

# Data Mining

Working notes for the hands-on course at the  
Faculty of Economics at University of Ljubljana

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

The notes were written by the members of the Bioinformatics Lab at University of Ljubljana, and are extensions of the notes by Blaz Zupan and Janez Demšar.

Welcome to the introductory workshop on Data Mining! This short course is designed for students marketing. 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.

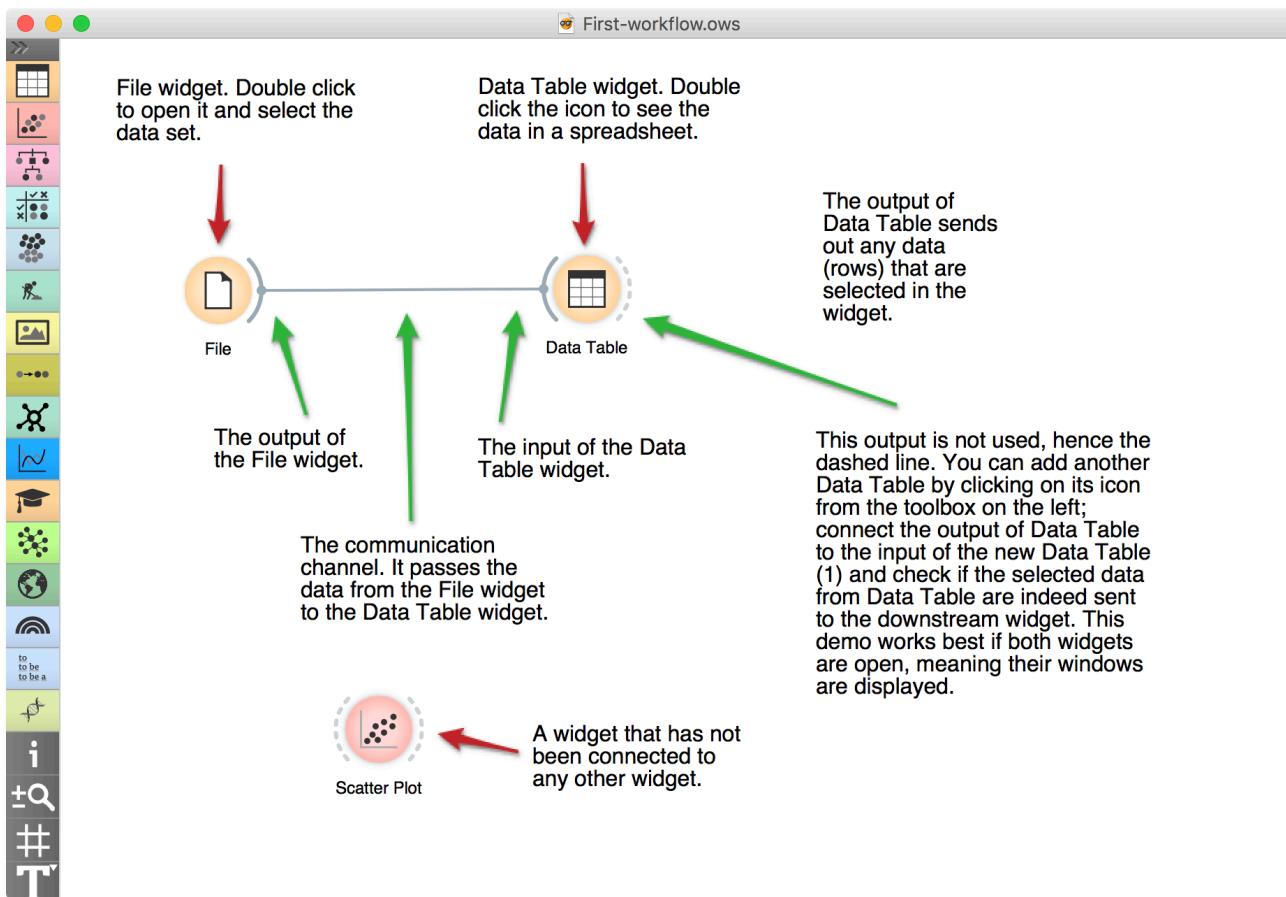
If you haven't already installed Orange, please download the installation package from <https://orange.biolab.si>.



Attribution-NonCommercial-NoDerivs  
CC BY-NC-ND

# Lesson 1: Workflows in Orange

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

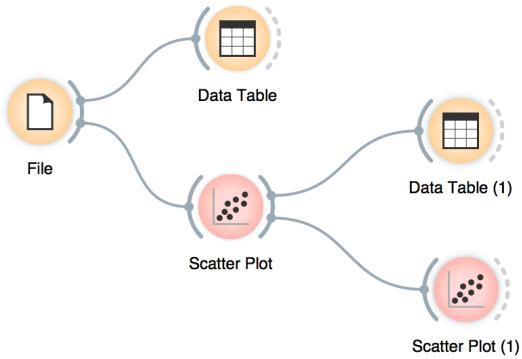


A simple workflow with two connected widgets and one widget without connections. The outputs of a widget appear on the right, while the inputs appear on the left.

We construct workflows by dragging widgets onto the canvas and connect 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.

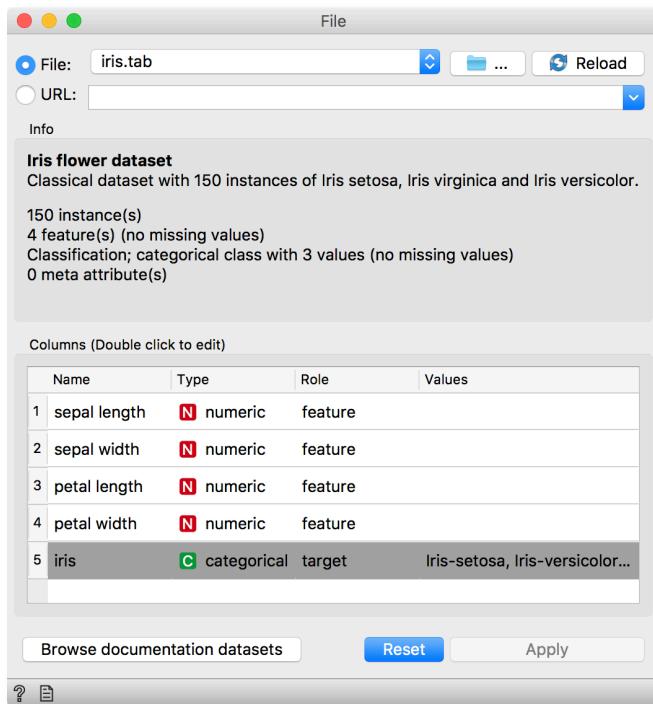
Start by constructing a workflow that consists of a File widget, two Scatter Plot widgets and two Data Table widgets:

**Workflow with a File widget that reads data from the disk and sends it to Scatter Plot and Data Table widgets. 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).**



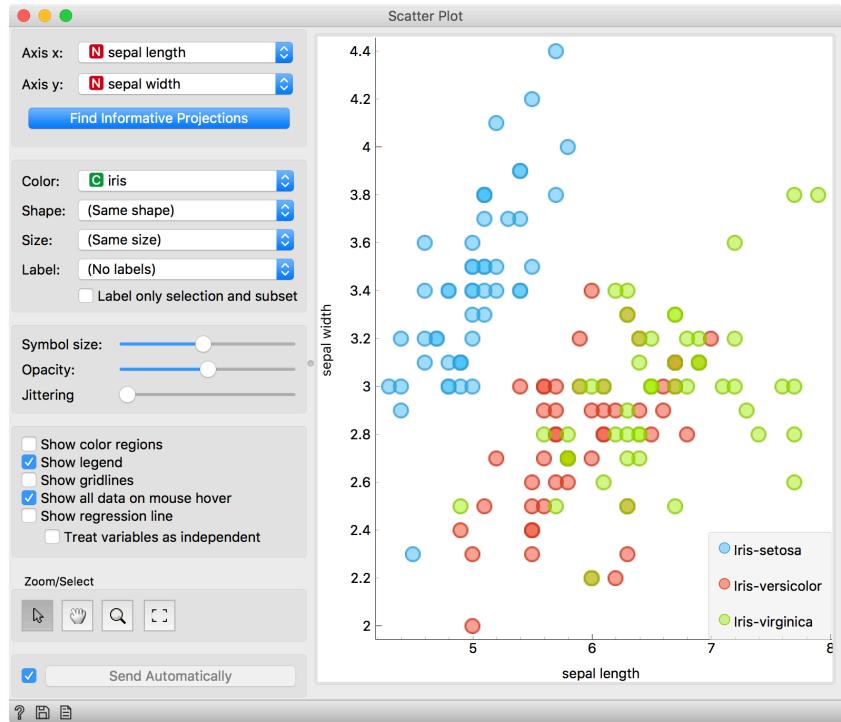
The File widget reads the data from your local disk. Open the File widget by double clicking its icon. You will see a pre-loaded data set named *iris*. This is a famous data set describing the subtypes of iris flowers.

Orange workflows often start with a File widget. The *iris* data set has 150 rows (flowers) and 5 columns. Out of the 5 columns, 4 contain measures of sepals and petals and 1 (marked as "categorical class") reports on the type of iris (*setosa*, *virginica* or *versicolor*).



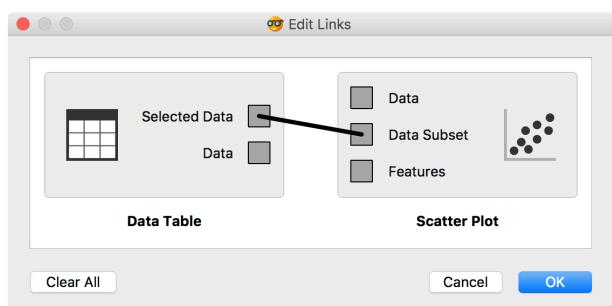
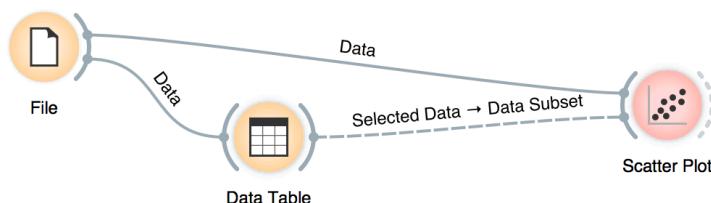
Now open the other widgets. In the Scatter Plot widget, select a few data points and watch as they appear in Data Table (1). Use a combination of two Scatter Plot widgets, where the second scatter plot shows a detail from a smaller region selected in the first scatter plot.

