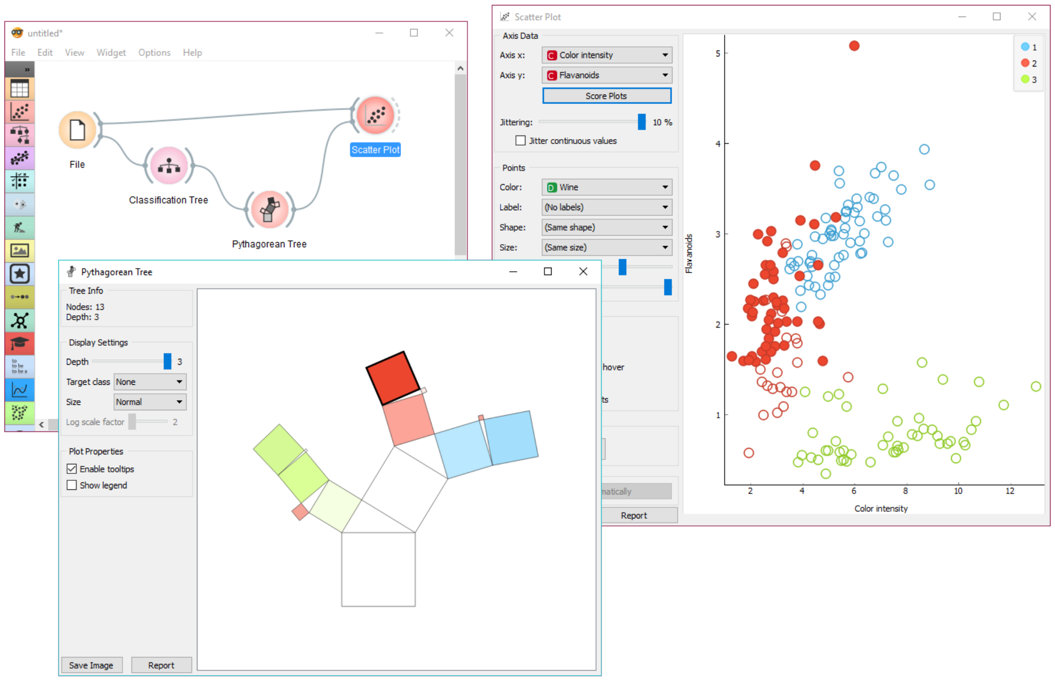


Introduction to Data Mining

Working notes for the hands-on course organized for students of engineering and natural sciences by BEST, Board of European Students of Technology

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

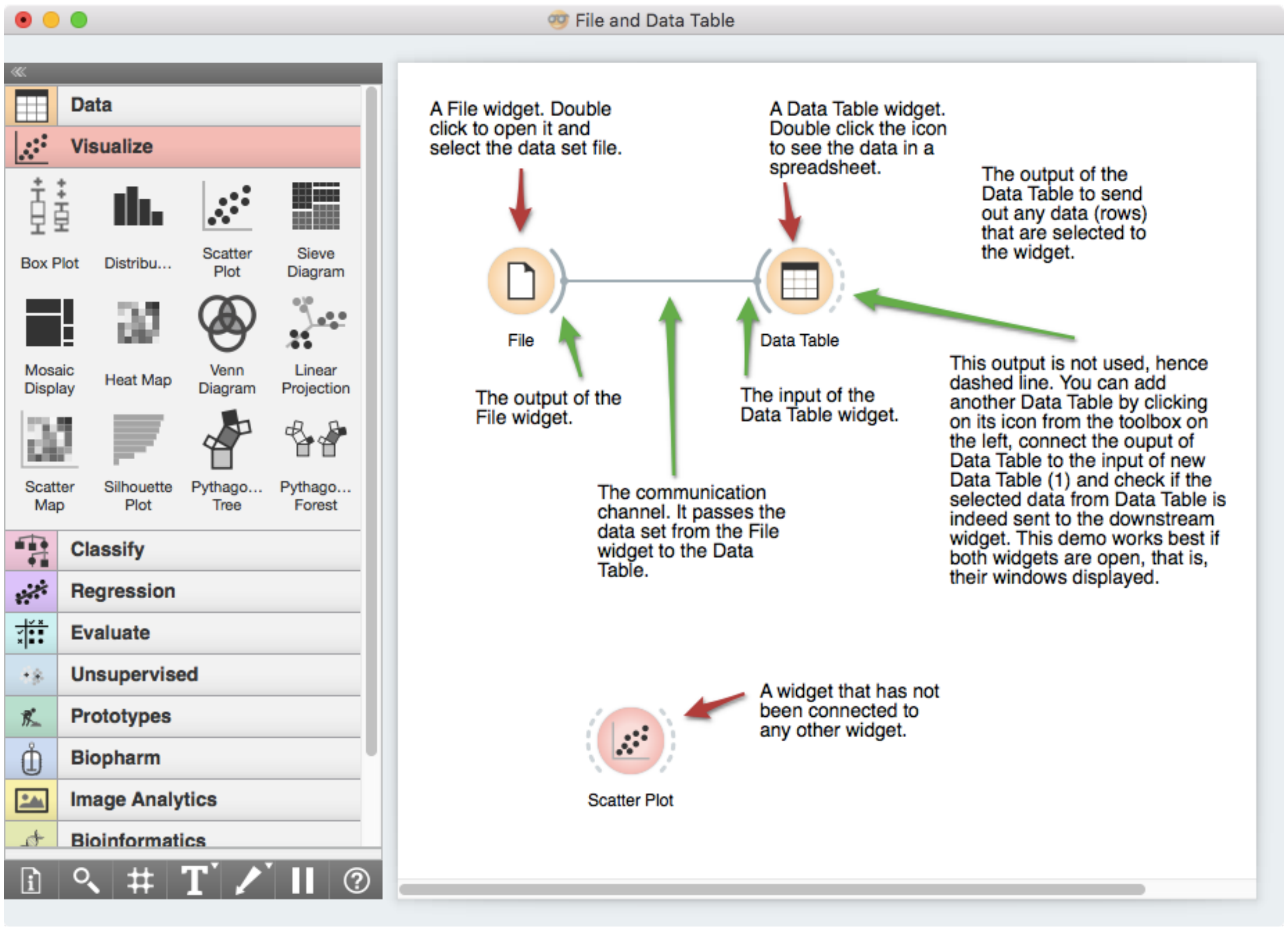
The working notes were prepared by Blaž Zupan and Janez Demšar with help from the members of the Bioinformatics Lab in Ljubljana that develop and maintain Orange.

Welcome to the course on Interactive and Visual Approaches to Data Mining! This course is designed for students and researchers of life sciences. You will see how common data mining tasks can be accomplished without programming. We will use Orange to construct visual data mining workflows. Many similar data mining environments exist, but the lecturer prefers Orange for one simple reason—he is one of its authors.

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

Lesson 1: Workflows in Orange

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



A File widget. Double click to open it and select the data set file.

A Data Table widget. Double click the icon to see the data in a spreadsheet.

The output of the Data Table to send out any data (rows) that are selected to the widget.

The output of the File widget.

The input of the Data Table widget.

This output is not used, hence dashed line. You can add another Data Table by clicking on its icon from the toolbox on the left, connect the ouput of Data Table to the input of new Data Table (1) and check if the selected data from Data Table is indeed sent to the downstream widget. This demo works best if both widgets are open, that is, their windows displayed.

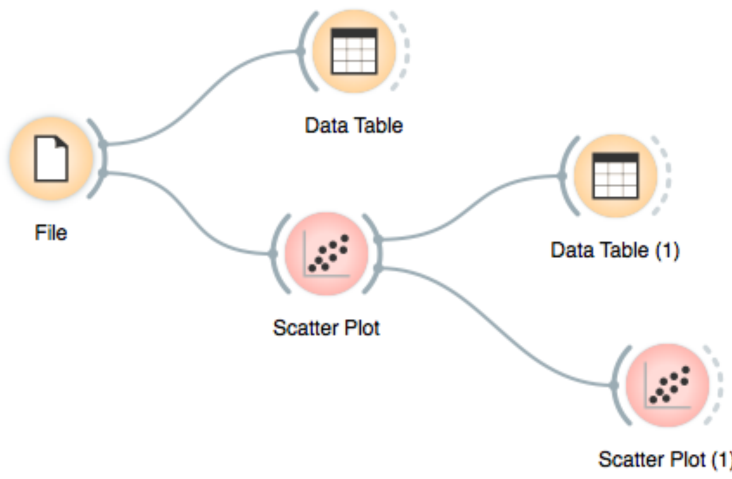
The communication channel. It passes the data set from the File widget to the Data Table.

A widget that has not been connected to any other widget.

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

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

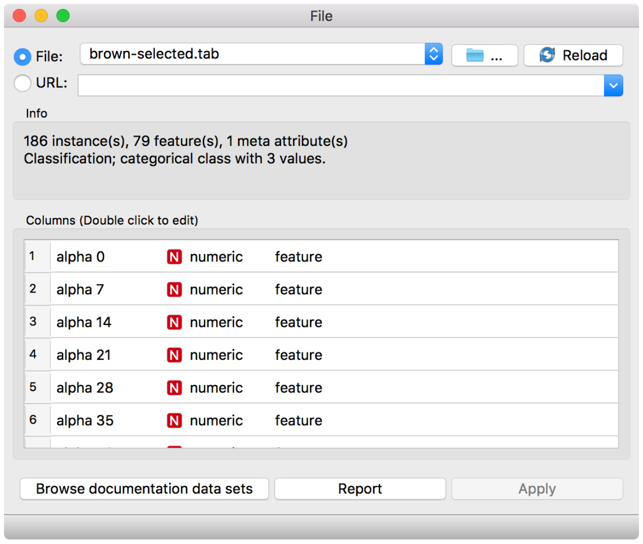
Start by constructing a workflow that consists of a File widget, two Scatter Plot widgets, and two Data Table widgets:



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

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

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


File
File:
URL:
...
Reload
Info
Columns (Double click to edit)
Browse documentation data sets
Report
Apply

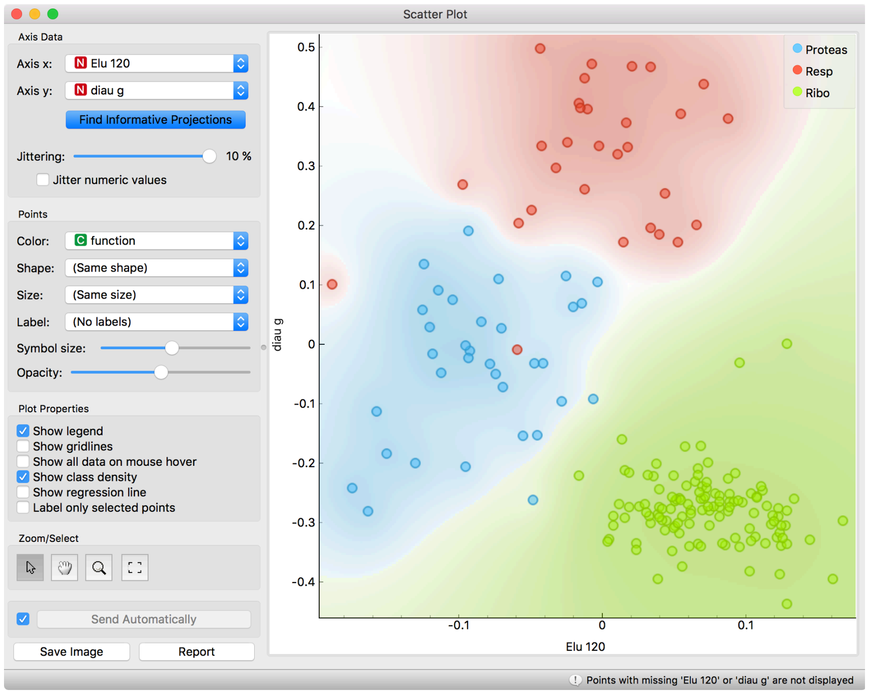
File: brown-selected.tab

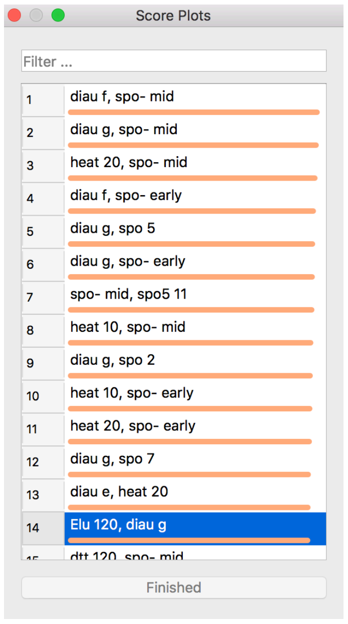
186 instance(s), 79 feature(s), 1 meta attribute(s)
Classification; categorical class with 3 values.

1	alpha 0	N numeric	feature
2	alpha 7	N numeric	feature
3	alpha 14	N numeric	feature
4	alpha 21	N numeric	feature
5	alpha 28	N numeric	feature
6	alpha 35	N numeric	feature

After you load the data, open the other widgets. In the Scatter Plot widget, select a few data points and watch as they appear in widget Data Table (1). Use a combination of two Scatter Plot widgets, where the second scatter plot shows a detail from a smaller region selected in the first scatterplot.

