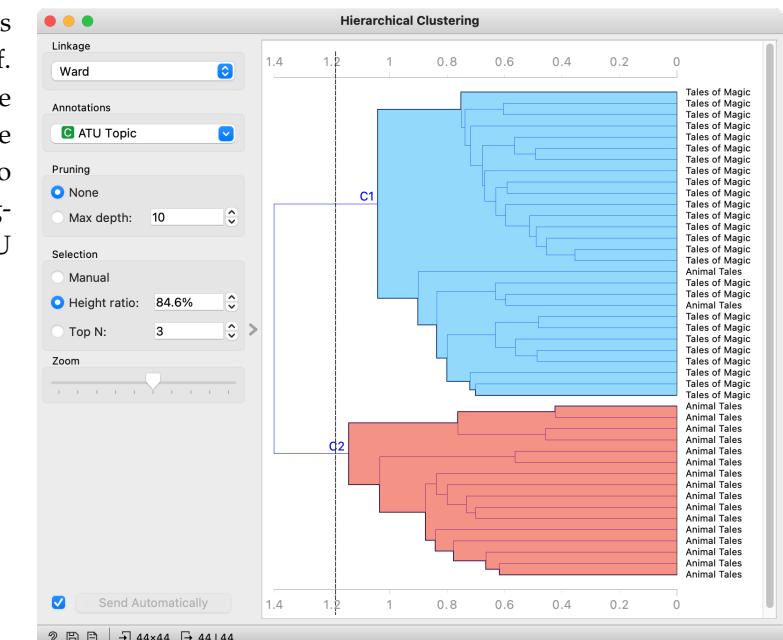
The *Hierarchical Clustering* widget visualizes the clustering in a dendrogram. Connect *Corpus Viewer* to *Hierarchical Clustering* and open both widgets. Now click on a cluster in the dendrogram and observe the documents from the selected cluster in *Corpus Viewer*.

Explore different clusters. Why are some Tales of Magic mixed with Animal Tales? What do they have in common?

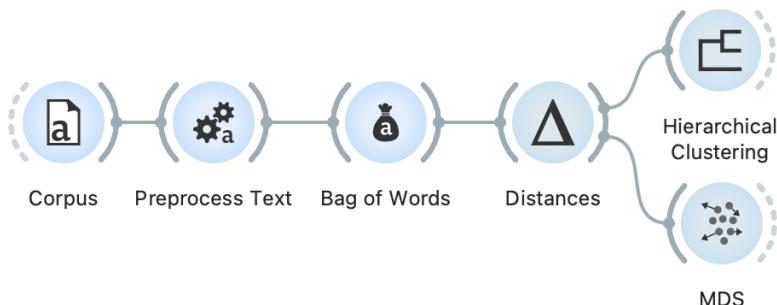


Hierarchical Clustering builds a hierarchy of documents and it is up to us to define what is similar enough to be in one cluster. We can set the appropriate degree of similarity by dragging the vertical line left or right in the visualization.

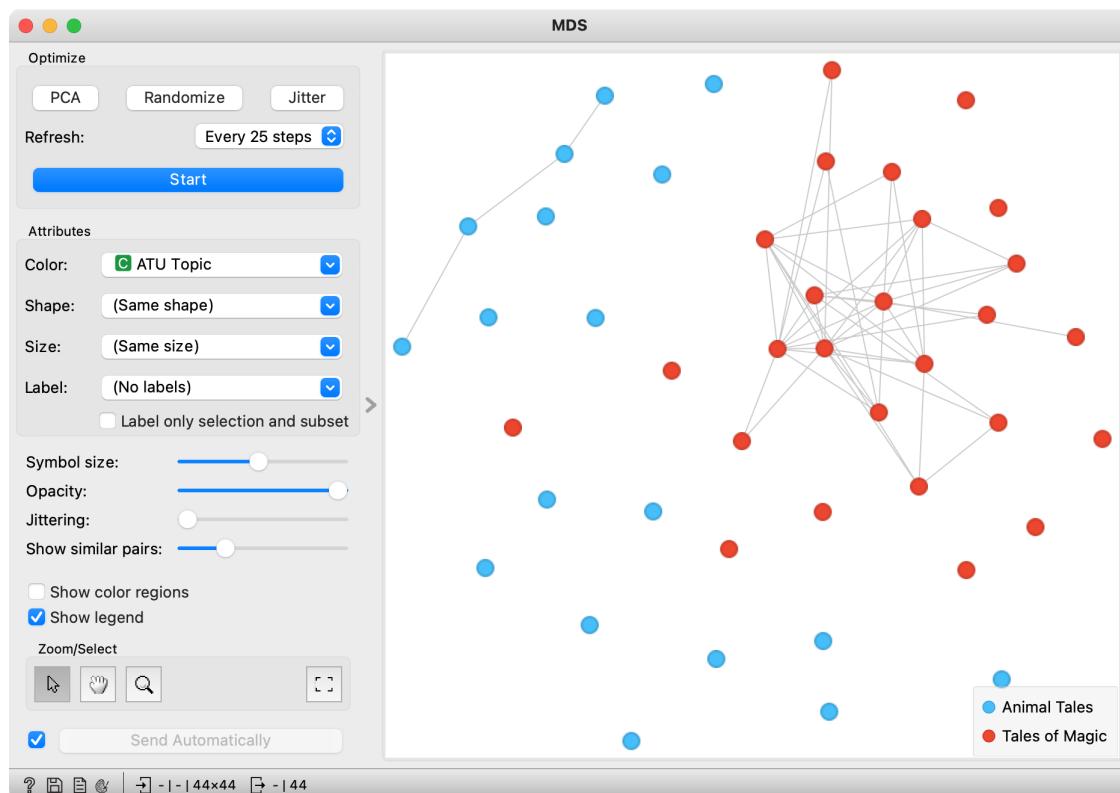
We went with two groups, since the distance between the clusters increases at that particular cut-off. Compare this to the cut-off we would require for three, four or five clusters. The clusters we get also correspond nicely with the designated Aarne-Thompson type (ATU Topic).



But how close are the animal tales from the third and animal tales from the last cluster? Let us see the documents on a plane, where similar documents would lie close to each other. This visualization is called Multidimensional Scaling or *MDS* in short.



Tales of Magic form one group and Animal Tales another - just as we expected. Interestingly, Tales of Magic seem to be more similar to each other than Animal Tales are (they are connected). Inspect similar tales by selecting them in the visualization and reading them in Corpus Viewer.



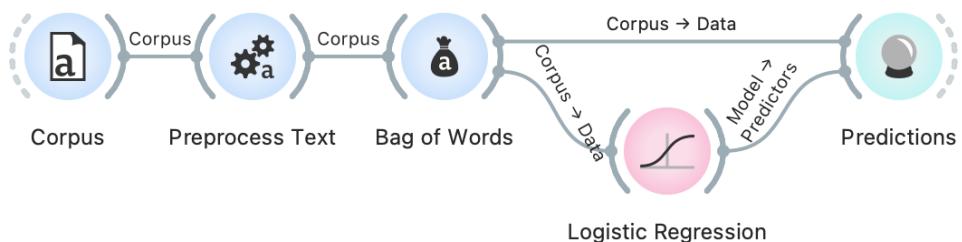
Text Classification

Grimm's tales corpus contains the attribute for Aarne-Thompson type (ATU). This is the index of folk-tale motifs and the tale have a high-level (genre) and a mid-level ATU type (subgenre).

Could we perhaps predict the ATU type based on the content of the tale? Let us see.

First, we need a target variable. This is the feature we are trying to predict, in our case an ATU type. We also need a numerical representation of each document - something we already have from the *Bag of Words*.

Aarne and Thompson were two folklorists, who invented and perfected the motif-based classification system of folk tales. This system has been in place since 1910 and is commonly used in comparative folkloristics. The final U in ATU stands for Uther, who was the last to update the index in 2004.



Now we will build a predictive model. A predictive model considers tokens (words) and predicts the target variable (ATU Topic). Every model also needs a learner, which is a method on how to consider the tokens. A commonly used classifier in text mining is *Logistic Regression*.

In Predictions, we can see a column with predicted values from Logistic Regression. Seems like our model got most of the tale types right.

Predictions					
Show probabilities for		Logistic Regression			
		ATU Topic	Title	Abstract	Content
1	0.00 : 1.00 → Tales of M...	Tales of Magic	A Tale About...	A simple boy...	A certain f...
2	0.00 : 1.00 → Tales of M...	Tales of Magic	Brier Rose	An offended ...	A king and...
3	1.00 : 0.00 → Animal Tales	Animal Tales	Cat and Mou...	A mouse live...	A certain c...
4	0.00 : 1.00 → Tales of M...	Tales of Magic	Cinderella	The familiar ...	The wife o...
5	0.00 : 1.00 → Tales of M...	Tales of Magic	Hansel and ...	A poor wood...	Hard by a s...
6	1.00 : 0.00 → Animal Tales	Animal Tales	Herr Korbes	A hen and a ...	Once upor...
7	0.00 : 1.00 → Tales of M...	Tales of Magic	Jorinda and ...	A witch lures...	There was...
8	0.00 : 1.00 → Tales of M...	Tales of Magic	Little Red Ri...	A girl known ...	Once upor...
9	0.00 : 1.00 → Tales of M...	Tales of Magic	Mother Holle	A widow spo...	Once upor...
10	1.00 : 0.00 → Animal Tales	Animal Tales	Old Sultan	A farmer dec...	A shepherd...