Scatter Plot can be used only with numerical features. We see the relationship between sepal length and sepal width and color the points by the type of iris. What does this visualization tell us?



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

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



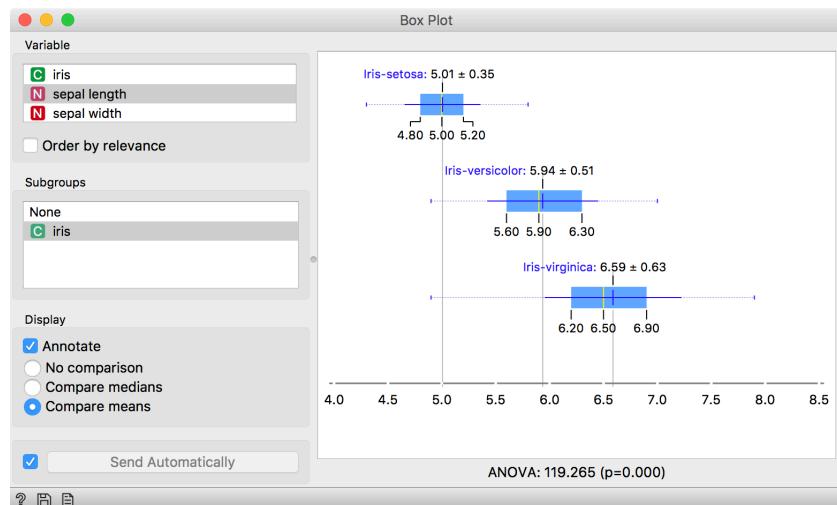
How does Orange distinguish between the primary data source and the data selection? It uses the first connected signal as the entire data set and the second one as its subset. To make changes or to check what is happening under the hood, double click on the line connecting the two widgets.

## Lesson 2: Data Exploration

A nice way to get the first impression of our data is to visualize it. Box Plot can help us discover outliers, distributions and interesting features.



Looks like the average sepal length is 5.84, but our standard deviation (thick blue line) is quite high, so we can expect a lot of variance in our samples.



**Tick 'Order by relevance' box under Variable list to order variable by how well they split subgroup values. Here, the mean of iris setosa is quite separate, while the means of iris versicolor and iris virginica still overlap somewhat.**

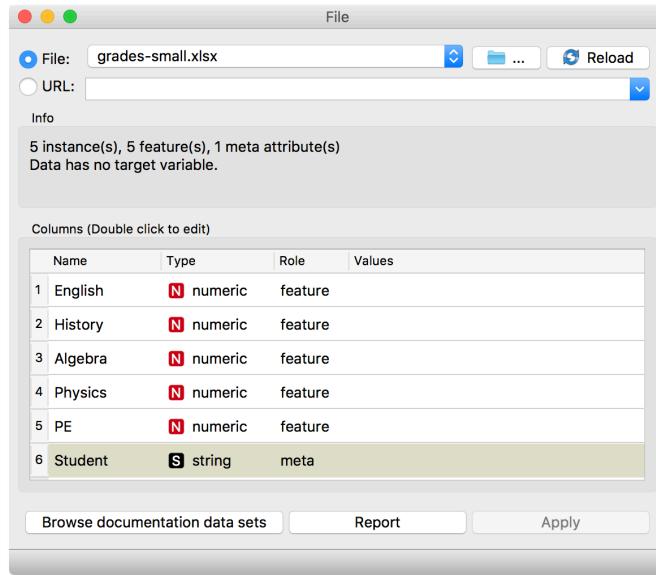
But Box Plot is even more powerful than that. We can compare distribution by subgroups. Select Iris in Subgroups box and see how irises differ by sepal length. Seems like iris setosas have the smallest sepal length and virginicas the highest.

## Lesson 3: Loading Your Own Data Set

Orange can read the data from spreadsheet file formats which include tab and comma separated and Excel files. Let us prepare a data set (with school subjects and grades) in Excel and save it on a local disk.

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

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



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

It is always good to check if all the data was read correctly. 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' visualization window. On the left, there's an 'Info' panel with details: 5 instances, 6 features (no missing values), no target variable, and 1 meta attribute (no missing values). Below it are sections for 'Variables' (checkboxes for 'Show variable labels (if present)', 'Visualize continuous values', and '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 titled 'Data Table' with columns: Student, English, History, Algebra, Physics, Physical, and GPA. The data rows are:

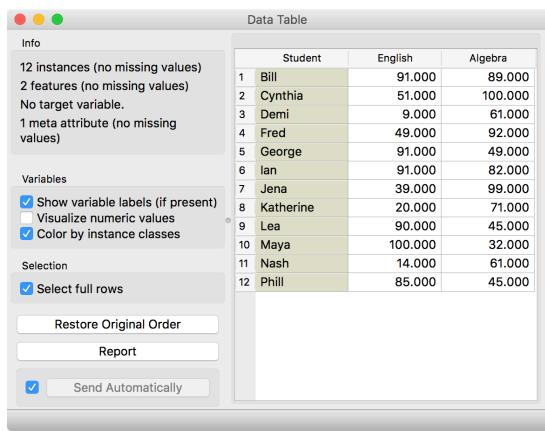
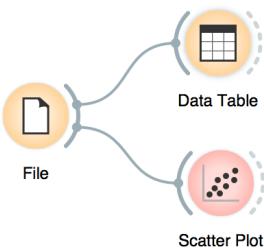
	Student	English	History	Algebra	Physics	Physical	GPA
1	George	22.000	32.000	21.000	46.000	99.000	3.000
2	John	91.000	65.000	89.000	11.000	29.000	3.000
3	Thomas	51.000	21.000	100.000	100.000	27.000	3.000
4	James	9.000	18.000	61.000	90.000	8.000	2.000
5	John	93.000	39.000	12.000	17.000	63.000	1.000

Nice, everything is here.

Instead of using Excel, we could also use Google Sheets, a free online spreadsheet alternative. Then, instead of finding the file on the local disk, we would enter its URL address in 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 actually a web address of an image, and more. But enough for now. If you would really like to dive in for more, check out the [documentation page on Loading your Data](#), or a [video](#) on this subject.

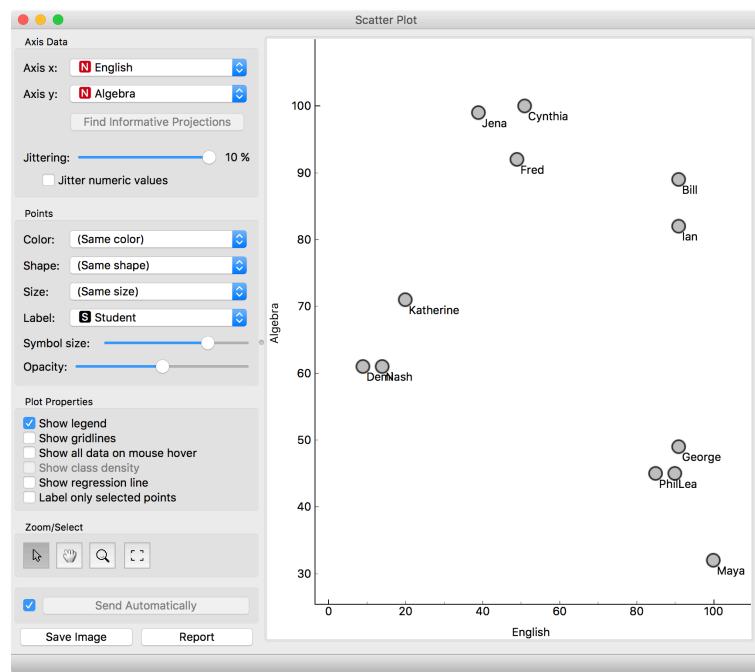
We will introduce clustering with a simple data set on students and their grades in English and Algebra. Load the data set from <https://file.biobab.si/text/grades.tab>.



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

## Lesson 4: Hierarchical Clustering

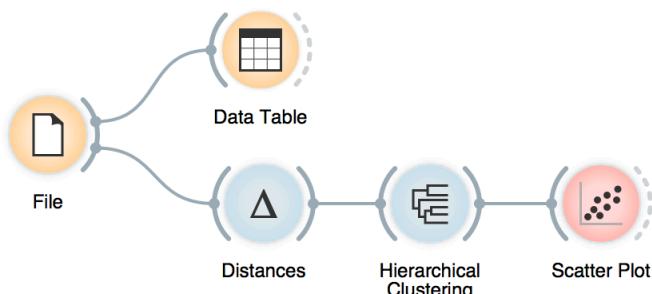
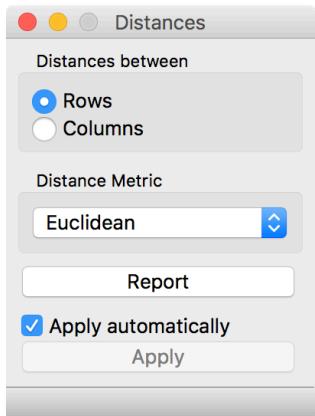
Say that we are interested in finding clusters in our data. That is, we would like to identify groups of data instances that are close together, similar to each other. Consider a simple, two-featured data set (see the side note) and plot it in the Scatter Plot. How many clusters do we have? What defines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



