

Introduction to Data Mining

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

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

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

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

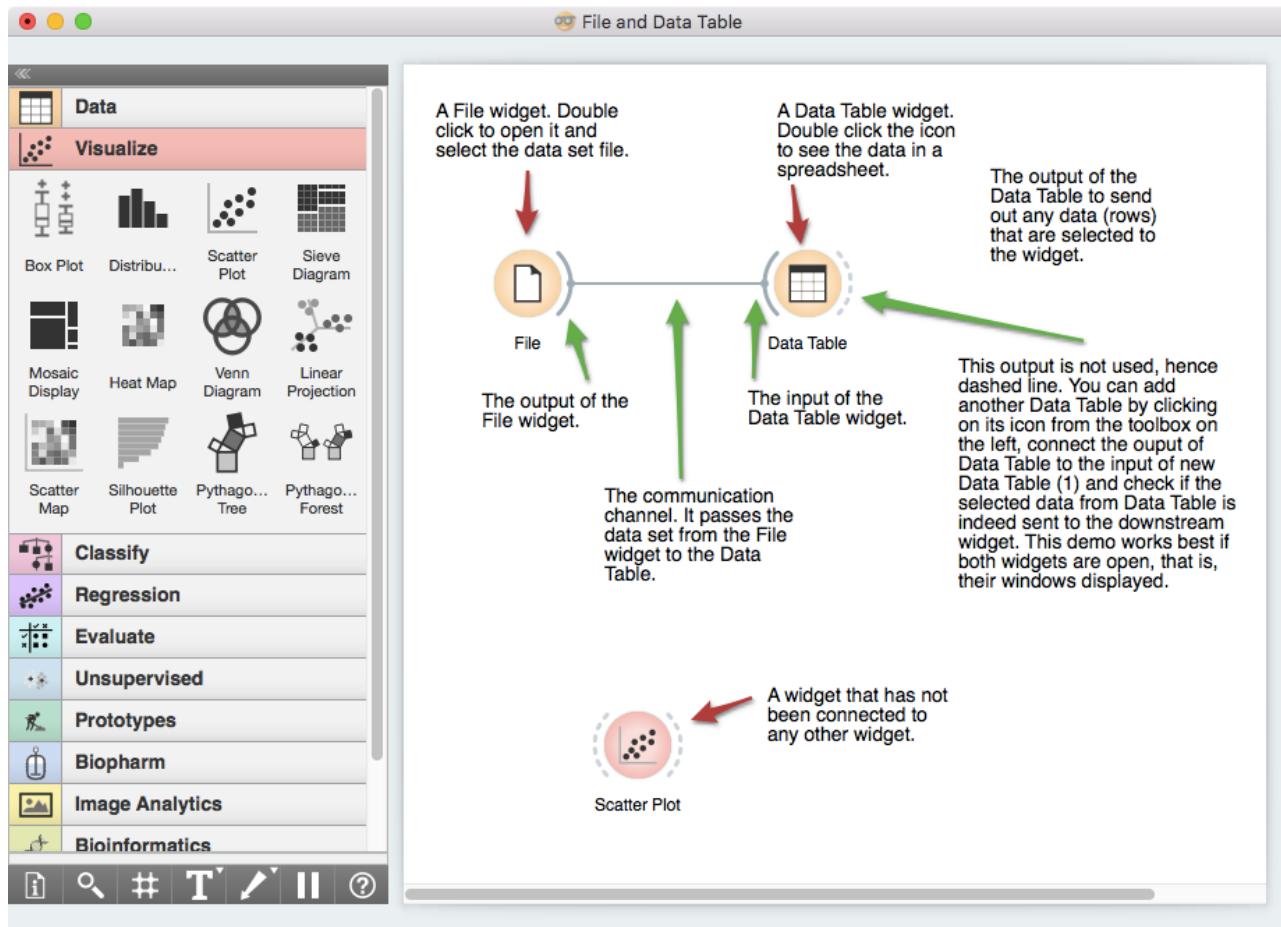
If you haven't already installed Orange, please download the installation package from <http://orange.biolab.si>.



Attribution-NonCommercial-NoDerivs
CC BY-NC-ND

Lesson 1: Workflows in Orange

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

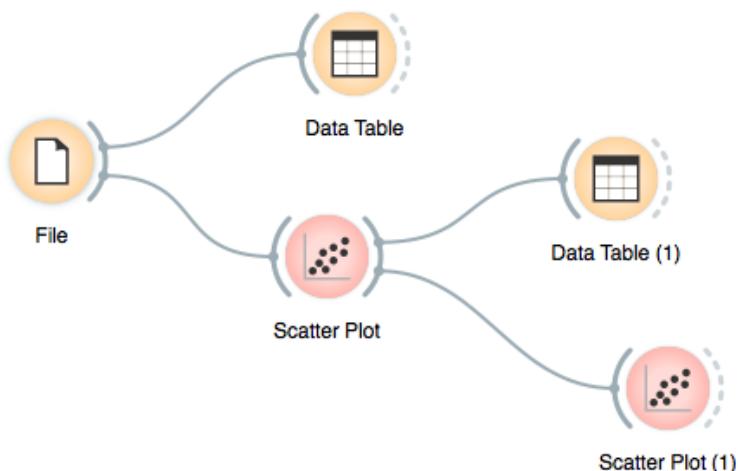


A screenshot above shows a simple workflow with two connected widgets and one widget without connections. The outputs of a widget appear on the right, while the inputs appear on the left.

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

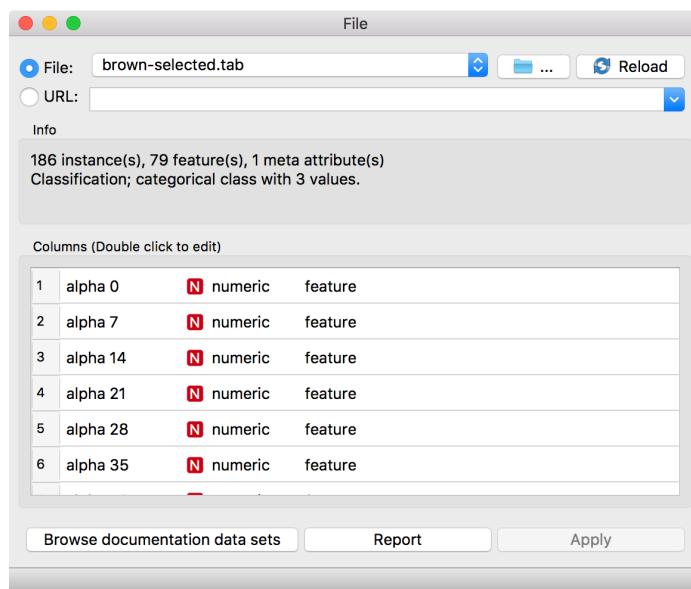
Start by constructing a workflow that consists of a File widget, two Scatter Plot widgets, and two Data Table widgets:

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



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

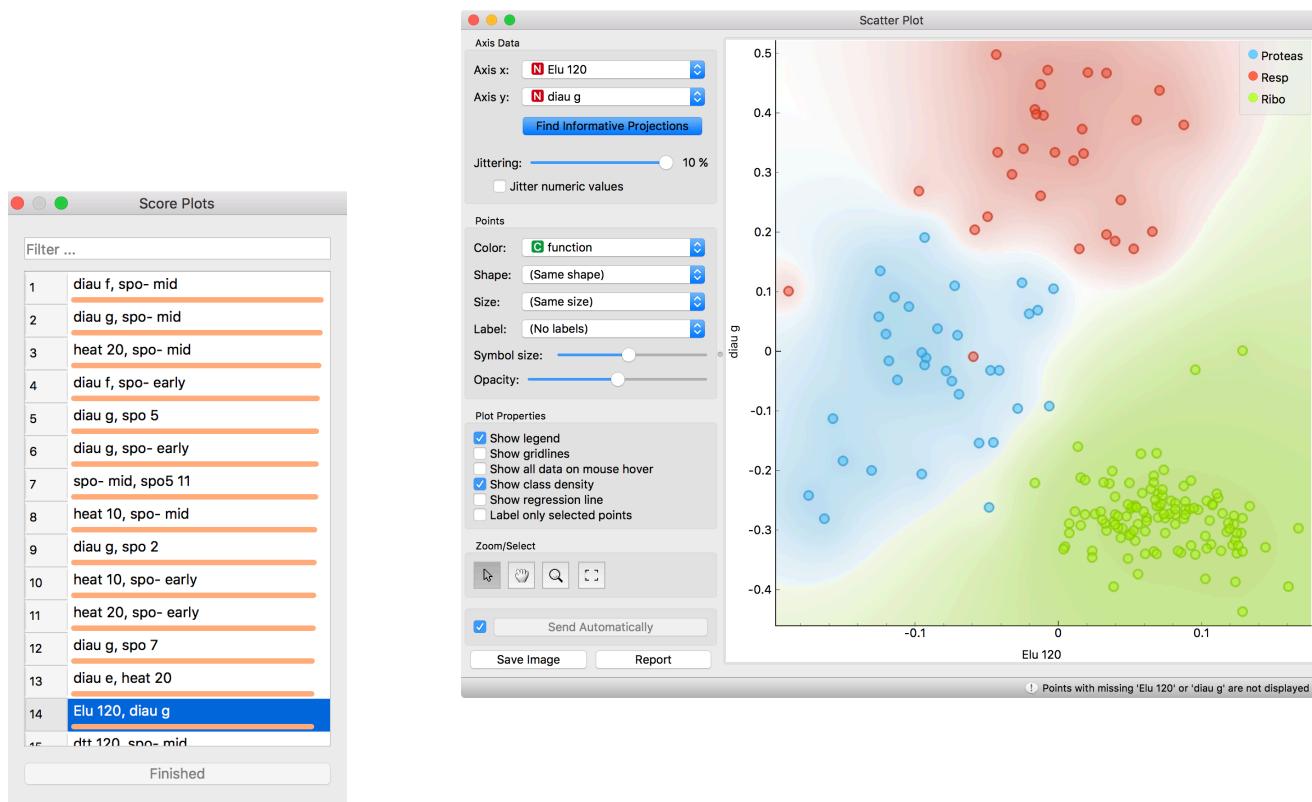
Orange workflows often start with a File widget. The brown-selected data set comprises 186 rows (genes) and 81 columns. Out of the 81 columns, 79 contain gene expressions of baker’s yeast under various conditions, one column (marked as a “meta attribute”) provides gene names, and one column contains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



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

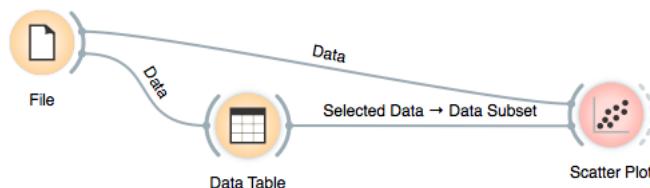
where the second scatter plot shows a detail from a smaller region selected in the first scatterplot.

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

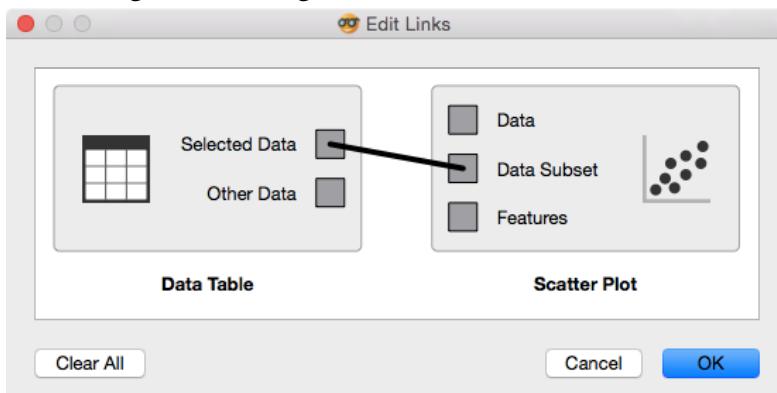


We can connect the output of the Data Table widget to the Scatter Plot widget to highlight the chosen data instances (rows) in the scatter plot.

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

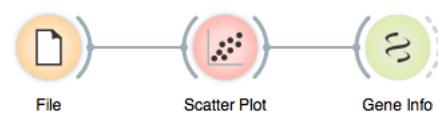


How does Orange distinguish between the primary data source and the data selection? It uses the first connected signal as the entire data set and the second one as its subset. To make changes or to check what is happening under the hood, double click on the line connecting the two widgets.



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

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

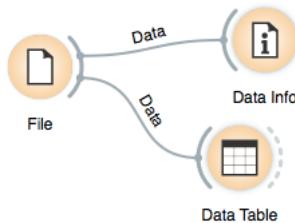


Lesson 2: Basic Data Exploration

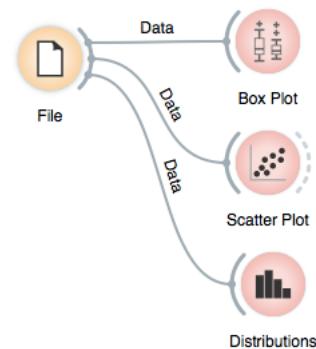
Let us consider another problem, this time from clinical medicine. We will dig for something interesting in the data and explore it a bit with various widgets. You will get to know Orange better and also learn about several interesting visualizations.

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

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

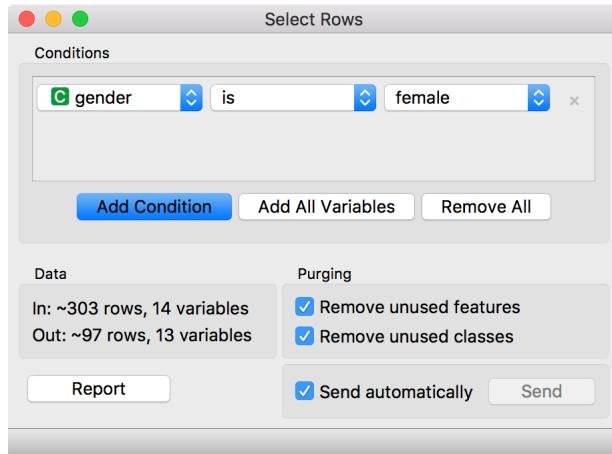
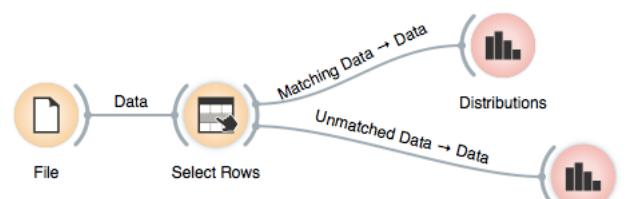


Let us check whether standard visualizations tell us anything interesting. (Hint: look for gender differences. These are always interesting and occasionally even real.)

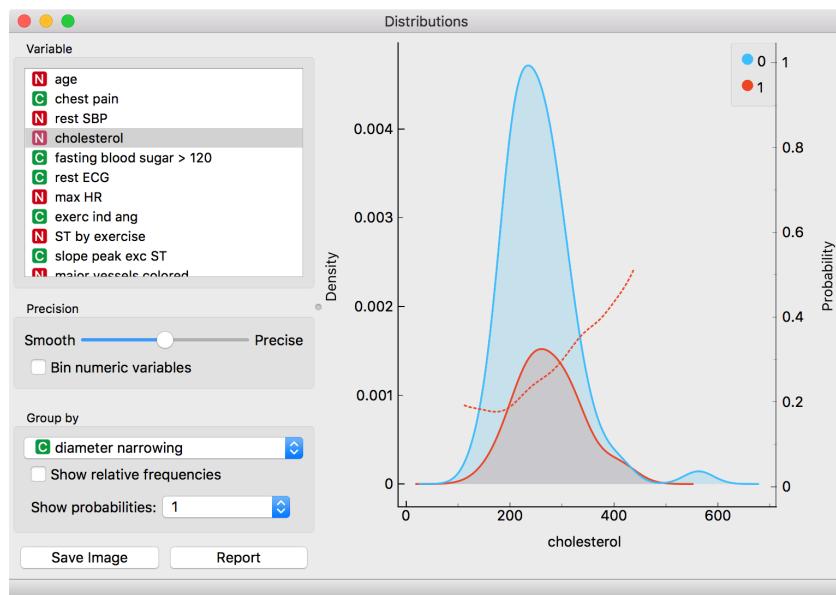


The two Distributions widgets get different data: the upper gets the selected rows, and the lower gets the others. Double-click the connection between the widgets to access setup dialog, as you've learned in the previous lesson.

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

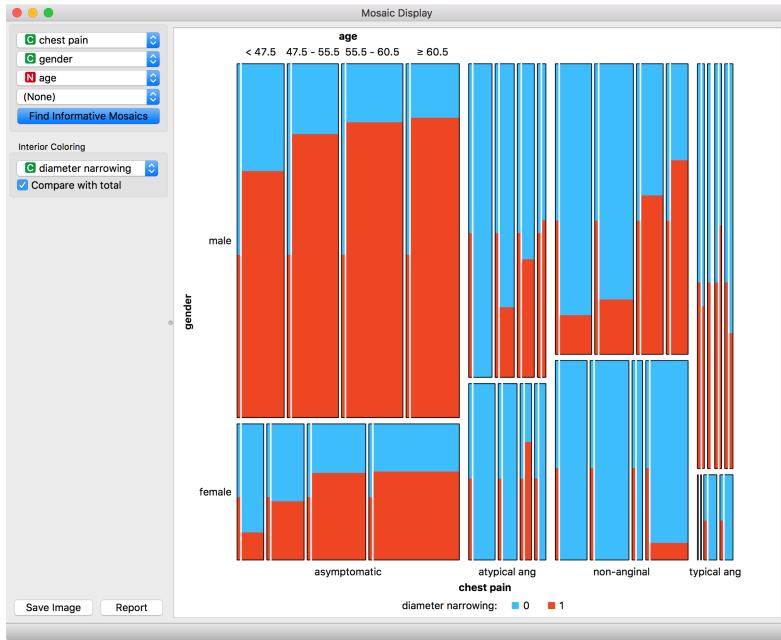


In the Select Rows widget, we choose the female patients. You can also add other conditions. Selection of data instances works well with visualization of data distribution. Try having at least two widgets open at the same time and explore the data.



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

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

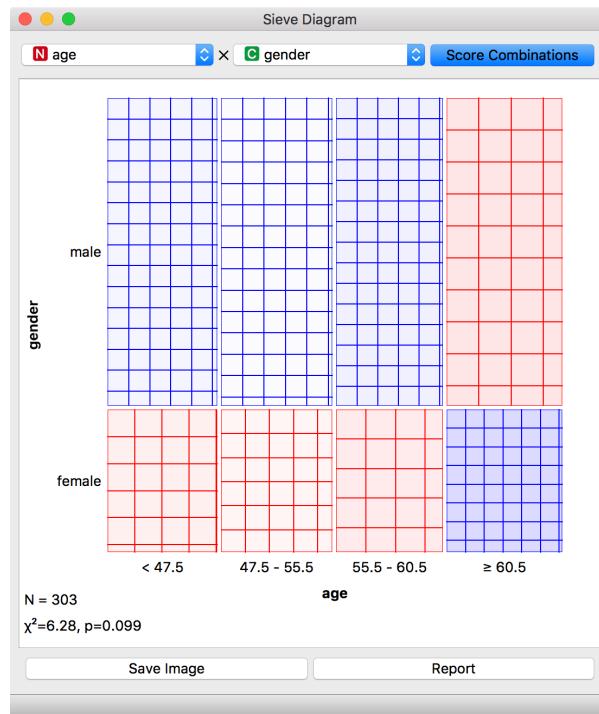


The mosaic display shows a rectangle split into columns with widths reflecting the prevalence of different types of chest pain. Each column is then further split vertically according to gender distributions within the column. The resulting rectangles are divided again horizontally according to age group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

What can we conclude from this diagram?