Model AUC CA F1 Precision Recall
Logistic Regression 1.000 1.000 1.000 1.000 1.000

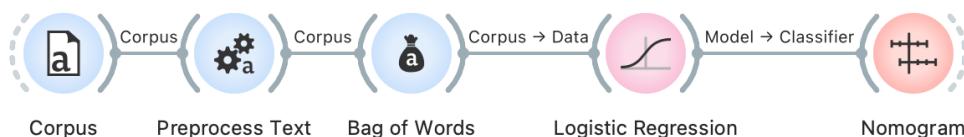
Restore Original Order

Logistic Regression

This visualization is called a nomogram. It displays scores (votes) of each attribute for the selected target value at the top left.

In the workflow above we have used logistic regression, a popular machine learning method. It is often used in text mining for its speed and predictive performance. How does it work?

It lets the words vote. For example, the word 'fox' in the text votes for the tale being an animal tale. So do cats and birds and wolves, but not so strongly (lines in the visualization are much shorter). We can see that the word 'fox' is the best clue for the text being an animal tale.

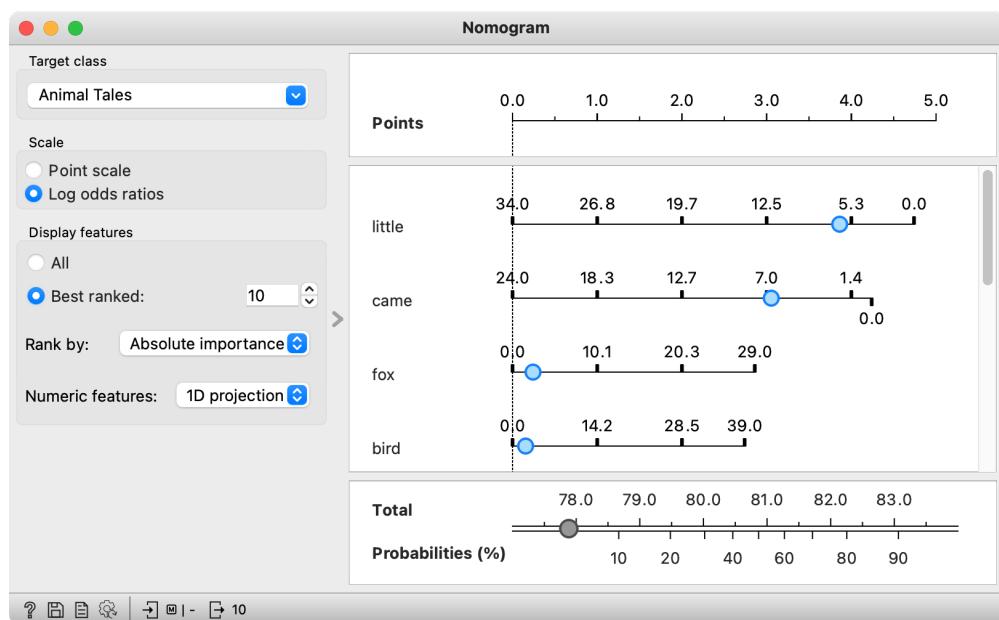


In Nomogram, you can interactively observe the model's classification. Drag the blue dots left or right so that the accumulated sum of points (Total) is as high as possible.

The word 'little' votes for the opposition. So does 'came' (see how they have zeros at the right end of the scale?). The more little things there are in a story, the less likely it is an animal tale.

Each word gives a score. If there are 29 foxes in the text, then the model will give it 3 points for being an animal tale. And if there are no words 'little', it will give it an additional 4.5 points.

Of course the real method is a bit more complicated - it tries to find appropriate vote weights and thresholds. But this involves some linear algebra and other scarier-than-wolves words, so let's not stroll down this path.



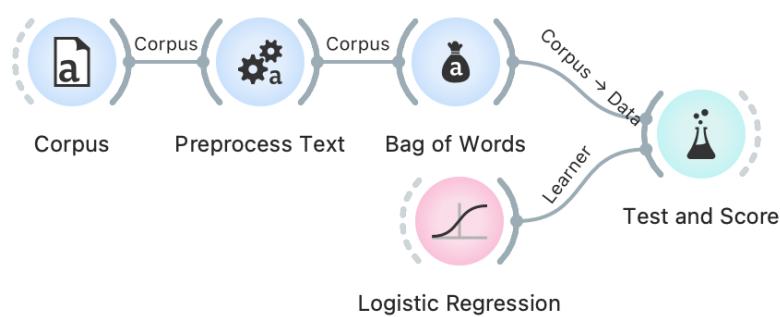
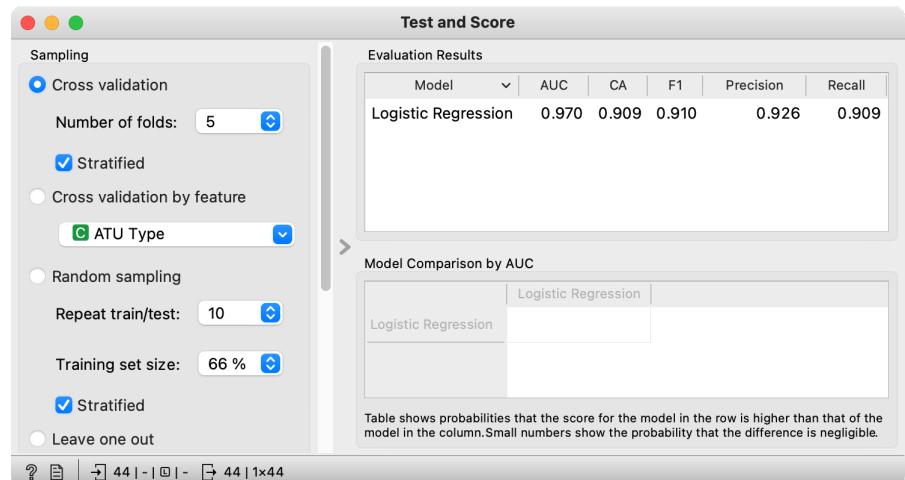
Model Evaluation

We have observed the logistic model - foxes, birds, little things and all that. The voting schema looked reasonable. And before that, we have seen the model's predictions. They looked good. But how would we quantify the performance of the model?

Perhaps we should just compute the proportion of the stories for which the model gave the correct answer? This score is called classification accuracy. For example, if we correctly predicted 40 tales out of 44, the classification accuracy would be $40/44$ or 91%.

AUC is another good measure to consider. In essence, the closer these two measures are to 1, the better the model's performance.

The widget that computes the classification accuracy is called *Test and Score*. It needs two inputs: the data to test the model on and the modeling algorithm.



This time, Logistic Regression doesn't need a data input. Instead, it provides a learner, which is a procedure for constructing the model. Test and Score then applies the learner multiple times on different data subsets. Each time, it constructs the model on a selected subset and uses the left-out data for testing. It would make little sense asking the model whether Rapunzel is an Animal Tale, after already telling it that it is not.

It would make little sense asking whether Rapunzel is about animals after already telling the model that it is not.

Didn't we do this above in Predictions? Indeed, and this is why the predictions there were so excellent. Models should never be tested on the data from which they were constructed.

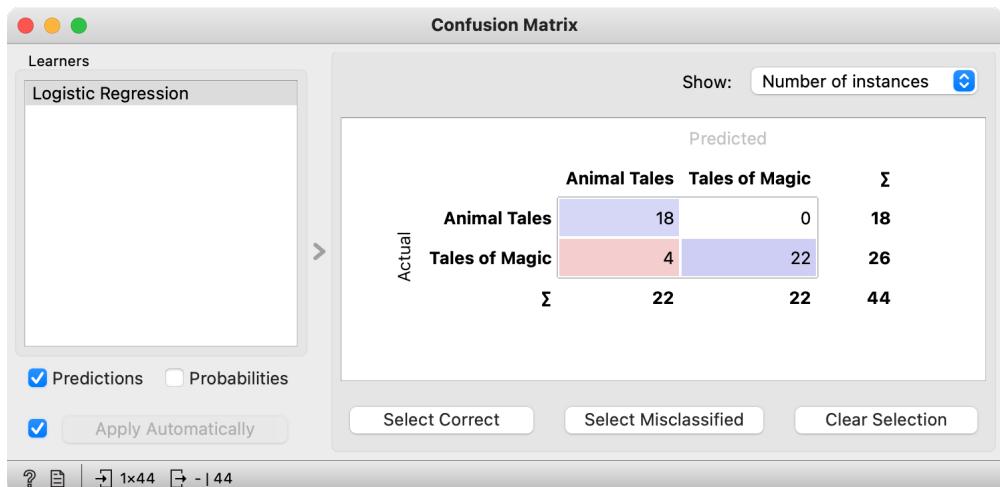
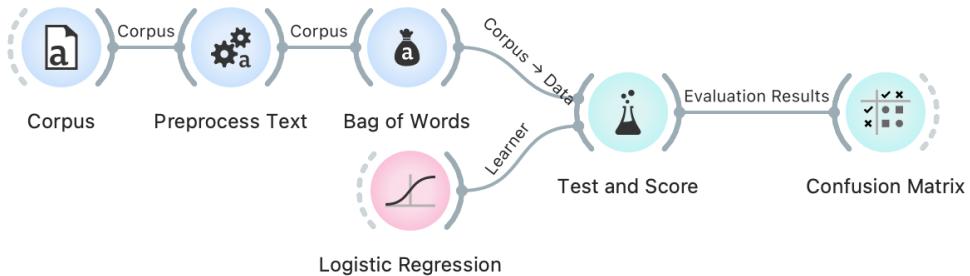
Word Enrichment

To observe a particular misclassification, click on a cell in the matrix.