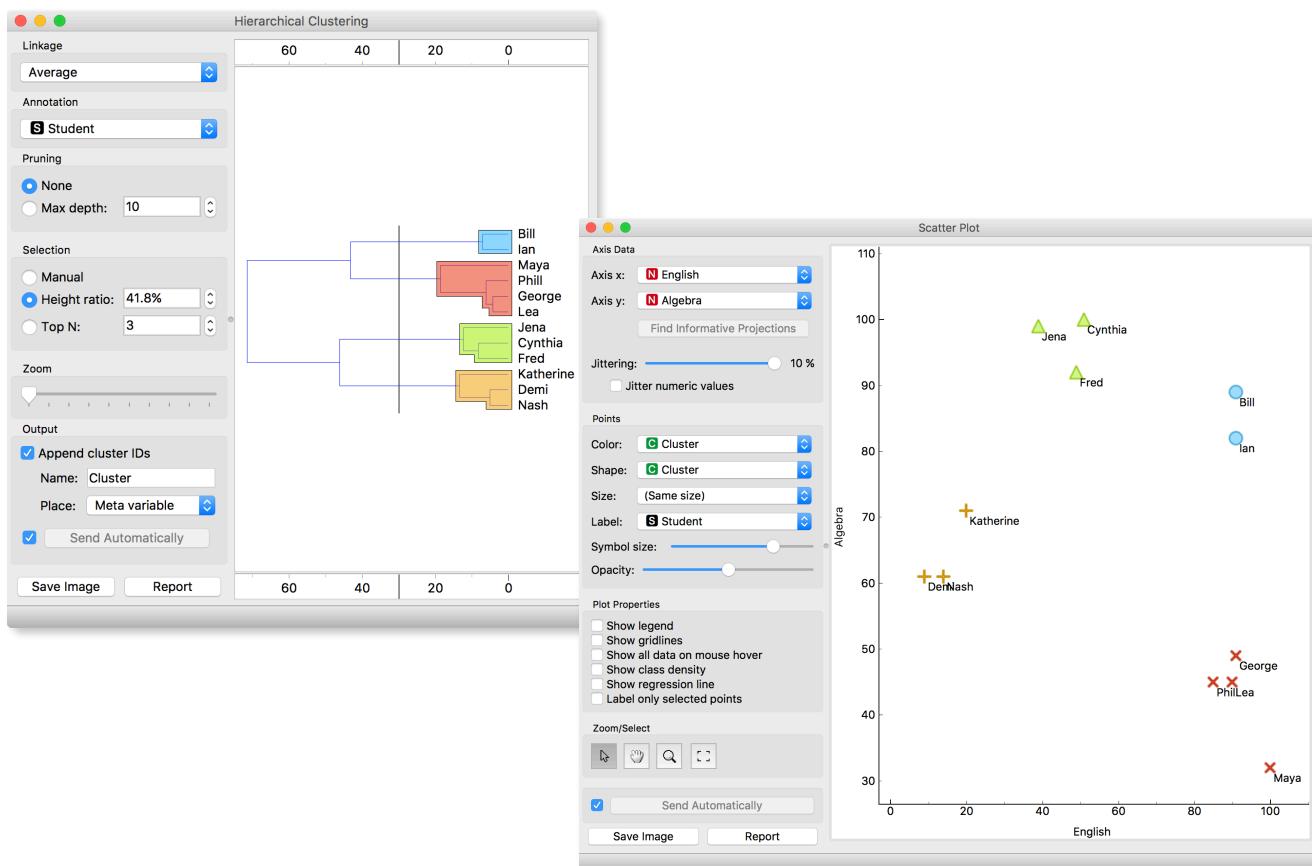
First, we need to define what we mean by “similar”. We will assume that all our data instances are described (profiled) with continuous features. One simple measure of similarity is the Euclidean distance. So, we would like to group data instances with small Euclidean distances.

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

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



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

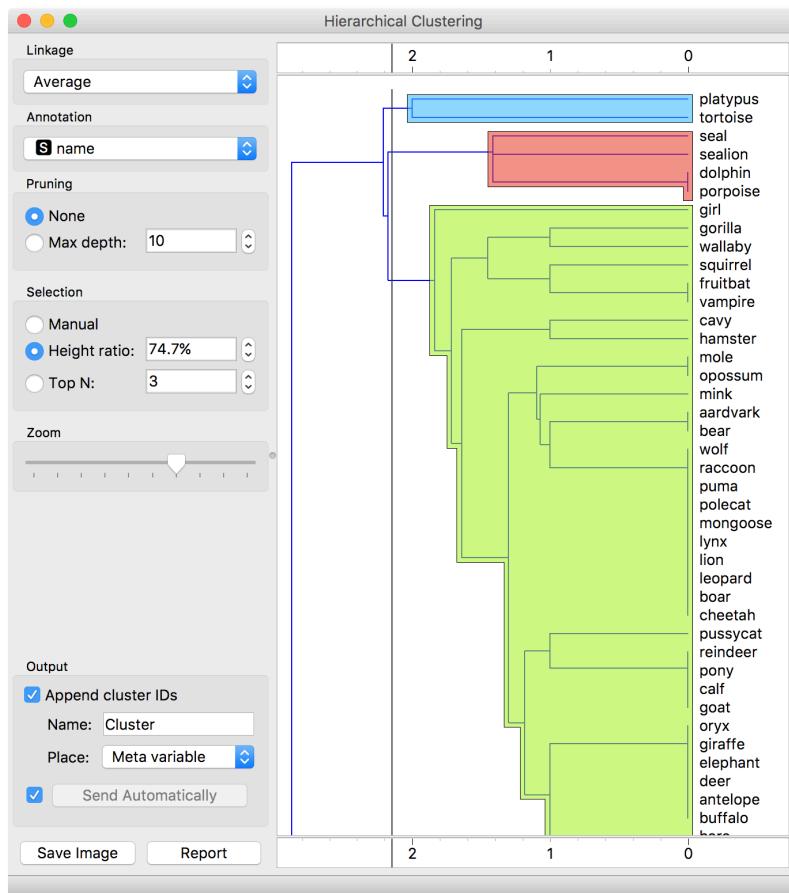
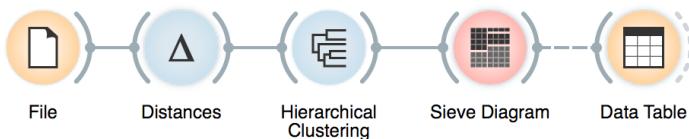




Hierarchical clustering works fast for smaller data sets. But for bigger ones it fails. Simply, it cannot be used. Why?

## Lesson 5: Animal Kingdom

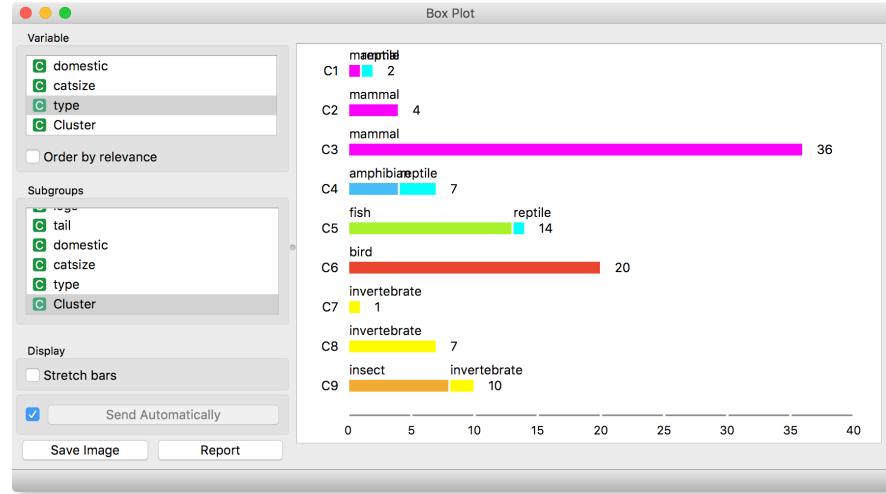
Your lecturers spent substantial part of their youth admiring a particular Croatian chocolate called Animal Kingdom. Each chocolate bar came with a card — a drawing of some (random) animal, and the associated album made us eat a lot of chocolate. With the next generation, the story repeated. Some things stay forever. Funny thing is, 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.



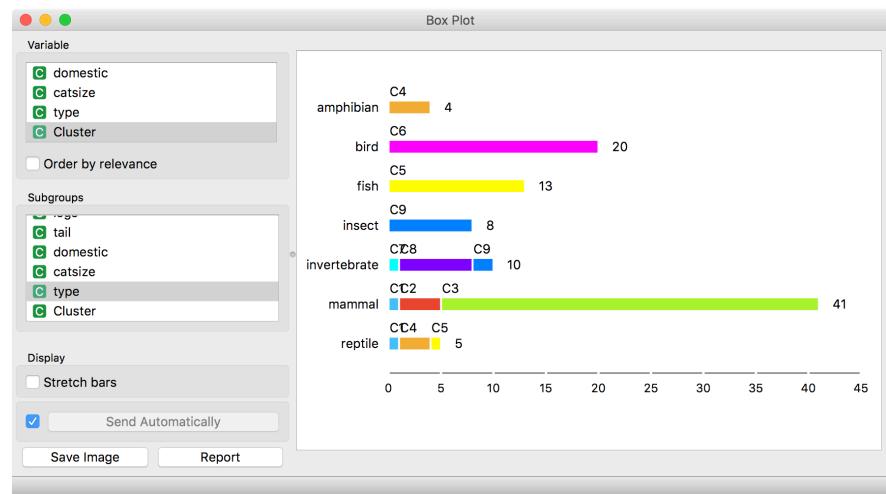
Here we use *zoo* data (from documentation data sets) with attributes that report on various features of animals (has hair, has feathers, lays eggs). We measure the distance and compute the clustering. Animals in this data set are annotated with type (mammal, insect, bird, and so on). It would be cool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

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

We can observe the discovered clusters in a Box Plot. We can get a distribution of animal types in each cluster:



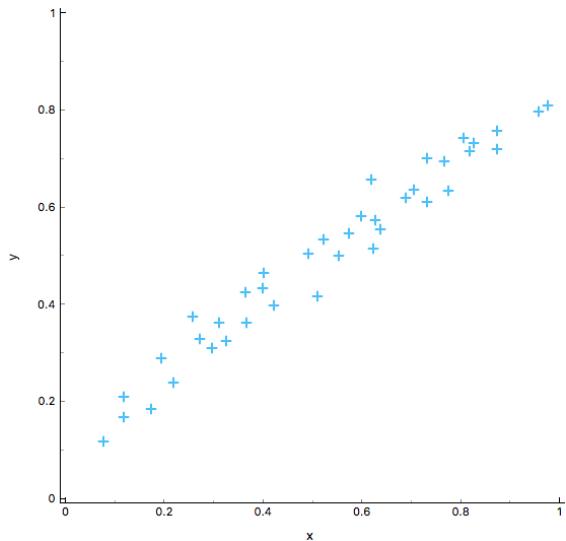
Or we can turn it around and see how different types of animals are spread across clusters.