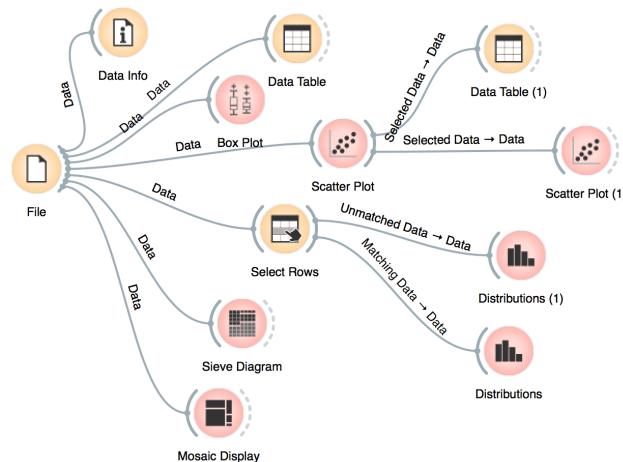
Another visualization, Sieve diagram, also splits a rectangle horizontally and vertically, but with independent cuts, so the areas correspond to the expected number of data instances assuming the observed variables are independent. For example, $1/4$ of patients are older than 60, and $1/3$ of patients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

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

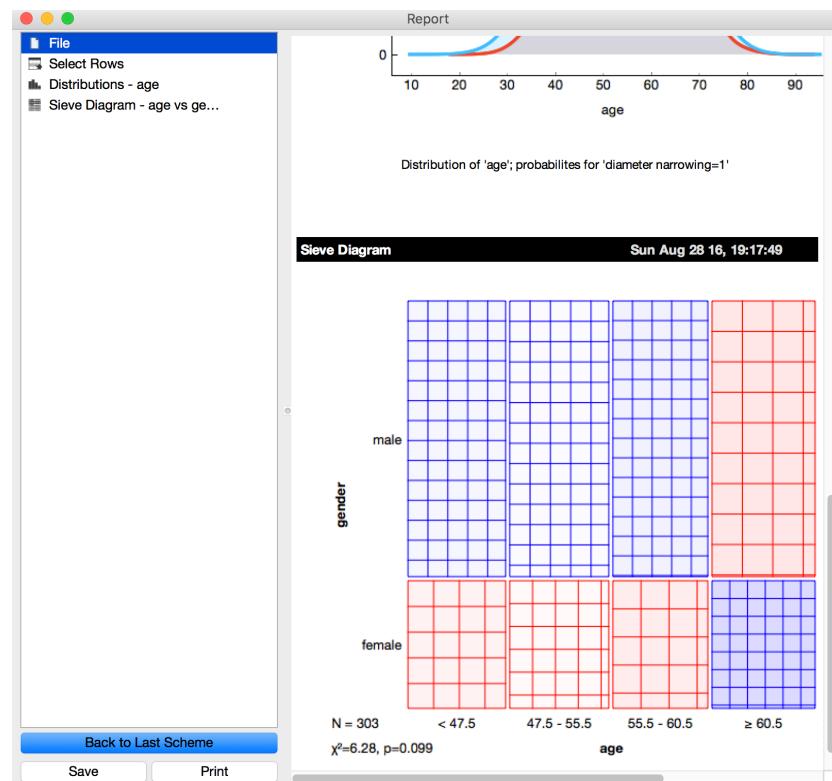


Lesson 3: Saving Your Work

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



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

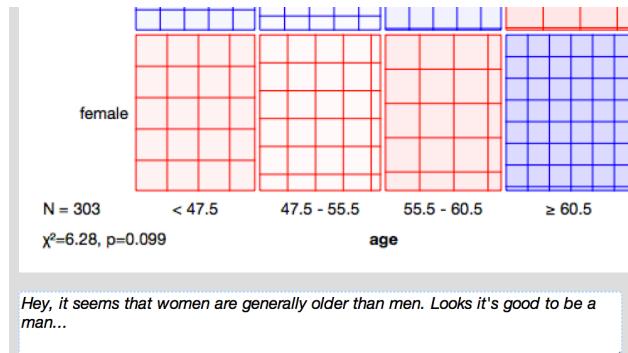


One more trick: Pressing Ctrl-C (or ⌘-C, on Mac) copies a visualization to the clipboard, so you can paste it to another application.

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

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

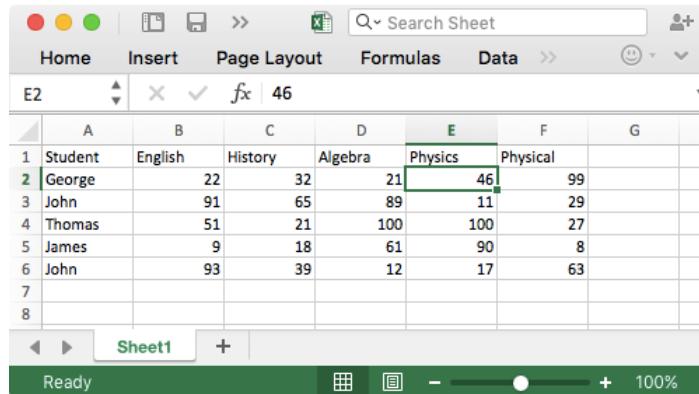
Clicking on a part of the report also allows you to add a comment.



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

Lesson 4: Loading Your Own Data Set

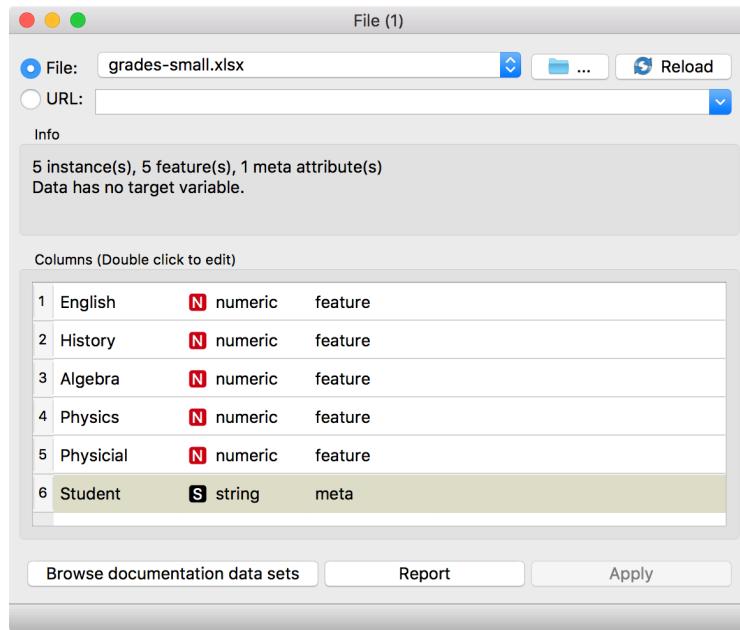
The data sets we have worked with in previous lessons come with Orange installation. Orange can read data from spreadsheet file formats which include tab and comma separated and Excel files. Let us prepare a data set (with school subjects and grades) in Excel and save it on a local disk.



A screenshot of Microsoft Excel showing a data table. The table has columns labeled A through G. Row 1 contains column headers: Student, English, History, Algebra, Physics, and Physical. Rows 2 through 6 contain data for five students: George, John, Thomas, James, and John. The 'Physics' column for George contains the value '46'. The table is displayed on 'Sheet1'.

	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.



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

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



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

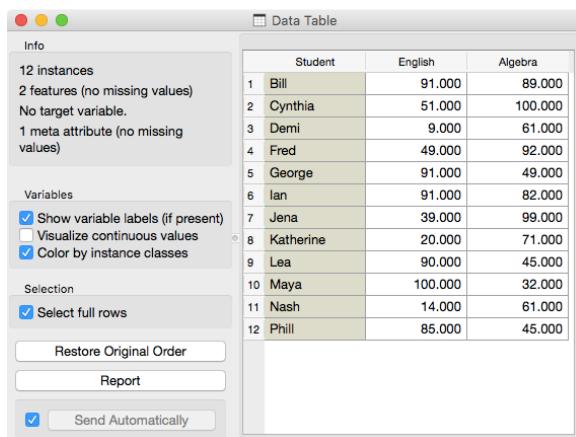
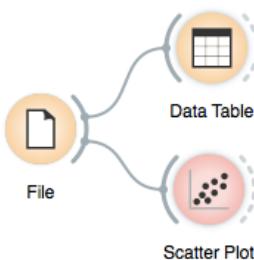
	Student	English	History	Algebra	Physics	Physical
1	George	22.000	32.000	21.000	46.000	99.000
2	John	91.000	65.000	89.000	11.000	29.000
3	Thomas	51.000	21.000	100.000	100.000	27.000
4	James	9.000	18.000	61.000	90.000	8.000
5	John	93.000	39.000	12.000	17.000	63.000

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.

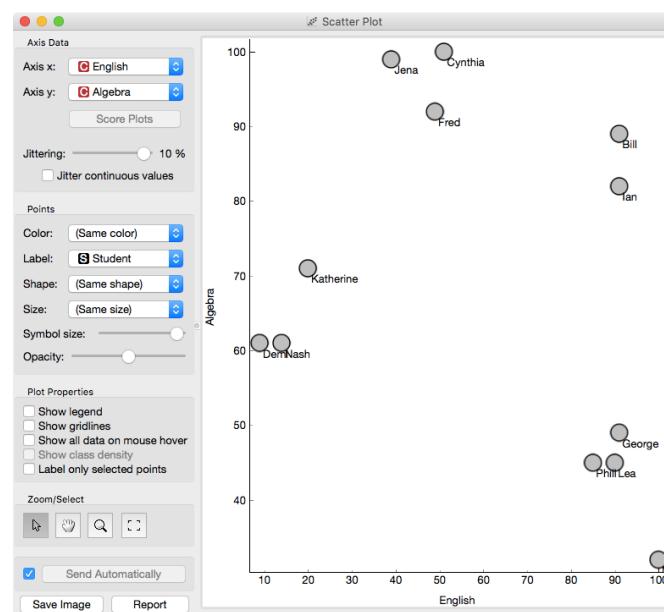
In the class, we will introduce clustering using a simple data set on students and their grades in English and Algebra. Load the data set from <http://file.biolab.si/files/grades2.tab>.



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).

Lesson 5: Hierarchical Clustering

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?



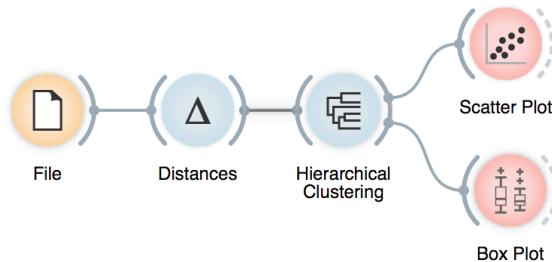
We need to start with a definition of “similar”. One simple measure of similarity for such data is the Euclidean distance: square the differences across every dimension, sum them and take the square root, just like in Pythagorean theorem. So, we would like to group data instances with small Euclidean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

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

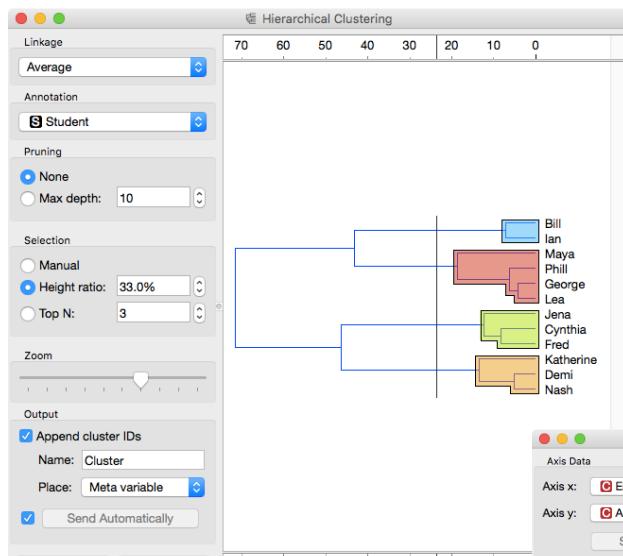
This procedure constructs a hierarchy of clusters, which explains why we call it hierarchical clustering. After it is done, we can

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

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



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



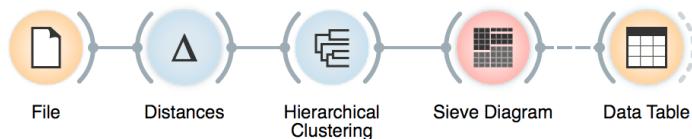
You can also observe the properties of the clusters
- that is, the average grades in Algebra and English
- in the box plot.



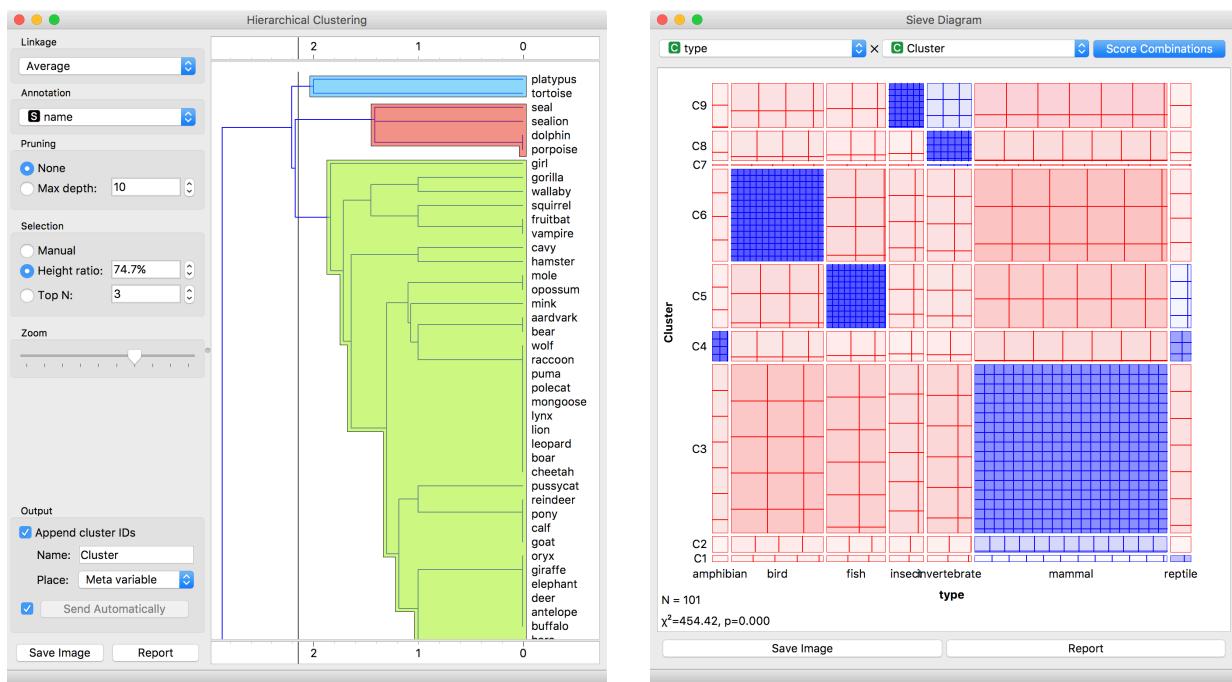
Lesson 6: Animal Kingdom



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

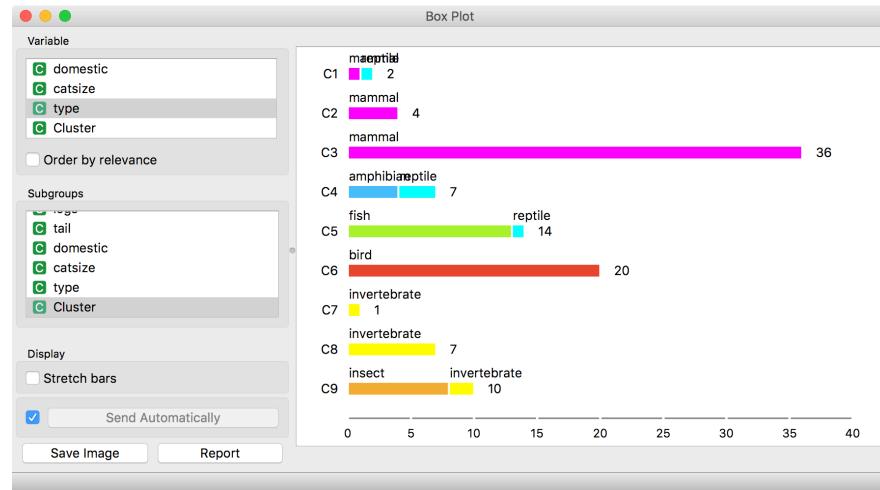


Here we use zoo data (from documentation data sets) with attributes that report on various features of animals (has hair, has feathers, lays eggs). We measure the distance and compute the clustering. Animals in this data set are annotated with type (mammal, insect, bird, and so on). It would be cool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

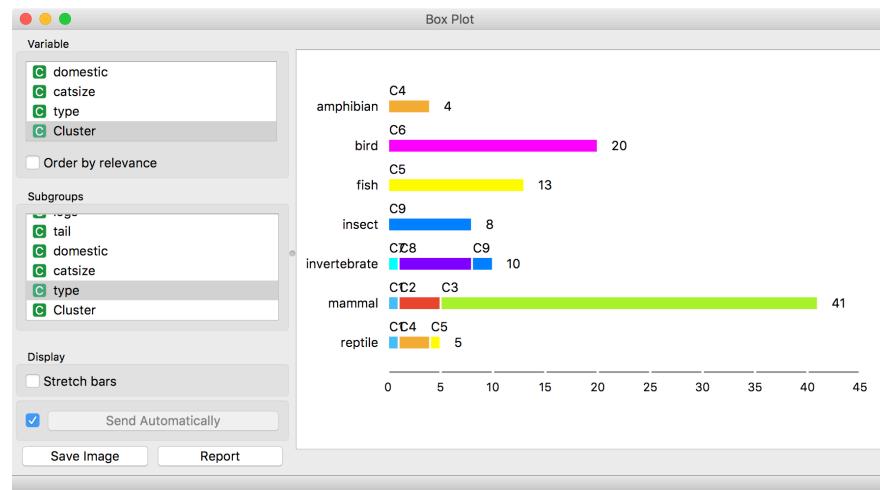


Looks great. Birds, say, are in cluster C6. Cluster C4 consists of amphibians and some reptiles. And so forth.