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

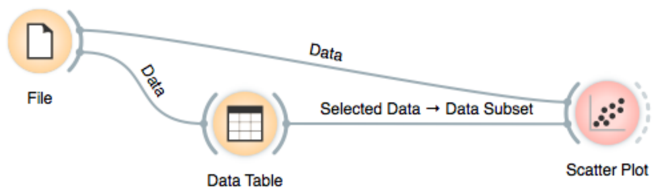




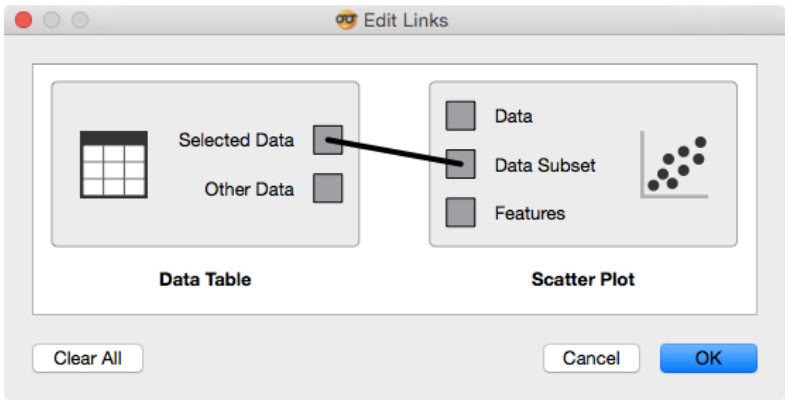
1	diau f, spo- mid
2	diau g, spo- mid
3	heat 20, spo- mid
4	diau f, spo- early
5	diau g, spo 5
6	diau g, spo- early
7	spo- mid, spo5 11
8	heat 10, spo- mid
9	diau g, spo 2
10	heat 10, spo- early
11	heat 20, spo- early
12	diau g, spo 7
13	diau e, heat 20
14	Elu 120, diau g
15	dtt 120, spo- mid

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

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

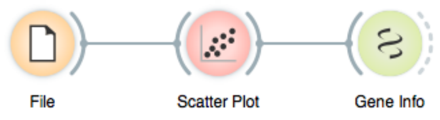


How does Orange distinguish between the primary data source and the data selection? It uses the first connected signal as the entire data set and the second one as its subset. To make changes or to check what is happening under the hood, double click on the line connecting the two widgets.



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

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

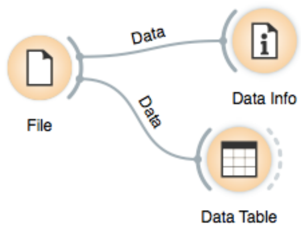


Lesson 2: Basic Data Exploration

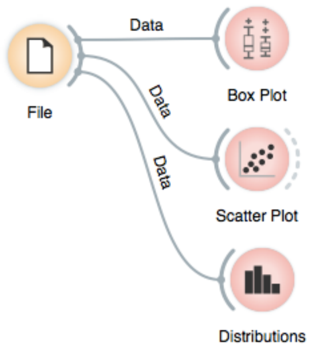
Let us consider another problem, this time from clinical medicine. We will dig for something interesting in the data and explore it a bit with various widgets. You will get to know Orange better and also learn about several interesting visualizations.

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

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

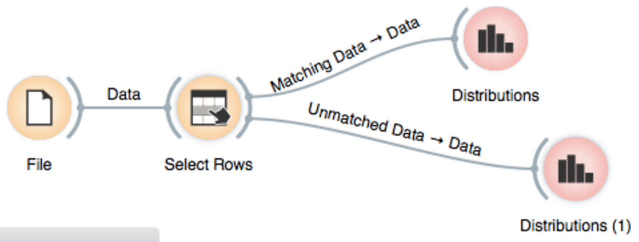


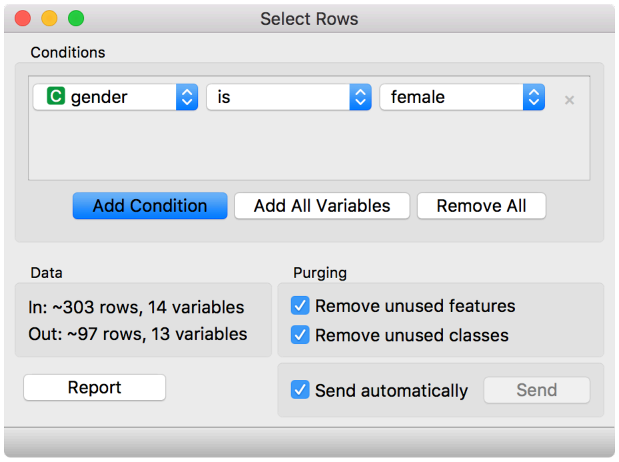
Let us check whether standard visualizations tell us anything interesting. (Hint: look for gender differences. These are always interesting and occasionally even real.)



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

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




Select Rows

Conditions

gender is female

Add Condition · Add All Variables · Remove All

Data

In: ~303 rows, 14 variables
Out: ~97 rows, 13 variables

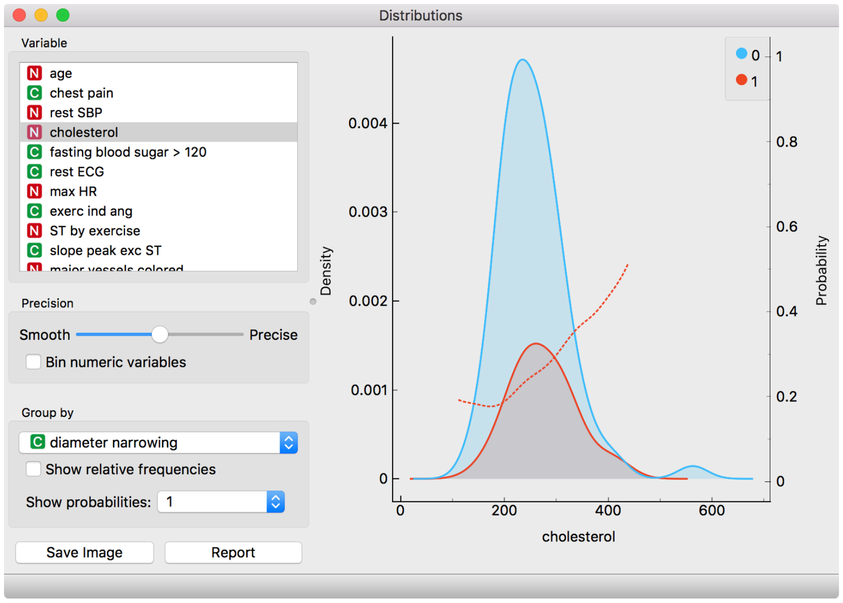
Purging

✓ Remove unused features
✓ Remove unused classes

Report

✓ Send automatically · Send

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


Distributions

Variable: age, chest pain, rest SBP, cholesterol (selected), fasting blood sugar > 120, rest ECG, max HR, exerc ind ang, ST by exercise, slope peak exc ST, major vessels colored

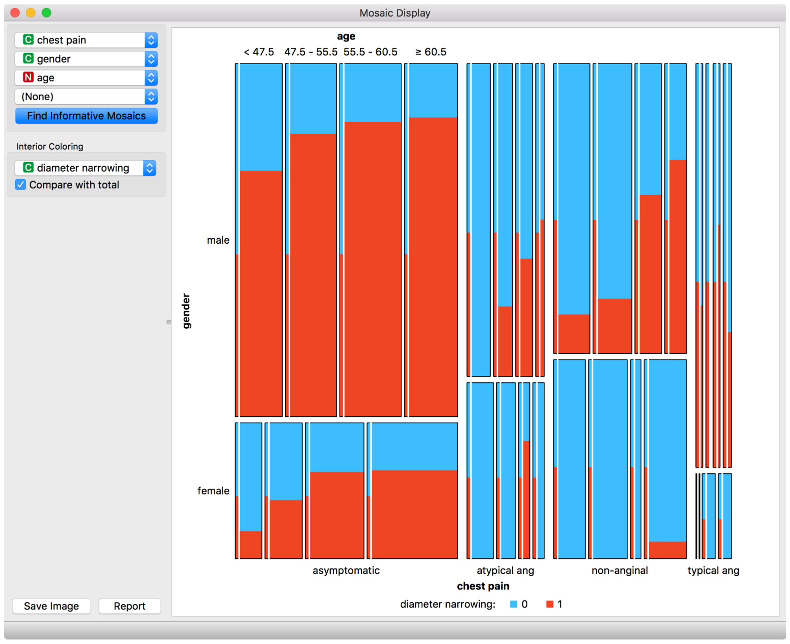
Precision: Smooth — Precise; ☐ Bin numeric variables

Group by: diameter narrowing; ☐ Show relative frequencies; Show probabilities: 1

Save Image · Report

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

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

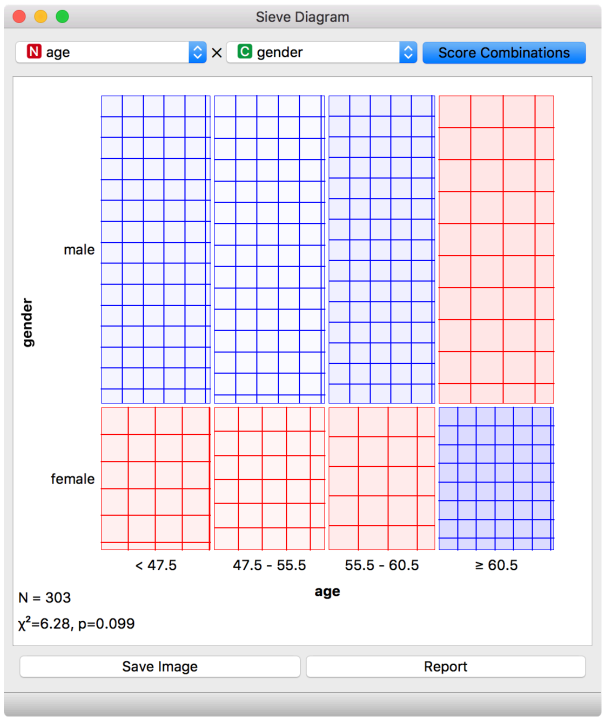


The mosaic display shows a rectangle split into columns with widths reflecting the prevalence of different types of chest pain. Each column is then further split vertically according to gender distributions within the column. The resulting rectangles are divided again horizontally according to age group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

What can you read from this diagram?

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

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

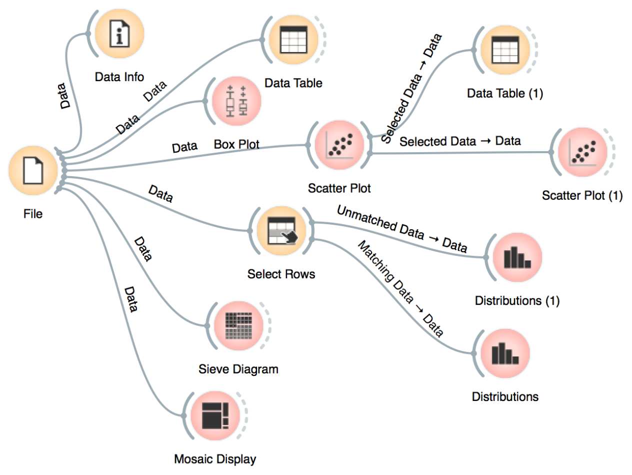


N = 303

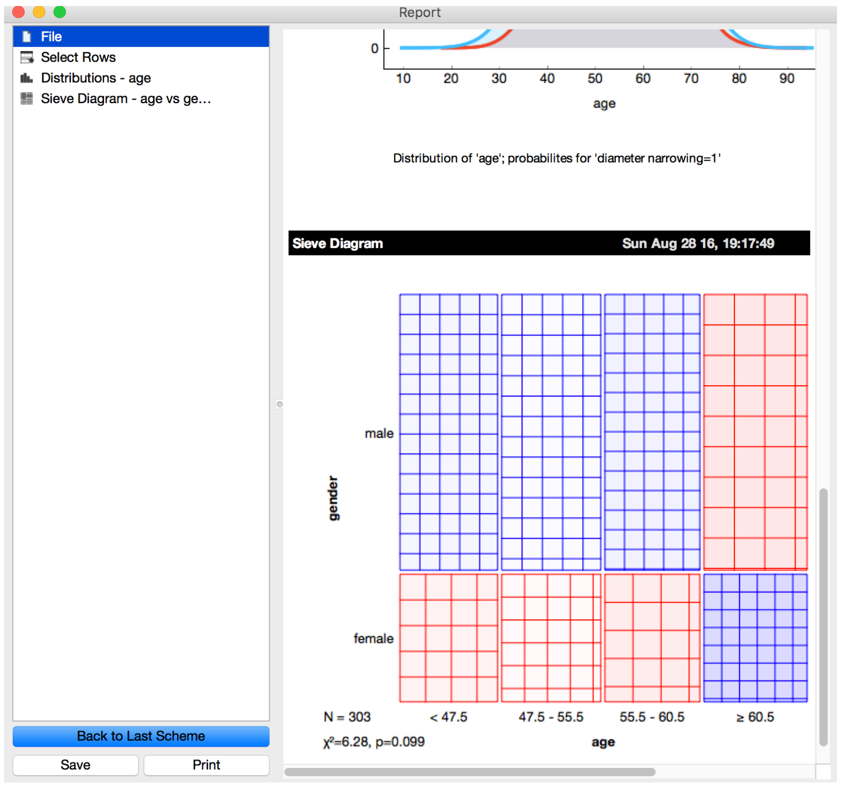
χ^2=6.28, p=0.099

Lesson 3: Saving Your Work

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



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

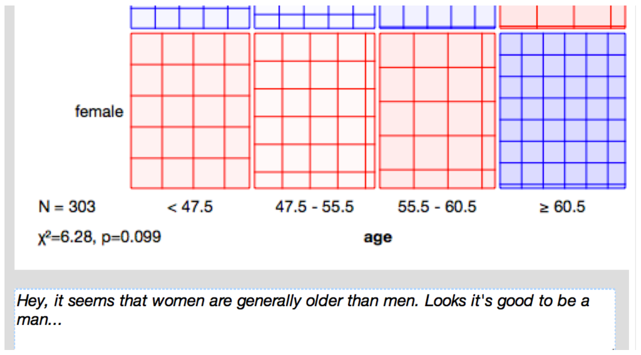


One more trick: Pressing Ctrl-C (or ⌘-C, on Mac) copies a visualization to the clipboard, so you can paste it to another application.