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 6: PCA

Let us observe a simple 2D data set. We already know scatter plot shows relationship between two numerical variables, x and y in our case.



Do you notice anything unusual here? It looks like x and y are highly correlated. Could we perhaps simplify the data set by describing the data with a single variable? Perhaps like this?



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 squint our eyes, we see just a line. We lose one dimension, essentially keeping just a single coordinate for each dot.

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. In particular we want to remove those that are correlated and tell us the same thing. And we want our final data points to be as spread out as possible.

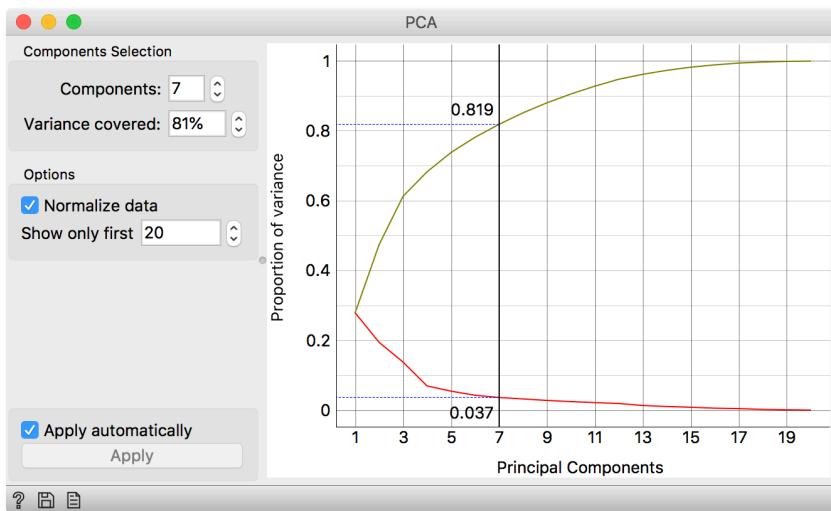
How do we do this? Imagine you are observing a swarm of flies; your data are their exact coordinates in the room, so the position of each fly is described by three numbers. Then you discover that your flies actually fly in a formation: they are (almost) on the same line. You could then describe the position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

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

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

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

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.



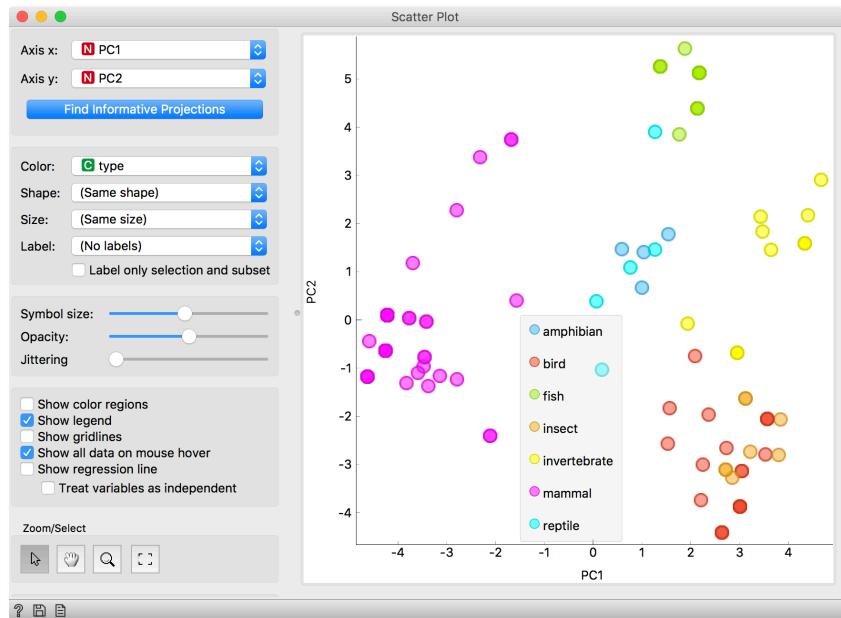
The widget allows you to select the number of components and helps you by showing how much information (technically: explained variance) you retain with respect to the number of components (brownish line) and the amount of information (explained variance, red line) in each component.

The PCA on the left shows the scree diagram for zoo data. Set like this, the widget replaces the 16 features with just seven - and still keeping 81% of information.

**PCA is particularly useful in visualizations and for speeding up the analysis.**

The axes, PC<sub>1</sub> and PC<sub>2</sub>, do not correspond to particular features in the original data, but to their linear combination. What we are looking at is a projection onto the plane, defined by the first two components. When you consider only two components, you can imagine that PCA puts a hyperplane into multidimensional space and projects all data onto 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.



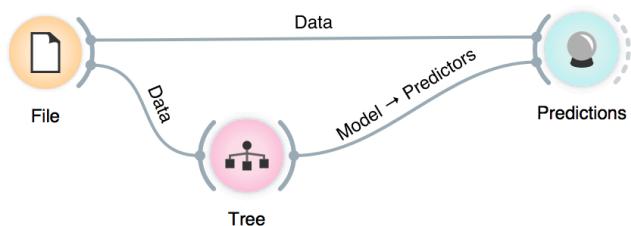
Not bad, really.

# Lesson 7: Classification

In the previous lessons, we explored iris data, but have never constructed actual prediction models. For this lesson, load the *churn-filtered.tab* data set from this [link](#).

Retaining customers is the bread and butter of most companies so they like to keep as much of them as possible. Can we predict which customers will leave before they actually do?

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



Let us create this simple workflow.

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

Tree	churn	state	area code	phone number	account length
1 0.96 : 0.04 → False	False	KS	415.000	382-4657	128.000
2 0.96 : 0.04 → False	False	OH	415.000	371-7191	107.000
3 0.96 : 0.04 → False	False	NJ	415.000	358-1921	137.000
4 0.96 : 0.04 → False	False	OK	415.000	330-6626	75.000
5 0.96 : 0.04 → False	False	AL	510.000	391-8027	118.000
6 0.96 : 0.04 → False	False	MA	510.000	355-9993	121.000
7 0.96 : 0.04 → False	False	MO	415.000	329-9001	147.000
8 0.96 : 0.04 → False	False	LA	408.000	335-4719	117.000
9 0.03 : 0.97 → True	True	IN	415.000	329-6603	65.000
10 0.96 : 0.04 → False	False	RI	415.000	344-9403	74.000
11 0.96 : 0.04 → False	False	IA	408.000	363-1107	168.000
12 0.96 : 0.04 → False	False	MT	510.000	394-8006	95.000
13 0.03 : 0.97 → True	True	IA	415.000	366-9238	62.000
14 0.01 : 0.99 → True	True	NY	415.000	351-7269	161.000
15 0.96 : 0.04 → False	False	ID	408.000	350-8884	85.000
16 0.96 : 0.04 → False	False	VT	510.000	386-2923	93.000
17 0.96 : 0.04 → False	False	VA	510.000	356-2992	76.000
18 0.96 : 0.04 → False	False	TX	415.000	373-2782	73.000
19 0.96 : 0.04 → False	False	FL	415.000	396-5800	147.000
20 0.03 : 0.97 → True	True	CO	408.000	393-7984	77.000
21 0.96 : 0.04 → False	False	AZ	415.000	358-1958	130.000
22 0.96 : 0.04 → False	False	SC	415.000	350-2565	111.000

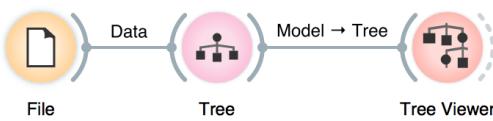
The Predictions widget also receives the data from the File widget. The widget uses the model to make predictions about the data, and shows them in the table.

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

But (and even before answering these very important questions), what is a classification tree? And how does Orange create one? Is this algorithm something we should really use?

So many questions to answer!

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



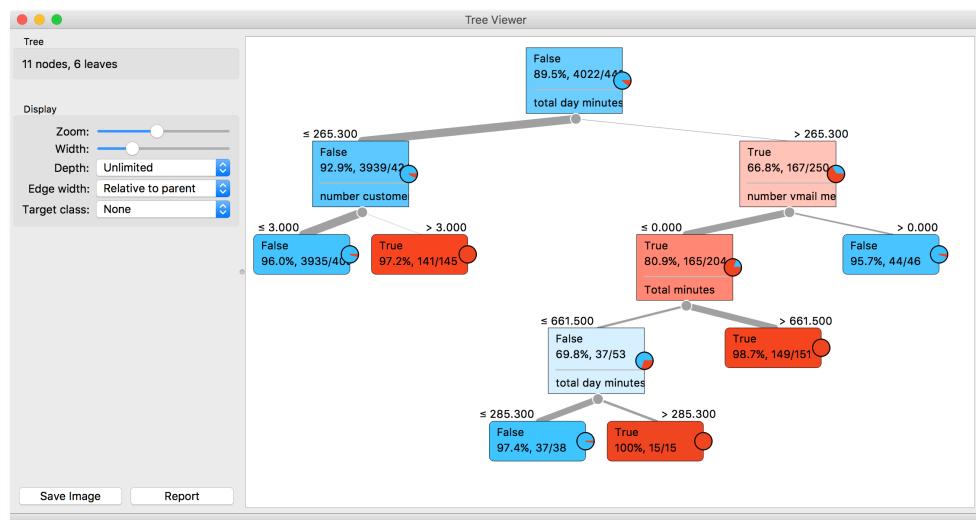
Some relations inferred from the classification tree may not be too reliable for small data sets. What should the size of the data set be to acquire stronger conclusions?

