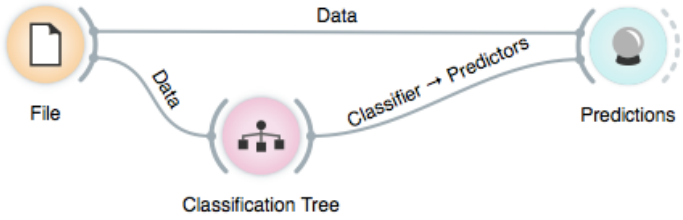
Lesson 5: Classification

In one of the previous lessons, we explored the heart disease data. We wanted to predict which persons have clogged arteries — but we actually did not make any predictions. We observed some potentially interesting relations between the features and the condition, but have never constructed an actual model.

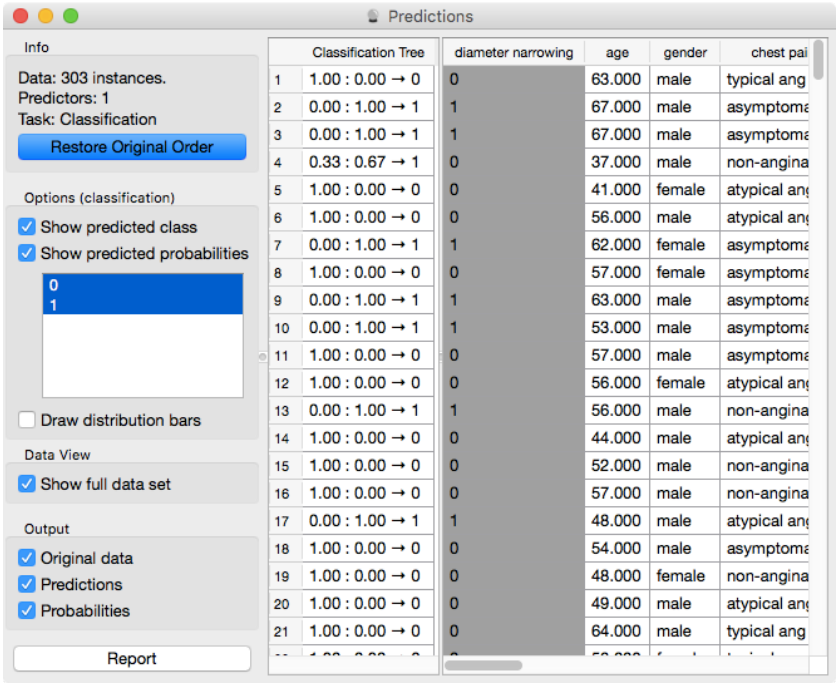
Let us create one now.

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



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

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



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

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

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

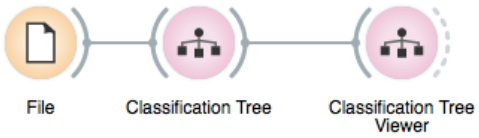
So many questions to answer today!