To output all the misclassified documents, use 'Select Misclassified'. This will output selected documents for further inspection.

We already know a lot about our model, but wouldn't it be great to see where it was right and where it was wrong? The *Confusion Matrix* widget can do exactly that!

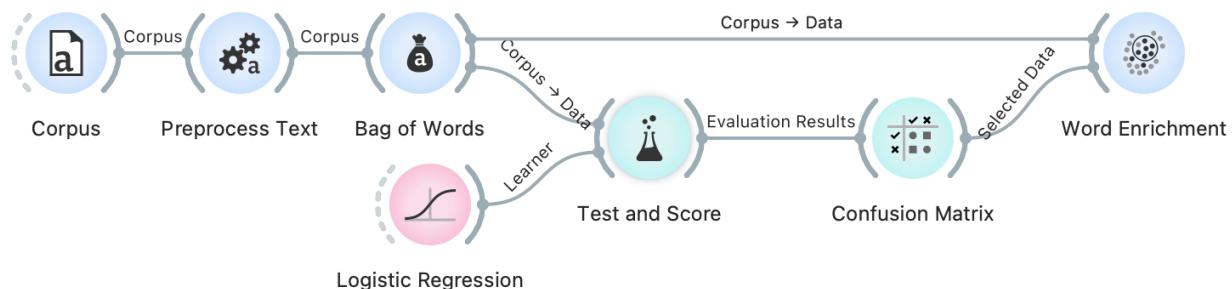
The matrix has correctly predicted documents in the diagonal (blue) and incorrectly predicted documents in off-diagonal (red). We can see that no Animal Tales were wrongly predicted as Tales of Magic, but 4 Tales of Magic were incorrectly predicted as Animal Tales.



But why? What is so different about these documents, that the model failed to predict the right class?

We can reuse *Corpus Viewer* to inspect the misclassified documents. Or even better, observe which words are significant in each subset!

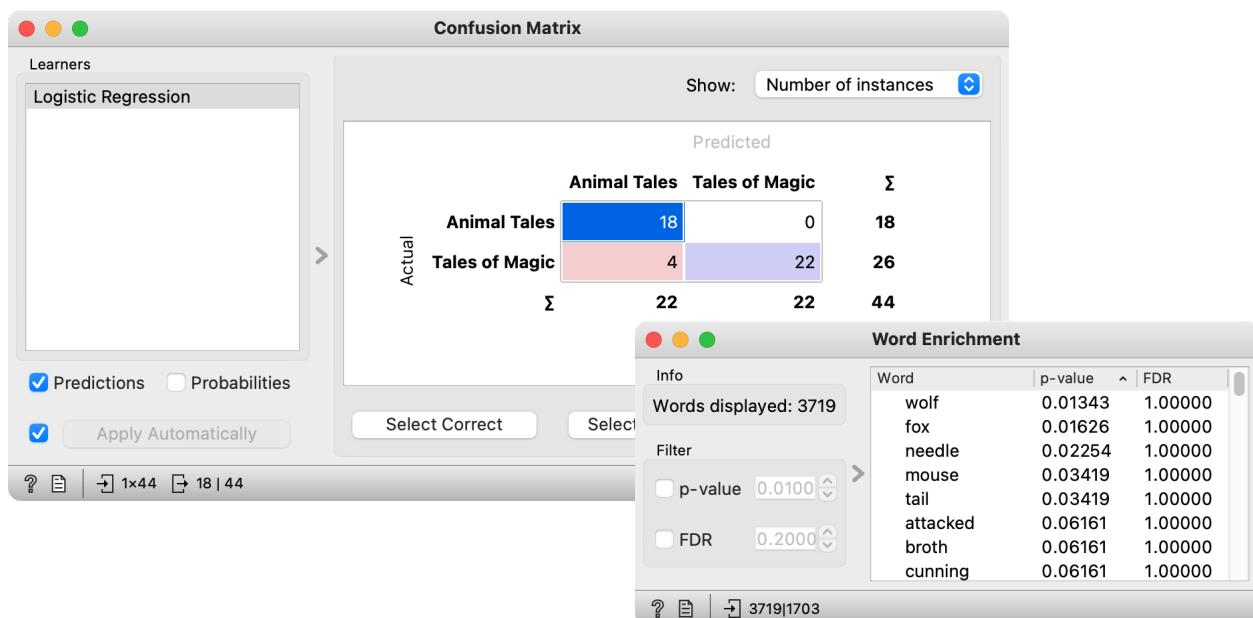
Word Enrichment compares a subset of documents against the entire corpus and finds statistically significant words for the selected subset.



For correctly classified Animal Tales, *wolf* is a significant word representing those documents. For correctly classified Tales of Magic, the list is much longer and contains words such as *king*, *beautiful*, *man*, etc. These results are very similar to what we have observed in a Nomogram. This is indeed just a different way of inspecting the model!

So the next time you see the word *wolf* in a text, you can bet the text is an animal tale! :)

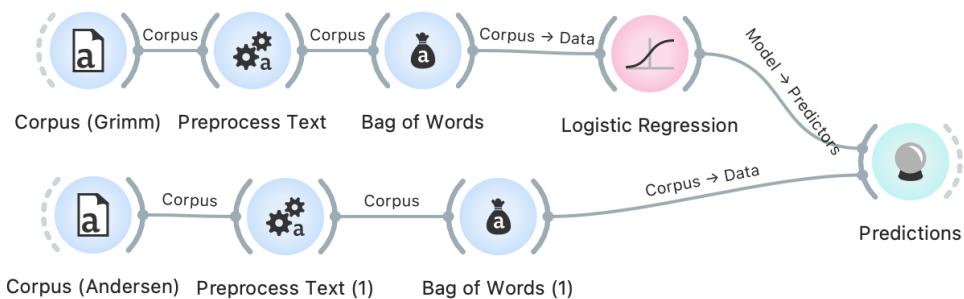
Word Enrichment works on any kind of subset. In Corpus Viewer, browse for documents containing the word *queen*. Now pass this subset to *Word Enrichment*. Don't forget to connect *Bag of Words* as well - Word Enrichment needs the entire data set against which to compare the subset.



Predictions

All we did so far is predict what we already knew - the tale type of each text. We had the right answers already in our ATU Topic column. But what if we don't have this information? Could we perhaps predict tale types for unlabelled texts?

Open a new *Corpus* widget and load the *andersen.tab* corpus. Here we have three tales from H. C. Andersen. Inspect them in *Corpus Viewer* and try to guess the tale type yourself.



Now connect them to *Predictions* the same way as before - with *Logistic Regression* passing the constructed model and the new Corpus widget passing the data for prediction. Do not forget to copy-paste the preprocessing widgets, namely *Preprocess Text* and *Bag of Words* to repeat data preparation.

Logistic Regression predicted all the tales to be Tales of Magic. The Ugly Duckling as a Tale of Magic? Sounds strange!

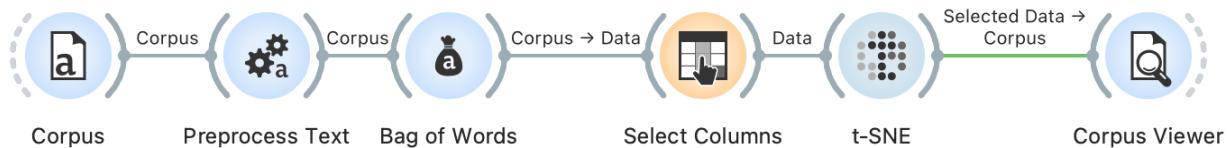
Why do you think this happens?

	Title	Content	{...}
The Little Match-Seller	It was terribl...	across=2,ah=1,almost=2,along=...	
The Philosopher's Stone	Far away to...	abilities=1,able=2,absent=1,abs...	
The Ugly Duckling	It was lovely...	able=1,absurd=1,across=2,advic...	