## Lesson 8: Classification Trees

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

Let us show how classification tree is built for predicting churn. We will use Tree Viewer to visualize the tree.

We read the tree from top to bottom. The majority, 89.5% of customers in this data set will stay, hence the label *False* in the root of the tree. The best feature that would split this data set into two data sets, where one of the classes would prevail, is *total day minutes*. See the node in the right branch? The majority (66.8%) of customers will leave the company. The data subset in the left branch is cleaner than its root, with 92.9% customers staying. The tree uses the *number customer service calls* feature to break this subset into two data subsets that are even cleaner (red and blue nodes at the next level). Building the tree is therefore a procedure which uses a few most informative features to hierarchically break the data into smaller subsets where one of the classes is prevailing.



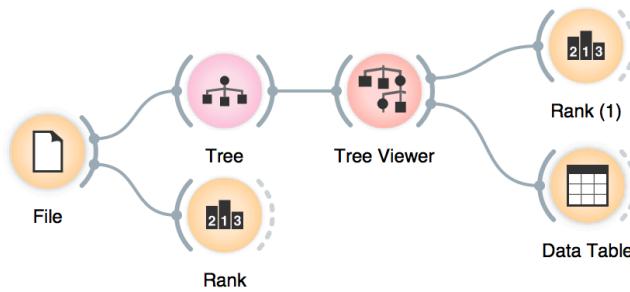
Trees place the most useful feature at the root. What would be the most useful feature? The feature that splits the data into two purest possible subsets. It then splits both subsets further, again by

their most useful features, and keeps doing so until it reaches a subset in which all data belongs to the same class (leaf nodes in strong blue or red) or until it runs out of data instances to split or out of useful features.

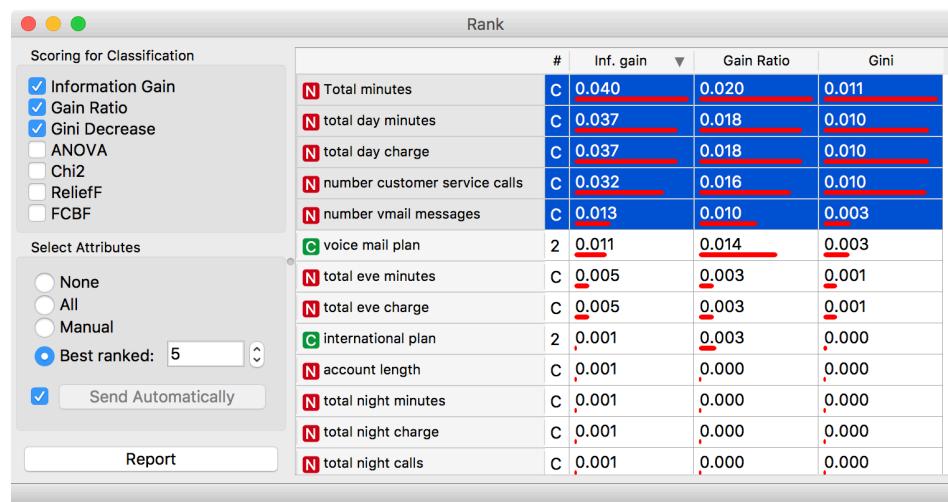
We still have not been very explicit about what we mean by “the most useful” feature. There are many ways to measure the quality of features, based on how well they distinguish between classes. We will illustrate the general idea with information gain. We can compute this measure in Orange using the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

**Rank widget could be used on its own. Say, to figure out which features are best predictors of a customer who will leave. In the future, we could spot similar customers and provide them special offers or discounts to persuade them to stay.**

In this course, we won't learn how to compute information gain. There's a good explanation of this concept with formulas and graphs on [stackoverflow.com](https://stackoverflow.com) (google it).



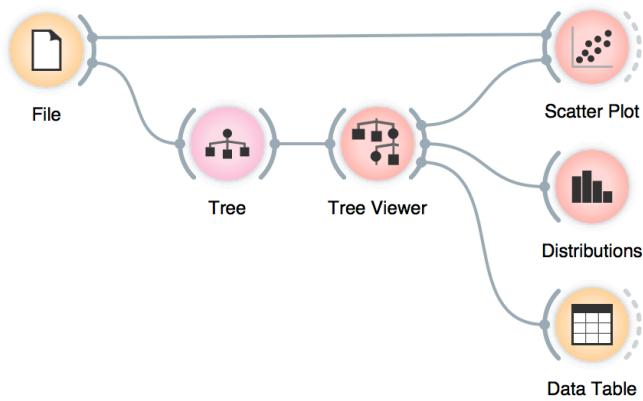
Besides the information gain, Rank displays several other measures (including Gain Ratio and Gini), which are often quite in agreement and were invented to better handle categorical features with many different values.



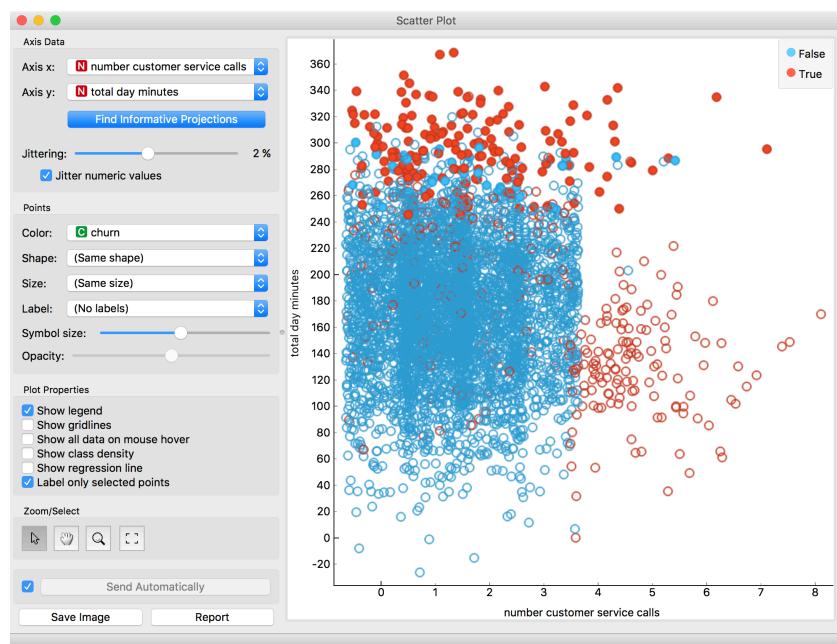
## Lesson 9: Model Inspection

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

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, an approach favored by the data visualization guru Edward Tufte.



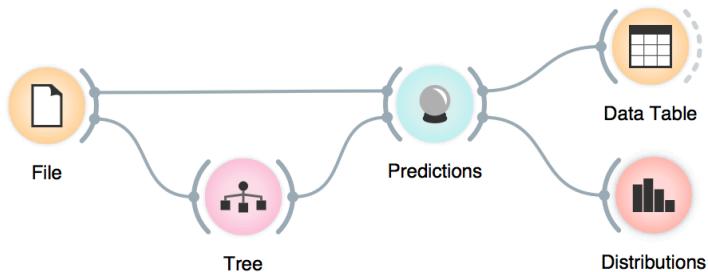
Just for fun, we have included a few other widgets in this workflow. The Tree Viewer selects data instances by inferring rules from the data itself and optimizing to obtain purer data subsets.



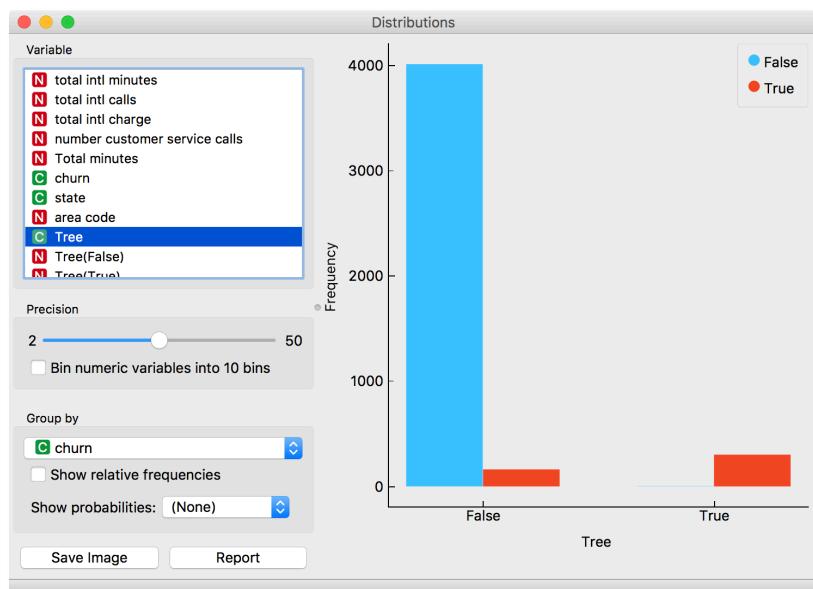
# Lesson 10: Classification Accuracy

Now that we know what classification trees are, the next question is what is the quality of their predictions. First, 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 of classification accuracy with the widgets we already know.