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

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

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

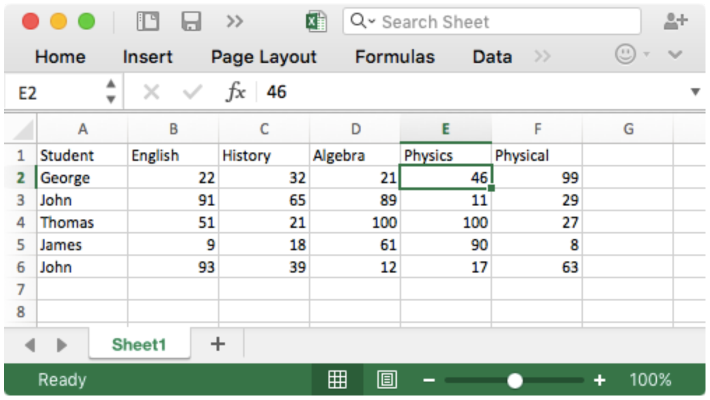


Hey, it seems that women are generally older than men. Looks it's good to be a man...

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

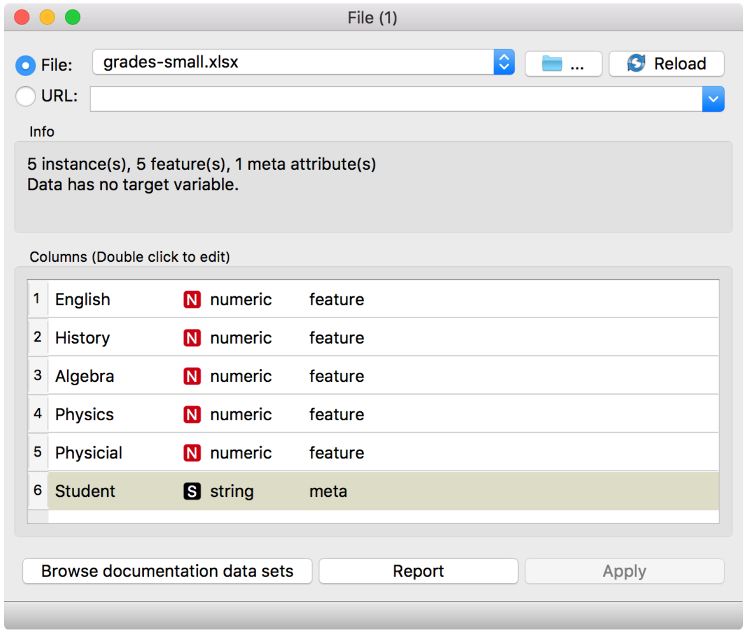
Lesson 4: Loading Your Own Data Set

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



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

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


File (1)
File:
URL:
Reload
Info
Columns (Double click to edit)
Browse documentation data sets
Report
Apply

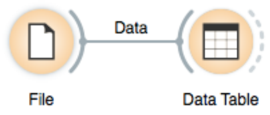
File: grades-small.xlsx

5 instance(s), 5 feature(s), 1 meta attribute(s)
Data has no target variable.

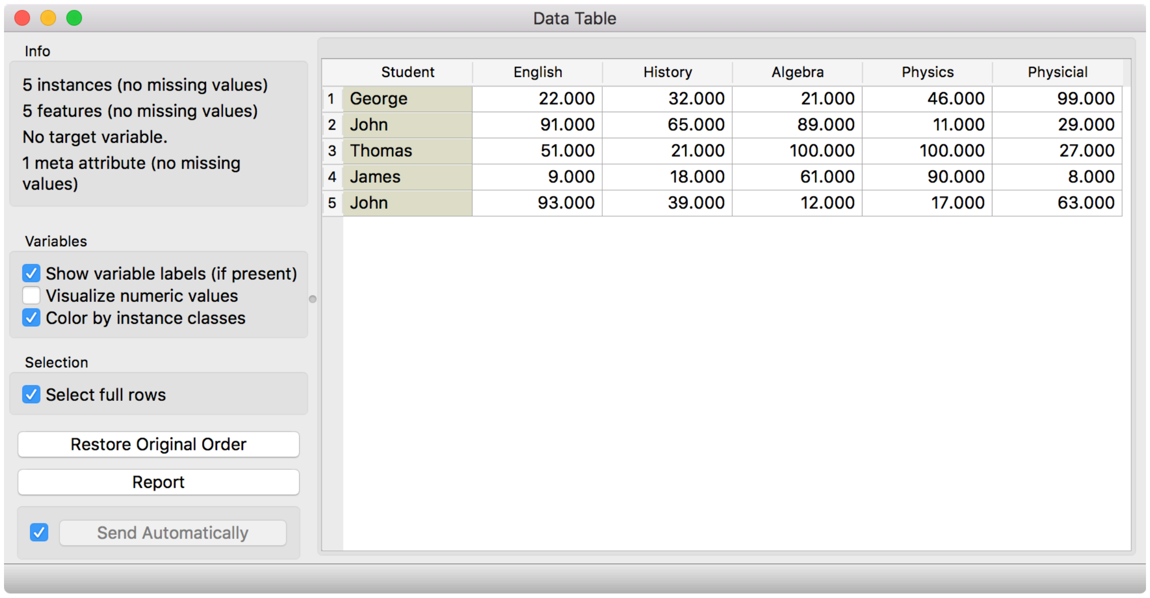
1	English	N numeric	feature
2	History	N numeric	feature
3	Algebra	N numeric	feature
4	Physics	N numeric	feature
5	Physicial	N numeric	feature
6	Student	S string	meta

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

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



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



Info

5 instances (no missing values)
5 features (no missing values)
No target variable.
1 meta attribute (no missing values)

Variables

- [✓] Show variable labels (if present)
- [ ] Visualize numeric values
- [✓] Color by instance classes

Selection

- [✓] Select full rows

Restore Original Order

Report

[✓] Send Automatically

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

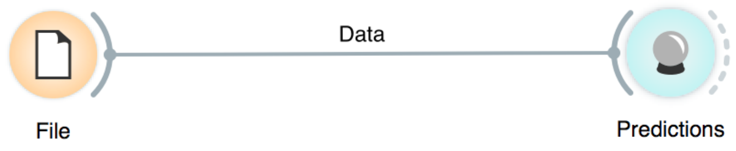
Nice, everything is here.

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

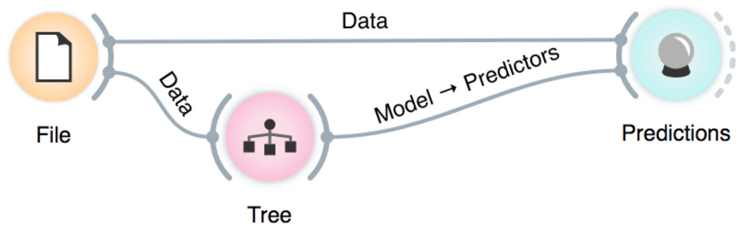
There is more to input data formatting and loading. We can define the type and kind of the data column, specify that the column is a web address of an image, and more. But enough for the first day. If you would like to dive deeper, check out the documentation page on Loading your Data, or a video on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

Lesson 5: Classification

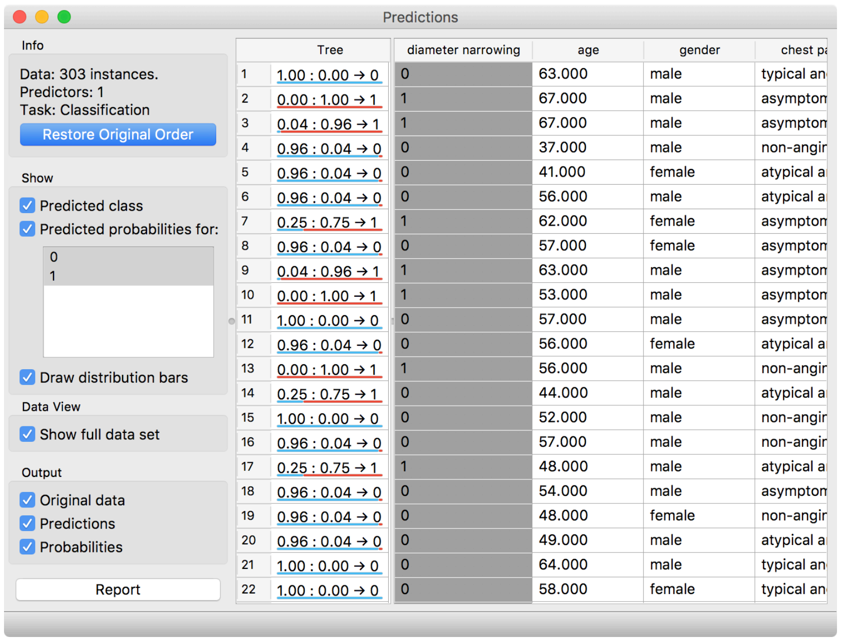
In one of the previous lessons, we explored the heart disease data. We wanted to predict which persons have clogged arteries — but we did not make any predictions. Let's try it now.



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



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



Data: 303 instances.
Predictors: 1
Task: Classification

✓ Predicted class
✓ Predicted probabilities for:
0
1
✓ Draw distribution bars

✓ Show full data set

✓ Original data
✓ Predictions
✓ Probabilities

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

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

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

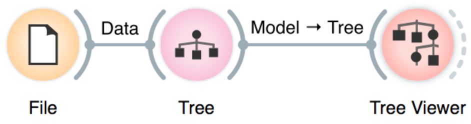
But (and even before answering these critical questions), what is a tree? How does it look like? How does Orange create one? Is this algorithm something we should use? So many questions to answer today!

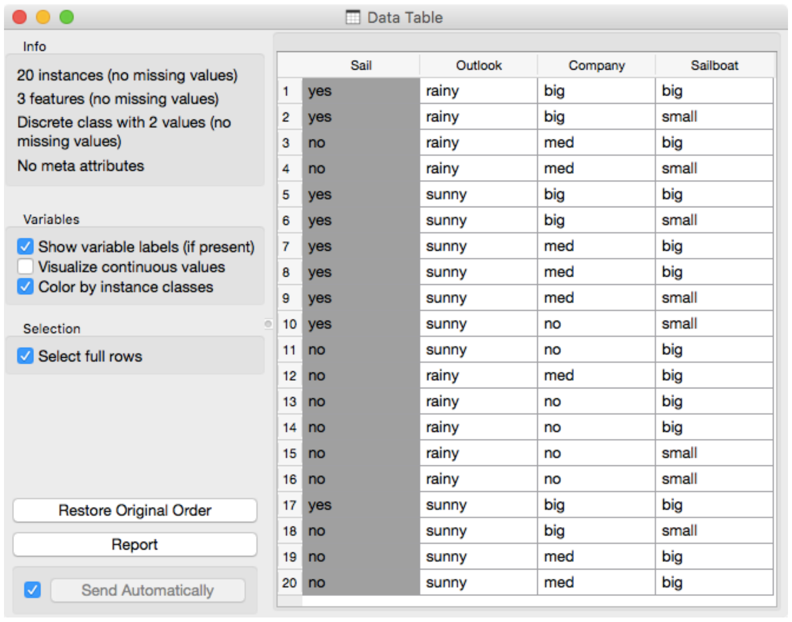
Lesson 6: Classification Trees

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

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

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





Info

20 instances (no missing values)
3 features (no missing values)
Discrete class with 2 values (no missing values)
No meta attributes

Variables

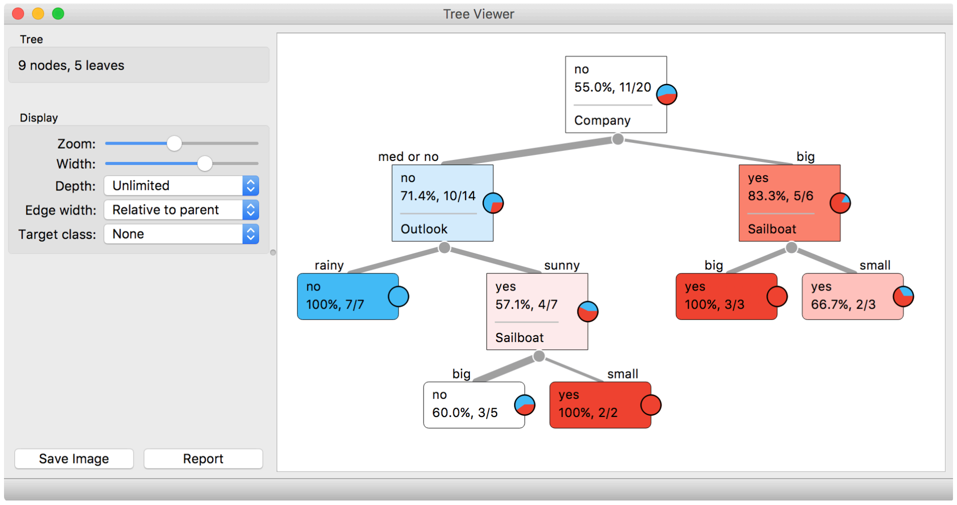
- [✓] Show variable labels (if present)
- [ ] Visualize continuous values
- [✓] Color by instance classes

Selection

- [✓] Select full rows

	Sail	Outlook	Company	Sailboat
1	yes	rainy	big	big
2	yes	rainy	big	small
3	no	rainy	med	big
4	no	rainy	med	small
5	yes	sunny	big	big
6	yes	sunny	big	small
7	yes	sunny	med	big
8	yes	sunny	med	big
9	yes	sunny	med	small
10	yes	sunny	no	small
11	no	sunny	no	big
12	no	rainy	med	big
13	no	rainy	no	big
14	no	rainy	no	big
15	no	rainy	no	small
16	no	rainy	no	small
17	yes	sunny	big	big
18	no	sunny	big	small
19	no	sunny	med	big
20	no	sunny	med	big

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



Tree

9 nodes, 5 leaves

Display

Depth: Unlimited
Edge width: Relative to parent
Target class: None

We read the tree from top to bottom. It looks like this skipper is a social person; as soon as there's company, the probability of her sailing increases. When joined by a smaller group of individuals, there is no sailing if there is rain. (Thunderstorms? Too dangerous?) When she has a smaller company, but the boat at her disposal is big, there is no sailing either.

