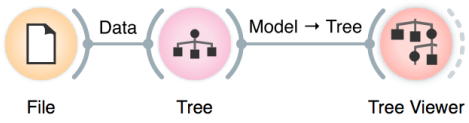
Lesson 15: Classification

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.

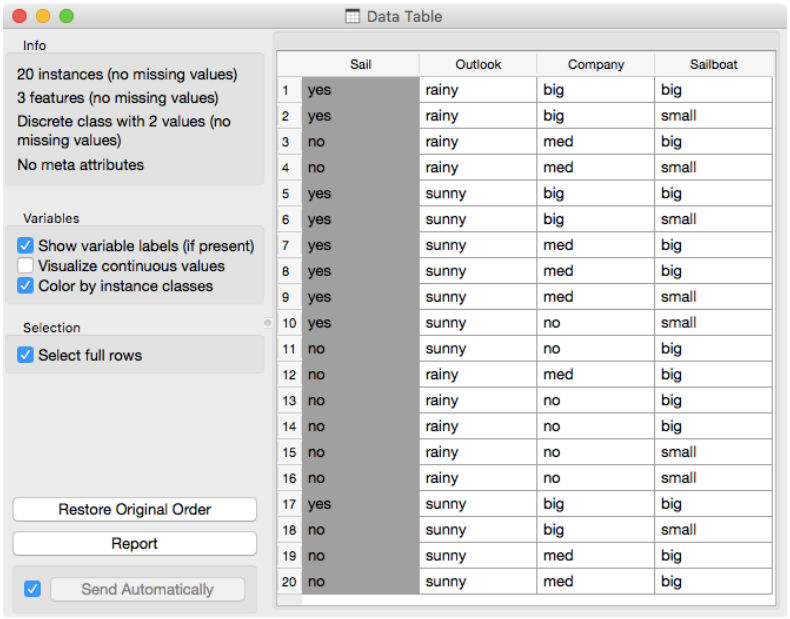
Classification tree is one of the oldest, but still popular, machine learning methods. We like it since the method is easy to explain and gives rise to random forests, one of the most accurate machine learning techniques. So, what kind of model is a classification tree?

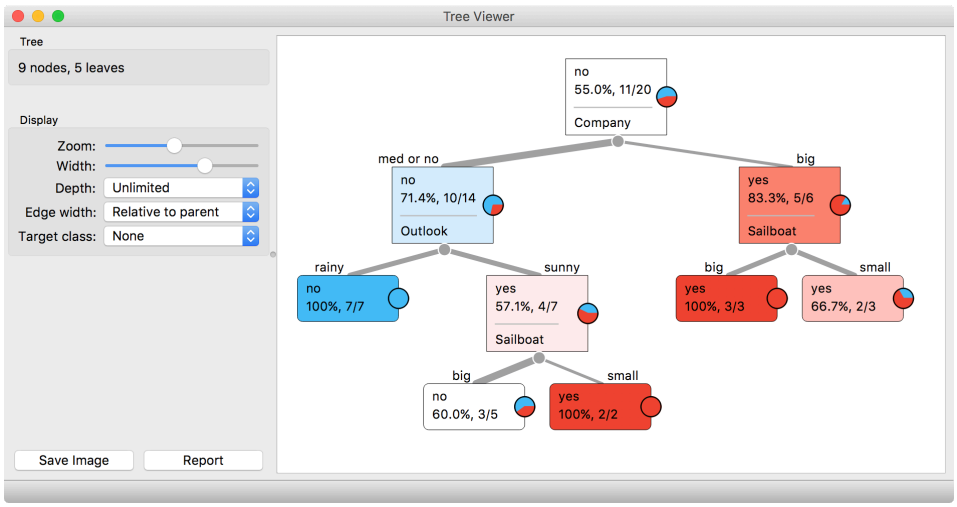
Let us load a data set from http://file.biolab.si/datasets/sailing.tab that records the conditions under which a skipper went sailing, build a tree with a Tree widget and visualize it in the Tree Viewer.

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.



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?





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

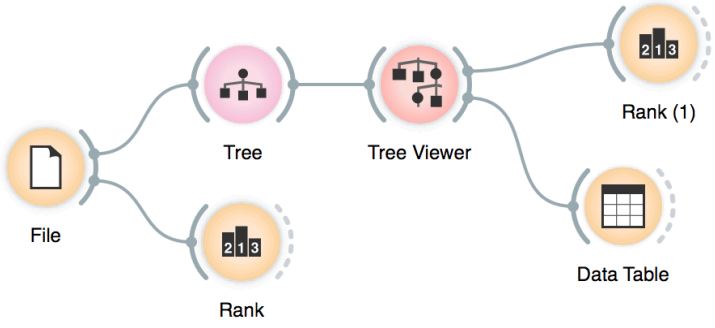
sailing if there is rain. (Thunderstorms? Too dangerous?) When she has a smaller company, but the boat at her disposal is big, there is no sailing either.