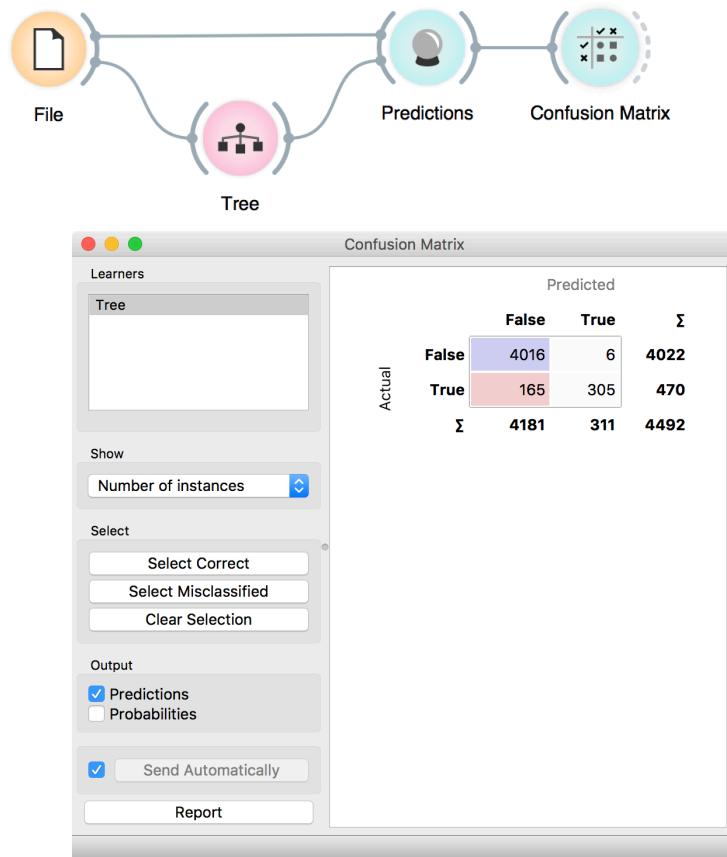
We do not yet know Distributions widget, but it is similar to Box Plot. It shows distribution of a selected variable and it also enables grouping by categorical variables. You can substitute Distributions for Box Plot, if you wish.



Predictions outputs a data table augmented with a column that includes predictions. In Data Table, 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.



But sometimes we need something more exact than a rough estimate observed in the Distributions widget. To see the statistics of correctly and incorrectly classified examples open the Confusion Matrix widget.

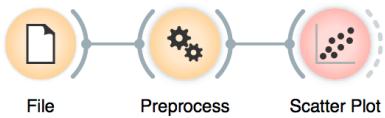


We see that for 4016 customers who stayed and 305 who left, the tree predicted the outcome correctly. However, it misclassified 165 persons that left the company as not at risk and marked 6 persons that stayed as potential resisters. Since classification accuracy is the proportion of correctly classified examples, we calculate it by dividing the number of correct classifications ( $4016 + 305 = 4321$ ) with the number of all examples (4492). Hence classification accuracy is  $4321/4492 = 96.1\%$ . Sounds good, but is it really?

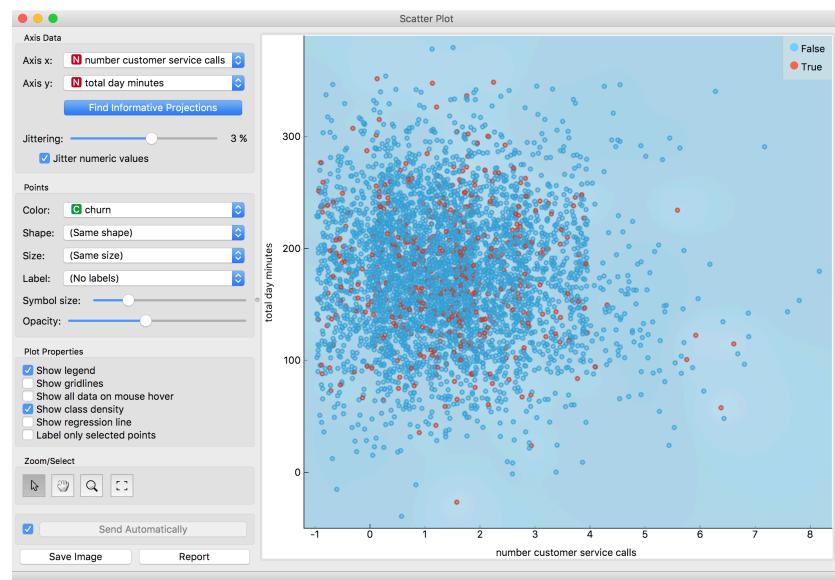
## Lesson II: How to Cheat

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

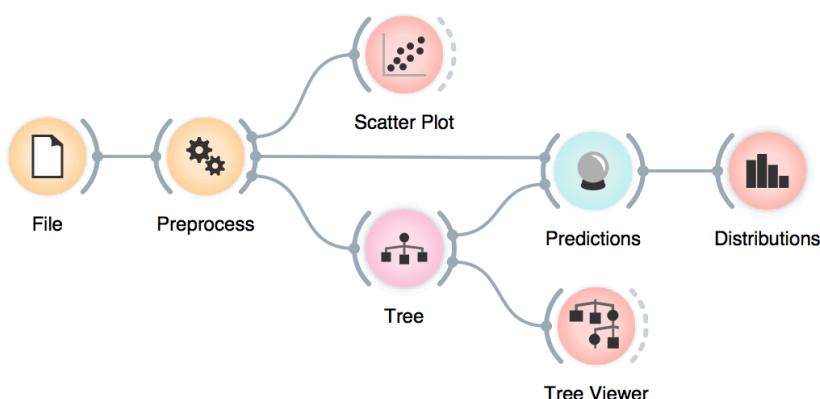
At this stage, the classification tree looks very good. There are only a couple of data points where it makes a mistake. Can we mess up the data set so bad that the trees will ultimately fail? Like, remove any existing correlation between employee characteristics and churn? We can! There's the Preprocessing widget with randomize class preprocessor. Check out the chaos it creates in the Scatter Plot visualization where there were nice clusters before randomization!



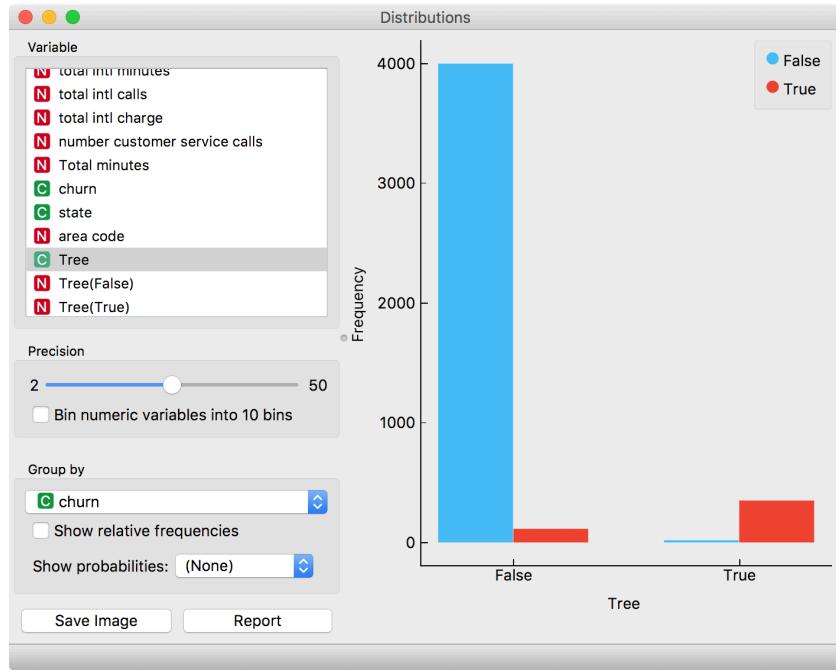
Why is the background in this scatter plot so blue? Why has red disappeared after class randomization?



Fine. There can be no classifier that can model this mess, right? Let's make sure. (When connecting Preprocess to Tree, Orange will connect the Preprocessor signals. You will have to manually correct this by connecting the output Preprocessed Data to Data. Connections in this dialog are removed by clicking on them.)



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



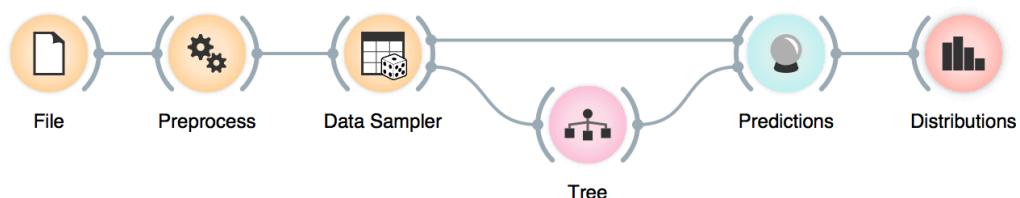
Most unusual. Almost no mistakes. How is this possible? On a class-randomized data set?

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

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

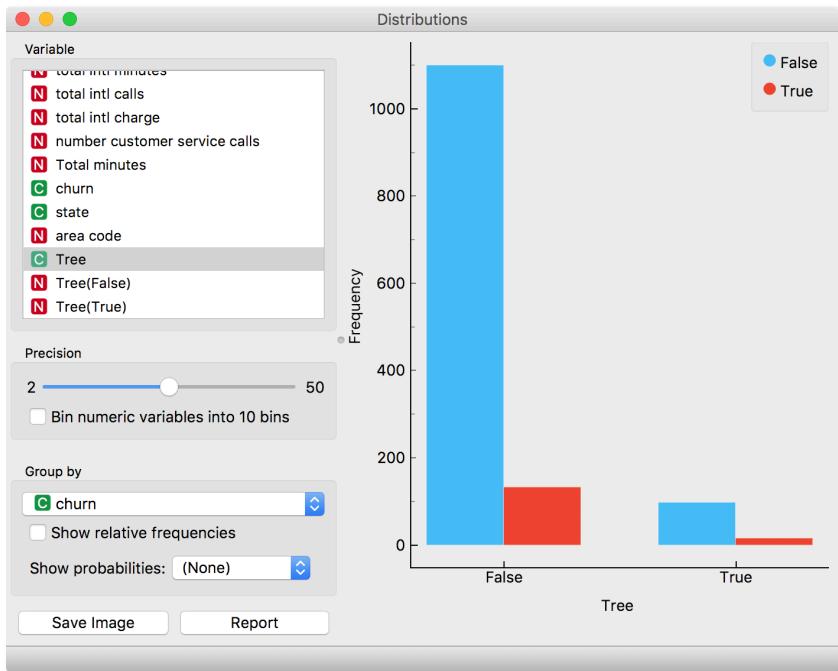
Ha, if this is so, if a classifier remembers everything from a data set without discovering any general patterns, it should perform miserably on any new data set. Let us check this out. We will split our data set into two sets, training and testing, train the classification tree on the training data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

**Signals from the Data Sampler** widget have not been named in our workflow to save space.  
Data Sampler split 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 sent to the Predictions widget. Set the Data Sampler so that the size of these two data sets is about equal.