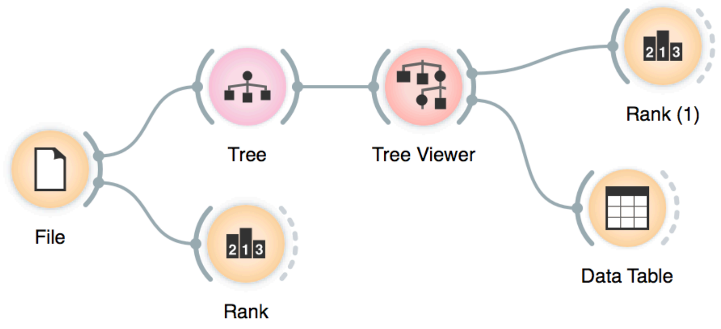
Classification trees were hugely

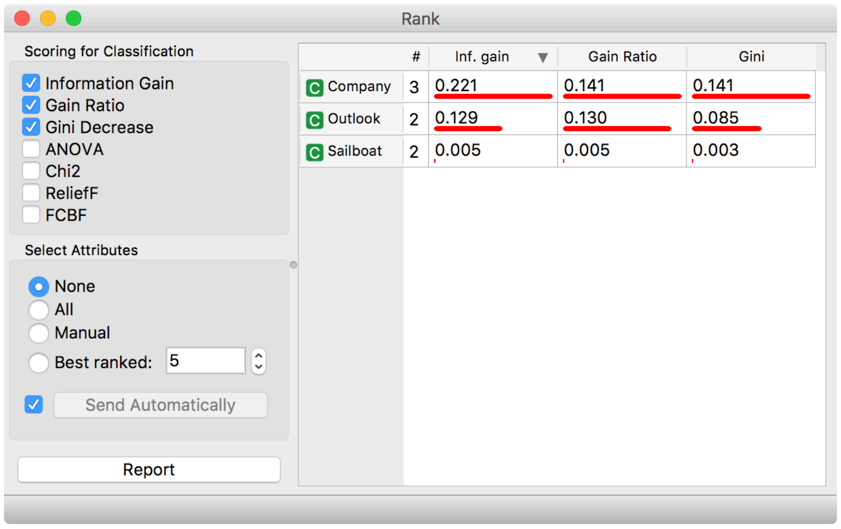
The Rank widget could be used on its own. Say, to figure out which genes are best predictors of the phenotype in some gene expression data set. Or what experimental conditions to consider to profile the genes and assign their function. Oh, but we have already worked with a data set of this kind. What does Rank tell us about it?

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

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

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




Rank

Scoring for Classification
- [✓] Information Gain
- [✓] Gain Ratio
- [✓] Gini Decrease
- [ ] ANOVA
- [ ] Chi2
- [ ] ReliefF
- [ ] FCBF

Select Attributes
- [✓ None]
- All
- Manual
- Best ranked: 5

[✓] Send Automatically

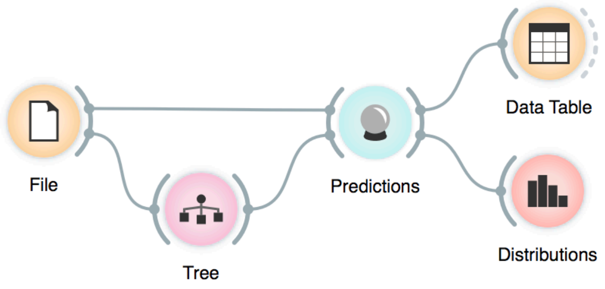
Report

	#	Inf. gain ▼	Gain Ratio	Gini
C Company	3	0.221	0.141	0.141
C Outlook	2	0.129	0.130	0.085
C Sailboat	2	0.005	0.005	0.003

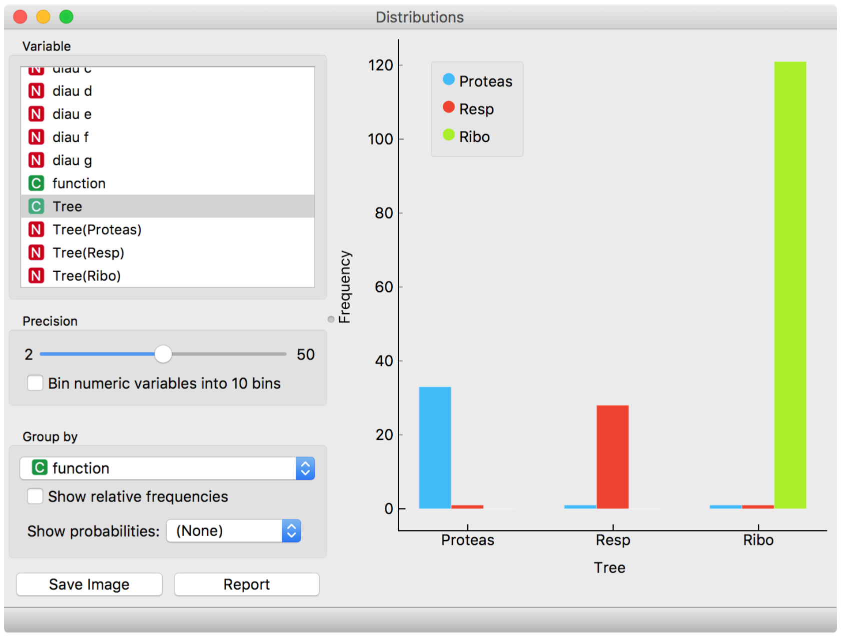
Lesson 7: Classification Accuracy

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

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



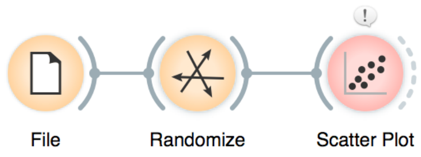
Let us try this schema with the brown-selected data set. The Predictions widget outputs a data table augmented with a column that includes predictions. In the Data Table widget, we can sort the data by any of these two columns, and manually select data instances where the values of these two features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



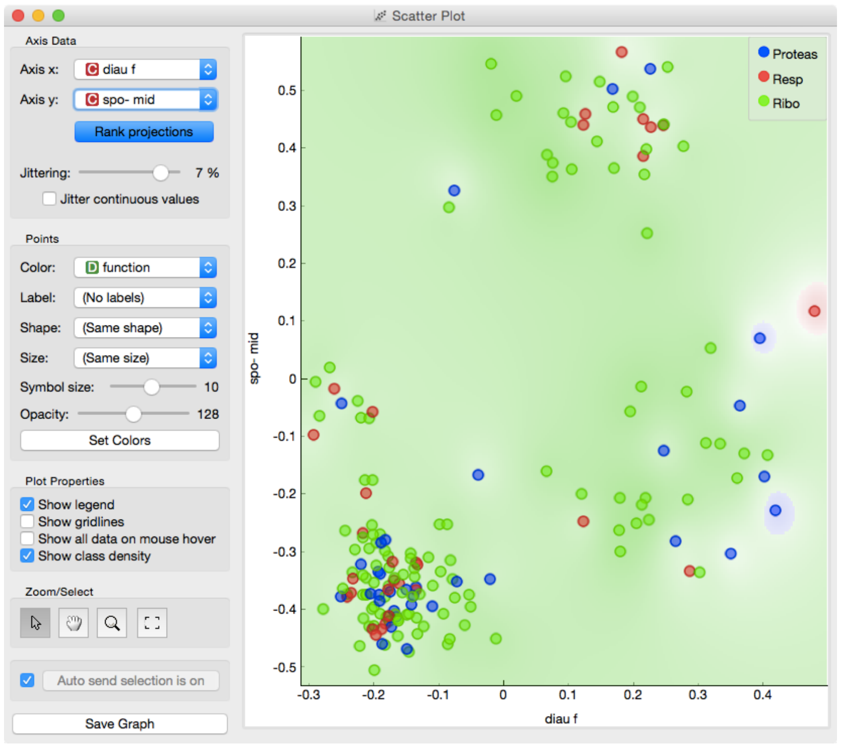
Lesson 8: How to Cheat

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

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

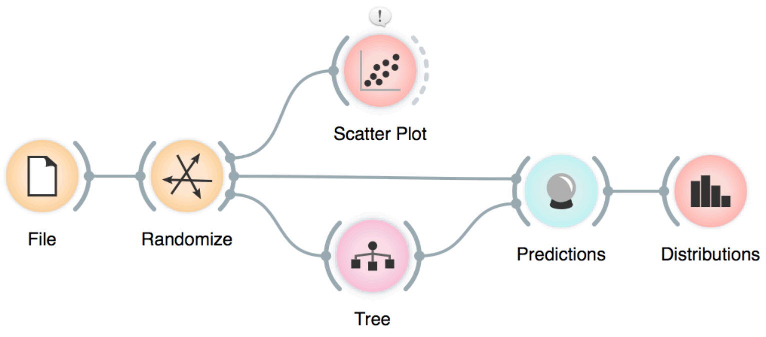


Randomize widget shuffles the column in the data table. It can shuffle the class column, columns with data features or columns with meta information. Shuffling the class column breaks any relation between features and the class, keeping the data points (genes profiles) intact.



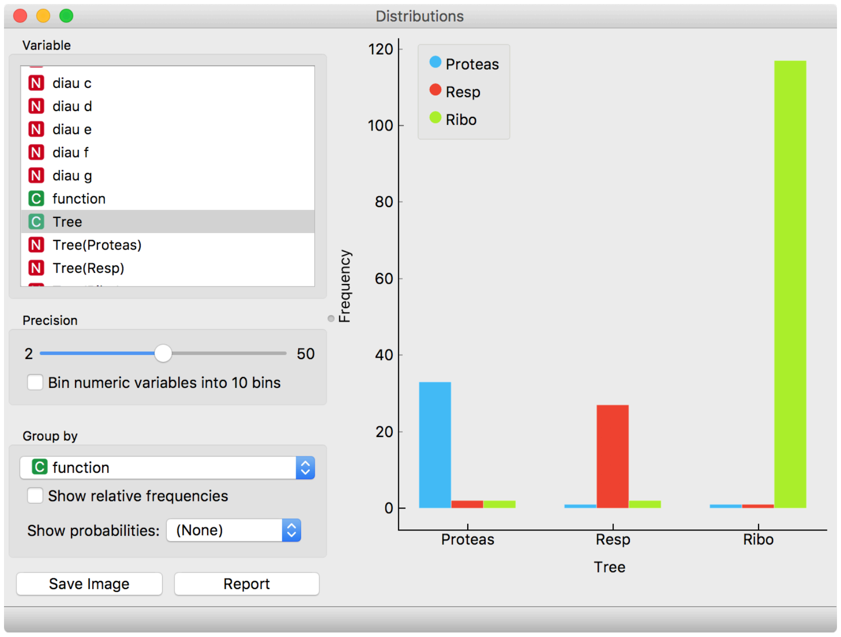
Why is the background in this scatter plot so green, and only green? Why have the other colors disappeared after the class randomization?

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



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

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



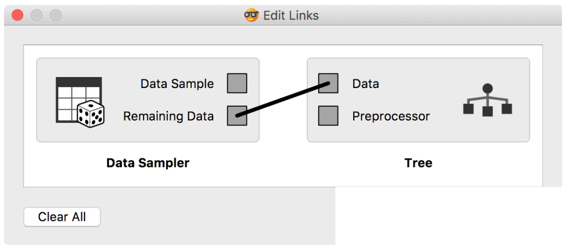
Most unusual. Almost no mistakes. How is this possible? On a class-randomized data set?

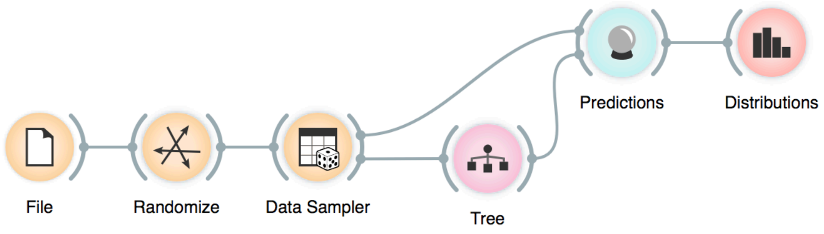
The signals from the Data Sampler widget have not been named in our workflow to save space. The Data Sampler splits the data to a sample and out-of-sample (so called remaining data). The sample was given to the Tree widget, while the remaining data was handed to the Predictions widget. Set the Data Sampler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

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

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

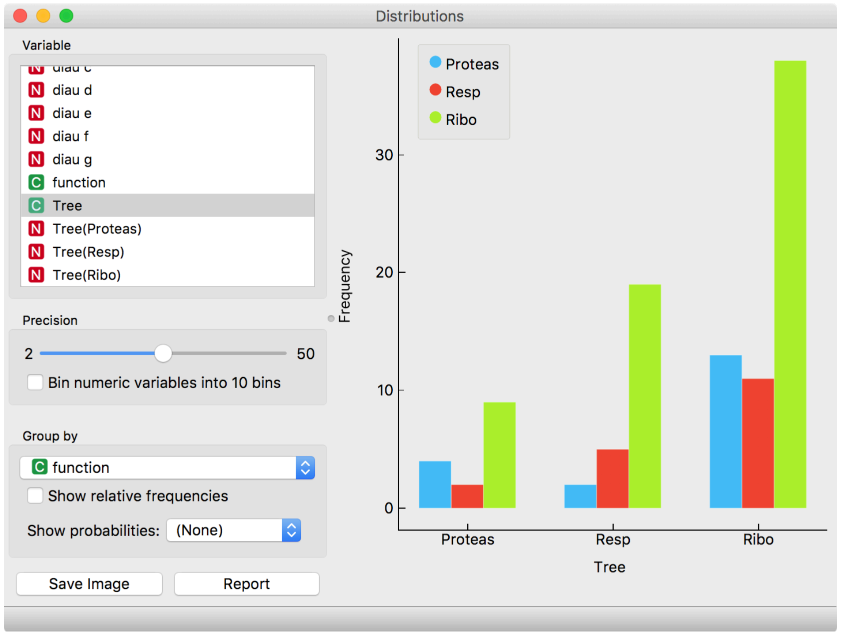
Ha, if this is so, that is, if a classifier remembers everything from a data set but without discovering any general patterns, it should perform miserably on any new data set. Let us check this out. We will split our data set into two sets, training and testing, train the classification tree on the training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





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

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



The first two classes are a complete fail. 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, this is just as we would expect.