Lesson 6: Classification Trees

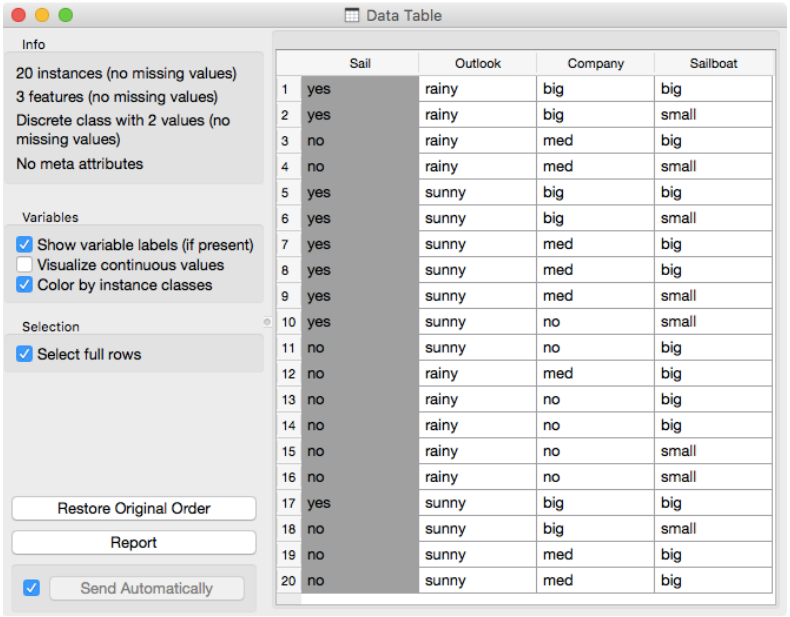
In the previous lesson, we used a classification tree, one of the oldest, but still popular, machine learning methods. We like it since the method is easy to explain and gives rise to random forests, one of the most accurate machine learning techniques (more on this later). So, what kind of model is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

Let us load a data set from http://bit.ly/2bvhKNr that records the conditions under which a friend skipper went sailing, build a tree and visualize it in the Classification Tree Viewer.

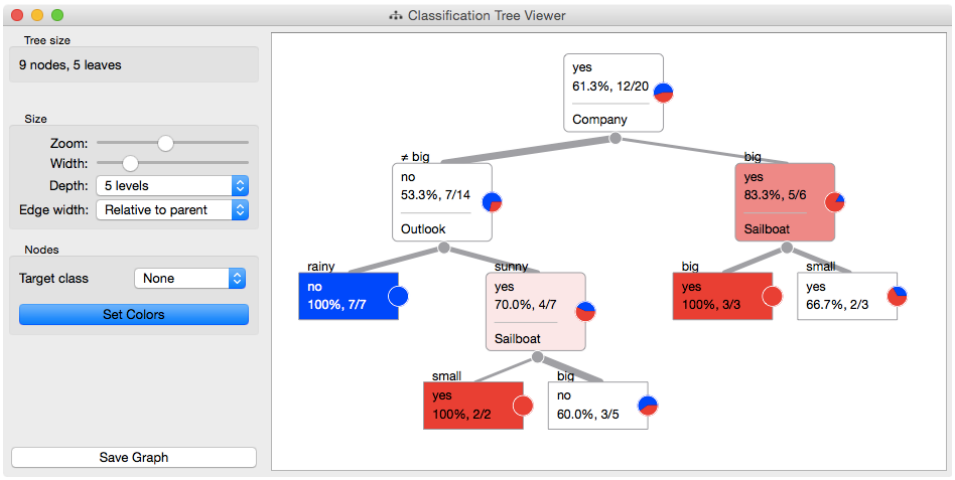
The data set we will use comes from Google Spreadsheet. Copy the web address and paste it into URL entry box in the File widget.



Here's a warning: this sailing data is very 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



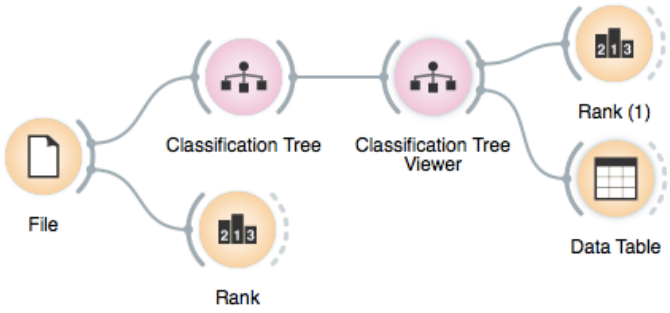
We read the tree from top to bottom. Looks like this skipper is a social person; as soon as there's company, the probability of her sailing increases. When joined by a smaller group of people, there is no sailing if there is rain. (Thunderstorms? Too dangerous?) When she has smaller company but the boat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

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

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.