Turns out that the majority of data instances have been predicted to False (will not leave). Why? Blue again (like blue 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 predictions of the users that are going to leave are a complete fail. On the class-randomized training data our classifier fails miserably. Finally, just as we would expect.

To really 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 actually learn a general model.

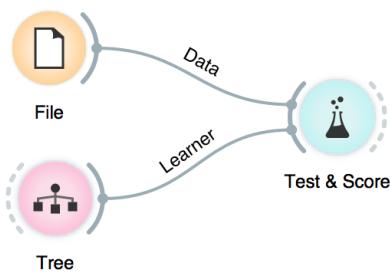
Learning is not simply memorizing. Rather, it is discovering patterns that govern the data and applying it to new data as well. To estimate the accuracy of a classifier, we therefore need a separate test set. This estimate should not depend on just one division of the input data set to training and test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

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

## Lesson 12: Cross-Validation

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

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

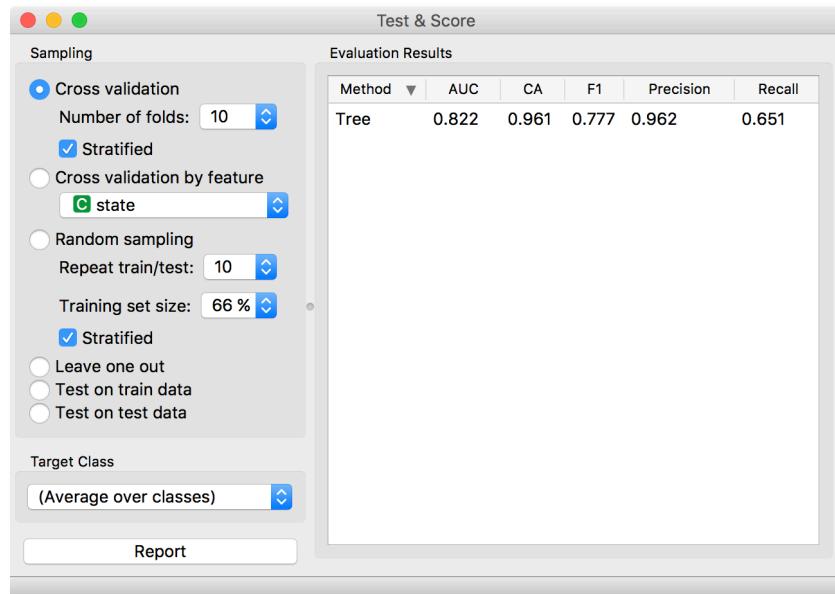


**For geeks:** a learner is an object that, given the data, outputs a model. Just what Test & Score needs.

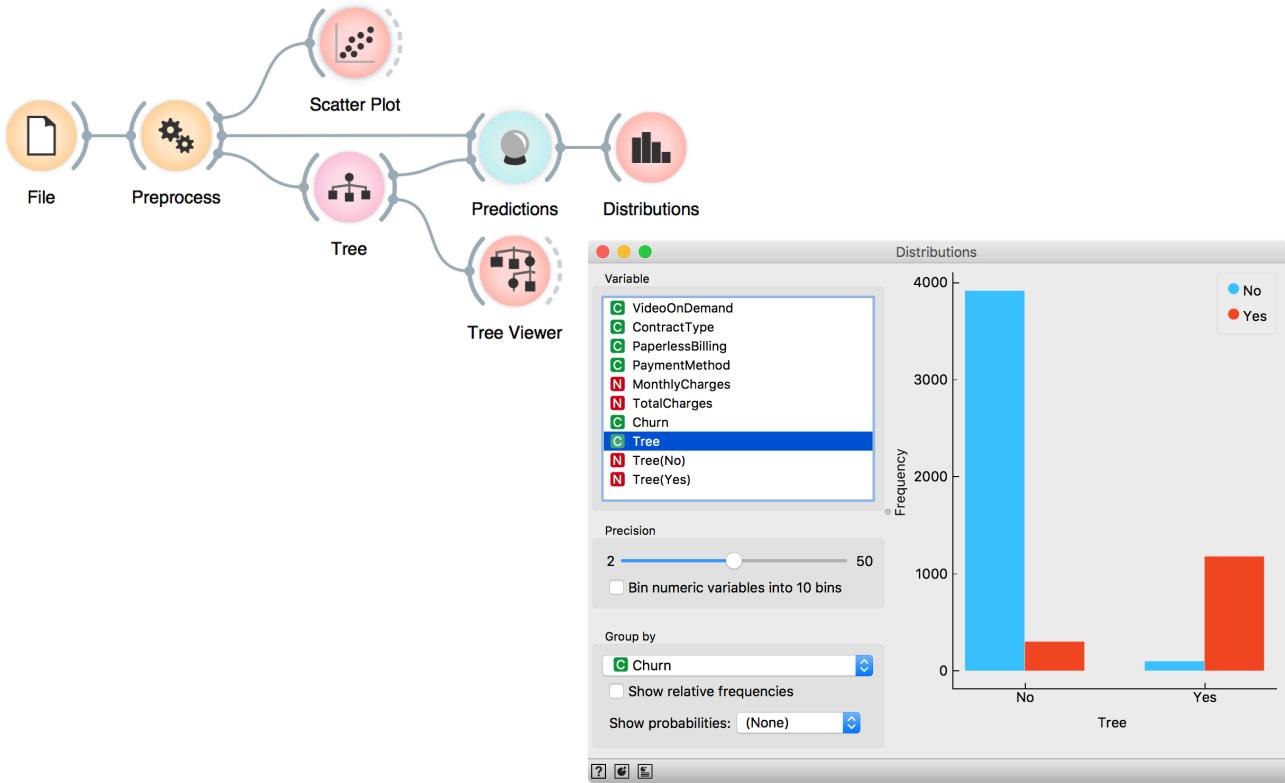
Cross validation splits the data sets into, say, 10 different non-overlapping subsets we call folds. In each iteration, one fold will be used for testing, while the data from all other folds will be used for training. In this way, each data instance will be used for testing exactly once.

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

Here is how the Test & Score widget looks like. CA stands for classification accuracy, and this is what we really care for now.



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

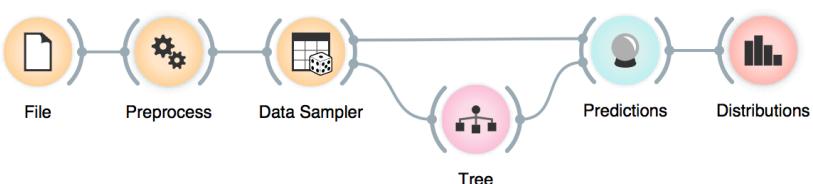


Most unusual. Almost no mistakes. How is this possible? On a class-randomized data set?

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

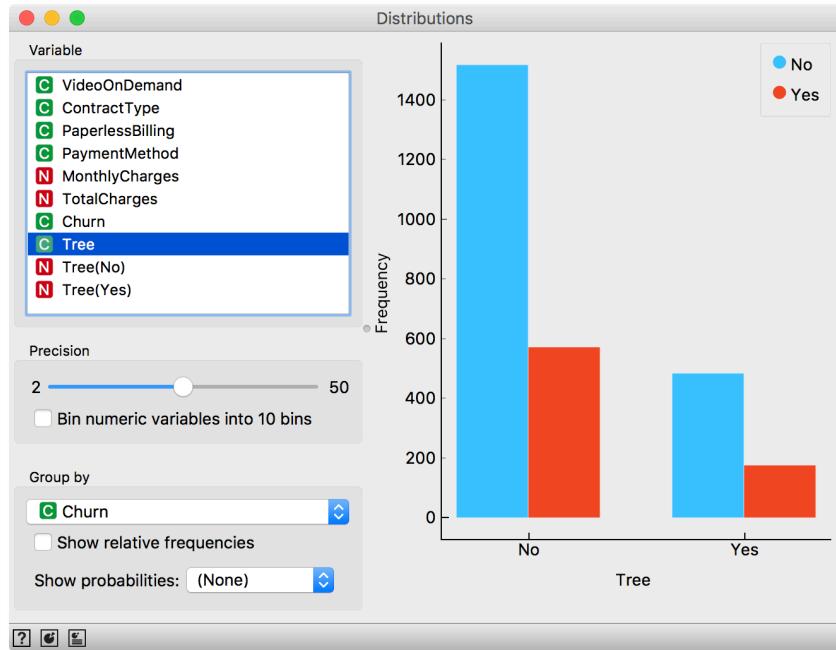
Looks like the tree just memorized every data instance from the data set. No wonder the predictions were right. The tree makes no sense, and it is complex because it simply remembered everything.

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



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

Turns out that for both class values the majority of data instances have been predicted to No (blue). Why? Here is a hint: use the Box Plot widget to answer this question.



Predictions are a complete fail. On the class-randomized training data our classifier performs miserably. Finally, just as we would expect.

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

To really 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 actually learn a general model.

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

## Lesson 13: Case Studies

Q1: Which model performs the best?

Q2: What are the three most important variables for this model?

Q3: Use Attrition - Predict and identify who is most likely to leave and come up with measures to prevent this.

Q1: Use Rank to identify three most important attributes.

Q2: How can we explain these attributes? Use Box Plot to describe the leaving customers and come up with a strategy to prevent this.

Q1: Can success of the projects submitted in April and May be predicted from models inferred from data on previous months?

Q2: What contributes the most to the success of the campaign?

Q3: Which models performs the best?

Q4: Select those instances the model said will be funded (positives). Now inspect how are true positives different from false positives (use Box Plot).

Data scientists are made with lots and lots of practice. Here we have some typical business use cases where data mining and machine learning are typically employed. All data sets are available in the Datasets widget.