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



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



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

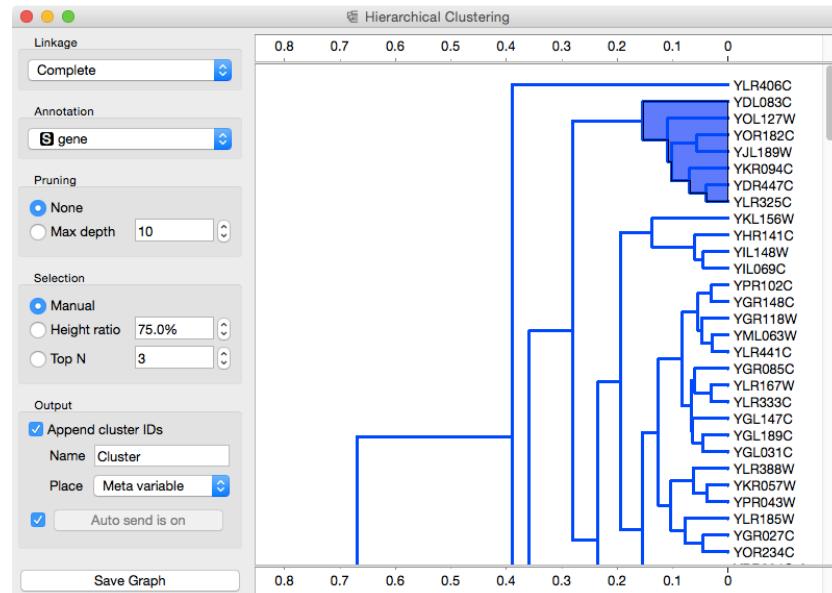
Lesson 7: Discovering clusters

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

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



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



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

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

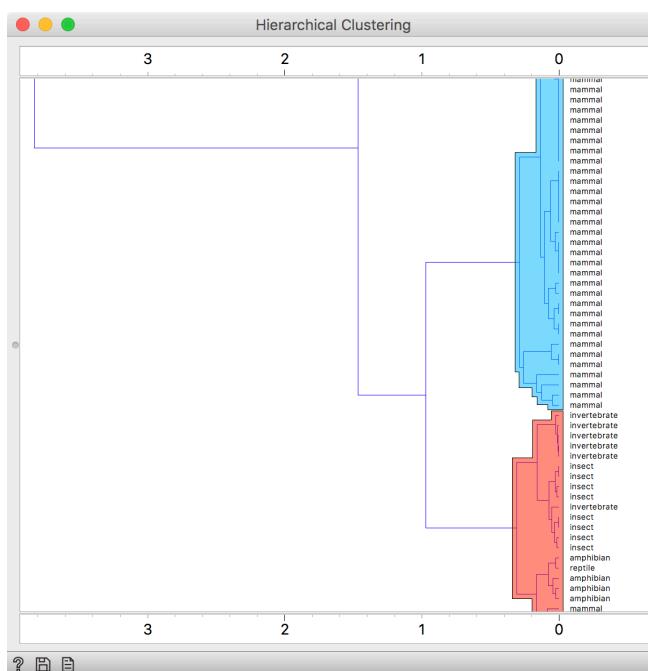
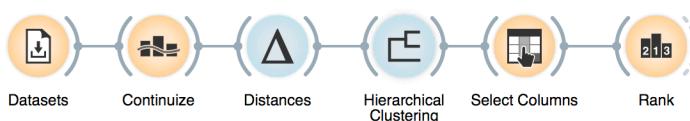
well the clusters match the classes - which we know, but clustering did not.

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

This lesson is not a recipe for what you should be doing in practice. If your data already contains group 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 meaningful subgroups in the data; we pretend we did not know the groups, use the clustering to discover them, and checked how well the correspond to the actual groups.

Lesson 8: Cluster interpretation

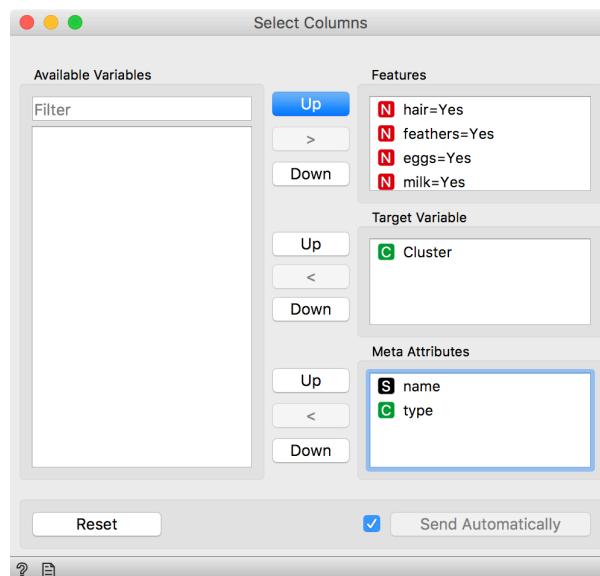
Once we have inferred the clusters, we would like to know what are the distinguishing features. For the zoo data set, we could, for instance, mark two clusters, and then ask for the features that distinguish among these. Having data marked with cluster identifiers takes us back to classification, and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



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

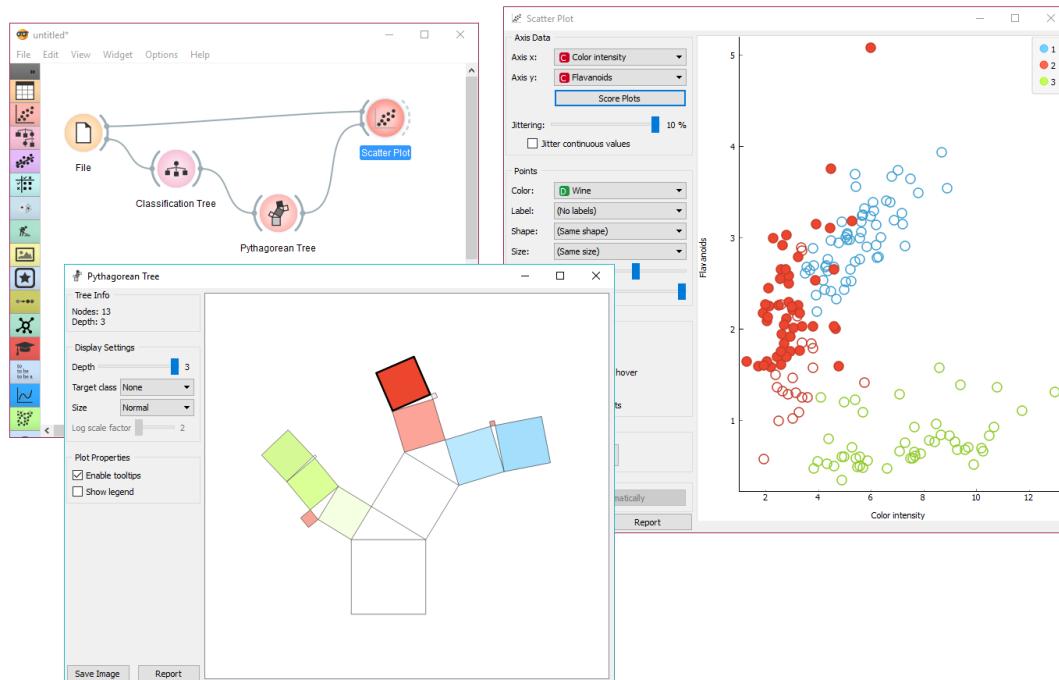
We load zoo data from Datasets, continuize the categorical features (this will be changed in Orange soon, Distances should automatically perform continuization), estimate the distances, and feed everything in Hierarchical Clustering. So far, nothing new.

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



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

	#	Gain ratio ▼	Gini
N milk=Yes		0.892	0.428
N eggs=Yes		0.892	0.428
N toothed=Yes		0.663	0.310
N hair=Yes		0.663	0.310
N backbone=Yes		0.583	0.259
N legs		0.329	0.244
N breathes=Yes		0.323	0.077
N tail=Yes		0.310	0.181
N venomous=Yes		0.297	0.060
N catsize=Yes		0.273	0.162
N aquatic=Yes		0.245	0.116
N airborne=Yes		0.126	0.048
N fins=Yes		0.090	0.005
N domestic=Yes		0.068	0.023
N predator=Yes		0.005	0.003
N feathers=Yes		0.000	0.000



Introduction to Data Mining

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

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

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

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

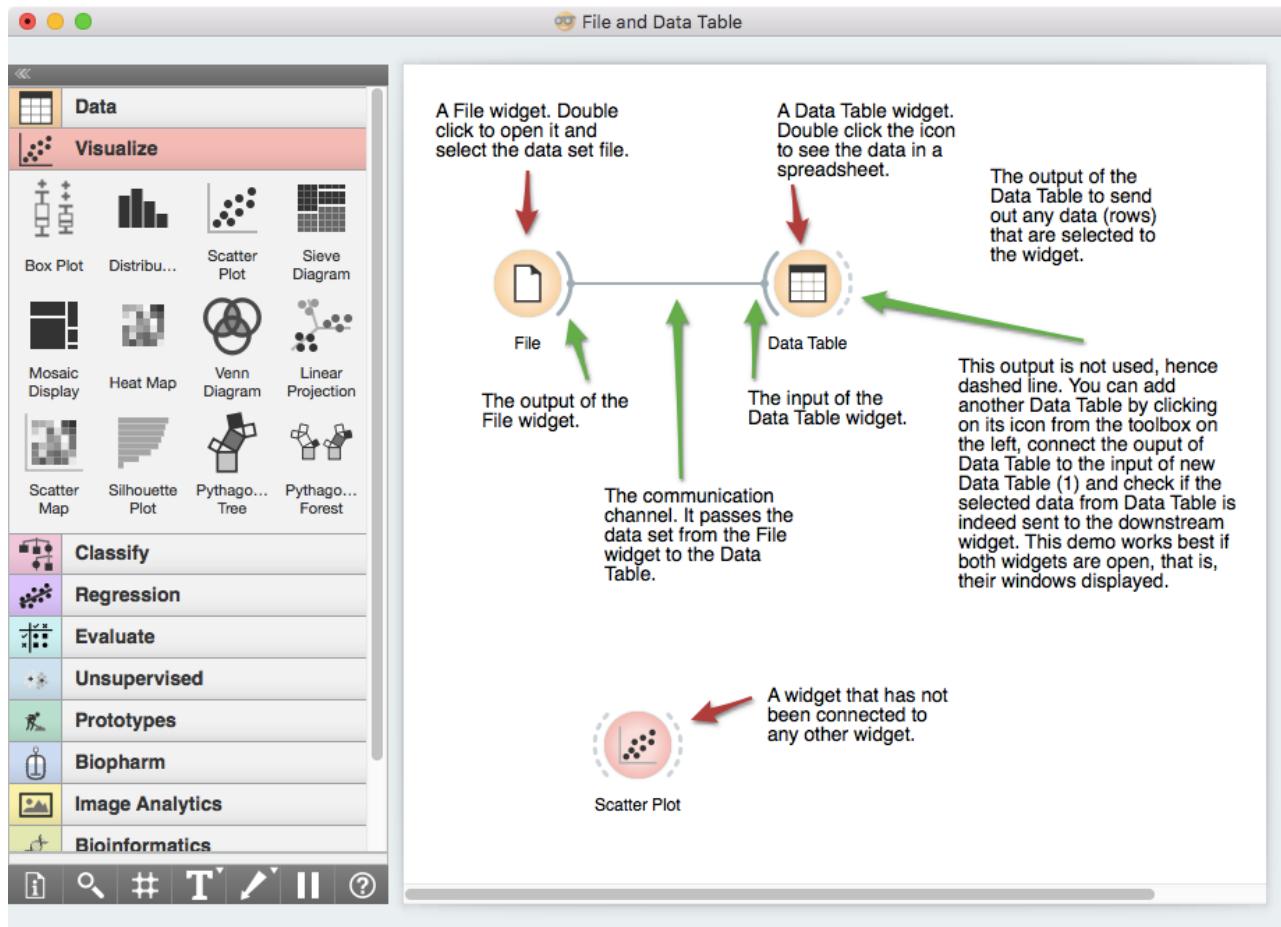
If you haven't already installed Orange, please download the installation package from <http://orange.biolab.si>.



Attribution-NonCommercial-NoDerivs
CC BY-NC-ND

Lesson 1: Workflows in Orange

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

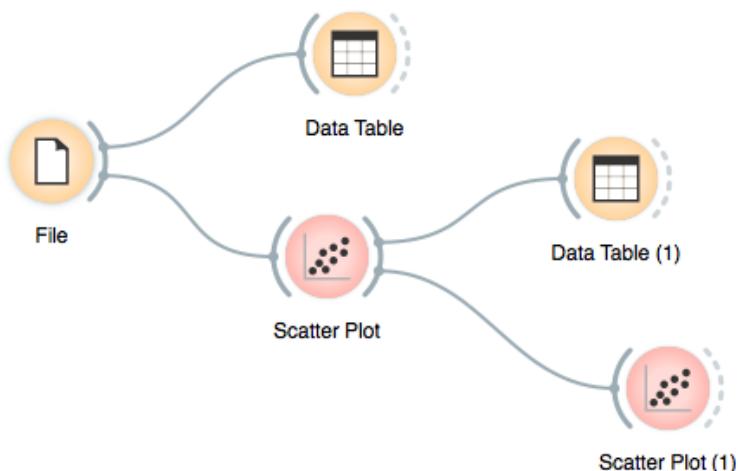


A screenshot above shows a simple workflow with two connected widgets and one widget without connections. The outputs of a widget appear on the right, while the inputs appear on the left.

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

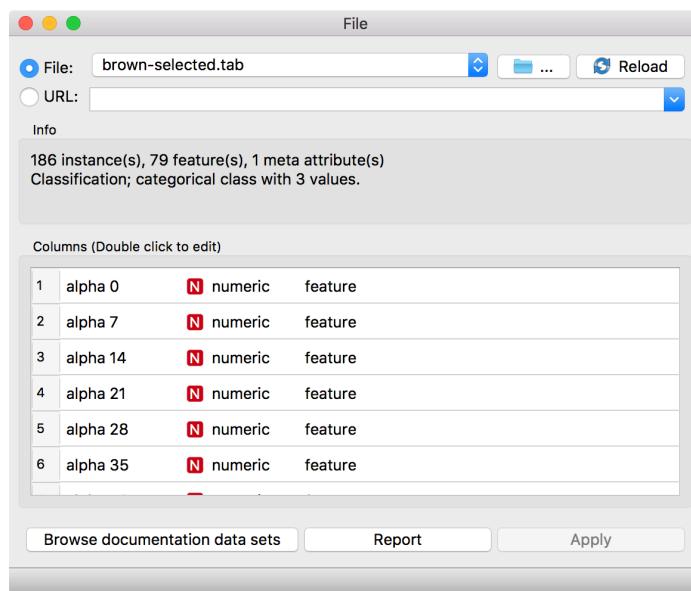
Start by constructing a workflow that consists of a File widget, two Scatter Plot widgets, and two Data Table widgets:

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



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

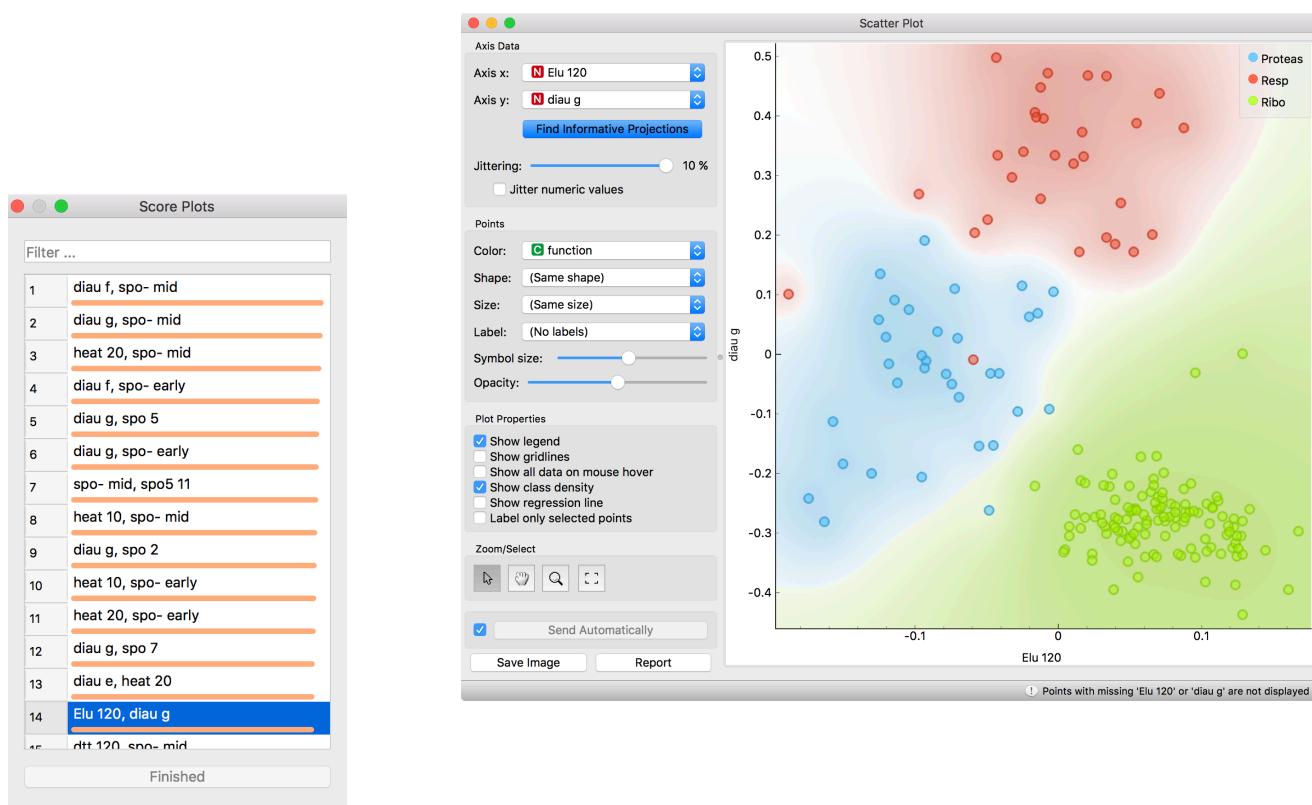
Orange workflows often start with a File widget. The brown-selected data set comprises 186 rows (genes) and 81 columns. Out of the 81 columns, 79 contain gene expressions of baker’s yeast under various conditions, one column (marked as a “meta attribute”) provides gene names, and one column contains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



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

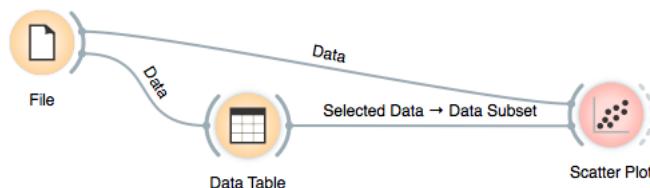
where the second scatter plot shows a detail from a smaller region selected in the first scatterplot.

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

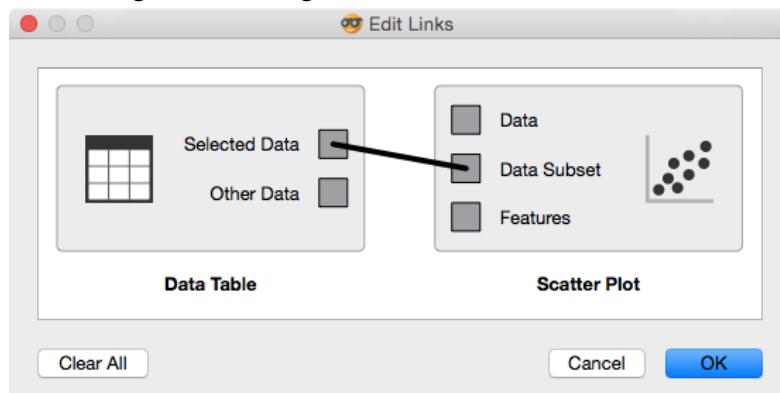


We can connect the output of the Data Table widget to the Scatter Plot widget to highlight the chosen data instances (rows) in the scatter plot.