Twitter

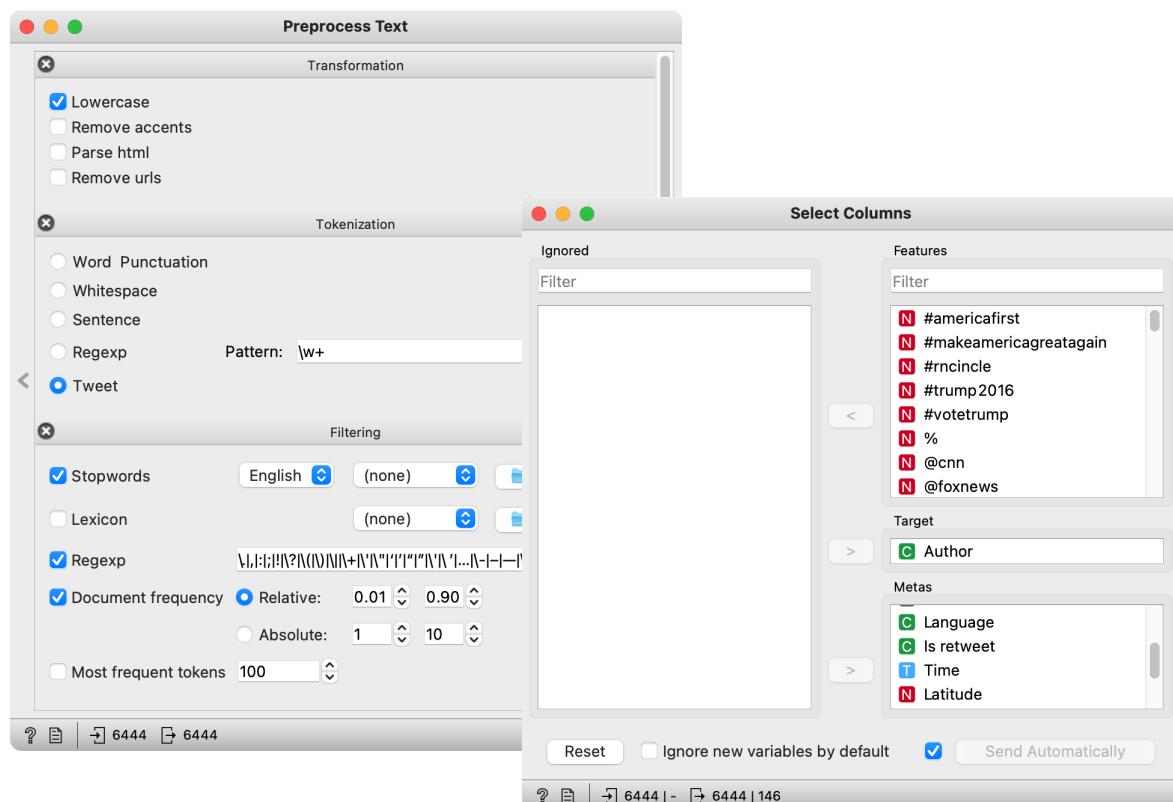
Ok, how about a fresh data set? Let us use Twitter data for the next example. In *Corpus*, go to *Browse documentation data sets* and select *election-2016-tweets.tab*. This corpus has over 6000 tweets of Hillary Clinton and Donald Trump from the pre-election period in 2016.

You can also use the *Twitter* widget to retrieve tweets on any topic. To do this, you will need your own API key: <https://developer.twitter.com/>



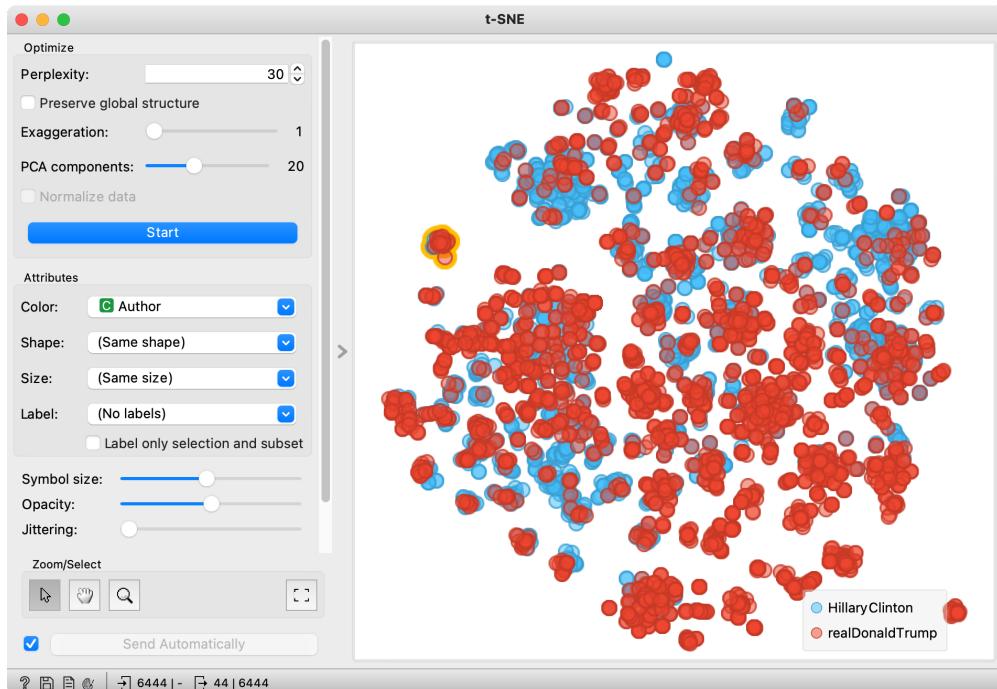
First, we need to preprocess the data with *Preprocess Text*. We will use a special Tweet tokenizer, that was pre-trained on millions of tweets. This will retain special words, such as @mentions, #hashtags, and emojis :). As the Tweet tokenizer doesn't remove punctuation, we will remove it with the pre-set Regexp filter.

This time we have a lot of unique tokens and things might get painfully slow. To narrow down the tokens even further, we can filter out all the infrequent words with *Document frequency*. Set the lower threshold to 0.01 - this means we will omit all the words appearing in less than 1% of the tweets.



Great, now we are left with manageable 146 tokens. Pass the pre-processed data to *Bag of Words*. If we check the results of BoW in a *Data Table*, we can see there are some features we don't wish to consider in the analysis. To remove them from the analysis, use *Select Columns* and put all those non-textual features to meta attributes.

Next, we will use t-SNE, a special dimensionality reduction technique that allows us to observe the data in a 2D plane. In this visualization, data instances that are similar to each other, will be put closer together. This usually results in distinct clusters, just like in our example.



Select a cluster and inspect it in a *Corpus Viewer*. What are the documents about? Does the cluster make sense?

Content: Nothing on emails. Nothing on the corrupt Clinton Foundation. And nothing on #Benghazi. #Debates2016 #debatenight

Content: A Clinton economy = more taxes and more spending! #Debate-Night <https://t.co/oFlaAhrwe5>

Content: It's still legal to pay disabled workers less than the minimum wage and Clinton announces her plan to end that: <https://t.co/vEglqMkmmg>

Content: Philly FOP Chief On Presidential Endorsement: Clinton 'Blew The Police Off' <https://t.co/ATBY343pS1>

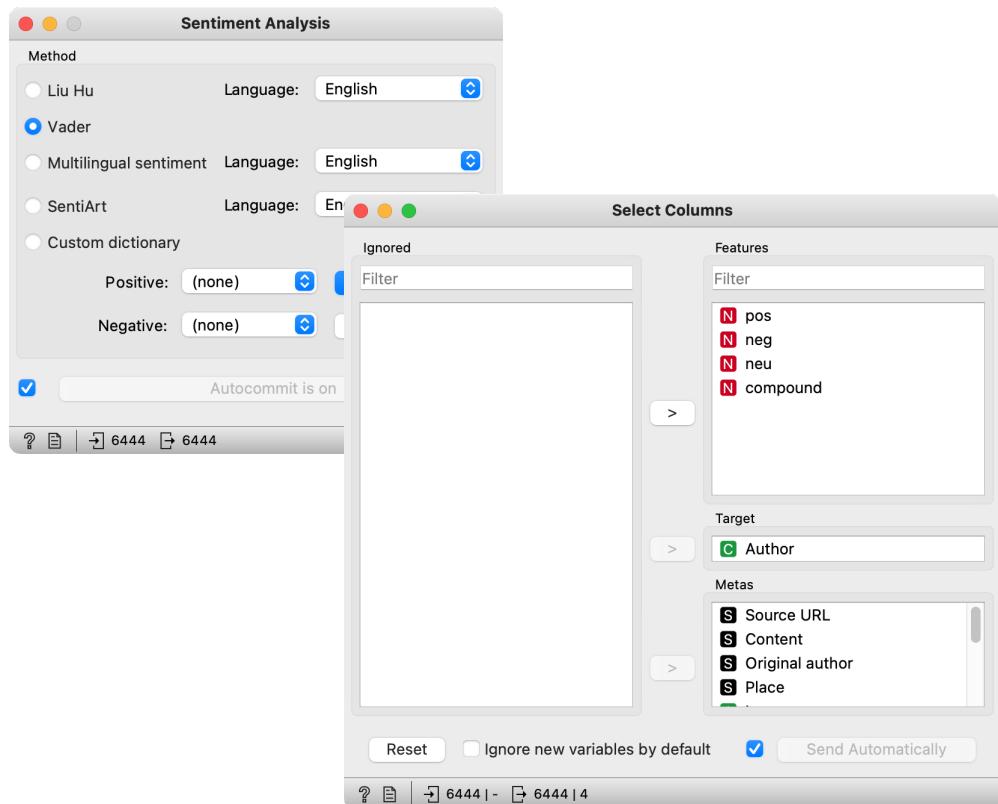
Sentiment Analysis

SENTIMENT ANALYSIS IS USEFUL IN MANY DIFFERENT SITUATIONS.