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

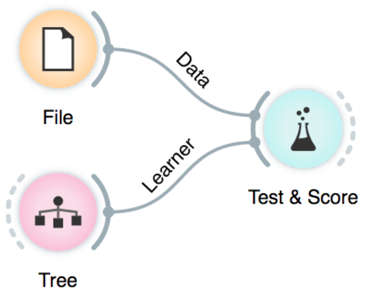
We needed to class-randomize only the training data set to fail in predictions. Try changing the workflow so that the classes are randomized only there, and not in the test set.

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

Lesson 9: Cross-Validation

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

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

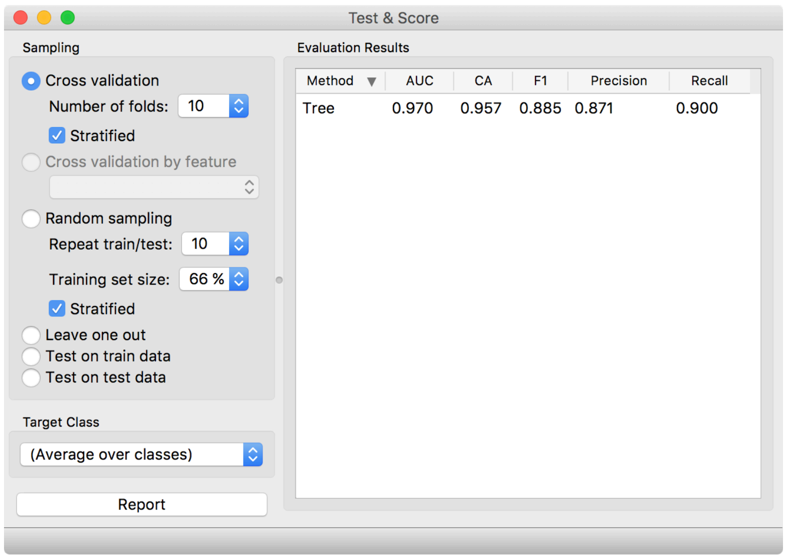


For geeks: a learner is an object that, given the data, outputs a classifier. Just what Test & Score needs.

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

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.

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



Sampling

- (●) Cross validation
  - Number of folds: 10
  - [✓] Stratified
- ( ) Cross validation by feature
- ( ) Random sampling
  - Repeat train/test: 10
  - Training set size: 66 %
  - [✓] Stratified
- ( ) Leave one out
- ( ) Test on train data
- ( ) Test on test data

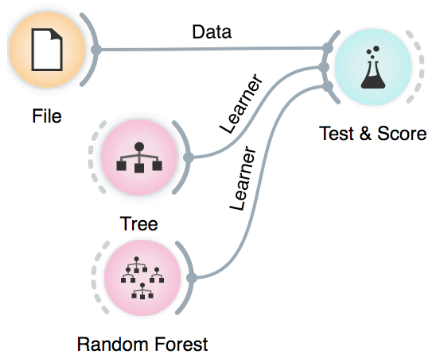
Target Class: (Average over classes)

Evaluation Results

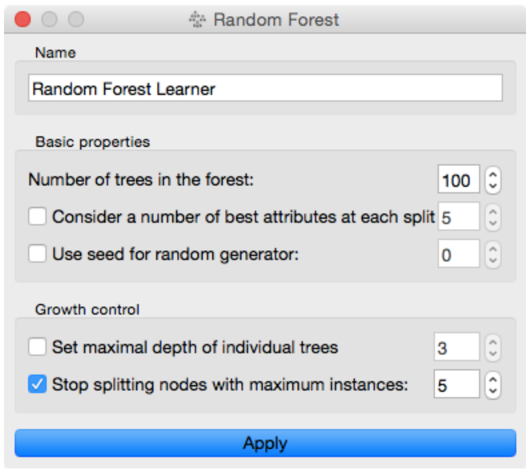
Method	AUC	CA	F1	Precision	Recall
Tree	0.970	0.957	0.885	0.871	0.900

Lesson 10: A Few More Classifiers

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.



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


Random Forest

Name

Random Forest Learner

Basic properties

Number of trees in the forest: 100

☐ Consider a number of best attributes at each split 5

☐ Use seed for random generator: 0

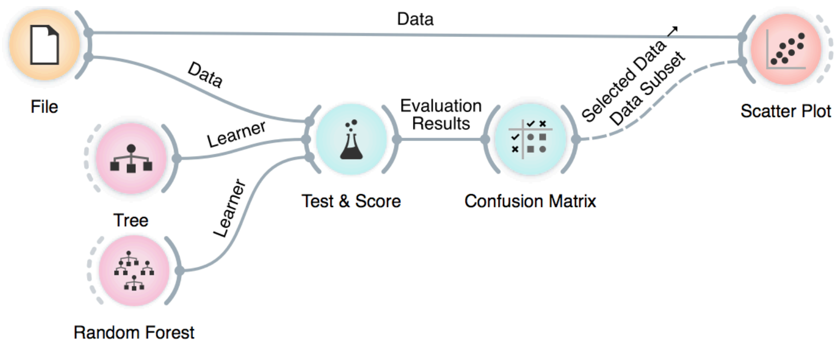
Growth control

☐ Set maximal depth of individual trees 3

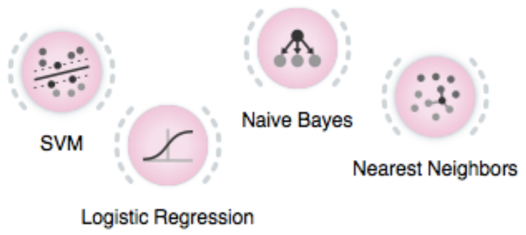
☑ Stop splitting nodes with maximum instances: 5

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

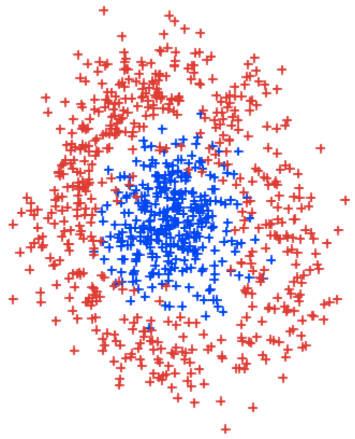
What kind of object is sent from the Test & Score widget to the Confusion Matrix widget? So far, we have used widgets that send data, or even learners. But what could the Test & Score widget communicate to other widgets?



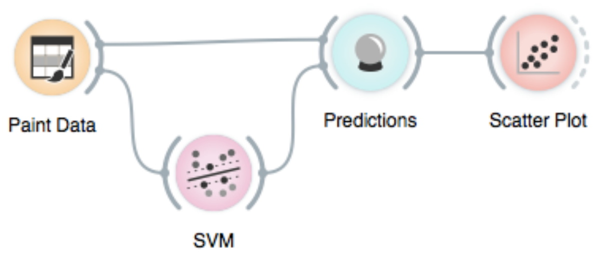
There are other classifiers we can try. We will briefly mention a few more, but instead of diving into what they do (we could spend a semester on this!), we'll pass on to other important topics in data mining. At this point, just add them to the workflow above and see how they perform.

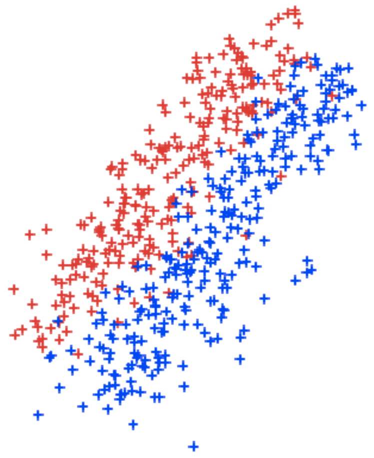


It would be nice if we could, at least on the intuitive level, understand the differences between all these methods and their variants (every method has some parameters). Remember, the classification tree finds hyperplanes orthogonal to the axis; those hyperplanes split the data space to regions with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

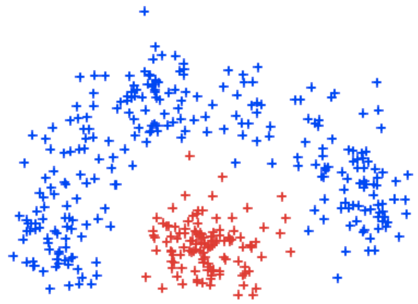


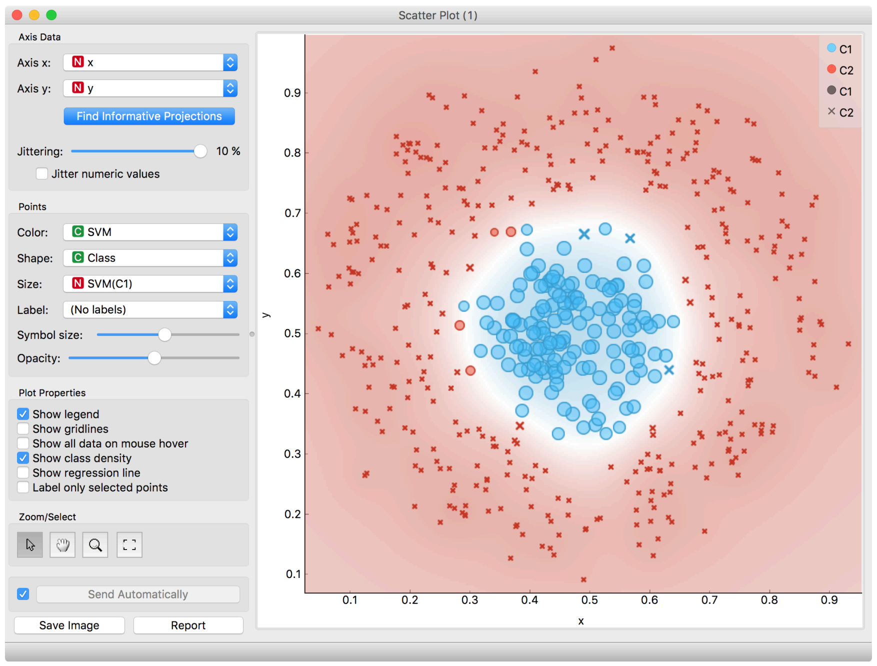
Ok, we have to admit: the above paragraph reads almost like gibberish. We would need a workflow where we could actually see the decision boundaries. And perhaps invent the data sets to test the classifiers. Best in 2D. Maybe, for a start, we could just paint the data. Time to stop writing this long passage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





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





Scatter Plot (1)

Axis Data

Axis x: x

Axis y: y

Find Informative Projections

Jittering: 10 %

Jitter numeric values

Points

Color: SVM

Shape: Class

Size: SVM(C1)

Label: (No labels)

Symbol size:

Opacity:

Plot Properties

- [✓] Show legend
- [ ] Show gridlines
- [ ] Show all data on mouse hover
- [✓] Show class density
- [ ] Show regression line
- [ ] Label only selected points

Zoom/Select

[✓] Send Automatically

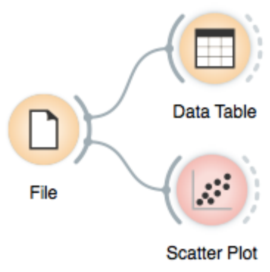
Save Image

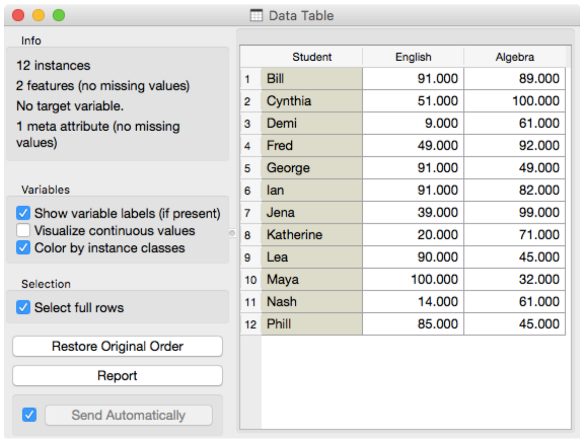
Report

Lesson 11: Hierarchical Clustering

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

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





Info

12 instances
2 features (no missing values)
No target variable.
1 meta attribute (no missing values)