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

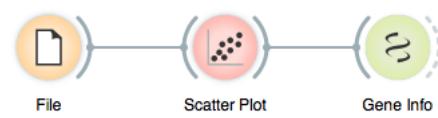


How does Orange distinguish between the primary data source and the data selection? It uses the first connected signal as the entire data set and the second one as its subset. To make changes or to check what is happening under the hood, double click on the line connecting the two widgets.



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

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

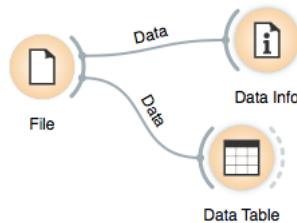


Lesson 2: Basic Data Exploration

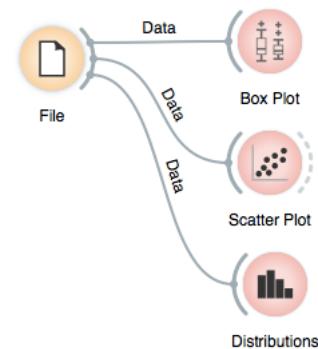
Let us consider another problem, this time from clinical medicine. We will dig for something interesting in the data and explore it a bit with various widgets. You will get to know Orange better and also learn about several interesting visualizations.

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

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

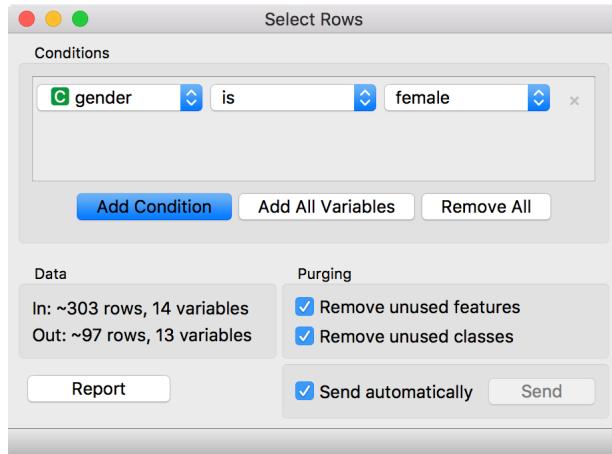
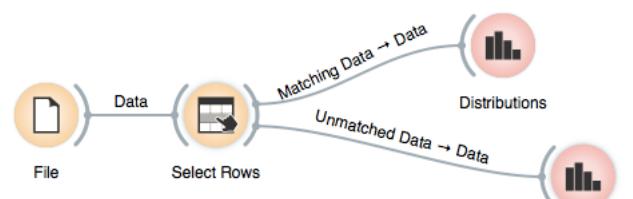


Let us check whether standard visualizations tell us anything interesting. (Hint: look for gender differences. These are always interesting and occasionally even real.)

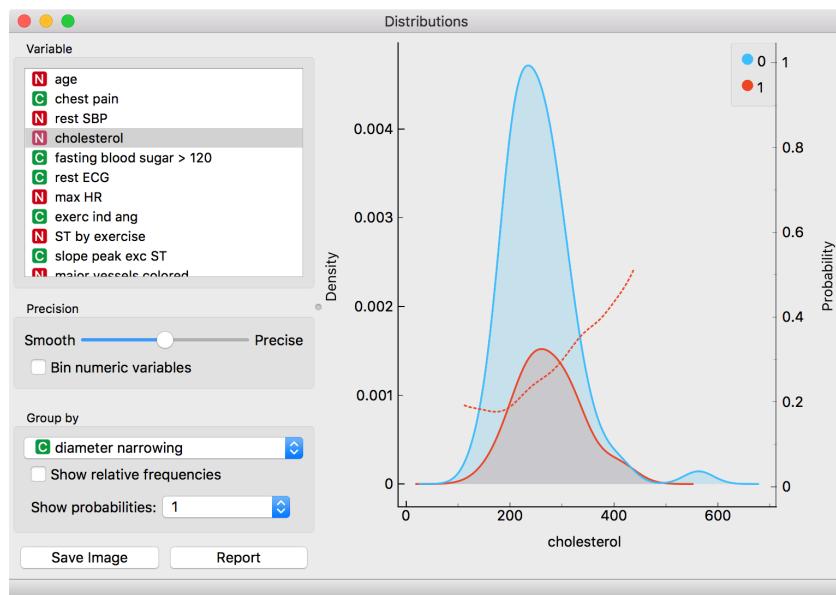


The two Distributions widgets get different data: the upper gets the selected rows, and the lower gets the others. Double-click the connection between the widgets to access setup dialog, as you've learned in the previous lesson.

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

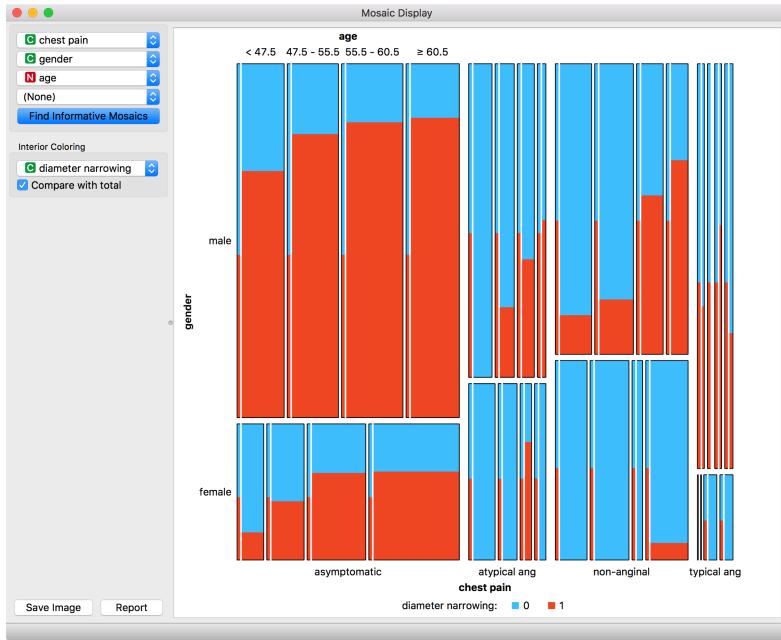


In the Select Rows widget, we choose the female patients. You can also add other conditions. Selection of data instances works well with visualization of data distribution. Try having at least two widgets open at the same time and explore the data.



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

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

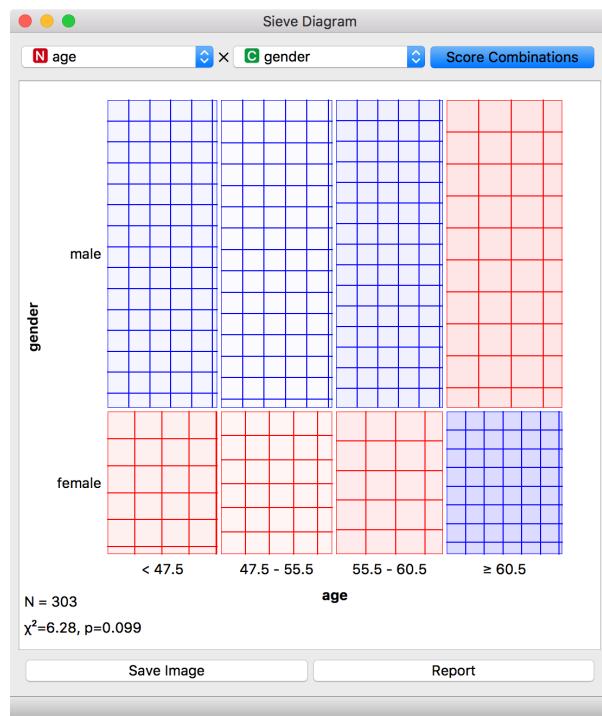


The mosaic display shows a rectangle split into columns with widths reflecting the prevalence of different types of chest pain. Each column is then further split vertically according to gender distributions within the column. The resulting rectangles are divided again horizontally according to age group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

What can we conclude from this diagram?

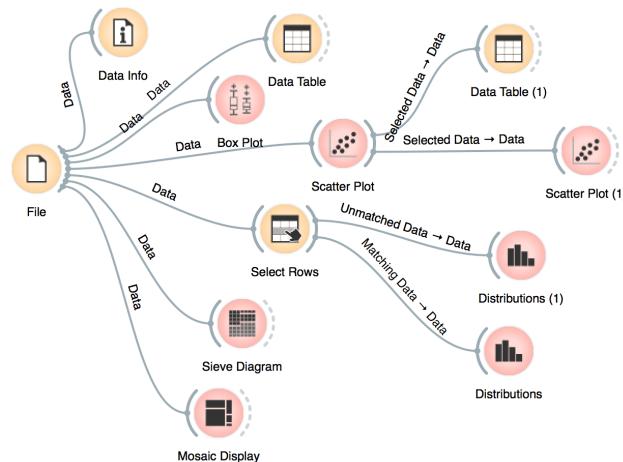
Another visualization, Sieve diagram, also splits a rectangle horizontally and vertically, but with independent cuts, so the areas correspond to the expected number of data instances assuming the observed variables are independent. For example, $1/4$ of patients are older than 60, and $1/3$ of patients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

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

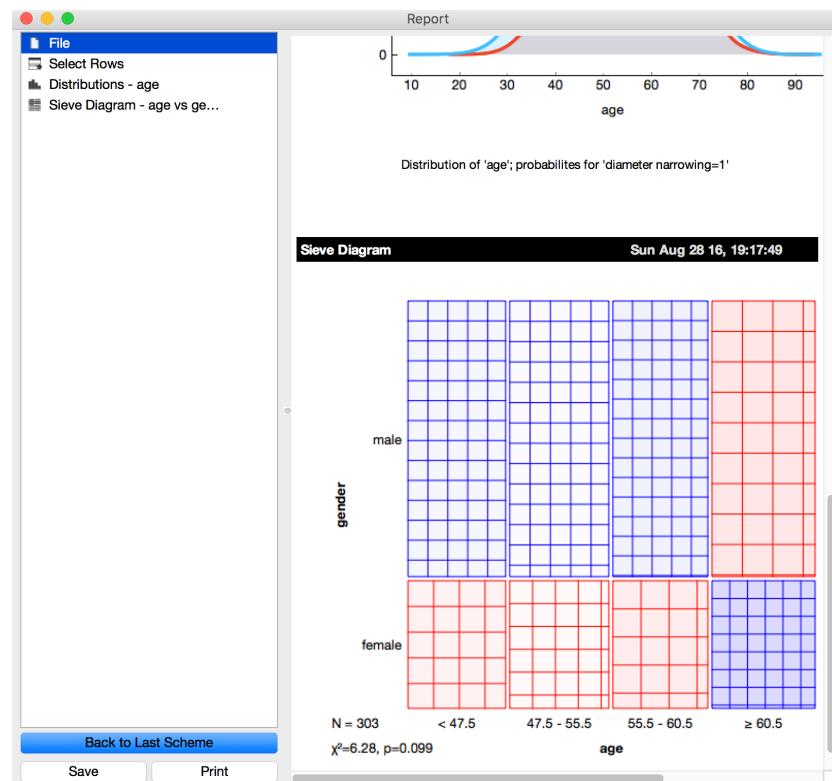


Lesson 3: Saving Your Work

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



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



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

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

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

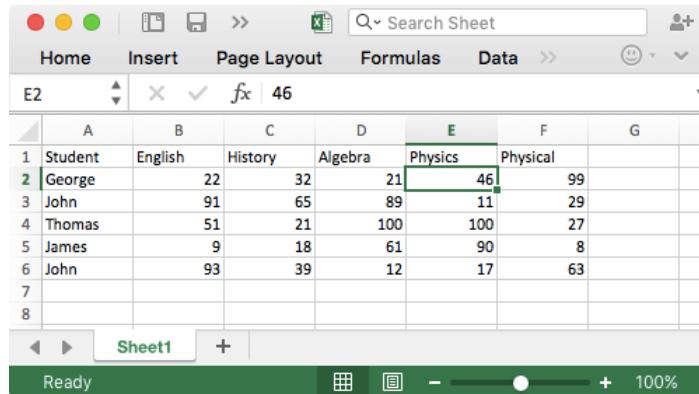
Clicking on a part of the report also allows you to add a comment.



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

Lesson 4: Loading Your Own Data Set

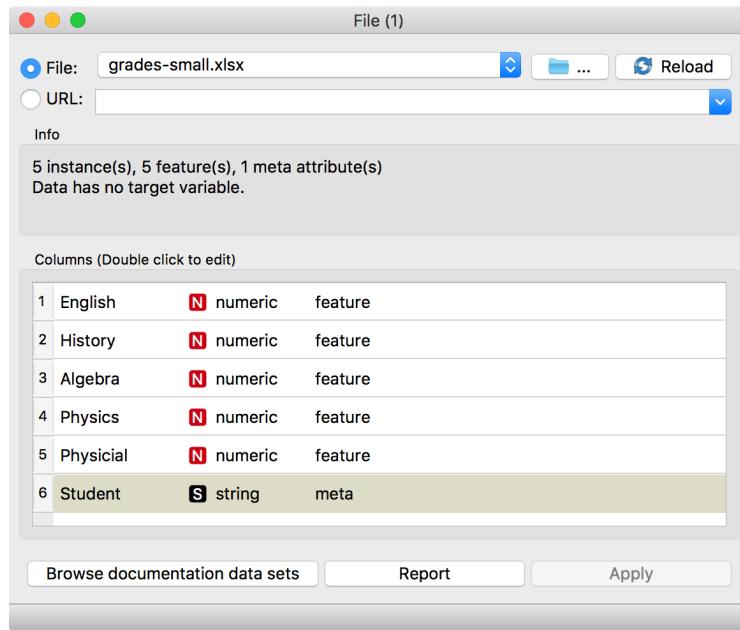
The data sets we have worked with in previous lessons come with Orange installation. Orange can read data from spreadsheet file formats which include tab and comma separated and Excel files. Let us prepare a data set (with school subjects and grades) in Excel and save it on a local disk.



A screenshot of Microsoft Excel showing a data table. The table has columns labeled A through G. Row 1 contains column headers: Student, English, History, Algebra, Physics, and Physical. Rows 2 through 6 contain data for five students: George, John, Thomas, James, and John. The 'Physics' column for George is highlighted in green, containing the value 46. The table is on 'Sheet1'. The status bar at the bottom shows 'Ready'.

	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.



A screenshot of the Orange Data Explorer interface. The 'File' configuration window is open, showing 'grades-small.xlsx' selected under 'File'. The 'Info' section indicates there are 5 instances, 5 features, and 1 meta attribute. The 'Columns' section lists the data columns: English, History, Algebra, Physics, Physical, and Student. The 'Student' column is identified as a string meta feature. At the bottom are 'Browse documentation data sets', 'Report', and 'Apply' buttons.

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.

The screenshot shows the 'Data Table' interface. On the left, there's a sidebar with 'Info' (5 instances, 5 features, 1 meta attribute), 'Variables' (checkboxes for Show variable labels, Visualize numeric values, Color by instance classes), 'Selection' (checkbox for Select full rows), and buttons for Restore Original Order, Report, and Send Automatically. The main area displays a table with columns: Student, English, History, Algebra, Physics, and Physical. The data rows are:

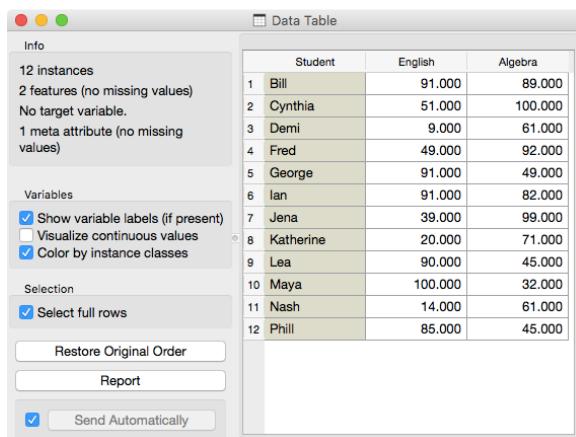
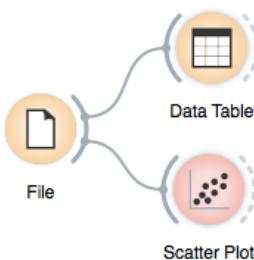
	Student	English	History	Algebra	Physics	Physical
1	George	22.000	32.000	21.000	46.000	99.000
2	John	91.000	65.000	89.000	11.000	29.000
3	Thomas	51.000	21.000	100.000	100.000	27.000
4	James	9.000	18.000	61.000	90.000	8.000
5	John	93.000	39.000	12.000	17.000	63.000

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.

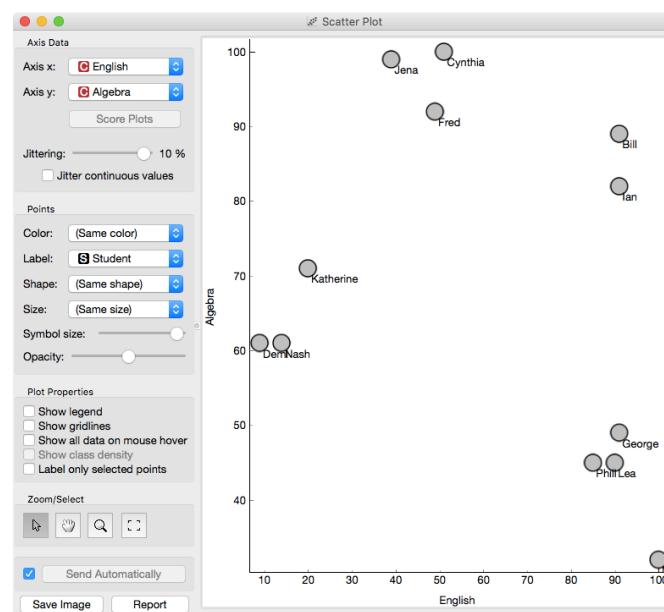
There is more to input data formatting and loading. We can define the type and kind of the data column, specify that the column is a web address of an image, and more. But enough for the first day. If you would like to dive deeper, check out the [documentation page on Loading your Data](#), or [a video](#)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

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>.