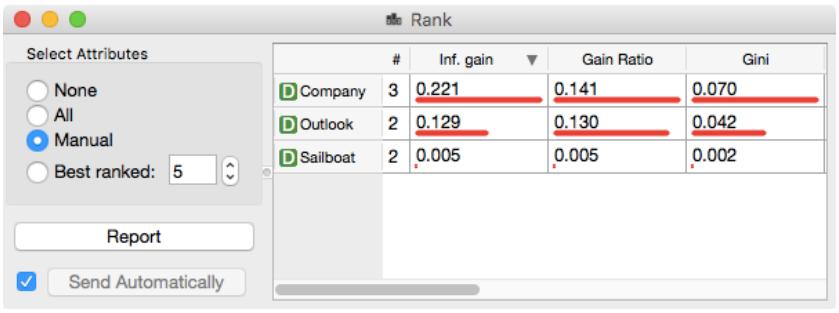
We still have not been very explicit about what we mean by "the most useful" feature. There are many ways to measure the quality of features, based on how well they distinguish between classes. We will illustrate the general idea with information gain. We can compute this measure in Orange using the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

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



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

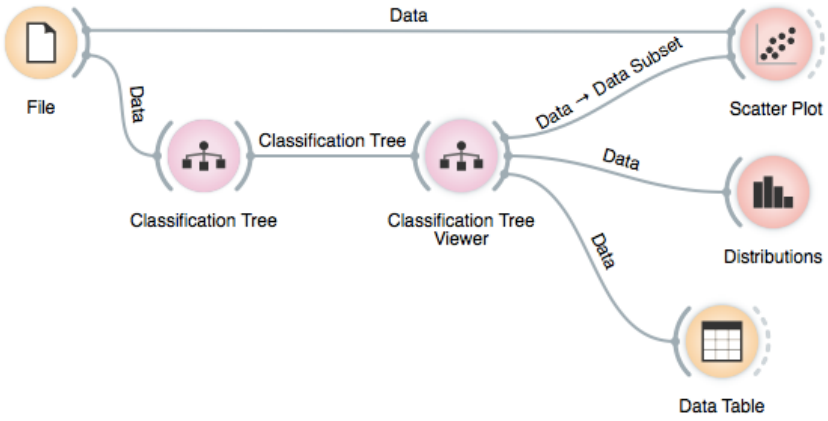
In this class, we will learn how to define and compute information gain. There's a good explanation of this concept with formulas and graphs on stackoverflow.com (google it).



Lesson 7: Model Inspection

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

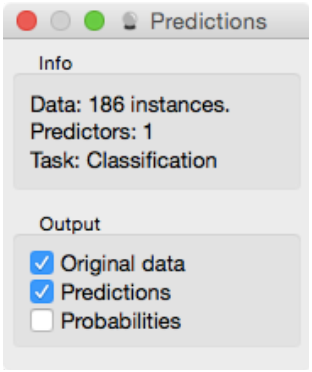
Wherever possible, visualizations in Orange are designed to support selection and passing of the data that applies to it. Finding interesting data subsets and analyzing their commonalities is a central part of explorative data analysis, a data analysis approach favored by the data visualization guru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

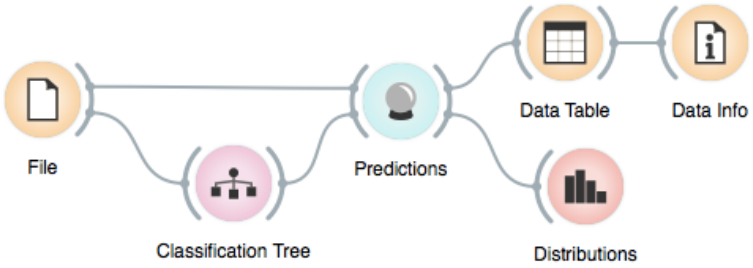


Just for fun, we have included a few other widgets in this workflow. In a way, the Classification Tree Viewer widget behaves like the Select Rows widget, except that the rules used to filter the data are inferred from the data itself and optimized to obtain purer data subsets.