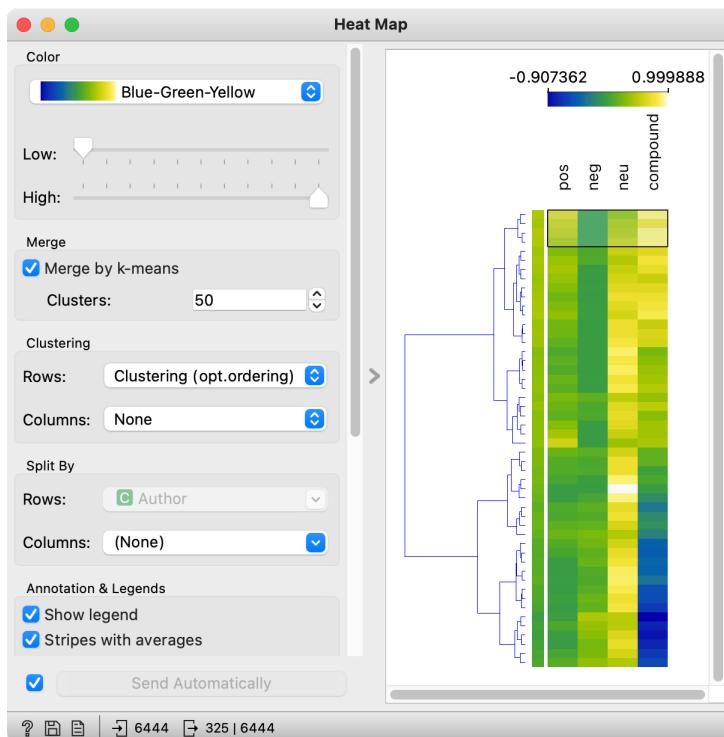
It can be used to explore brand sentiment, observe story arcs, recognize hate speech, and so on. Orange uses simple lexicon-based methods to compute the sentiment of the text.



Sentiment scores are computed in the *Sentiment Analysis* widget. We will use the default *Vader* method, but feel free to try the others. In *Select Columns*, we will keep only the sentiment scores and put all the other features in meta attributes.



A nice visualization for observing several numeric values at once is *Heat Map*. Heat map shows documents in rows and attributes in columns, with the color of the field corresponding to its value. In this case, high values are yellow/white and low values are blue.



selecting the top leaf of the dendrogram. Then, observe the output in *Corpus Viewer*.

Looks like our selection indeed contains positive tweets!



But our data is completely unorganized and it is difficult to make sense of the visualization. Hence we will use some tricks to make it interpretable. First, we will join similar rows into a single row. We will do this with k-Means and keep only 50 rows.

Now, our visualization is much more compact. But preferably, we would sort the rows in some logical order. For this, we will use *Clustering (opt. ordering)* setting. It will use hierarchical clustering to put similar rows together and optimally place the leaves of the dendrogram.

The visualization is finally interpretable. At the top, we see positive documents, while in the bottom, the negative ones. Select the most positive documents by

Networks

NETWORKS ARE STRUCTURES DESCRIBING CONNECTIONS BETWEEN ELEMENTS. For example, Twitter followers are a network describing who follows whom. Proteins interact with one another, which can be shown in a network, where protein interactions are connections between elements. Citations in text are a network, in which authors reference one another. And so on.

Elements in a network are called nodes. You can imagine nodes as rows in the data (i.e. Twitter users, proteins, authors). Connections between these elements are called edges (i.e. following someone, protein interaction, citation).

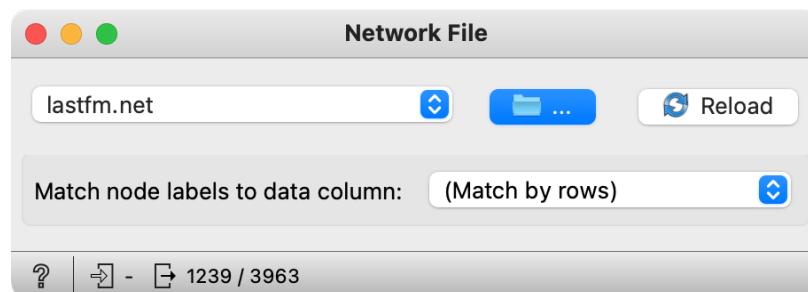
Installing the Network add-on

You will need the *Network add-on*, which introduces components for network visualization and analysis. To install Network add-on, go to Options → Add-ons and select Network from the list. You will have to restart Orange for widgets to appear.

A new pane with widgets from the Network add-on will appear on the left side of the canvas.

Loading networks

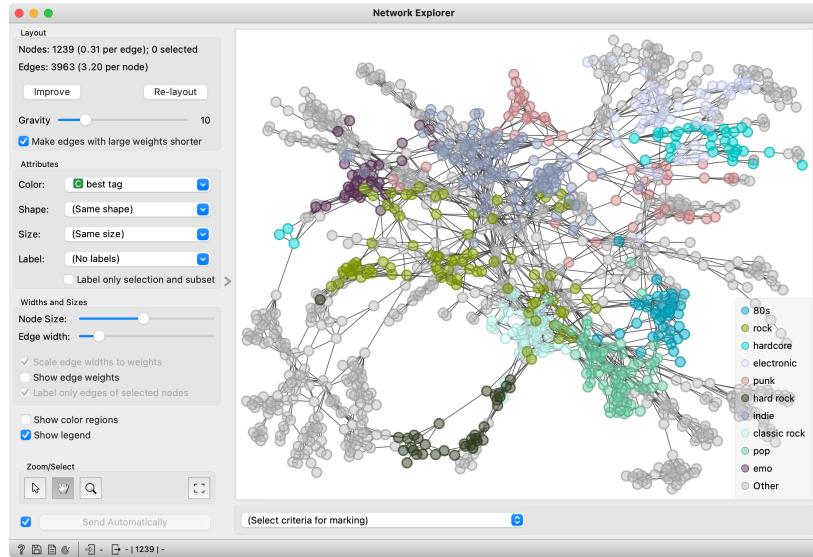
Let us observe a simple network. With the *Network File* widget, go to *Browse documentation networks...* and select *lastfm.net* from the list. Lastfm describes a network of artists and the genre they most likely belong to. Nodes in the network are artists, while edges are shared tags.



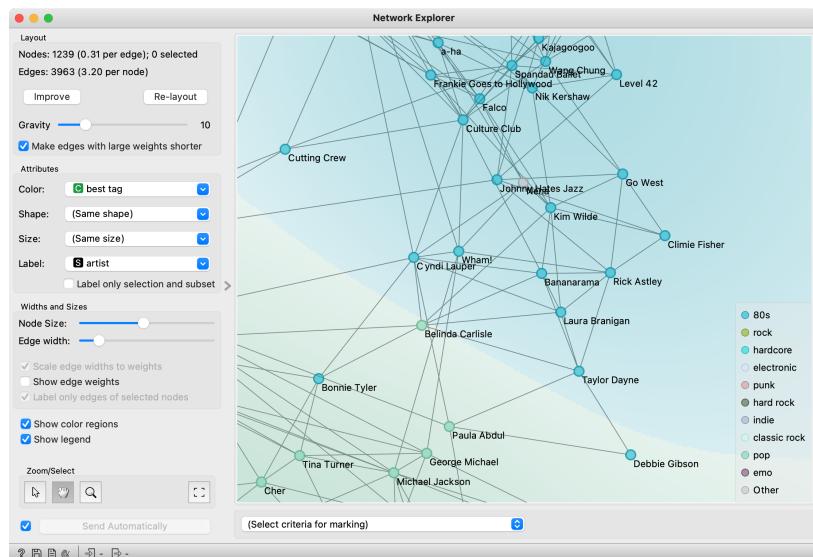
Lastfm network is derived from the Lastfm website, where users can tag tracks with labels. Here, *best tag* corresponds to the most frequent tag for the artists, and edges are the number of shared tags.

Visualizing networks

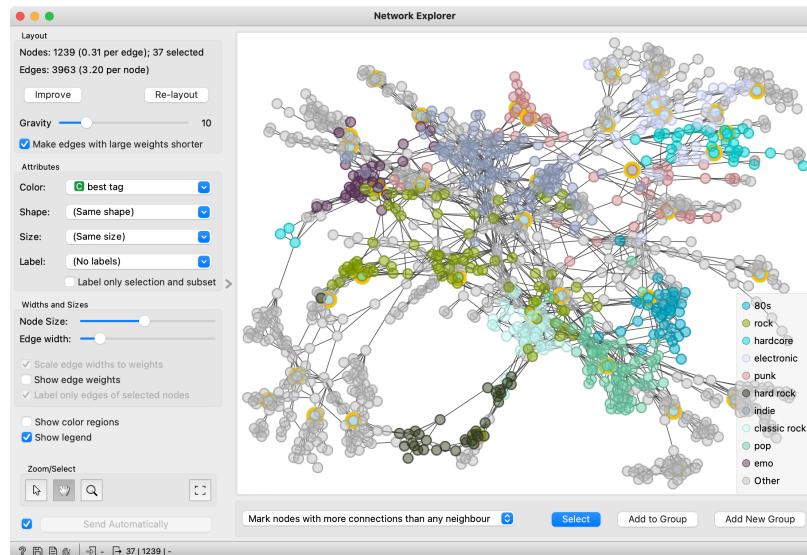
One can, of course, observe the data in a *Data Table*, but in case of networks, all one would see are metadata. A better way of exploring networks is with *Network Explorer*. This visualization shows network structure with points being artists, colored by the most frequent genre tag.