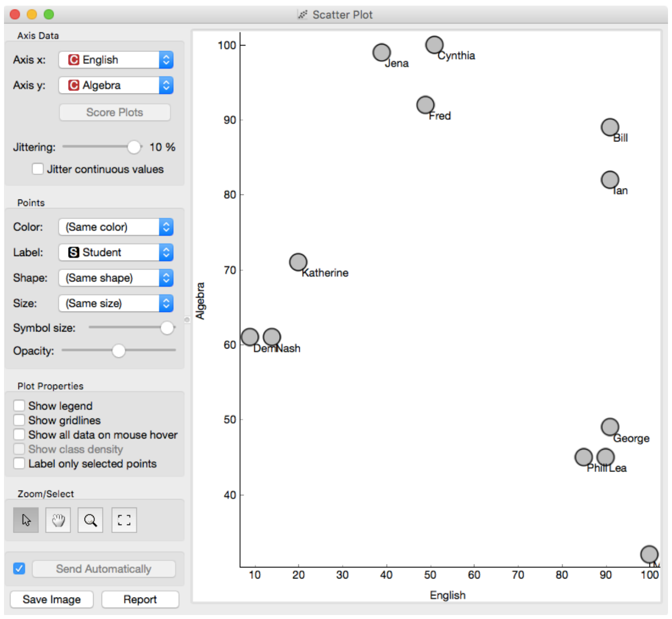
Variables

- [✓] Show variable labels (if present)
- [ ] Visualize continuous values
- [✓] Color by instance classes

Selection

- [✓] Select full rows

	Student	English	Algebra
1	Bill	91.000	89.000
2	Cynthia	51.000	100.000
3	Demi	9.000	61.000
4	Fred	49.000	92.000
5	George	91.000	49.000
6	Ian	91.000	82.000
7	Jena	39.000	99.000
8	Katherine	20.000	71.000
9	Lea	90.000	45.000
10	Maya	100.000	32.000
11	Nash	14.000	61.000
12	Phill	85.000	45.000



Axis Data
Axis x: English
Axis y: Algebra
Jittering: 10 %
- [ ] Jitter continuous values

Points
Color: (Same color)
Label: Student
Shape: (Same shape)
Size: (Same size)

Plot Properties
- [ ] Show legend
- [ ] Show gridlines
- [ ] Show all data on mouse hover
- [ ] Show class density
- [ ] Label only selected points

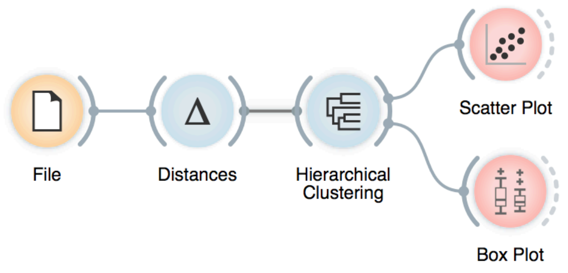
We need to start with a definition of "similar". One simple measure of similarity for such data is the Euclidean distance: square the differences across every dimension, some them and take the square root, just like in Pythagorean theorem. So, we would like to group data instances with small Euclidean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

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

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

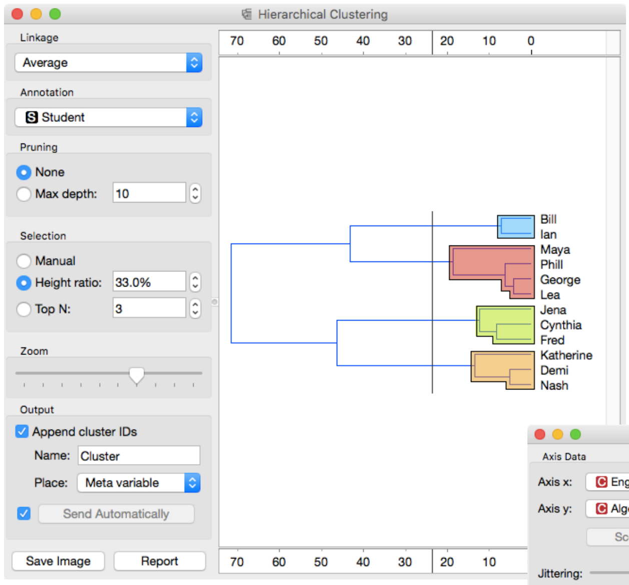
This procedure constructs a hierarchy of clusters, which explains why we call it hierarchical clustering. After it is done, we can observe the entire hierarchy and decide which would be a good point to stop. With this we decide the actual number of clusters.

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



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

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


Hierarchical Clustering

Linkage: Average

Annotation: Student

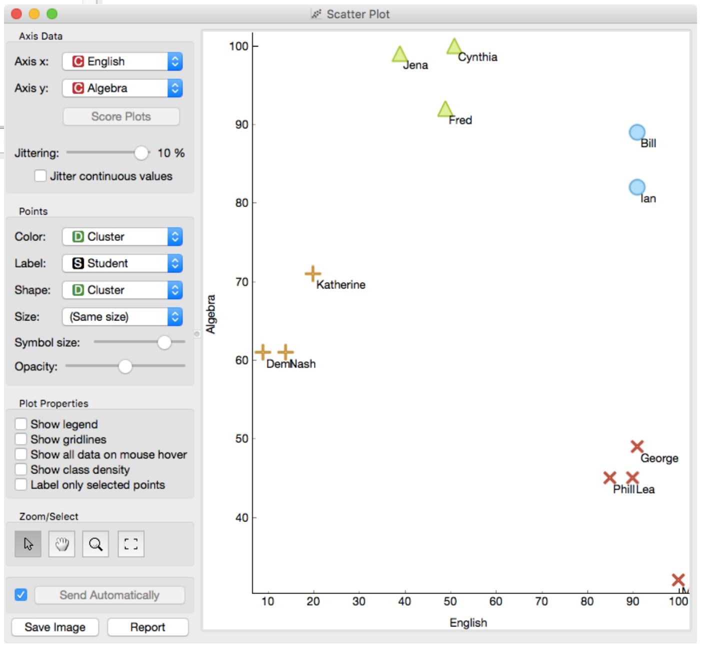
Pruning: [✓ None] Max depth: 10

Selection: Manual [✓ Height ratio: 33.0%] Top N: 3

Zoom

Output: ✓ Append cluster IDs; Name: Cluster; Place: Meta variable

✓ Send Automatically


Scatter Plot

Axis Data: Axis x: English; Axis y: Algebra

Jittering: 10 %

☐ Jitter continuous values

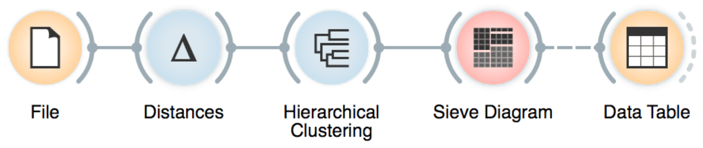
Points: Color: Cluster; Label: Student; Shape: Cluster; Size: (Same size); Symbol size; Opacity

Plot Properties: ☐ Show legend ☐ Show gridlines ☐ Show all data on mouse hover ☐ Show class density ☐ Label only selected points

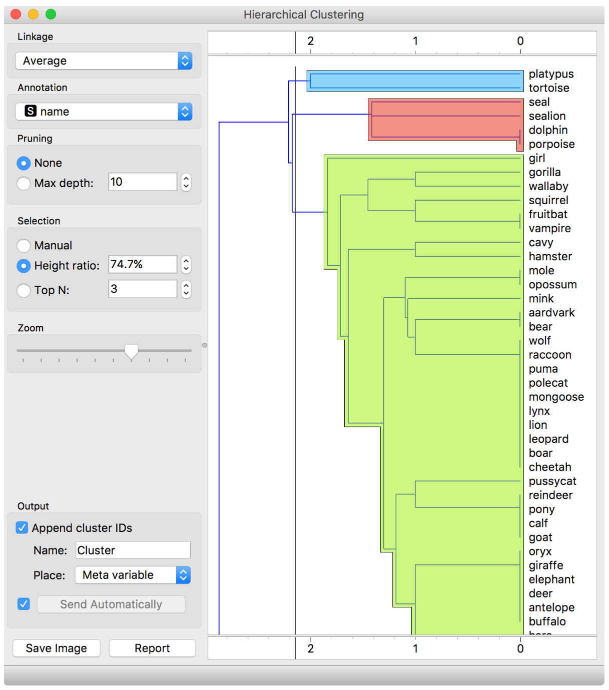
✓ Send Automatically

Lesson 12: Animal Kingdom

Your lecturers spent substantial part of their youth admiring a particular Croatian chocolate called Animal Kingdom. Each chocolate bar came with a card — a drawing of some (random) animal, and the associated album made us eat a lot of chocolate. Then our kids came, and the story repeated. Some things stay forever. Funny stuff was we never understood the order in which the cards were laid out in the album. We later learned about taxonomy, but being more inclined to engineering we never mastered learning it in our biology classes. Luckily, there's data mining and the idea that taxonomy simply stems from measuring the distance between species.



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



Hierarchical Clustering

Linkage: Average

Annotation: name

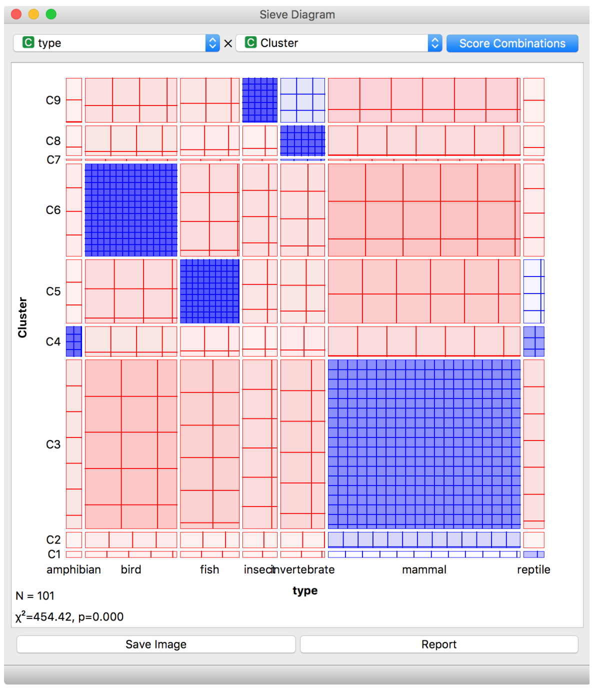
Pruning: [✓ None] Max depth: 10

Selection: Manual [✓ Height ratio: 74.7%] Top N: 3

Zoom

Output: [✓] Append cluster IDs, Name: Cluster, Place: Meta variable

[✓] Send Automatically



Sieve Diagram

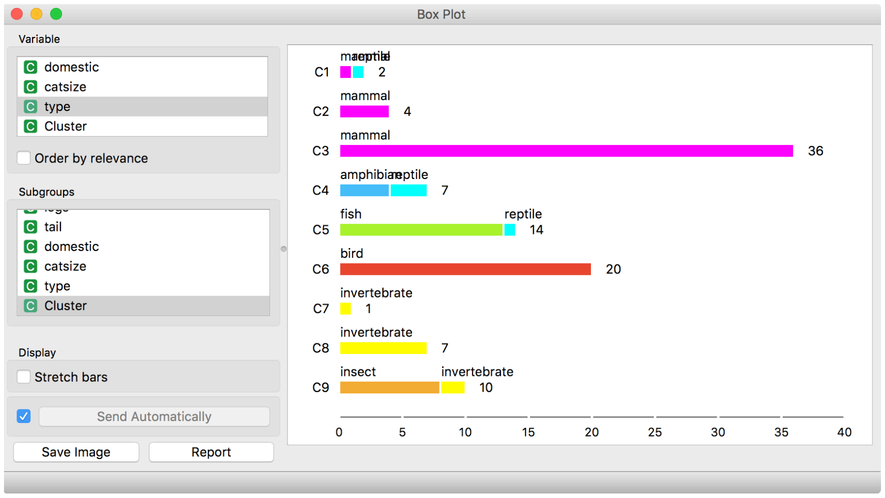
type × Cluster

N = 101

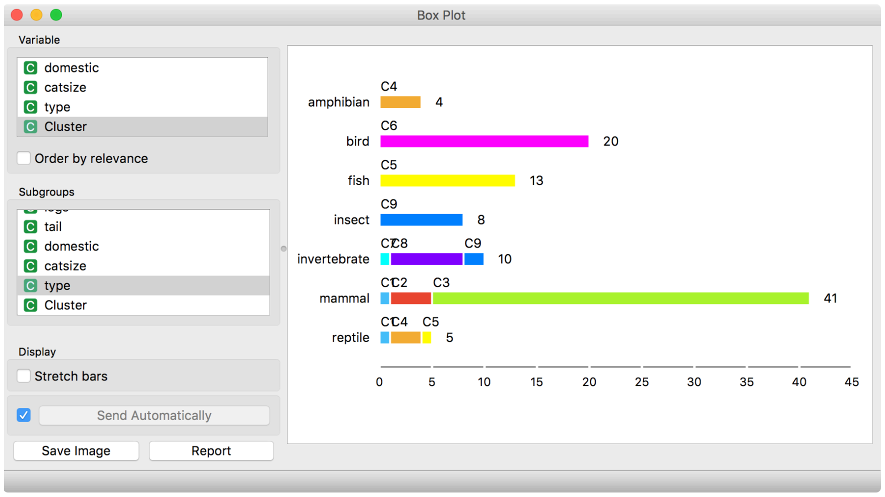
χ^2=454.42, p=0.000

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

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



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

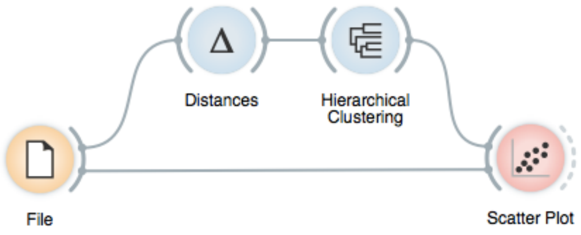


What is wrong with those mammals? Why can't they be in one single cluster? Two reasons. First, they represent 40 % of the data instances. Second, they include some weirdos. Click on the clusters in the box plot and discover who they are.