### 1. Attrition of employees

It is employers' worst nightmare when talented employees leave the company. In order to prevent this, we need to offer them some incentives to stay. But we cannot afford to give raises to everyone in the company. First, we must identify those employees who are most likely to leave.

Use *Attrition - Train* and *Attrition - Predict* data sets.

### 2. Customer churn

Key to a successful business is not only bringing in new customers, but also retaining the old ones. It is vital to identify customers who are most likely to switch to competition and provide incentives for them to stay with us.

Use *Telecom customer churn* data set.

### 3. Kickstarter

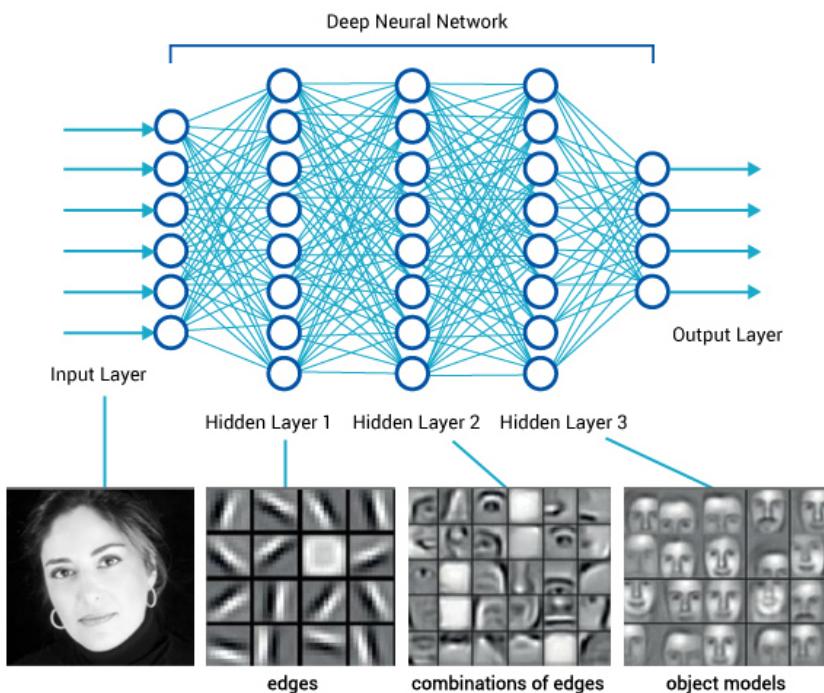
When launching a new product, there's always a high level of uncertainty, whether the product will succeed or fail. Could we mitigate at least a part of this uncertainty? We can pin point what contributes to a successful campaign (which campaign gets funded) and in this way increase the chances of our product to go from zero to hero.

Use *Kickstarter projects* data set.

## Lesson 14: Image Embedding

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

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



Until very recently, finding useful representation of complex objects such as images was a real pain. Now, technology called deep learning is used to develop models that transform complex objects to vectors of numbers. Consider images. When we, humans, see an image, our neural networks go from pixels, to spots, to patches, and to some higher order representations like squares, triangles, frames, all the way to representation of complex objects. Artificial neural networks used for deep learning emulate these through layers of computational units (essentially,

logistic regression models and some other stuff we will ignore here). If we put an image to an input of such a network and collect the outputs from the higher levels, we get vectors containing an abstraction of the image. This is called embedding.

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

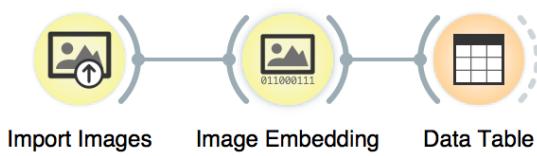


Image embedding describes the images with a set of 2048 features appended to the table with meta features of images.

**Data Table**

	image name	image vnlods/Images/d/ image	size	width	height	n0	n1	n2	n3	n4	n5	n6
1	duck	duck.png	39583	158	172	0.127	0.032	0.070	0.036	0.175	0.210	0.198
2	dog	dog.png	28745	129	125	0.041	0.218	0.191	0.149	0.161	0.584	0.647
3	horse	horse.png	69109	285	195	0.287	0.223	0.084	0.118	0.392	0.307	0.220
4	rabbit	rabbit.png	24294	97	174	0.271	0.024	0.171	0.014	0.403	0.181	0.419
5	ox	ox.png	56401	191	189	0.491	0.008	0.085	0.112	0.147	0.229	0.272
6	turkey	turkey.png	55072	171	182	0.351	0.051	0.263	0.063	0.137	0.532	0.260
7	sheep	sheep.png	58022	214	181	0.179	0.228	0.045	0.058	0.037	0.199	0.257
8	cow	cow.png	62159	210	189	0.482	0.129	0.053	0.090	0.132	0.599	0.242
9	calf	calf.png	45538	191	152	0.179	0.197	0.040	0.013	0.182	0.079	0.230
10	hen	hen.png	41716	134	168	0.387	0.058	0.042	0.097	0.388	0.203	0.133
11	foal	foal.png	39210	147	177	0.066	0.260	0.042	0.171	0.490	0.327	0.216
12	cat	cat.png	22193	105	137	0.044	0.173	0.724	0.000	0.158	0.119	0.300
13	goose	goose.png	34442	141	202	0.296	0.244	0.186	0.004	0.443	0.360	0.126
14	duckling	duckling.png	17109	99	119	0.058	0.048	0.055	0.047	0.196	0.184	0.154
15	rooster	rooster.png	41518	145	180	0.423	0.190	0.111	0.055	0.285	0.271	0.093

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.

**Distances**

Distances between

Rows  
 Columns

Distance Metric

Cosine

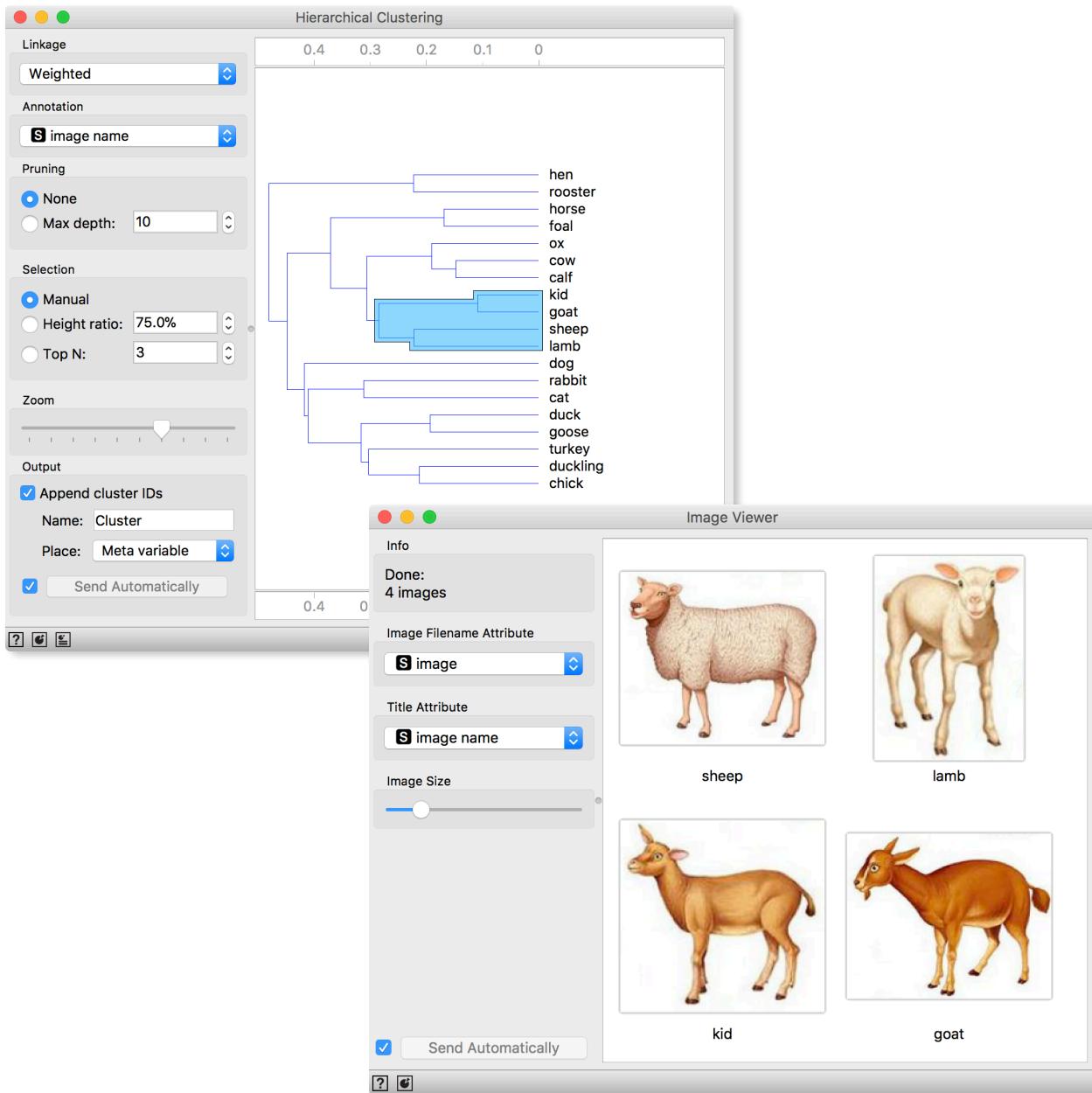
Normalized

Apply Automatically



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!



Now try to use image embedding and clustering for your own images. You may even try classification by placing the images in separate folders. For an example, check out <http://file.biolab.si/images/florida-houses.zip>.

## For the End

This Introduction to Data Mining workshop for students of marketing ends here. We have covered few important topics, including classification and clustering. Perhaps the most important topic under the surface of introduction is that of overfitting, something that all machine learning methods deal with and try to solve: how to learn the model that best generalizes across the data that it has not seen yet.

We have demonstrated how to use, in practice, few of the crucial algorithms 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 huge field and it takes years of study and practice to master it. You may never become a data scientist, but as an expert in economy and marketing it should now be easier to talk and collaborate with statisticians and computer scientists. And for those who want to go ahead with data science, well, now you know where to start.