Lesson 5: Hierarchical Clustering

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



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

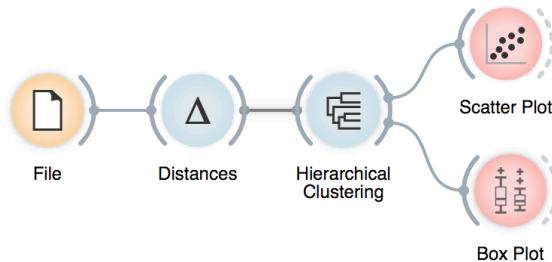
We need to start with a definition of “similar”. One simple measure of similarity for such data is the Euclidean distance: square the differences across every dimension, sum them and take the square root, just like in Pythagorean theorem. So, we would like to group data instances with small Euclidean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

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

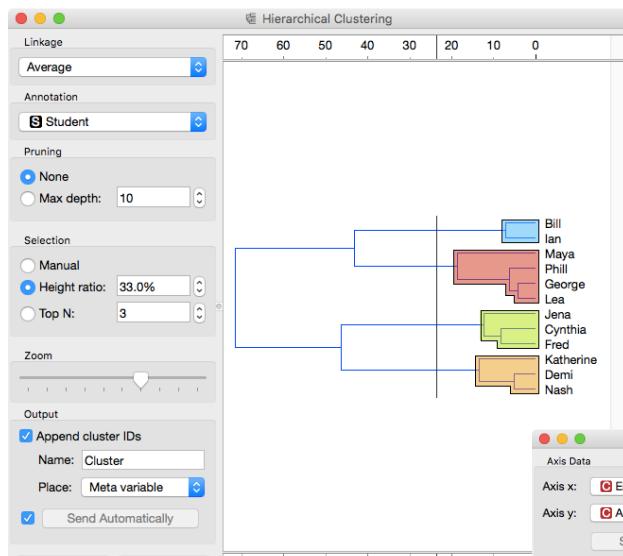
This procedure constructs a hierarchy of clusters, which explains why we call it hierarchical clustering. After it is done, we can

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

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



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



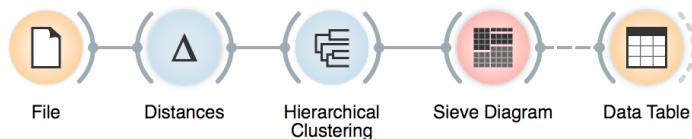
You can also observe the properties of the clusters
- that is, the average grades in Algebra and English
- in the box plot.



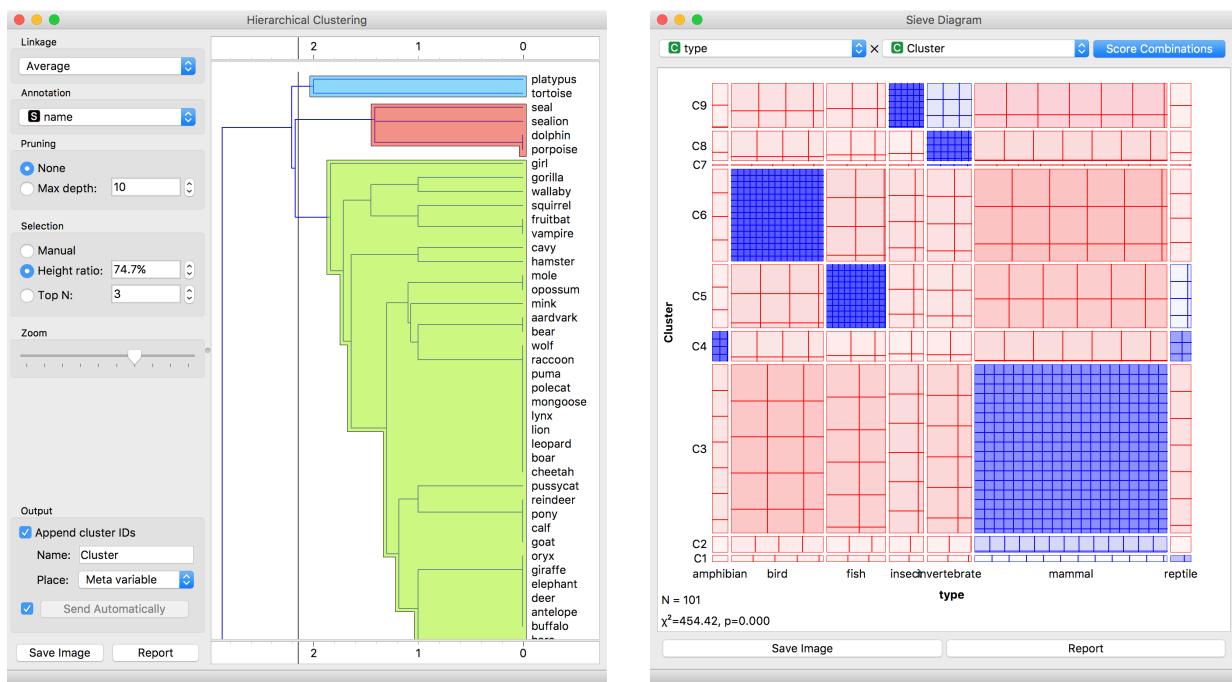
Lesson 6: Animal Kingdom



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

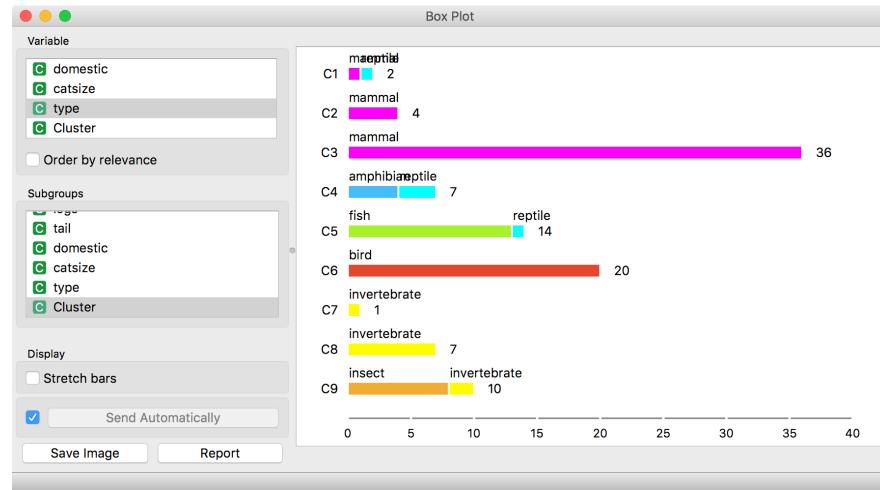


Here we use zoo data (from documentation data sets) with attributes that report on various features of animals (has hair, has feathers, lays eggs). We measure the distance and compute the clustering. Animals in this data set are annotated with type (mammal, insect, bird, and so on). It would be cool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

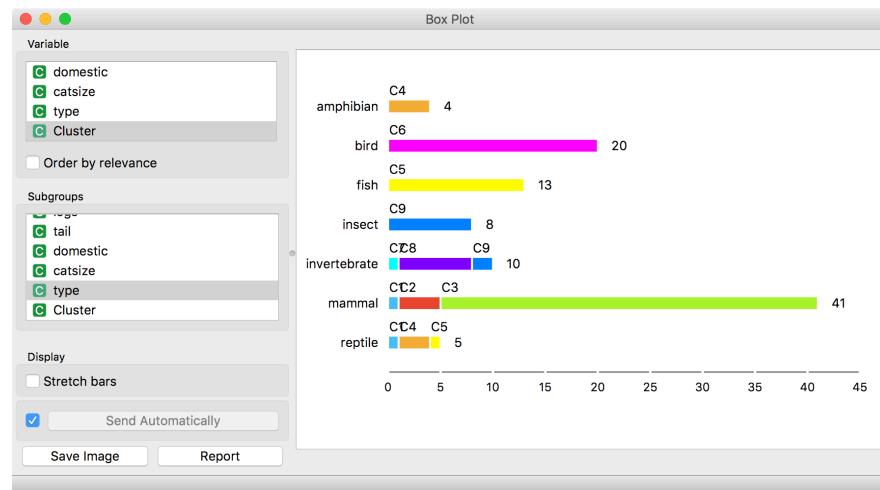


Looks great. Birds, say, are in cluster C6. Cluster C4 consists of amphibians and some reptiles. And so forth.

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



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



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

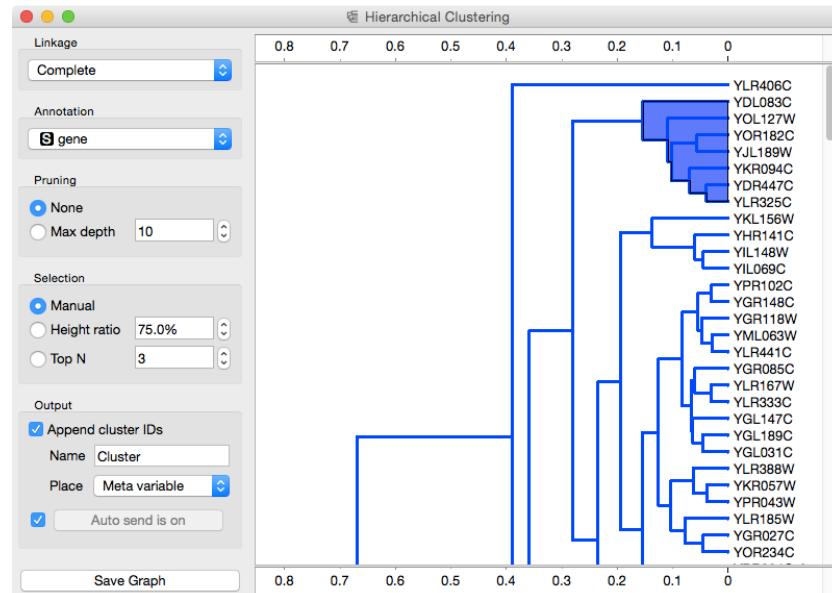
Lesson 7: Discovering clusters

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

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



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



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

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

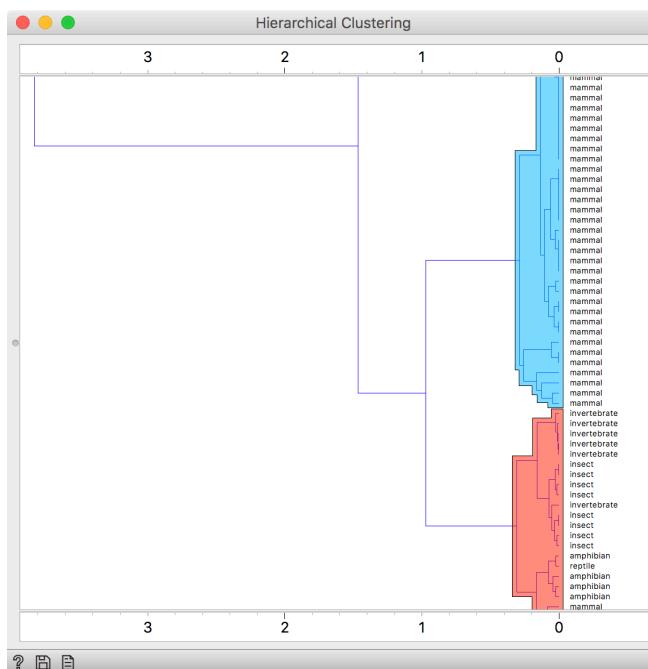
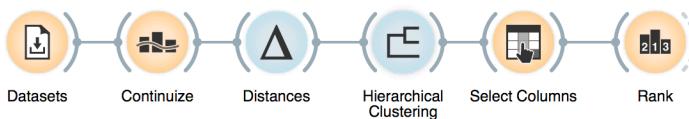
well the clusters match the classes - which we know, but clustering did not.

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

This lesson is not a recipe for what you should be doing in practice. If your data already contains group 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 meaningful subgroups in the data; we pretend we did not know the groups, use the clustering to discover them, and checked how well the correspond to the actual groups.

Lesson 8: Cluster interpretation

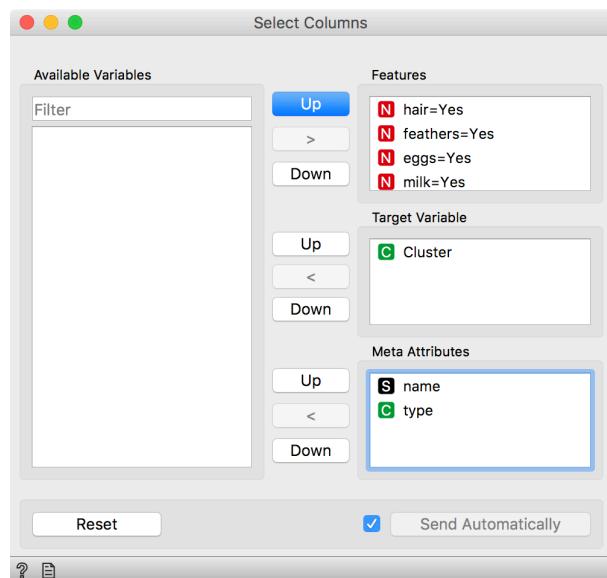
Once we have inferred the clusters, we would like to know what are the distinguishing features. For the zoo data set, we could, for instance, mark two clusters, and then ask for the features that distinguish among these. Having data marked with cluster identifiers takes us back to classification, and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



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

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

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



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

The Rank widget displays a list of attributes from highest to lowest gain ratio. The columns are labeled '#', 'Gain ratio ▼', and 'Gini'. The first two columns are sorted by Gain ratio in descending order, indicated by the downward arrow. The last column is sorted by Gini in ascending order, indicated by the upward arrow. The attributes listed are: milk=Yes, eggs=Yes, toothed=Yes, hair=Yes, backbone=Yes, legs, breathes=Yes, tail=Yes, venomous=Yes, catsize=Yes, aquatic=Yes, airborne=Yes, fins=Yes, domestic=Yes, predator=Yes, and feathers=Yes. The Gain ratio values range from 0.000 to 0.892, and the Gini values range from 0.000 to 0.428.

	#	Gain ratio ▼	Gini
N milk=Yes		0.892	0.428
N eggs=Yes		0.892	0.428
N toothed=Yes		0.663	0.310
N hair=Yes		0.663	0.310
N backbone=Yes		0.583	0.259
N legs		0.329	0.244
N breathes=Yes		0.323	0.077
N tail=Yes		0.310	0.181
N venomous=Yes		0.297	0.060
N catsize=Yes		0.273	0.162
N aquatic=Yes		0.245	0.116
N airborne=Yes		0.126	0.048
N fins=Yes		0.090	0.005
N domestic=Yes		0.068	0.023
N predator=Yes		0.005	0.003
N feathers=Yes		0.000	0.000

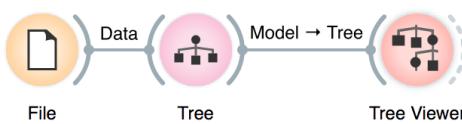
Lesson 15: Classification

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.

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

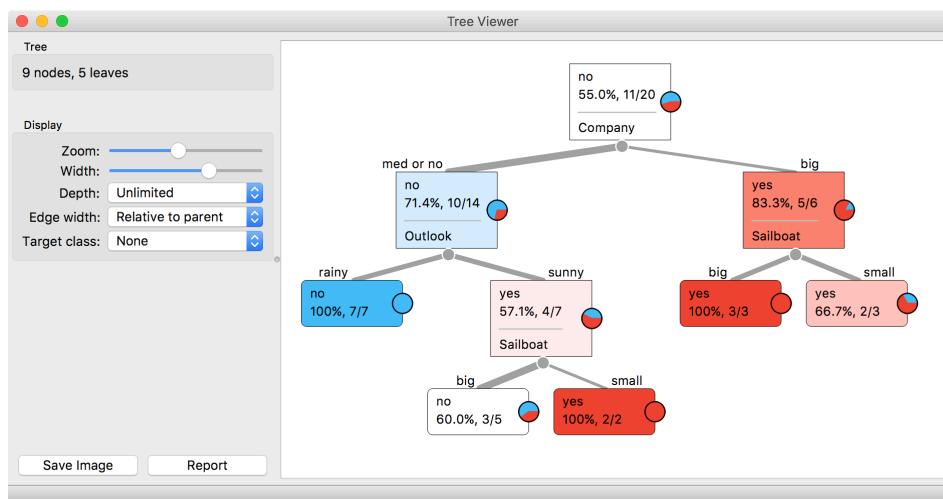
Classification tree is one of the oldest, but still popular, machine learning methods. We like it since the method is easy to explain and gives rise to random forests, one of the most accurate machine learning techniques. So, what kind of model is a classification tree?

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



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?

	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



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

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

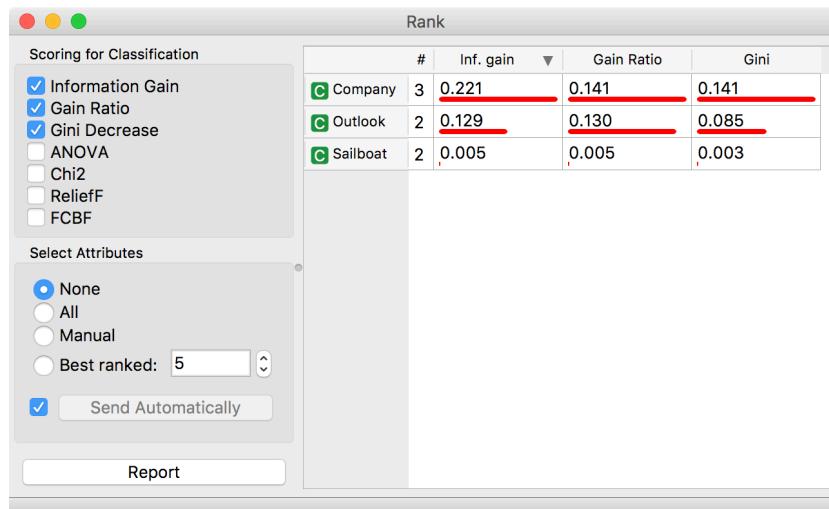
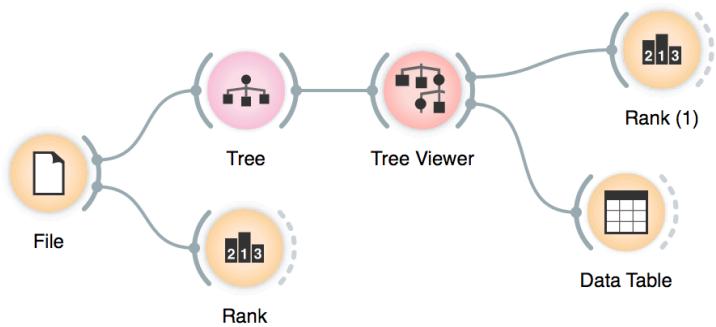
Classification trees were hugely popular in the early years of machine learning, when they were first independently proposed by the engineer Ross Quinlan (C4.5) and a group of statisticians (CART), including the father of random forests Leo Brieman.

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

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.



Lesson 16: Model Inspection

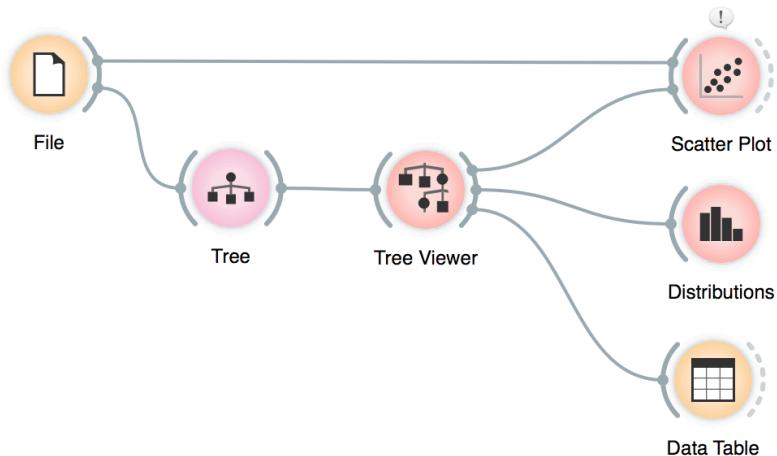
As Arthur Weasley used to say, you should never trust a thing if you don't know where it keeps its brains. For predictive models, it is always great to see them and understand how they make decisions. This may let us learn about patterns they spotted in the data, interpret predictions they make, it may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

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

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

Wherever possible, visualizations in Orange are designed to support selection and passing of the data that applies to it.