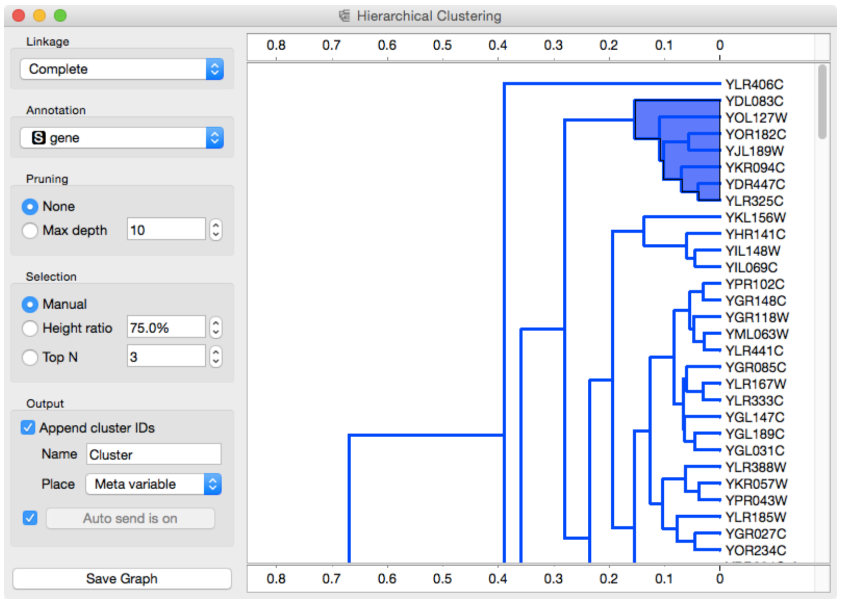
Lesson 13: Discovering clusters

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

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



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



Linkage: Complete

Annotation: gene

Pruning: [✓ None] Max depth 10

Selection: [✓ Manual] Height ratio 75.0% Top N 3

Output: ✓ Append cluster IDs
Name: Cluster
Place: Meta variable

✓ Auto send is on

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

For this data set, though, we can do something even better. The data already contains some predefined groups. Let us check how well the clusters match the classes - which we know, but clustering did not.

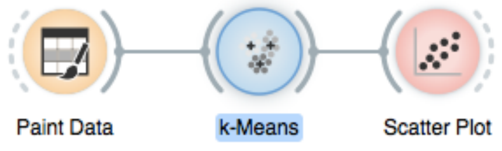
We will use the dendrogram to set a suitable threshold that splits the data into some three to five clusters. We can plot this data in a new scatter plot; we find a reasonable pair of attributes and then set the color of the points to represent the cluster they belong to. Do the clusters match the actual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

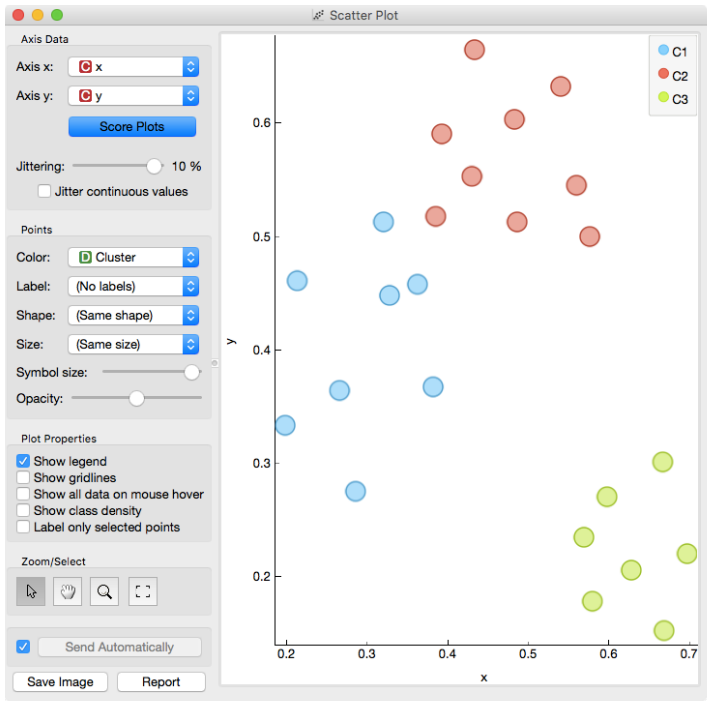
This lessons is not a recipe for what you should be doing in practice. If your data already contains groups labels, say gene group annotations, there is no need to discover them (again) by using clustering. In this case you should be interested in predictive models from previous lessons. If you do not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

Lesson 14: Silhouettes

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

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

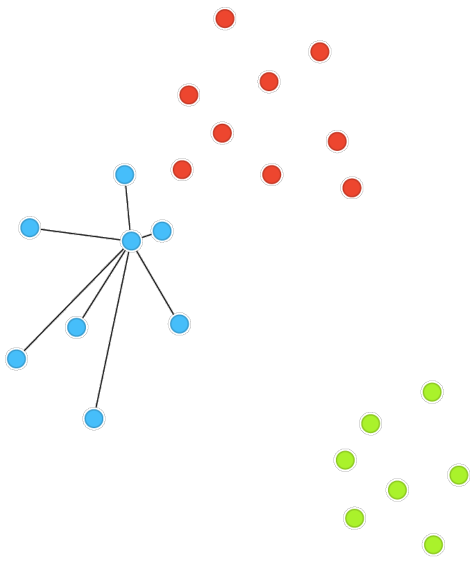




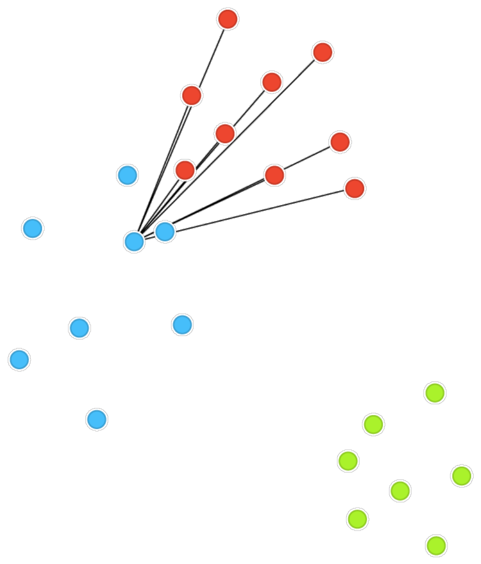
Axis Data
Axis x: x
Axis y: y
Jittering: 10 %
☐ Jitter continuous values
Points
Color: Cluster
Label: (No labels)
Shape: (Same shape)
Size: (Same size)
Symbol size:
Opacity:
Plot Properties
☑ Show legend
☐ Show gridlines
☐ Show all data on mouse hover
☐ Show class density
☐ Label only selected points
Zoom/Select
☑ Send Automatically

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 A, the better.



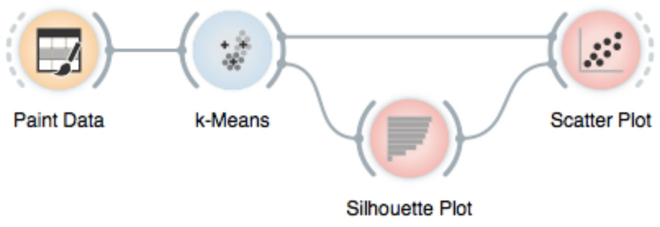
C3 is the green cluster, and all its points have large silhouettes. Not so for the other two.

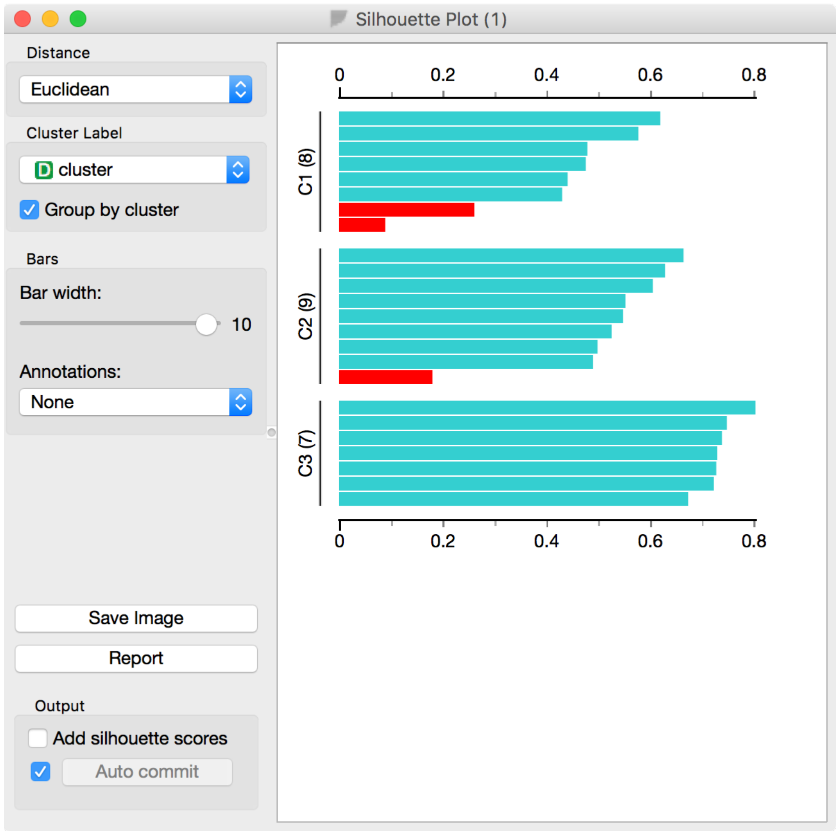
Below we selected three data instances with the worst silhouette scores. Can you guess where the lie in the scatter plot?

On the other hand, we would like a data point to be far away from the points in the closest neighboring cluster. The closest cluster to our blue data point is the red cluster. We can measure the distances between the blue data point and all the points in the red cluster, and again compute the average. Let us denote this average distance as B. The larger 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, $S = (B - A) / \max\{A, B\}$. Voilá, S is our silhouette score.

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




Silhouette Plot (1)

Distance: Euclidean

Cluster Label: cluster

Group by cluster: ✓

Bars

Bar width: 10

Annotations: None

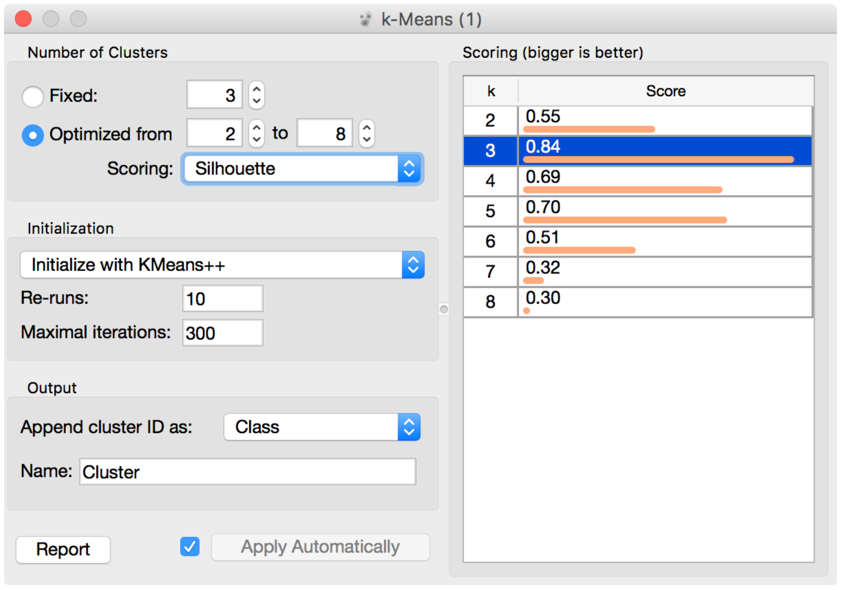
Output

Add silhouette scores: ☐

Auto commit: ✓

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 different number of clusters and displays the corresponding silhouette scores.