Lesson 8: Classification Accuracy

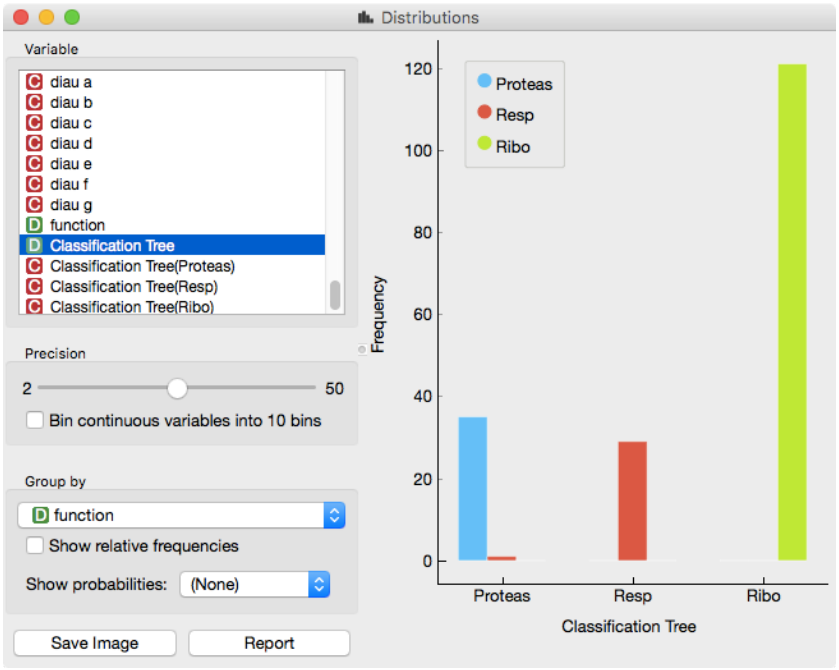
Now that we know what classification trees are, the next question is what is the quality of their predictions. For beginning, we need to define what we mean by quality. In classification, the simplest measure of quality is classification accuracy expressed as the proportion of data instances for which 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.





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.