Finding interesting data subsets and analyzing their commonalities is a central part of explorative data analysis, a data analysis approach favored by the data visualization guru Edward Tufte.

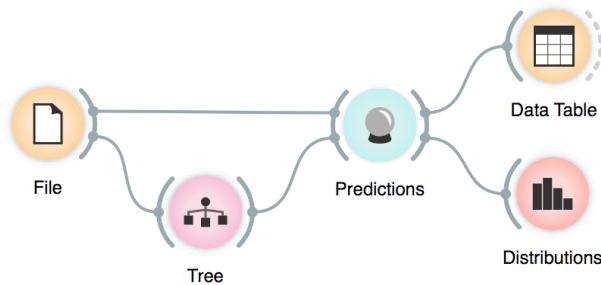


We can also use Distributions widget to see the distribution of classes in each node. In the case of the Brown data, this is pretty boring, though.

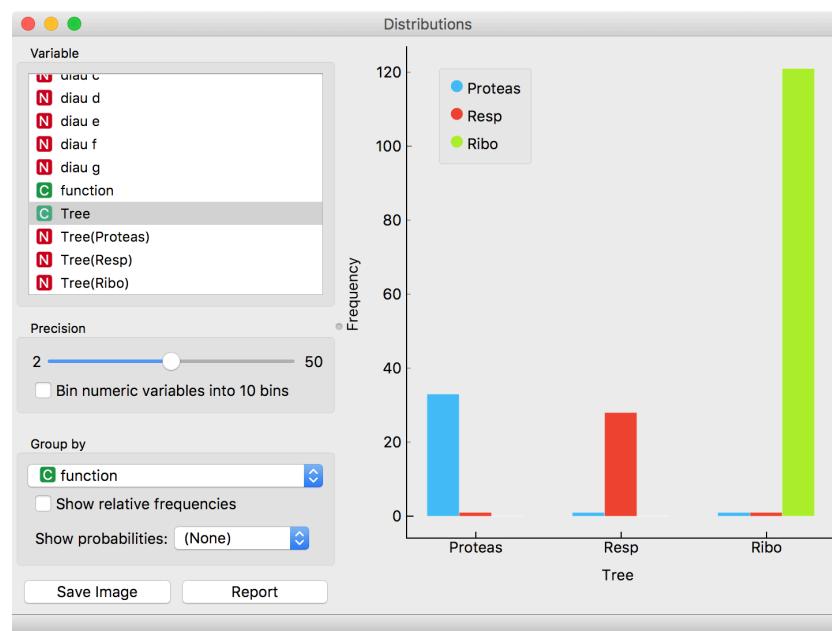
Lesson 17: Classification Accuracy

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

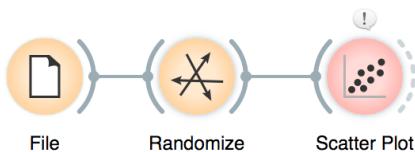
Measuring of accuracy is such an important concept that it would require its widget. But wait a while, there's educational value in reusing the widgets we already know.



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



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

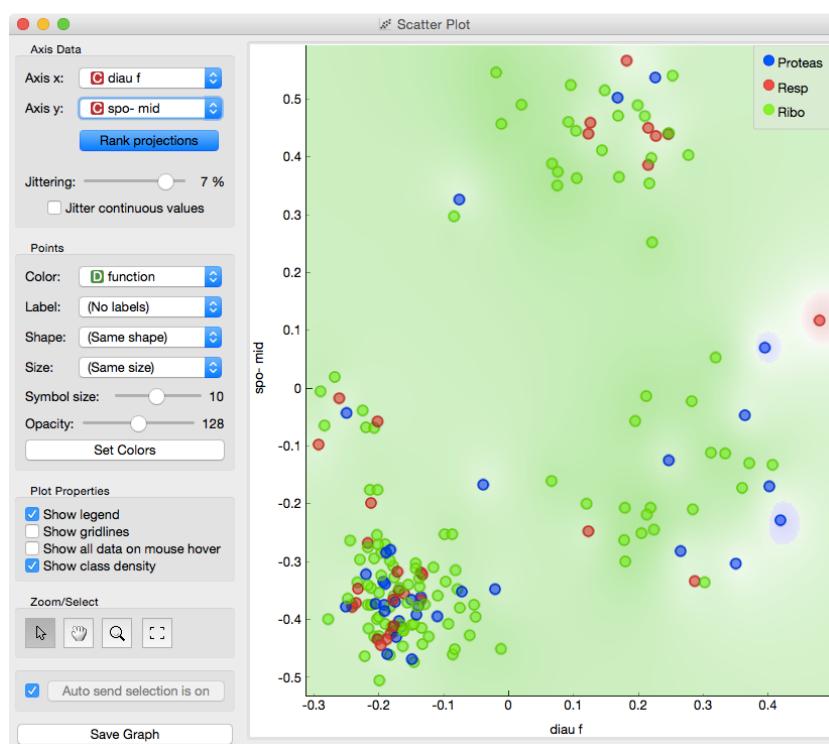


Randomize widget shuffles the column in the data table. It can shuffle the class column, columns with data features or columns with meta information. Shuffling the class column breaks any relation between features and the class, keeping the data points (genes profiles) intact.

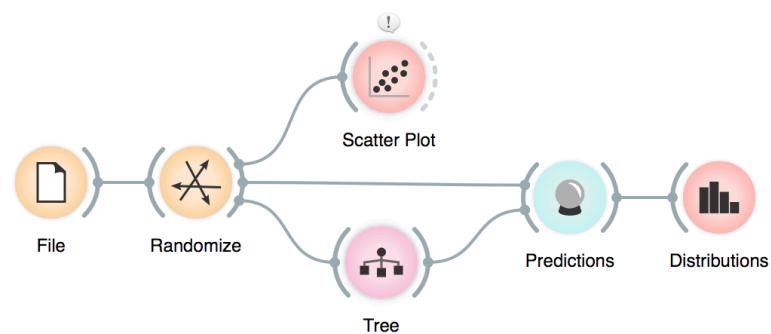
Why is the background in this scatter plot so green, and only green? Why have the other colors disappeared after the class randomization?

Lesson 18: How to Cheat

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

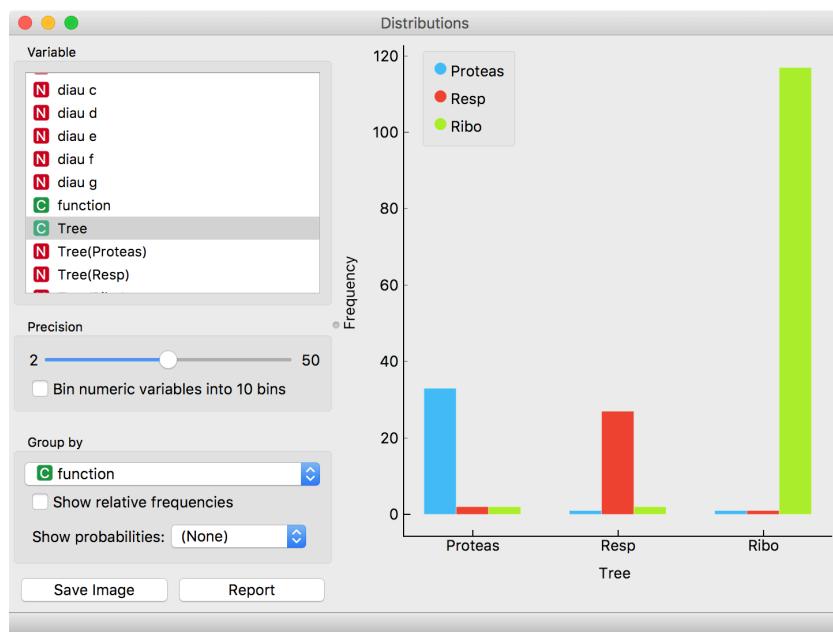


Fine. There can be no classifier that can model this mess, right? Let us test this. We will build classification tree and check its performance on the messed-up data set.



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

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



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

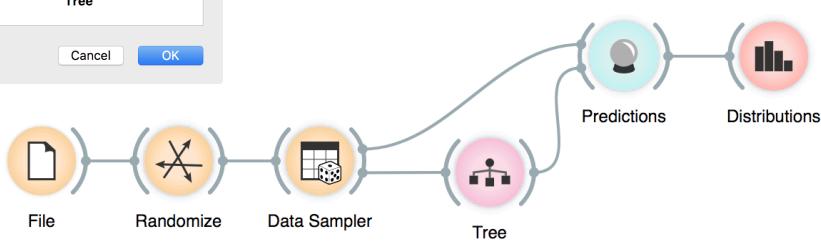
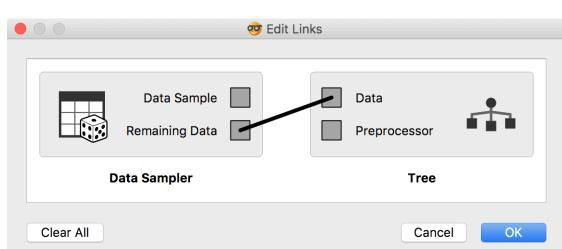
The signals from the Data Sampler widget have not been named in our workflow to save space. The Data Sampler splits the data to a sample and out-of-sample (so called remaining data). The sample was given to the Tree widget, while the remaining data was handed to the Predictions widget. Set the Data Sampler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

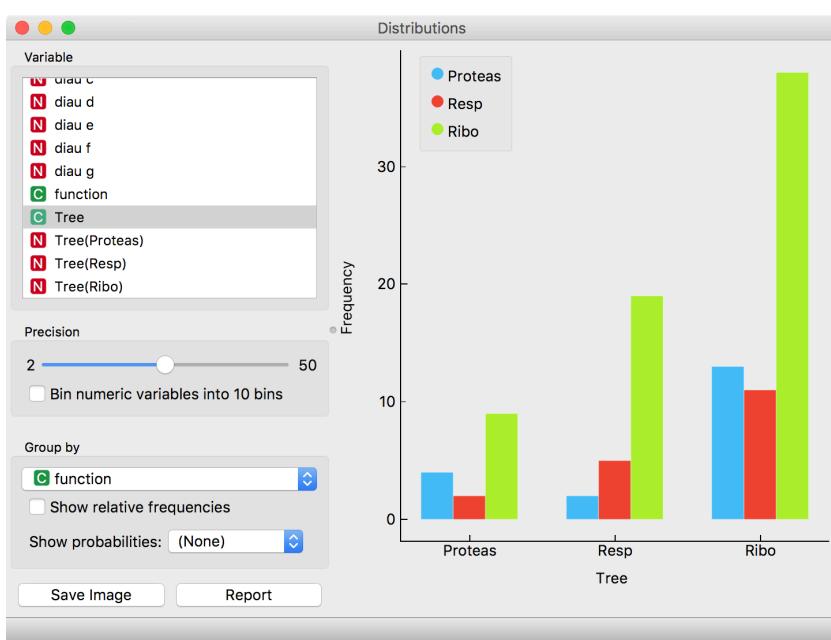
This should be a bit of *déjà vu*. Is not this the same as regression modelling with high degree polynomials?

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



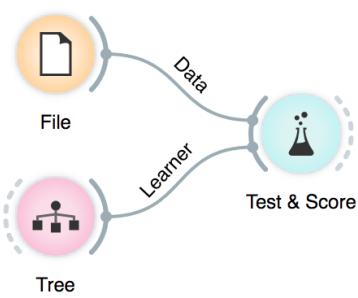
Turns out that for every class value the majority of data instances has been predicted to the ribosomal class (green). Why? Green again (like green from the Scatter Plot of the messed-up data)? Here is a hint: use the Box Plot widget to answer this question.

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



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

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



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

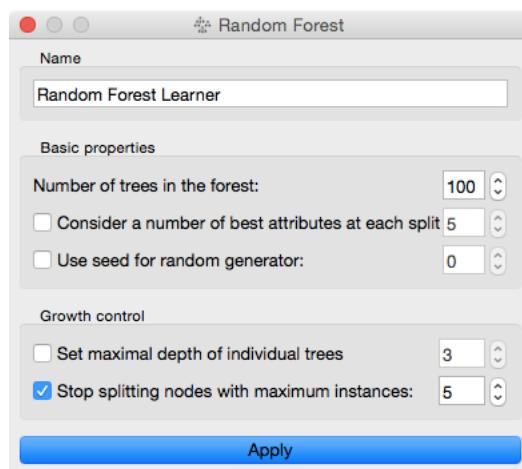
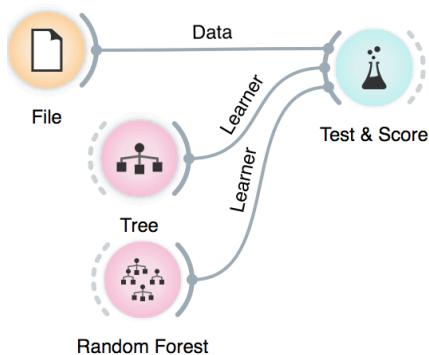
Testing classification models is thus the same as testing regression models, just with a different score. All other techniques we have seen before, such as cross-validation, apply here, too.

Lesson 19: A Few More Classifiers

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

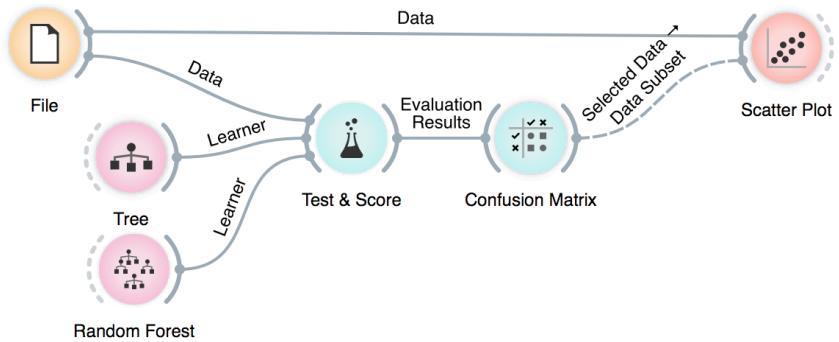
There are many other, much more accurate classifiers. A particularly interesting one is Random Forest, which averages across predictions of hundreds of classification trees. It uses two tricks to construct different classification trees. First, it infers each tree from a sample of the training data set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

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

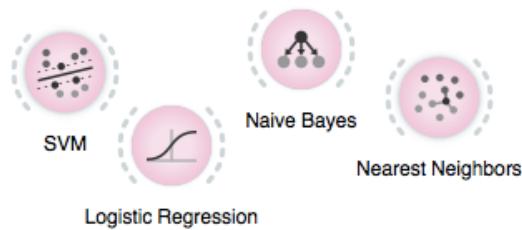


It may be interesting to compare where different classification methods make mistakes. We can use Confusion Matrix for this purpose, and then pass the signal from this widget to the Scatter Plot.

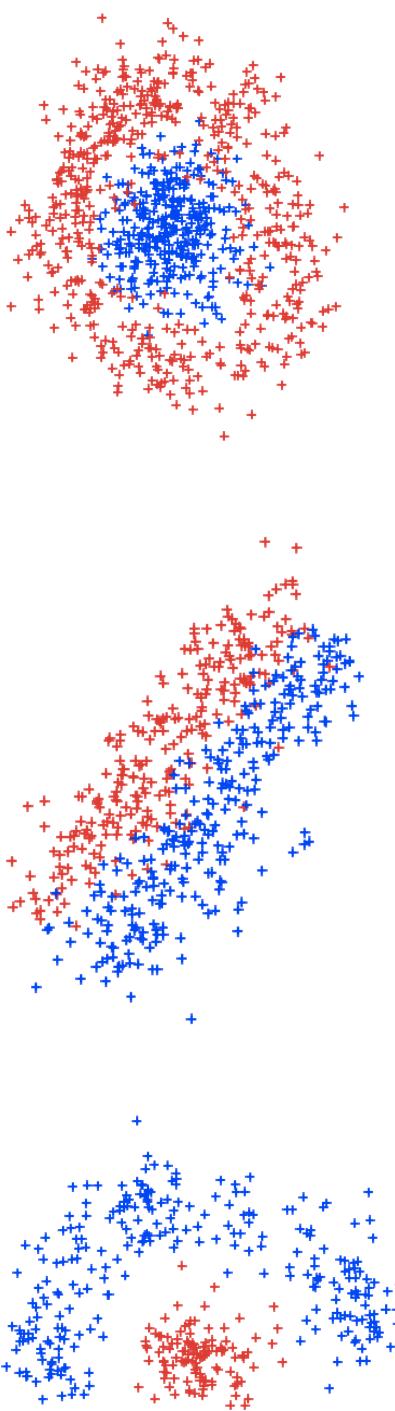
What kind of object is sent from the Test & Score widget to the Confusion Matrix widget? So far, we have used widgets that send data, or even learners. But what could the Test & Score widget communicate to other widgets?



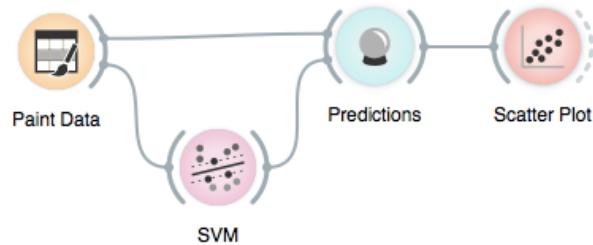
There are other classifiers we can try. We will briefly mention a few more, we won't dive too deep into how they work (we could spend a semester on this!).



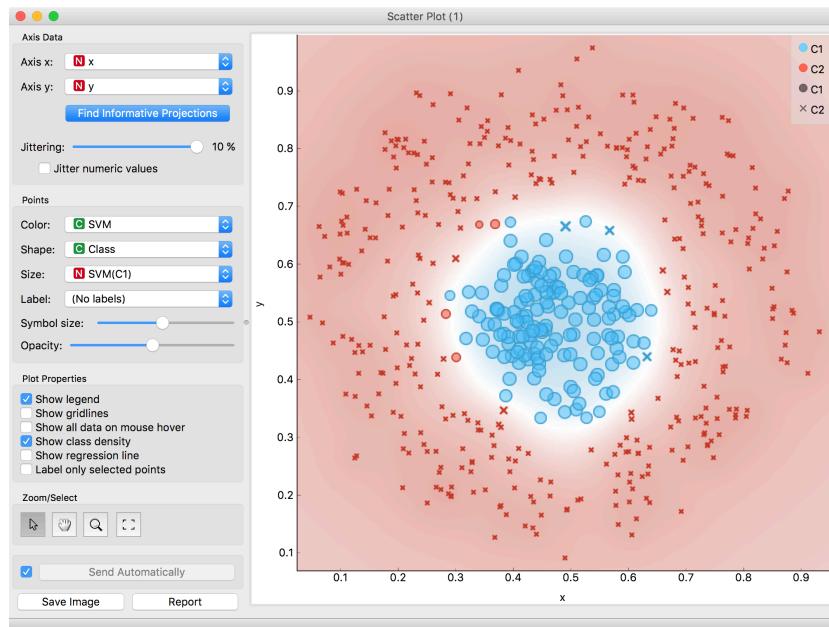
It would be nice if we could, at least on the intuitive level, understand the differences between all these methods and their variants (every method has some parameters). Remember, the classification tree finds hyperplanes orthogonal to the axis; those hyperplanes split the data space to regions with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



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



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

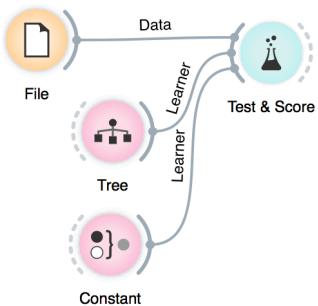


Lesson 20: Model Scoring

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

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

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



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

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

What do other columns represent? Keep reading!