Now let us zoom in onto a genre, say the 80s. We will turn on the *Show color regions* options, which will color the plot by the prevalent tag. Seems like the 80s are in close connection to pop artists.



But who are the genre representatives? If a node shares many similar tags with its neighbors, this would likely be the most typical representative of the genre. Let us select those nodes from the graph. At the bottom of the widget, set selection criteria to *Mark nodes with more connections than any neighbor*, then press *Select* to send the selection to the output.



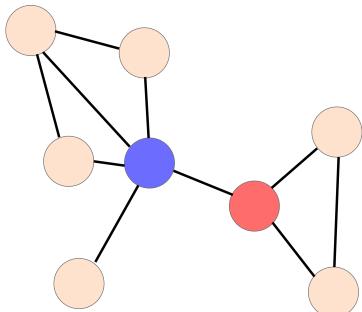
Connect *Data Table* to the widget and observe the subset. The Rolling Stones as typical classic rock representatives and Britney Spears as a typical pop artist? Sounds about right.

Data Table						
	best tag	artist	albums	proj-x	proj-y	Group
1	trip-hop	Massive Attack	Danny the D...	0.315585	0.665757	G1
2	metal	Disturbed	Indestructibl...	-0.730902	-0.181768	G1
3	funk	James Brown	Sho Is Funky...	0.795011	-0.663983	G1
4	minimal	Ricardo Villalobos	Fabric 36,Gr...	0.876711	0.825843	G1
5	post-punk	The Fall	Imperial Wax...	0.543135	0.243585	G1
6	classic rock	The Rolling Stones	Shine a Light...	0.0338871	-0.258395	G1
7	punk	Strike Anywhere	Dead FM,Exi...	-0.0229408	0.739445	G1
8	ninja tune	DJ Food	Now,Listen A...	0.502928	0.928039	G1
9	hardcore	Blacklisted	The Beat Go...	0.765575	0.619199	G1
10	singer-song...	Anna Ternheim	Halfway to Fi...	-0.554293	0.884381	G1
11	indie	Bloc Party	Intimacy,A W...	0.0559887	0.389012	G1
12	deathrock	Skeletal Family	Futile Comb...	0.88853	-0.058239	G1
13	experimental	Charalambides	Likeness,A V...	-0.651216	0.55356	G1
14	pop	Britney Spears	Blackout,B i...	0.314716	-0.500807	G1
15	electronic	LCD Soundsystem	45-33,Soun...	0.529959	0.707126	G1
16	chillout	Tosca	Souvenirs,J....	0.331192	0.905261	G1

Network Analysis

NETWORK ANALYSIS CAN TELL US A LOT ABOUT THE MOST IMPORTANT ELEMENTS OF THE GRAPH AND HOW WELL-CONNECTED EACH ELEMENT IS. A common way is to count the number of connections, shortest paths to neighbors, and shapes that the network takes (i.e. number of triangles).

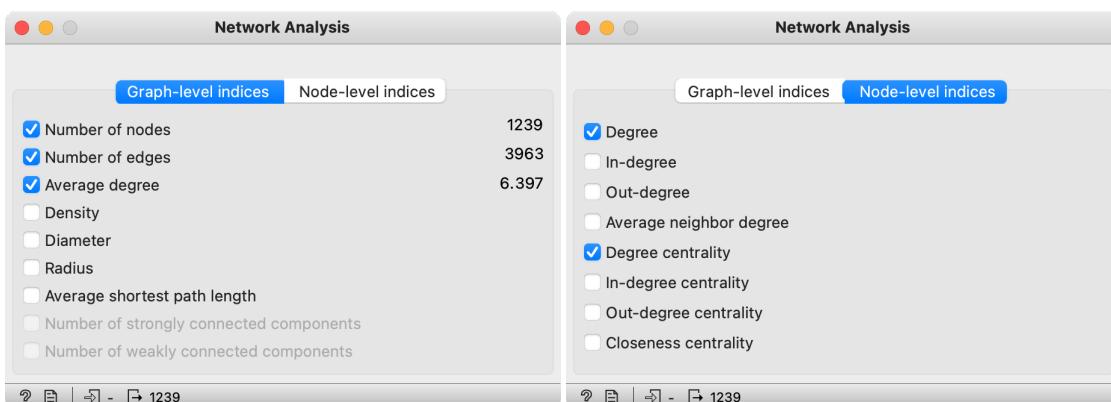
Some common measures include:



The average degree of the above graph is 2.5 (The number of edges for each node divided by the number of nodes.). The degree of the blue node is 5, while the degree for the red node is 3. The degree centrality for the blue node is 0.5 (half of all edges are connected to it!), while for the red node it is 0.3.

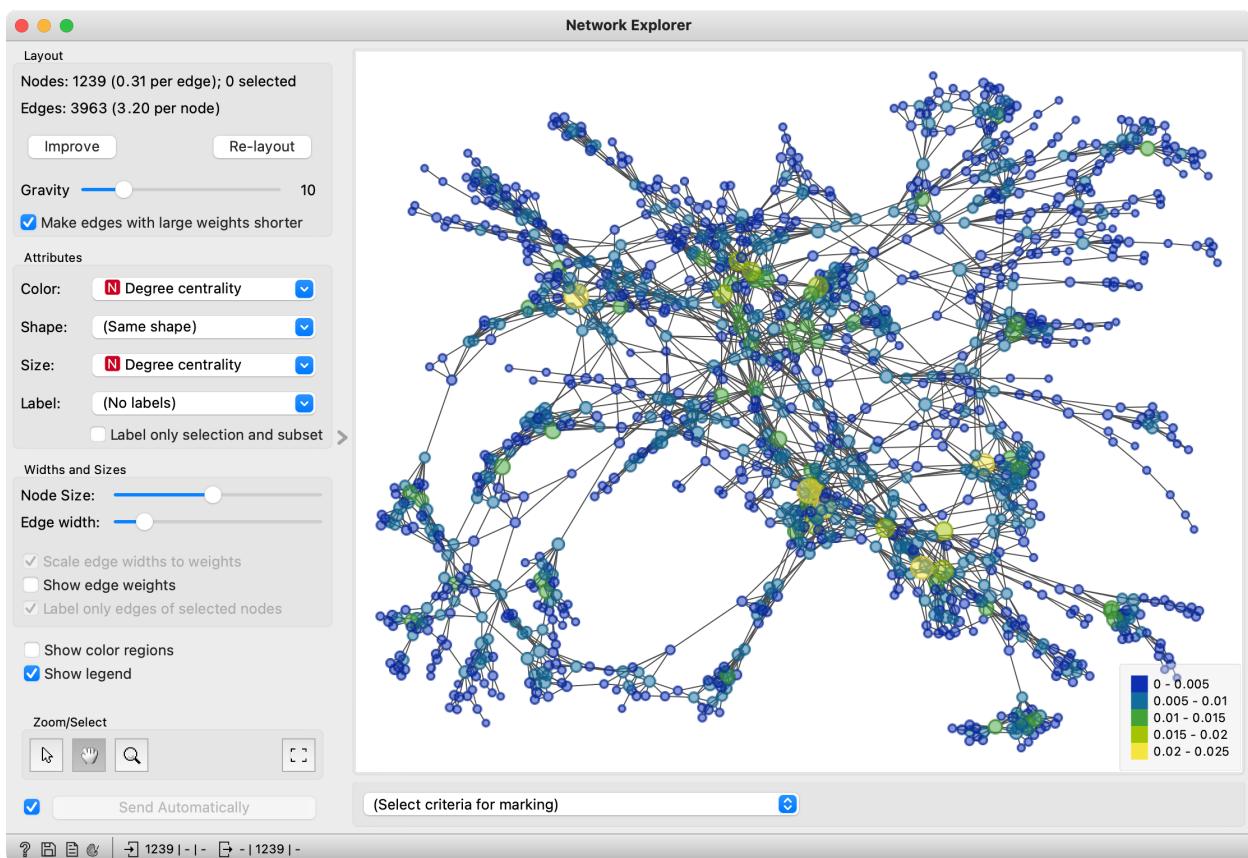
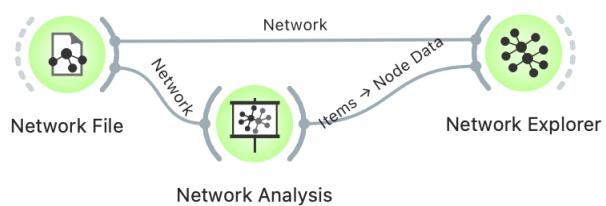
- *average degree*, which corresponds to the average number of edges per node. For lastfm, this would mean the average number of artists with whom the genre tag is shared. The higher the number, the more densely connected the network is. A similar measure is graph density, which corresponds to the ratio between number of edges and all possible number of edges.
- *degree*, which is the number of edges per node. The higher the number, the better connected the node is. For lastfm, this would mean the artist with the highest degree would be the one that shares the most genre tags with other artists.
- *degree centrality*, which is the ratio of nodes that connect to the given node. A high ratio would mean a node that can reach the most other nodes and that the node is very important in graph structure. In the case of lastfm, this would be the artist that represents the most widely used genre (perhaps pop, rock, jazz).