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

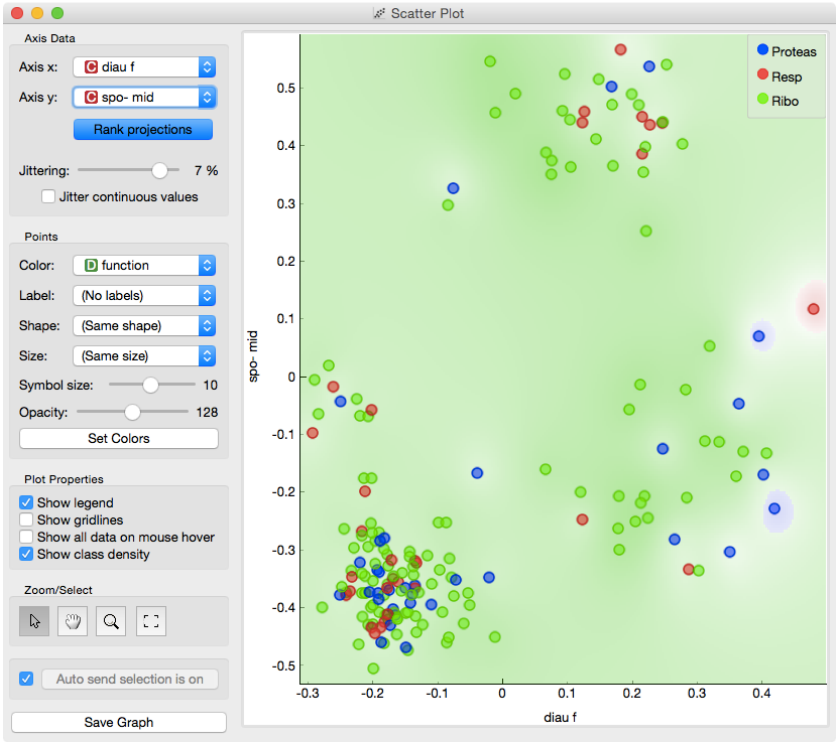


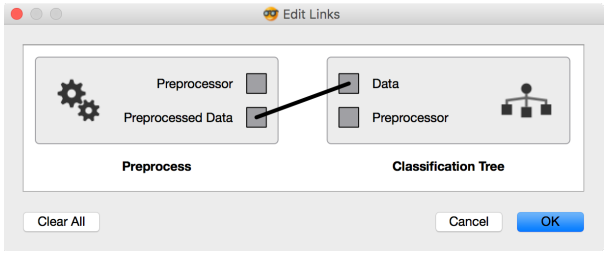
This lesson has a strange title and it is not obvious why it was chosen. Maybe you, the reader, should tell us what does this lesson have to do with cheating.

Lesson 9: How to Cheat

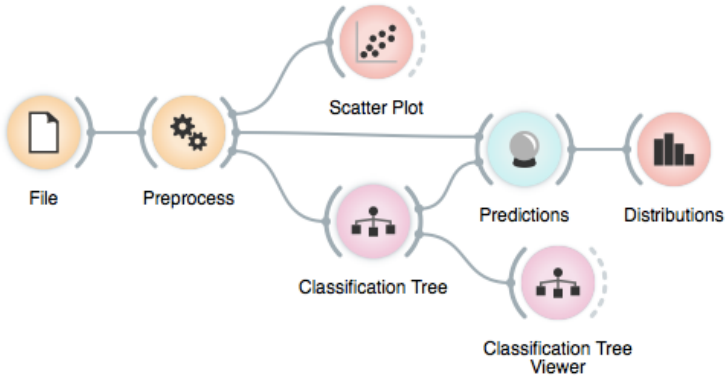
At this stage, the classification tree looks very good. There's only one data point where it makes a mistake. Can we mess up the data set so bad that the trees will ultimately fail? Like, remove any existing correlation between gene expression profiles and class? We can! There's the Preprocessing widget 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 green, and only green? Why have the other colors disappeared after the class randomization?