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

The problem with classification accuracy goes deeper, though.

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

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

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

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

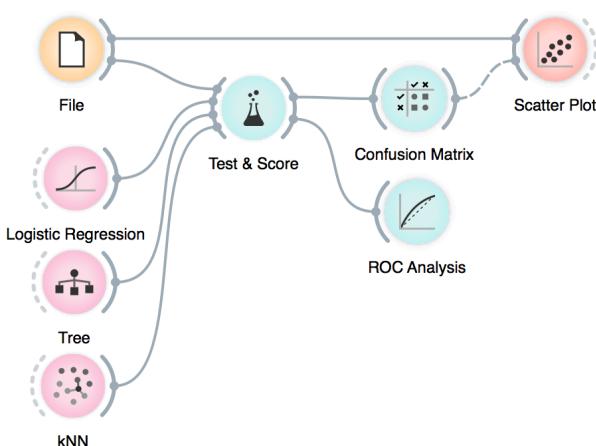
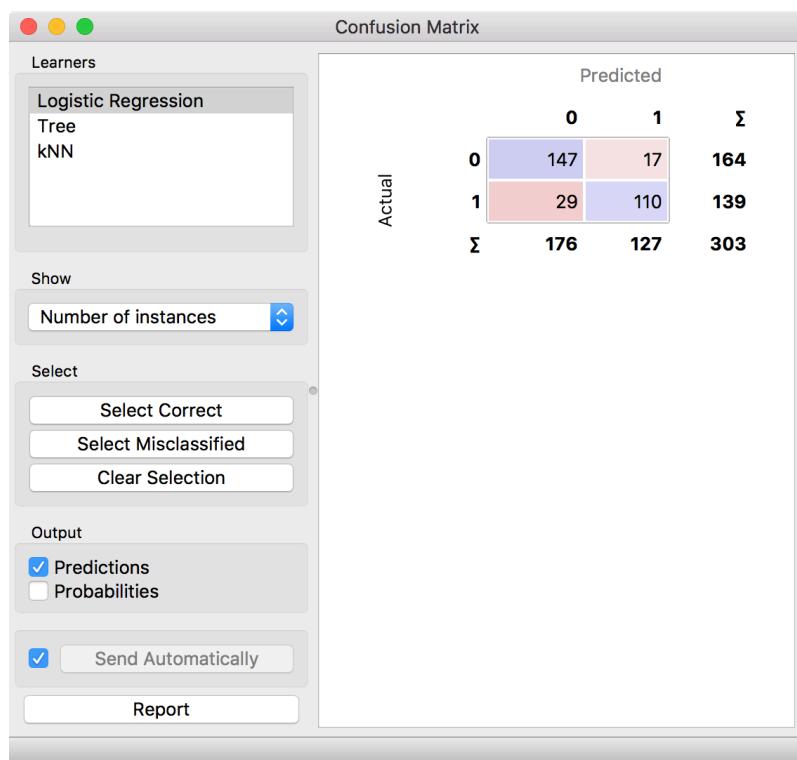
Say we decide that any leg with a 1% chance of being broken will be classified as broken. What will this do to our classification threshold? It is going to decrease badly — but we apparently do not care. What do we care about then? What kind of “accuracy” is important?

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

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

Identify them in the table!

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



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

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

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

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

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

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

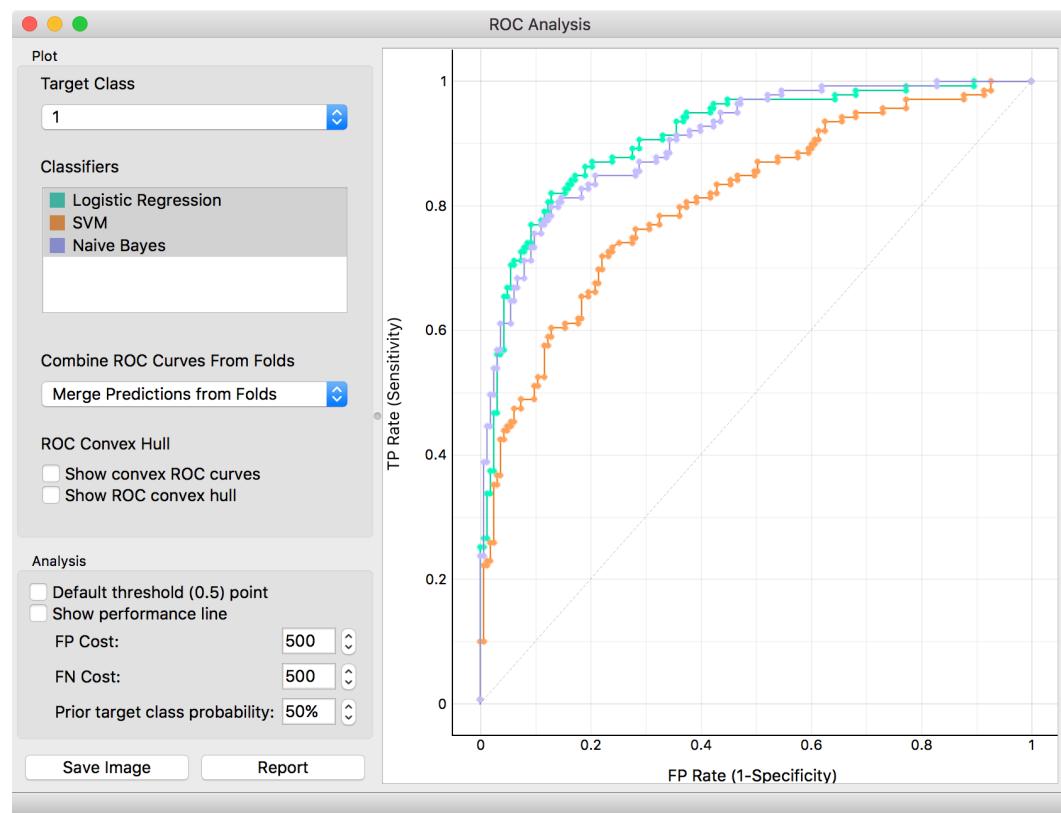
So, if you're classified as OK, you have a 90% chance of actually being OK? No, it's the other way around: 90% is the chance of being classified as OK, if you are OK. (Think about it, it's not as complicated as it sounds). If you're interested in your chance of being OK if the classifier tells you so,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

Lesson 16: Choosing the Decision Threshold

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

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

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



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

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

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

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

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

There is a popular score derived from the ROC curve, called Area under curve, AUC. It measures, well, the area under the curve.

This curve. If the curve goes straight up and then right, the area is 1; such an optimal AUC is not reached in practice. If the classifier guesses at random, the curve follows the diagonal and AUC is 0.5. Anything below that is equivalent to guessing + bad luck.

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

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

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

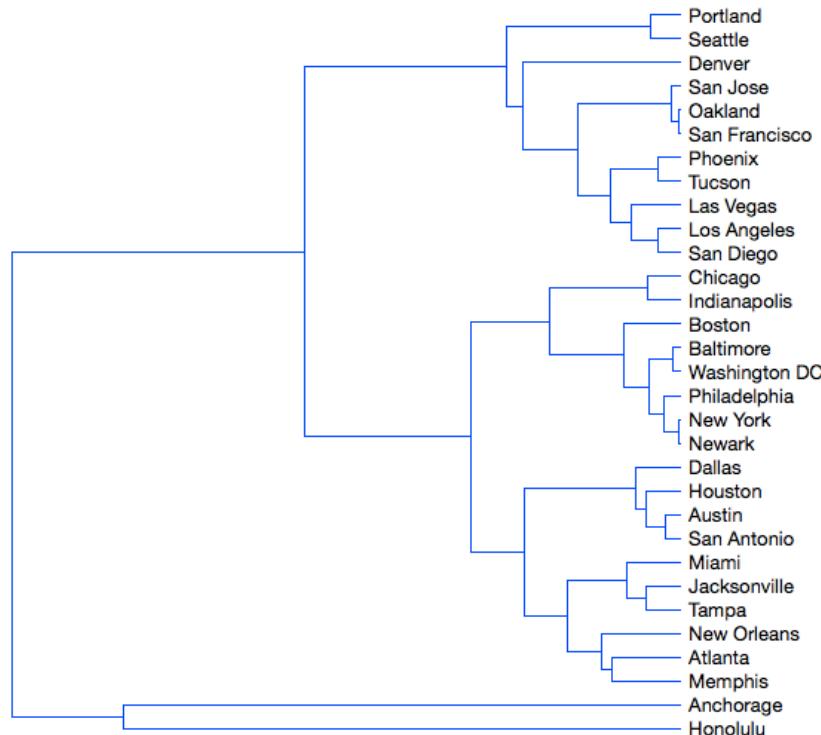
Lesson 18: Mapping the Data

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

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

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

The file is available at <http://file.biolab.si/files/us-cities.dst.zip>. To load it, unzip the file and use the File Distance widget from the Prototypes add-on.



How much sense does it make? Austin and San Antonio are closer to each other than to Houston; the tree is then joined by Dallas. On the other hand, New Orleans is much closer to Houston than to Miami. And, well, good luck hitchhiking from Anchorage to Honolulu.

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

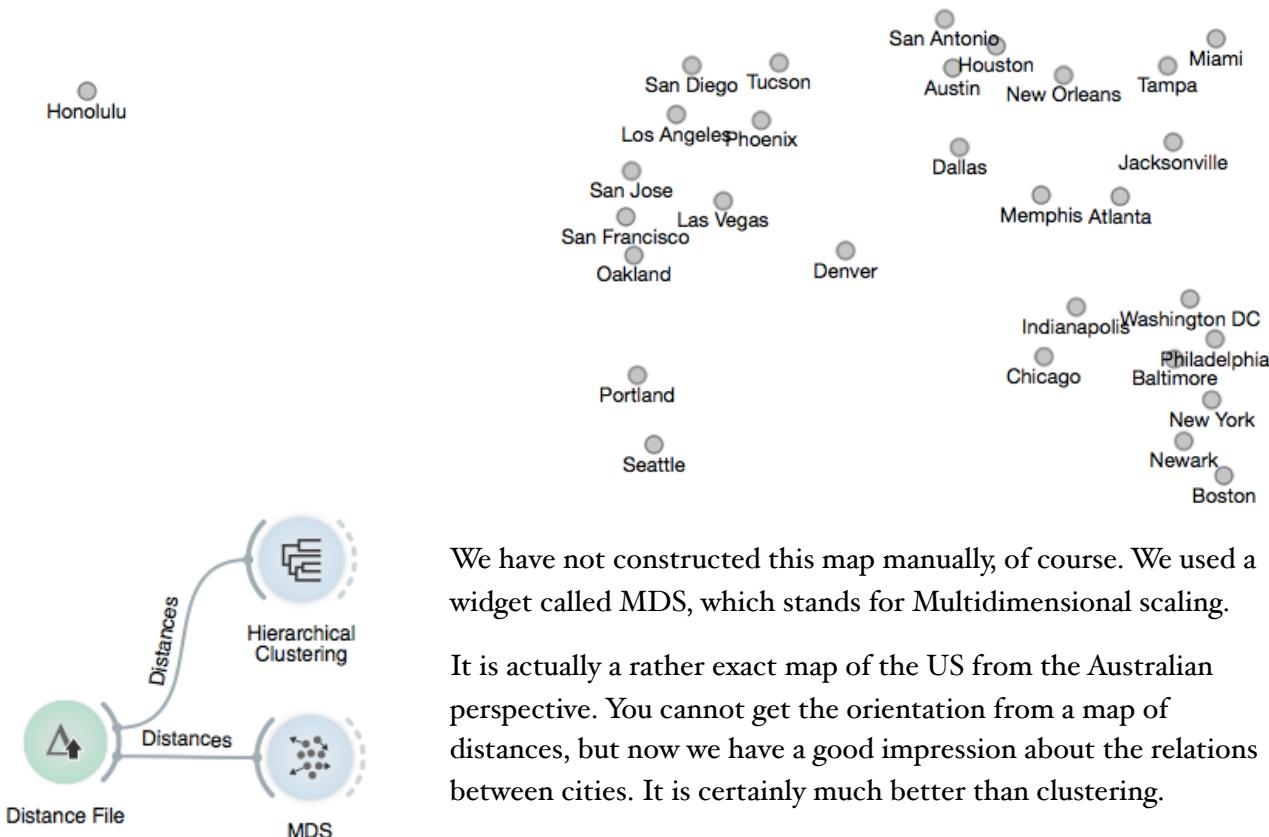
The real problem is New Orleans and San Antonio: New Orleans is close to Atlanta and Memphis, Miami is close to Jacksonville and

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.

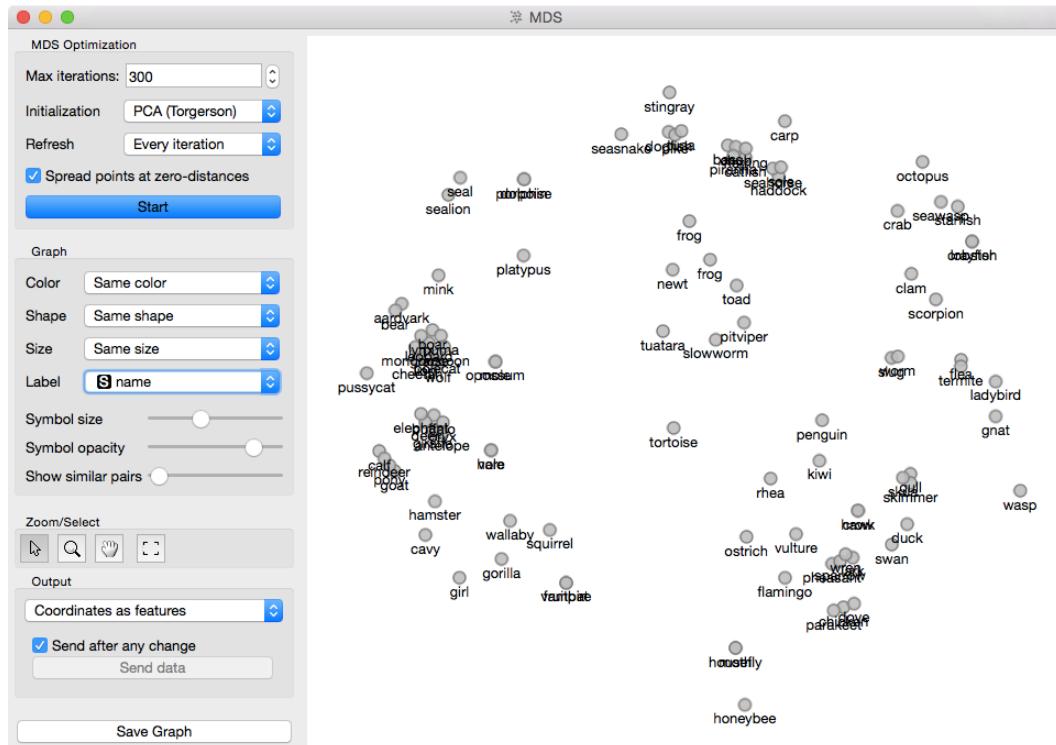
To get a better impression about the physical layout of cities, people have invented a better tool: a map! Can we reconstruct a map from a matrix of distances? Sure. Take any pair of cities and put them on paper with a distance corresponding to some scale. Add the third city and put it at the corresponding distance from the two. Continue until done. Excluding, for the sake of scale, Anchorage, we get the following map.



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

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

Remember the clustering of animals? Can we draw a map of animals?



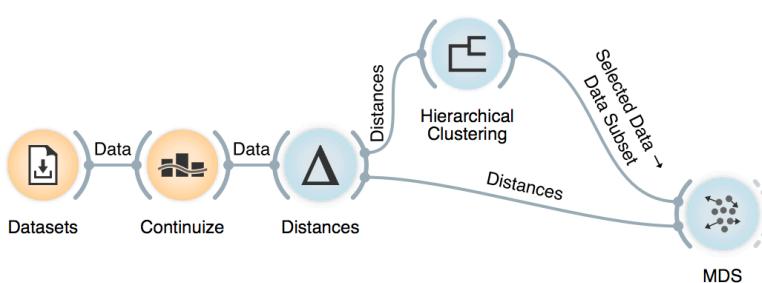
Does the map make any sense? Are similar animals together? Color the points by the types of animals and you should see.

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

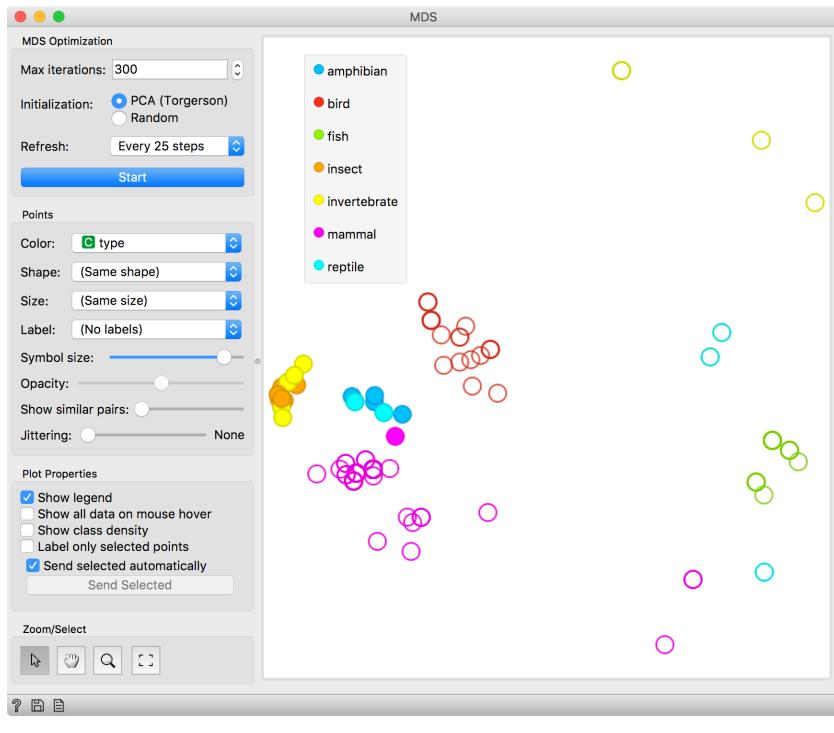
Lesson 19: Maps and Clusters

Remember the mixed cluster in the zoo data that contained invertebrates, reptiles, amphibian, and even a mammal. Was this a homogeneous cluster? Why the mammal there? And how far is this mammal to other mammals? And why is this cluster close to the cluster of mammals?



so we need to take care of the composition of the workflow and proper connections between widgets.