Number of Clusters

Fixed: 3

[✓ Optimized from] 2 to 8

Scoring: Silhouette

Initialization

Initialize with KMeans++

Re-runs: 10

Maximal iterations: 300

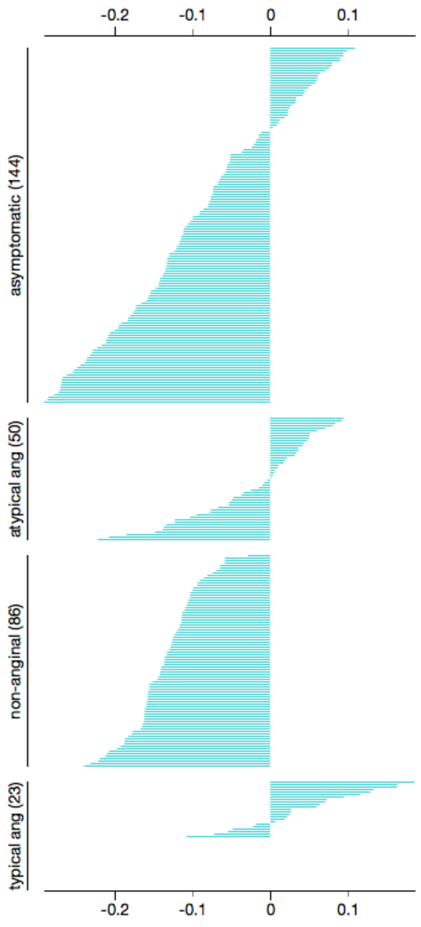
Output

Append cluster ID as: Class

Name: Cluster

Scoring (bigger is better)

k	Score
2	0.55
3	0.84
4	0.69
5	0.70
6	0.51
7	0.32
8	0.30



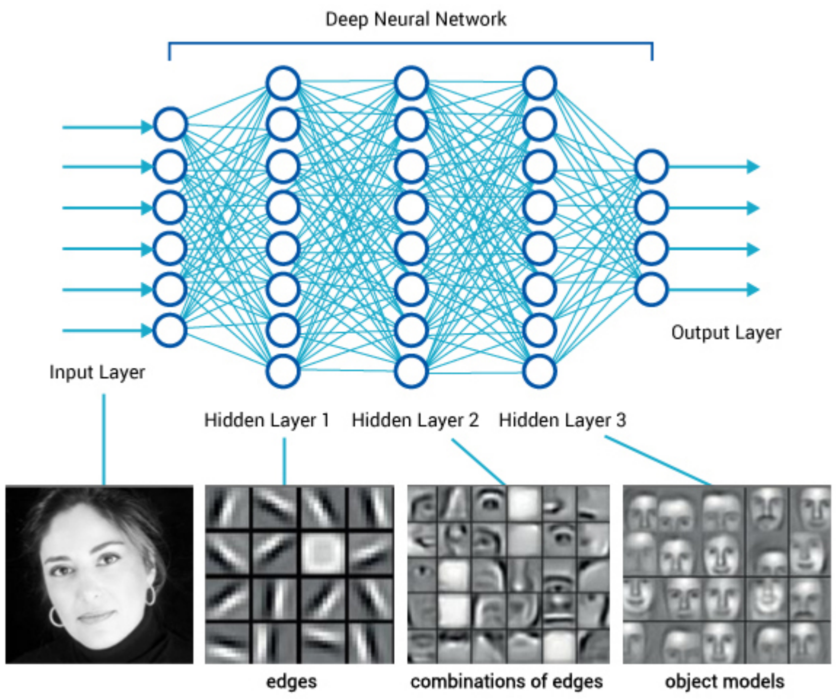
Ah, one more thing: Silhouette Plot can be used on any data, not just on data sets that are the output of clustering. We could use it with the iris data set and figure out which class is well separated from the other two and, conversely, which data instances from one class are similar to those from another.

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.

Lesson 15: Image Embedding

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

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



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

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

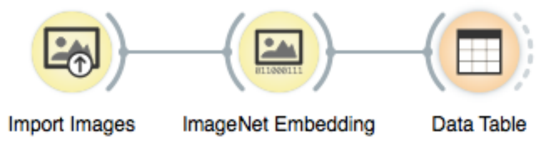
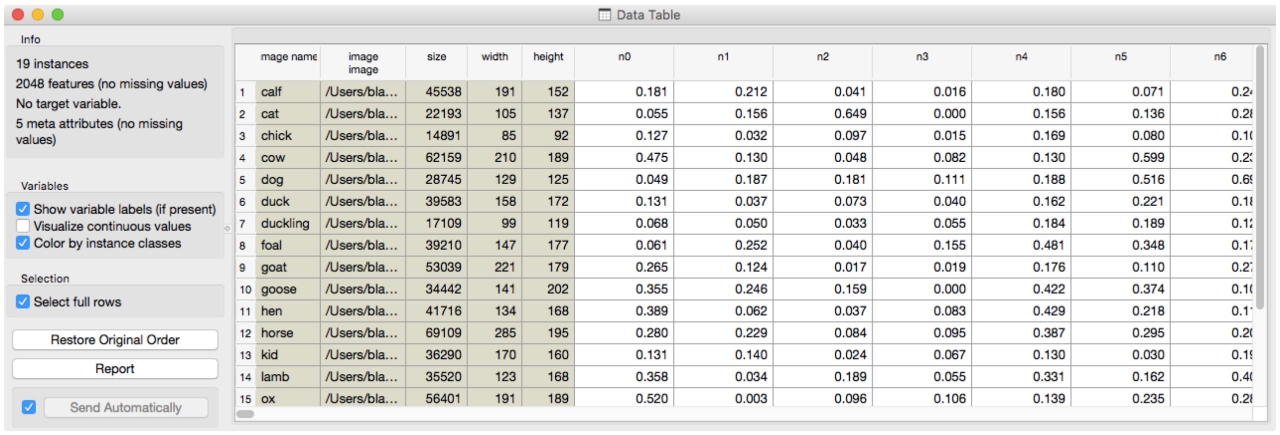


Image embedding describes the images with a set of 2048 features appended to the table with meta features of images.



Data Table

Info

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

Variables

- [✓] Show variable labels (if present)
- [ ] Visualize continuous values
- [✓] Color by instance classes

Selection

- [✓] Select full rows

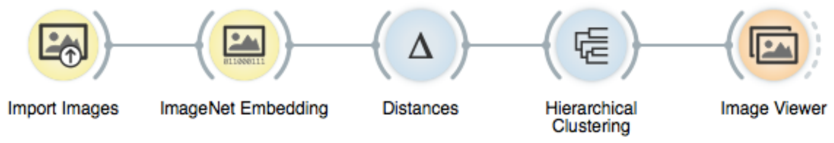
Restore Original Order

Report

[✓] Send Automatically

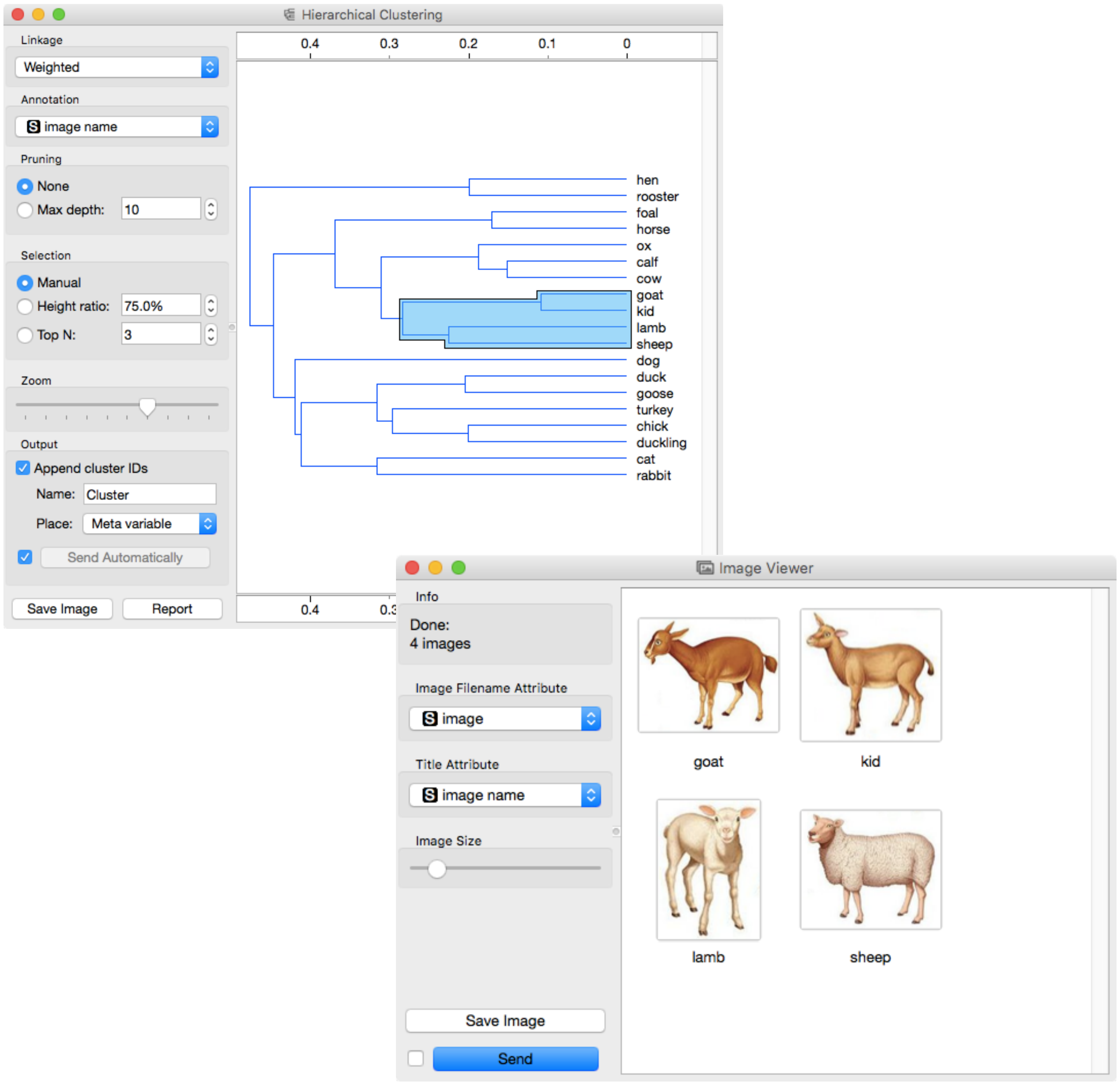
	mage name	image image	size	width	height	n0	n1	n2	n3	n4	n5	n6
1	calf	/Users/bla...	45538	191	152	0.181	0.212	0.041	0.016	0.180	0.071	0.2
2	cat	/Users/bla...	22193	105	137	0.055	0.156	0.649	0.000	0.156	0.136	0.2
3	chick	/Users/bla...	14891	85	92	0.127	0.032	0.097	0.015	0.169	0.080	0.1
4	cow	/Users/bla...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

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



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

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



Hierarchical Clustering

Linkage: Weighted

Annotation: image name

Pruning: [✓ None] Max depth: 10

Selection: [✓ Manual] Height ratio: 75.0% Top N: 3

Output: ✓ Append cluster IDs

Name: Cluster

Place: Meta variable

✓ Send Automatically

Image Viewer

Info

Done:
4 images

Image Filename Attribute: image

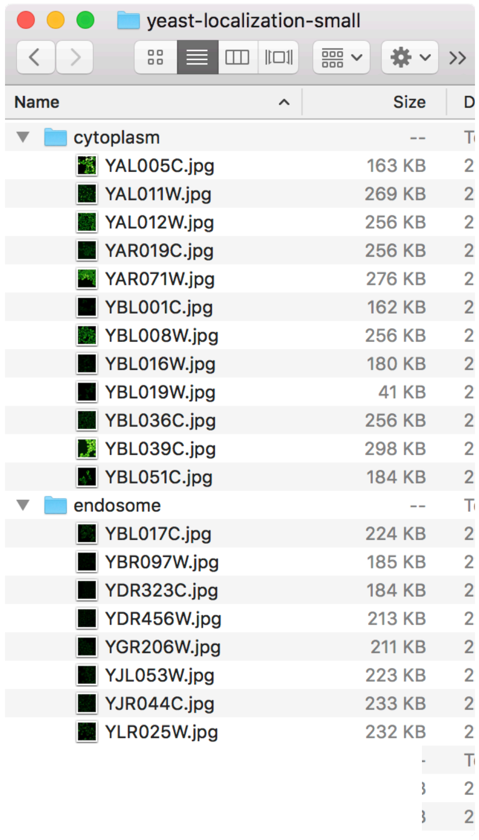
Title Attribute: image name

Image Size

Lesson 16: Images and Classification

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

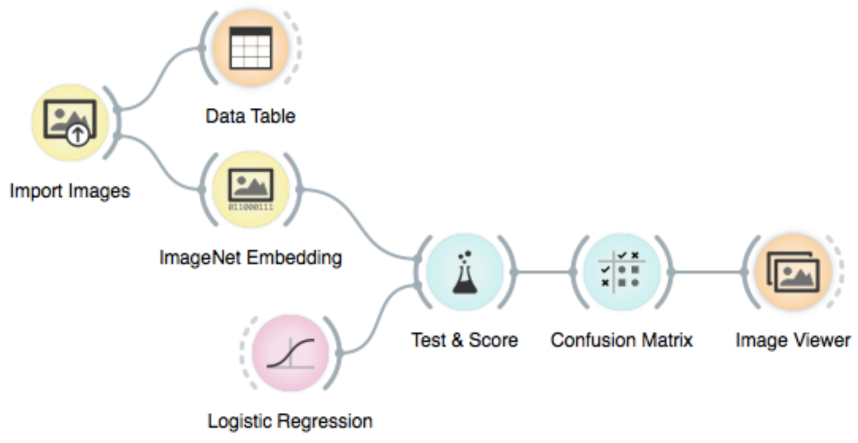
We can use image data for classification. For that, we need to associate every image with the class label. The easiest way to do this is by storing images of different classes in different folders. Take, for instance, images of yeast protein localization. Screenshot of the file names shows we have stored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



Name	Size	D
▼ cytoplasm	--	T
YAL005C.jpg	163 KB	2
YAL011W.jpg	269 KB	2
YAL012W.jpg	256 KB	2
YAR019C.jpg	256 KB	2
YAR071W.jpg	276 KB	2
YBL001C.jpg	162 KB	2
YBL008W.jpg	256 KB	2
YBL016W.jpg	180 KB	2
YBL019W.jpg	41 KB	2
YBL036C.jpg	256 KB	2
YBL039C.jpg	298 KB	2
YBL051C.jpg	184 KB	2
▼ endosome	--	T
YBL017C.jpg	224 KB	2
YBR097W.jpg	185 KB	2
YDR323C.jpg	184 KB	2
YDR456W.jpg	213 KB	2
YGR206W.jpg	211 KB	2
YJL053W.jpg	223 KB	2
YJR044C.jpg	233 KB	2
YLR025W.jpg	232 KB	2
	-	T
	3	2
	3	2

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

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



For the End

The course on Introduction to Data Mining ends here. We covered quite some mileage, and we hope we have taught you some essential procedures that should be on the stack of every data scientists. The goal was to get you familiar with basic techniques, tools, and concepts of data science and teach you have to visually explore data and models. Data science is a vast field, and it takes years of study and practice to master it. You may never become a data scientist, but as an expert in some other field, it should now be easier to talk and collaborate with statisticians and computer scientists. And for those who want to go ahead with data science, well, you now know where to start.