But enough talk, let us observe this in practice! A widget that computes these statistics is called *Network Analysis*.



Some statistics relate to graph properties, while others relate to node properties. Select the above three measures. Node statistics will be added to the data table, which means we can observe it in *Network Explorer*.

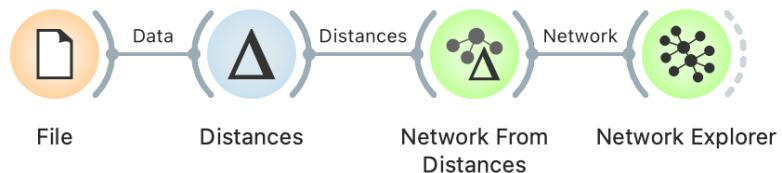
Sending additional node data to Network Explorer enables us to use the computed statistics in a visualization. We can color the nodes and set their size to degree centrality. This will expose the most central nodes - the ones that share the most tags with their neighbors. Unsurprisingly, we find The Rolling Stones and Britney Spears in the bunch.



Networks from Distances

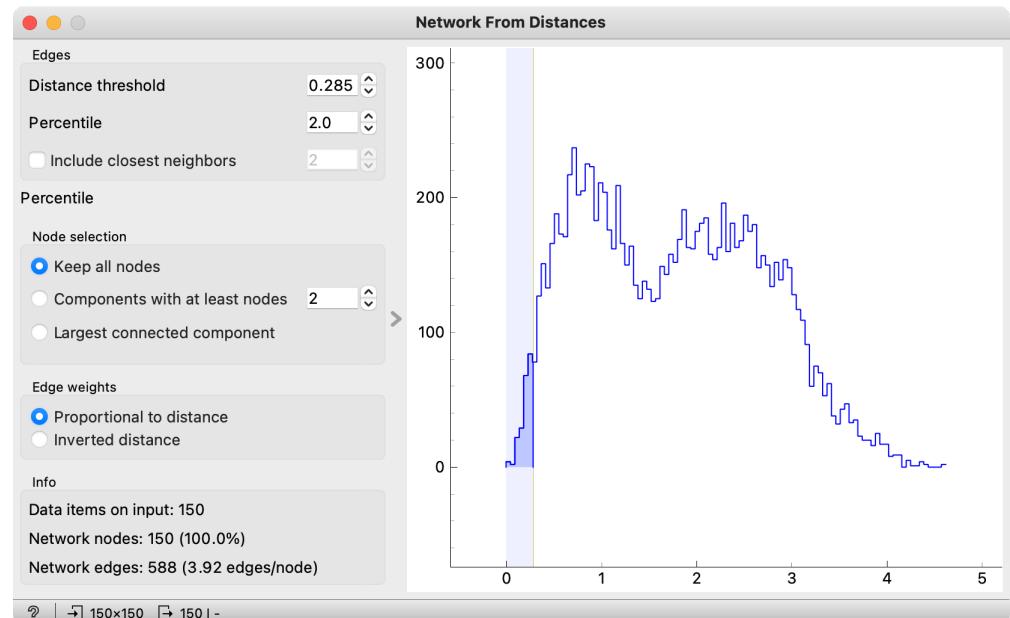
So far we have been using existing network graphs. But we can also generate a graph from regular tabular data. Let us take the *iris* data set. For now, we only have flowers as rows and their characteristics as columns. So how do we transform this to nodes and edges?

By looking at similarities. Rows will be nodes and the edge will be created between two rows if their similarity is above a certain threshold. Nice, isn't it?

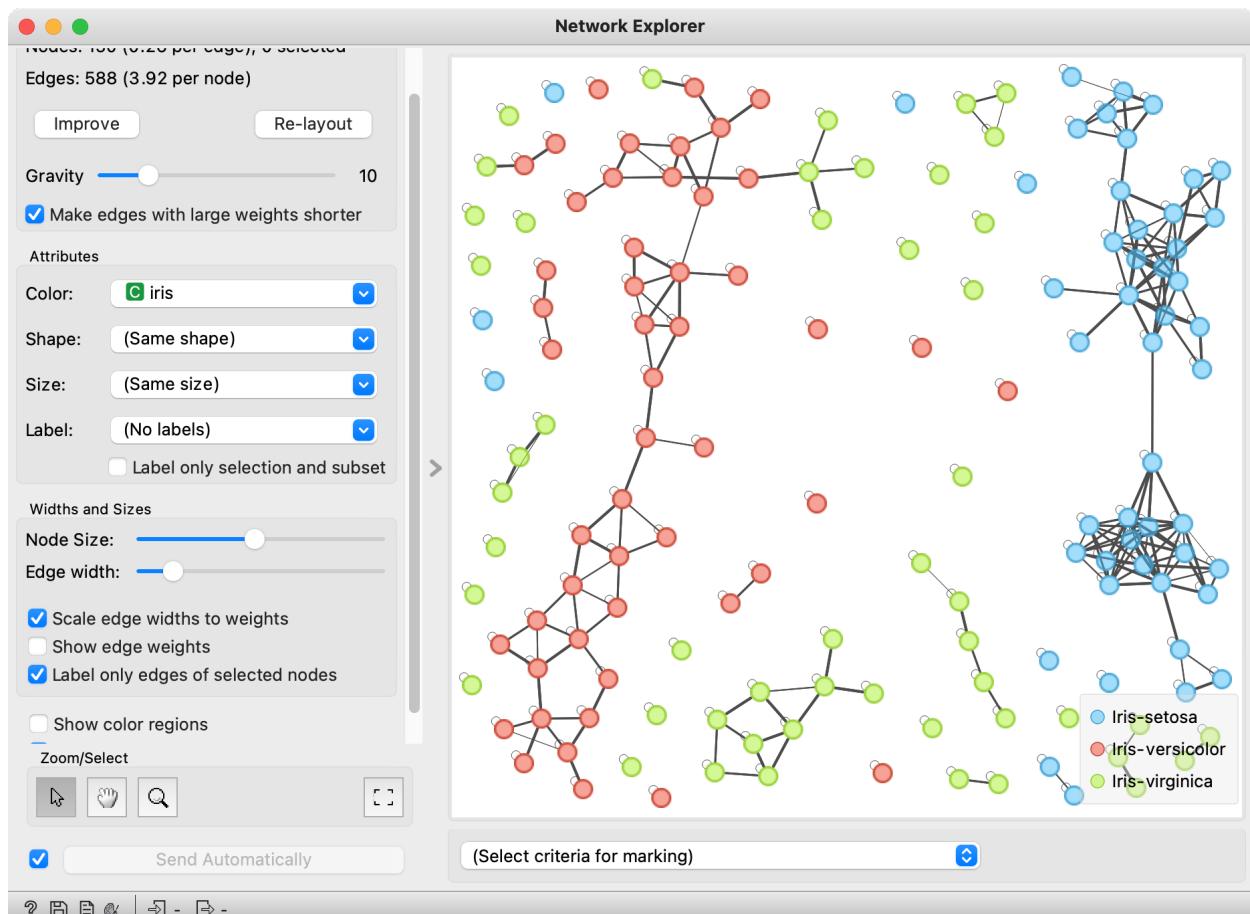


First, we need distance matrix, which we can compute with the *Distances* widget. Let's stick with Euclidean distances as it works well with this 4-dimensional data set.

Then pass the distance matrix to *Network from Distances*. The widget enables setting the distances threshold, which we will set to 2.0 percentiles (meaning we keep the top 2% of the edges that connect the most similar nodes).



Finally, pass the constructed graph to *Network Explorer*. Remember, this is not a scatter plot. The position of the points is not mapped to any sort of attribute. In the graph, you can simply see which elements are most similar to one another.