So many questions. But we can answer them all with a combination of clustering and multi-dimensional scaling. We would like to show any cluster that we selected from a dendrogram to be shown on the map of animals presented by MDS. And we would like to use cosine distances,

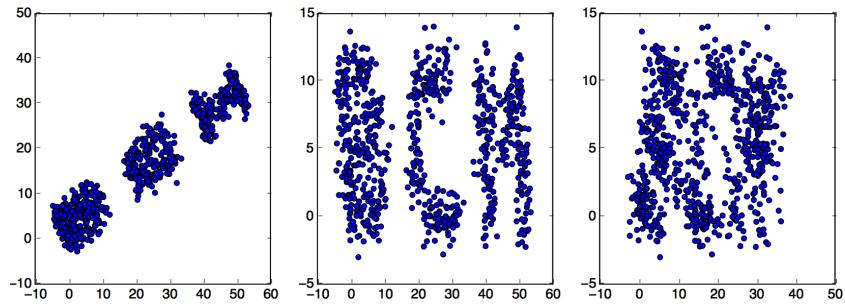


Clustering and two-dimensional embedding make a great combination for data exploration. Clustering finds the coherent groups, and embedding, such as MDS, reveals the relations between the clusters and positions the cluster on the data map. There are other dimensionality reduction and embedding techniques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

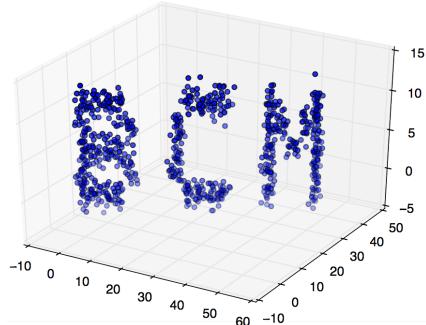
Can you change the workflow to explore the position of individual clusters found by k-means?

Lesson 20: Principal Component Analysis

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



Yes, the first scatter plot looks very useful: it tells us that x and y are highly correlated and that we have three clusters of somewhat irregular shape. But remember: this data is three dimensional. What if we saw it from another, perhaps better perspective?



Let's make another experiment. Go to <https://in-the-sky.org/ngc3d.php>, disable Auto-rotate and Show labels and select Zoom to show Local Milky Way. Now let's rotate the picture of the galaxy to find the layout of the stars.

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

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

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

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

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

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

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

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

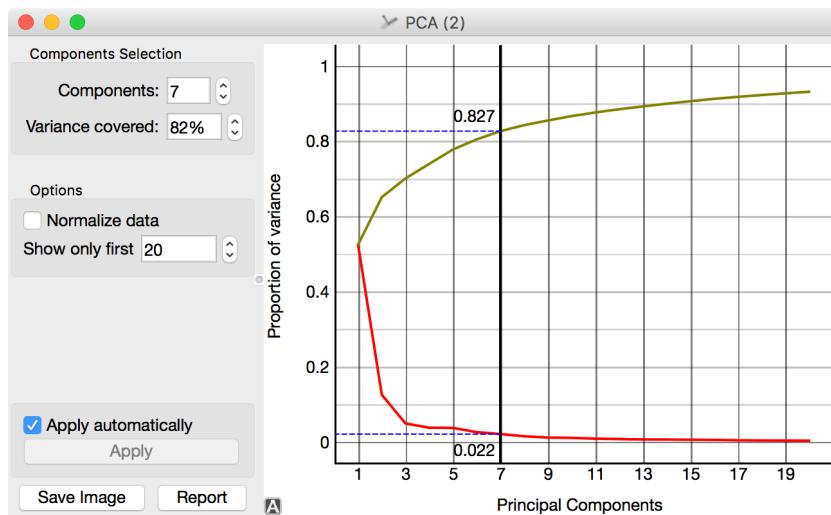
It turns out the flies are also spread in the third direction. Thus you need three numbers after all.

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

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

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

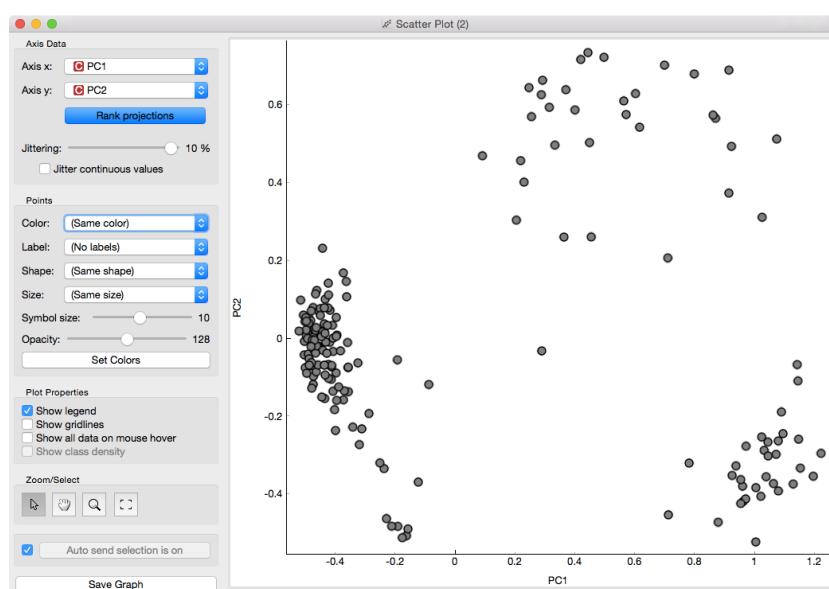
A technique that does this is called Principle Components Analysis, or PCA. The corresponding widget is simple: it receives the data and outputs the transformed data.



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

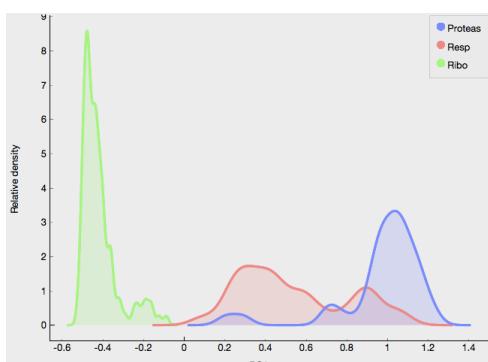
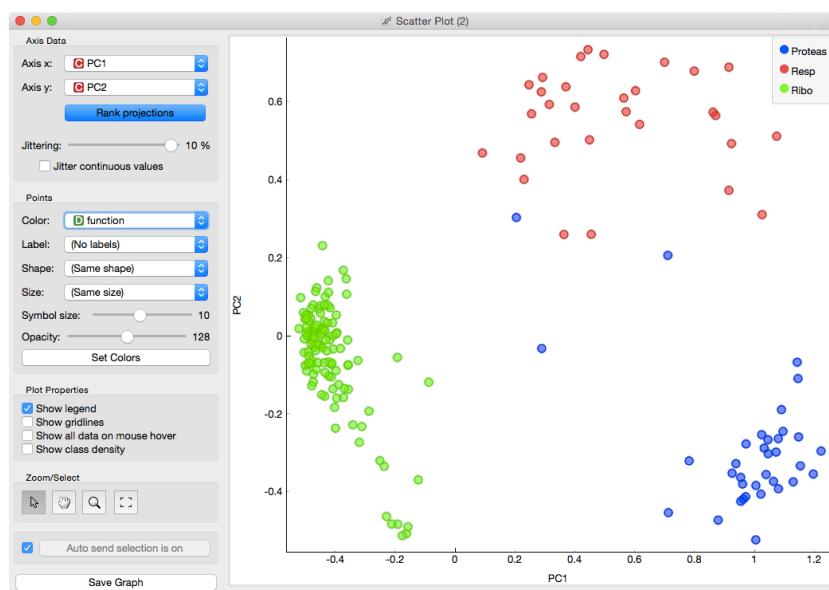
The PCA on the left shows the scree diagram for brown-selected data. Set like this, the widget replaces the 80 features with just seven - and still

keeping 82.7% of information. (Note: disable "Normalize data" checkbox to get the same picture.) Let us see a scatter plot for the first two components.

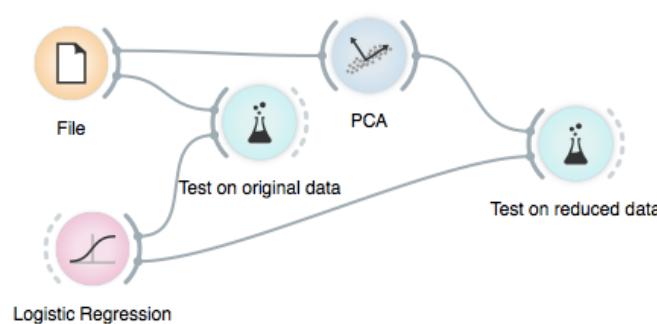


The axes, PC₁ and PC₂, do not correspond to particular features in the original data, but to their linear combination. What we are looking at is a projection onto the plane, defined by the first two components. When you consider only two components, you can imagine that PCA put a hyperplane into multidimensional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

Note that this is an unsupervised method: it does not care about the class. The classes in the projection may be well separated or not. Let's add some colors to the points and see how lucky we are this time.



The data separated so well that these two dimensions alone may suffice for building a good classifier. No, wait, it gets even better. The data classes are separated well even along the first component. So we should be able to build a classifier from a single feature!



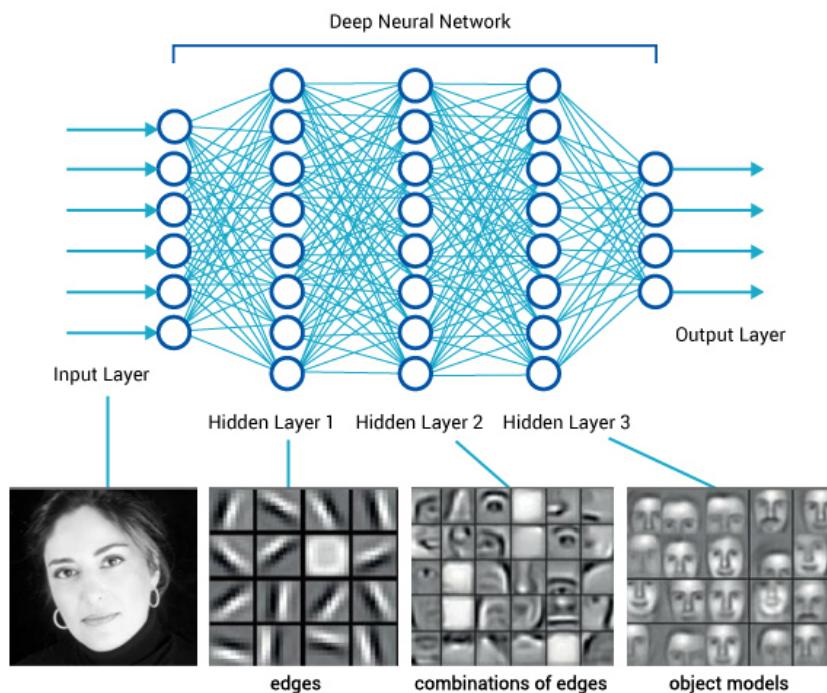
In the above schema we use the ordinary Test & Score widget, but renamed it to “Test on original data” for better understanding of the workflow.

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

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

Lesson 21: Image Embedding

This depiction of deep learning network was borrowed from
<http://wwwamax.com/blog/?>



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.

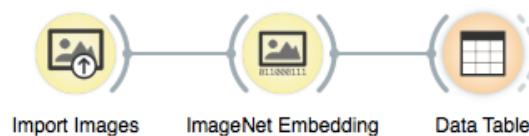


Image embedding describes the images with a set of 2048 features appended to the table with meta features of images.

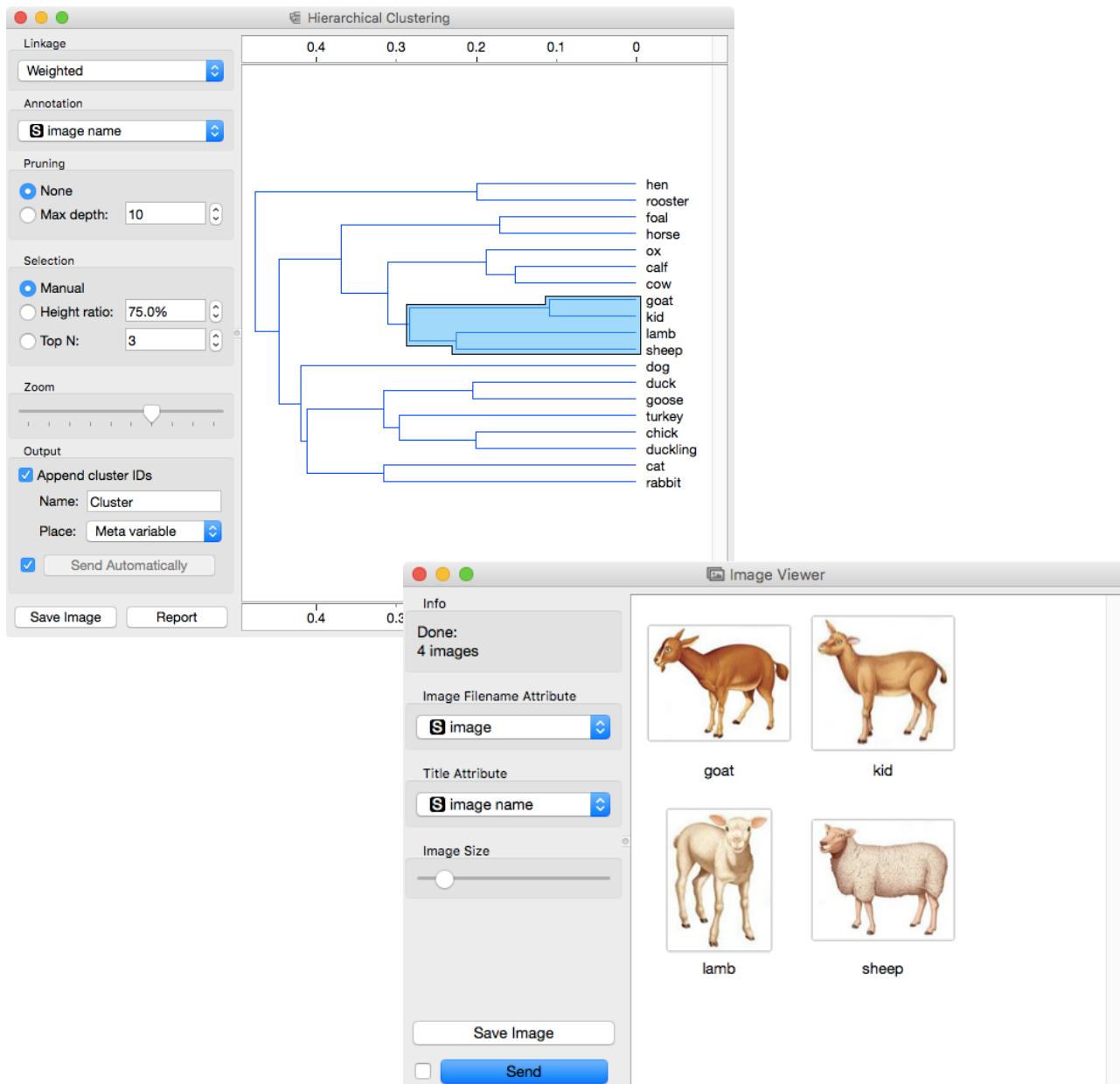
	image name	image image	size	width	height	n0	n1	n2	n3	n4	n5	n6
1	calf	/Users/bla...	45538	191	152	0.181	0.212	0.041	0.016	0.180	0.071	0.24
2	cat	/Users/bla...	22193	105	137	0.055	0.156	0.649	0.000	0.156	0.136	0.24
3	chick	/Users/bla...	14891	85	92	0.127	0.032	0.097	0.015	0.169	0.080	0.11
4	cow	/Users/bla...	62159	210	189	0.475	0.130	0.048	0.082	0.130	0.599	0.24
5	dog	/Users/bla...	28745	129	125	0.049	0.187	0.181	0.111	0.188	0.516	0.61
6	duck	/Users/bla...	39583	158	172	0.131	0.037	0.073	0.040	0.162	0.221	0.18
7	duckling	/Users/bla...	17109	99	119	0.068	0.050	0.033	0.055	0.184	0.189	0.11
8	foal	/Users/bla...	39210	147	177	0.061	0.252	0.040	0.155	0.481	0.348	0.11
9	goat	/Users/bla...	53039	221	179	0.265	0.124	0.017	0.019	0.176	0.110	0.24
10	goose	/Users/bla...	34442	141	202	0.355	0.246	0.159	0.000	0.422	0.374	0.11
11	hen	/Users/bla...	41716	134	168	0.389	0.062	0.037	0.083	0.429	0.218	0.11
12	horse	/Users/bla...	69109	285	195	0.280	0.229	0.084	0.095	0.387	0.295	0.21
13	kid	/Users/bla...	36290	170	160	0.131	0.140	0.024	0.067	0.130	0.030	0.11
14	lamb	/Users/bla...	35520	123	168	0.358	0.034	0.189	0.055	0.331	0.162	0.41
15	ox	/Users/bla...	56401	191	189	0.520	0.003	0.096	0.106	0.139	0.235	0.21

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



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

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



Lesson 22: Images and Classification

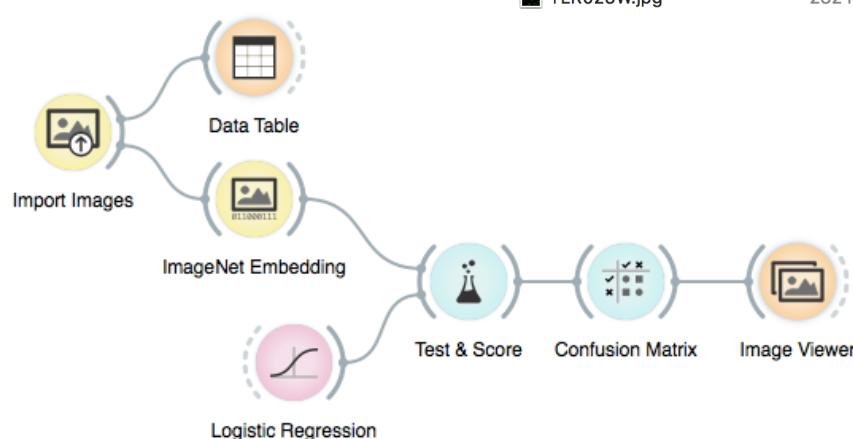
In this lesson, we are using images of yeast protein localization (<http://file.biobab.si/files/yeast-localization-small.zip>) in the classification setup. But this same data set could be explored in clustering as well. The workflow would be the same as the one from previous lesson. Try it out! Do Italian 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
cytoplasm	--
YAL005C.jpg	163 KB
YAL011W.jpg	269 KB
YAL012W.jpg	256 KB
YAR019C.jpg	256 KB
YAR071W.jpg	276 KB
YBL001C.jpg	162 KB
YBL008W.jpg	256 KB
YBL016W.jpg	180 KB
YBL019W.jpg	41 KB
YBL036C.jpg	256 KB
YBL039C.jpg	298 KB
YBL051C.jpg	184 KB
endosome	--
YBL017C.jpg	224 KB
YBR097W.jpg	185 KB
YDR323C.jpg	184 KB
YDR456W.jpg	213 KB
YGR206W.jpg	211 KB
YJL053W.jpg	223 KB
YJR044C.jpg	233 KB
YLR025W.jpg	232 KB

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

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



For the End

The course on Introduction to Data Mining at University of Ljubljana and its installment in 2018 ends here. We covered quite some mileage, and we hope we have taught you some essential procedures that should be on the stack of every data scientists. The goal was not to turn you into one but to get you familiar with some basic techniques, tools, and concepts. Data science is a vast field, and it takes years of study and practice to master it. You may never become a data scientist, but as an expert in biomedicine, it should now be more comfortable to talk and collaborate with statisticians and computer scientists. And for those who want to go ahead with data science, well, you now know where to start.