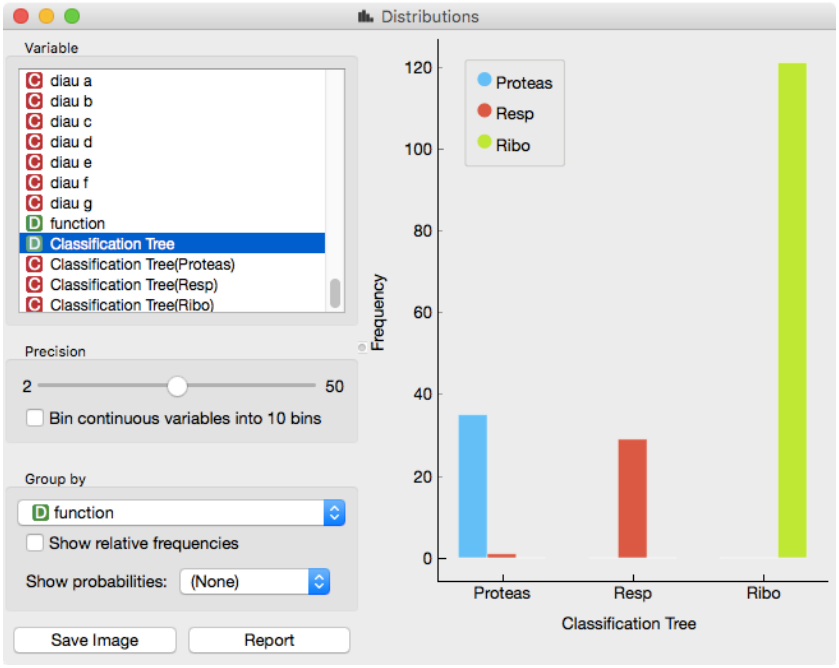




Fine. There can be no classifier that can model this mess, right?. Let's make sure. (When connecting Preprocess to Classification Tree, Orange will connect the Preprocessor signals. You will have to manually correct this by connecting output Preprocessed Data to Data. Connections in this dialog are removed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



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

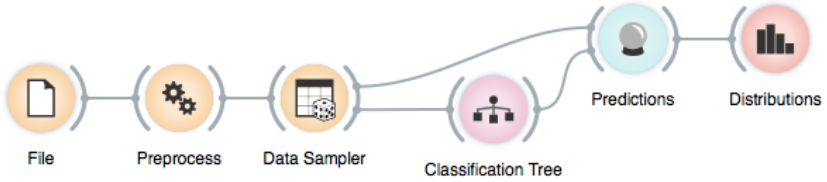


Most unusual. Almost no mistakes. How is this possible? On a class-randomized data set?

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

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

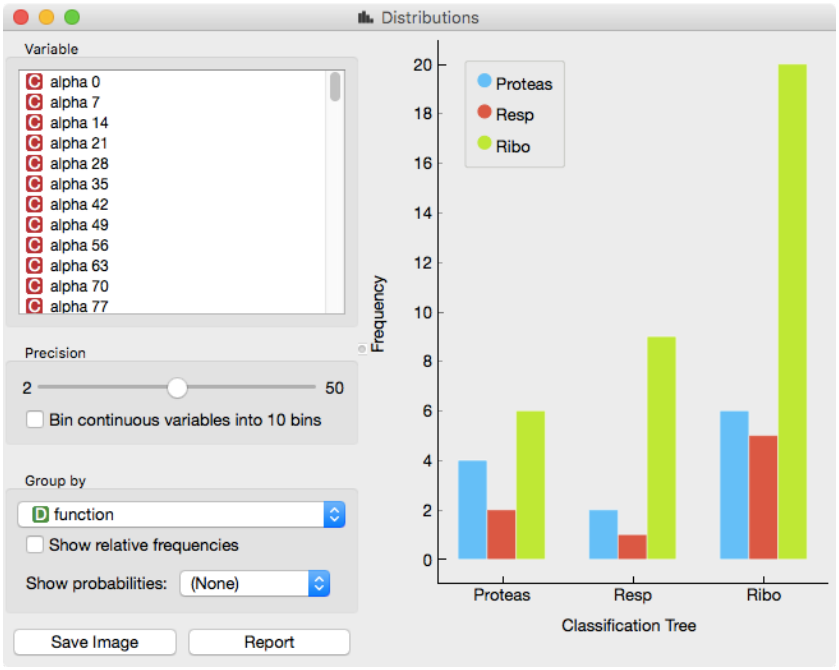
Ha, if this is so, if a classifier remembers everything from a data set but without discovering any general patterns, it should perform miserably on any new data set. Let us check this out. We will split our data set into two sets, training and testing, train the classification tree on the training data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



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

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

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



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

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.

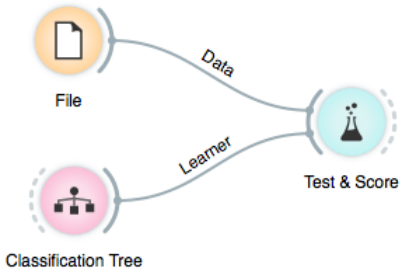
We needed to class-randomize only the training data set to fail in predictions. Try changing the workflow so that the classes are randomized only there, and not in the test set.

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

Lesson 10: Cross-Validation

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

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



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

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

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

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