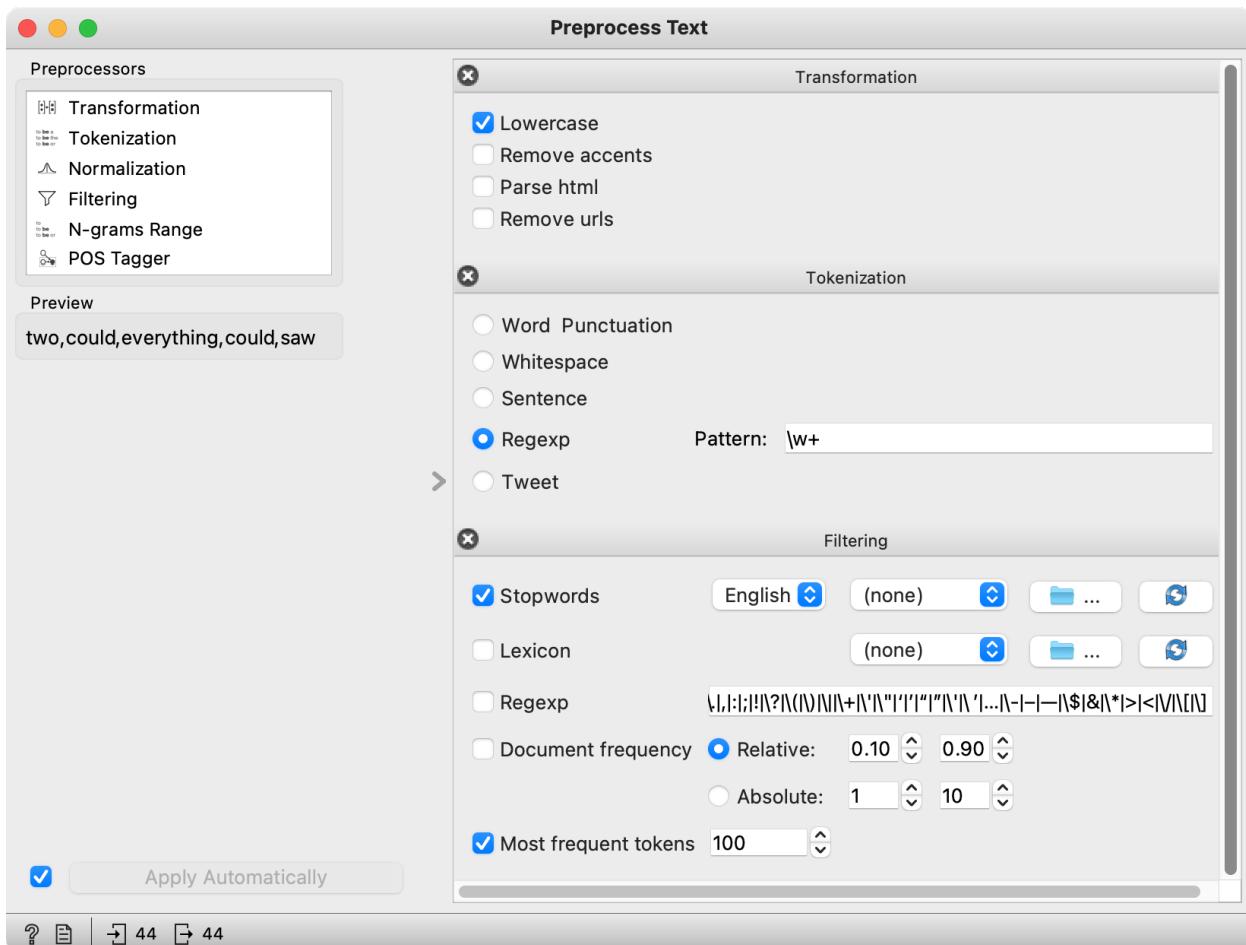


Network from Text

Now, let us try to generate networks from text. What do we mean by that? How can a text be transformed to a network? Well, documents are nodes, while edges can be the number of shared words. Again, we are back to similarity, but applied to text.

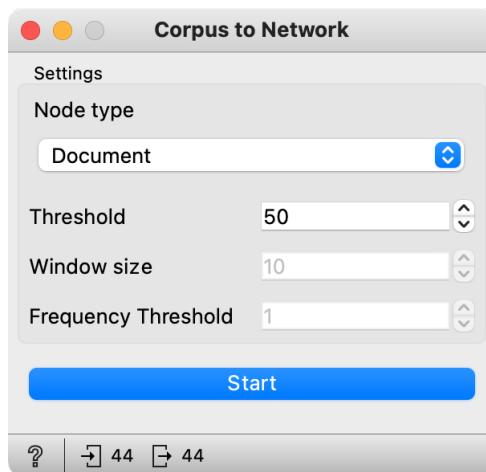
For this task, we will need the *Text add-on*. Load *grimm-tales-selected* with the *Corpus* widget. This data set contains 44 Grimm's tales, some of which are tales of magic and some are animal tales.

Every text needs to be preprocessed, that is we have to split the text to words and remove those words that have no meaning (such as stopwords). To speed up the analysis, we will keep only 100 most frequent tokens.

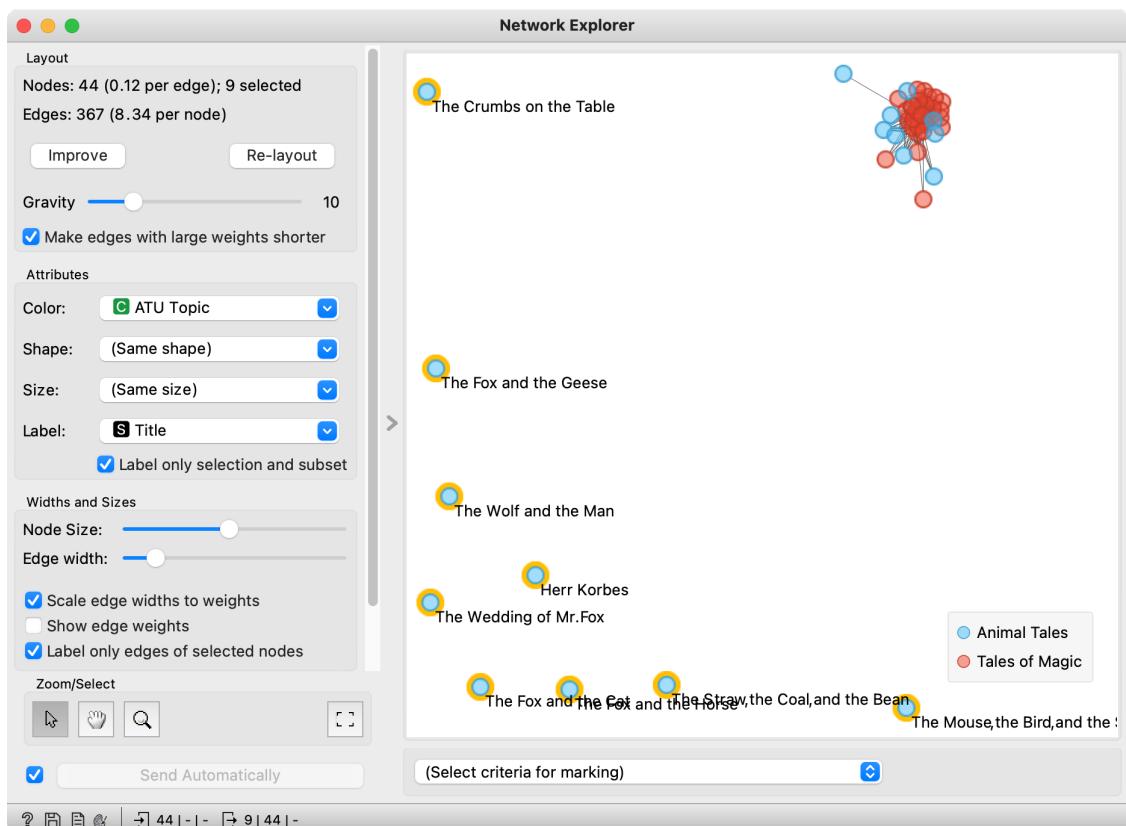


Now pass the data to *Corpus to Network*. With this widget, we will generate the graph. If we are generating network where nodes are documents, then we need to set a single parameter, namely the

threshold. This is similar to the similarity threshold in *Network from Distances*. *Threshold* will define how many words the documents have to share for them to have a connecting edge. In our case, we will set the threshold quite high - two documents have to share at least 50 words to be connected with an edge.

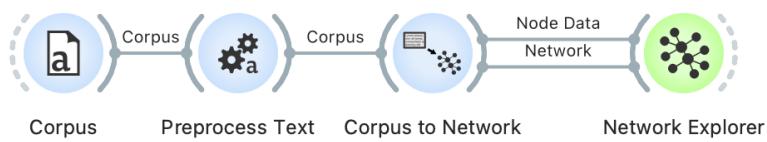


Let us observe the end result in *Network Explorer*. Seem like Tales of Magic are well-connected, even with some Animal Tale, while certain Animal Tales are quite distinct and don't share as many words with the other tales.



A task for the reader: play with the threshold and observe how

the graph changes. Does a lower threshold results in a more or less connected graph? What happens if words are used as Node types in *Corpus to Network*? What does such a graph show?



Bibliography

<https://github.com/biolab/orange3>, a.

<https://github.com/quasars>, b.

<https://quasar.codes>.

<https://quasar.codes>.

Robert Bringhurst. *The Elements of Typography*. Hartley & Marks, 3.1 edition, 2005. ISBN 0-88179-205-5.

Frank Mittelbach and Michel Goossens. *The L^AT_EX Companion*. Addison-Wesley, second edition, 2004. ISBN 0-201-36299-6.

Edward R. Tufte. *Envisioning Information*. Graphics Press, Cheshire, Connecticut, 1990. ISBN 0-9613921-1-8.

Edward R. Tufte. *Visual Explanations*. Graphics Press, Cheshire, Connecticut, 1997. ISBN 0-9613921-2-6.

Edward R. Tufte. *The Visual Display of Quantitative Information*. Graphics Press, Cheshire, Connecticut, 2001. ISBN 0-9613921-4-2.

Edward R. Tufte. *Beautiful Evidence*. Graphics Press, LLC, first edition, May 2006. ISBN 0-9613921-7-7.

Hideo Umeki. The *geometry* package. <http://ctan.org/pkg/geometry>, December 2008.

Index

license, [2](#)