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.

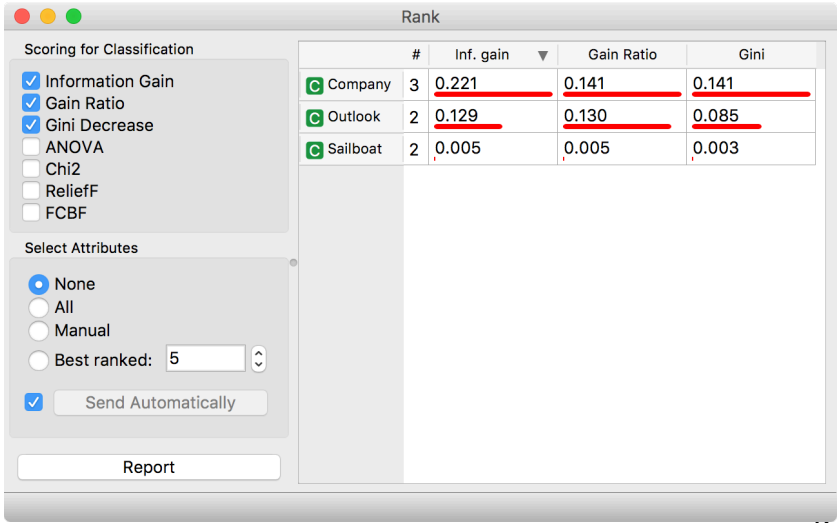
Trees place the most useful feature at the root. What would be the most useful feature? It is the feature that splits the data into two purest possible subsets. These are then split further, again by the most informative features. This process of breaking up the data subsets to smaller ones repeats until 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).

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

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.



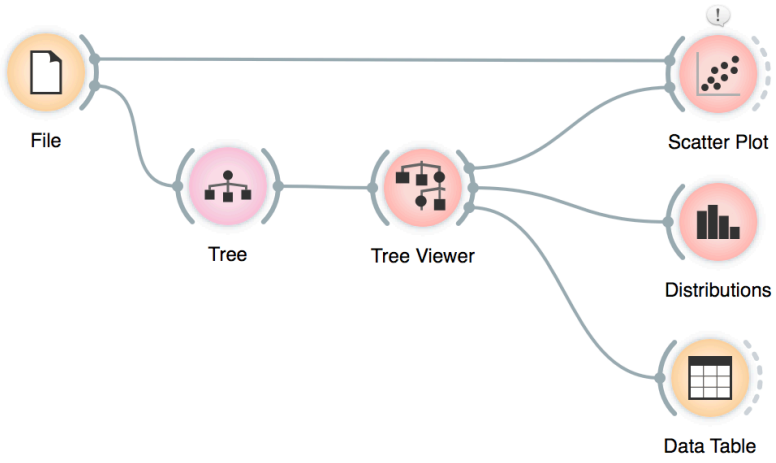
Lesson 16: Model Inspection

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.

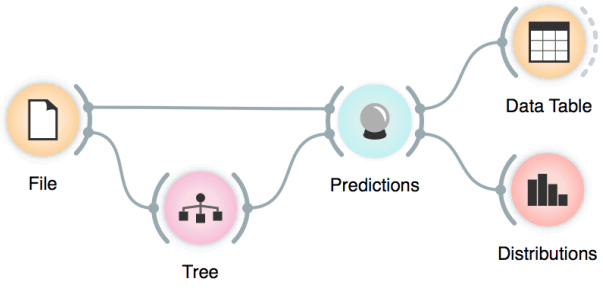


We can also use Distributions widget to see the distribution of classes in each node. In the case of the Brown data, this is pretty boring, though.

Lesson 17: Classification Accuracy

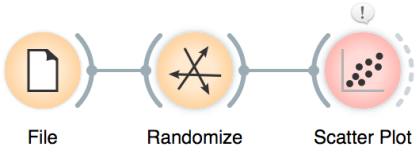
Now that we know what classification trees are, the next question is what is the quality of their predictions. For beginning, we need to define what we mean by quality. In classification, the simplest measure of quality is classification accuracy expressed as the proportion of data instances for which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

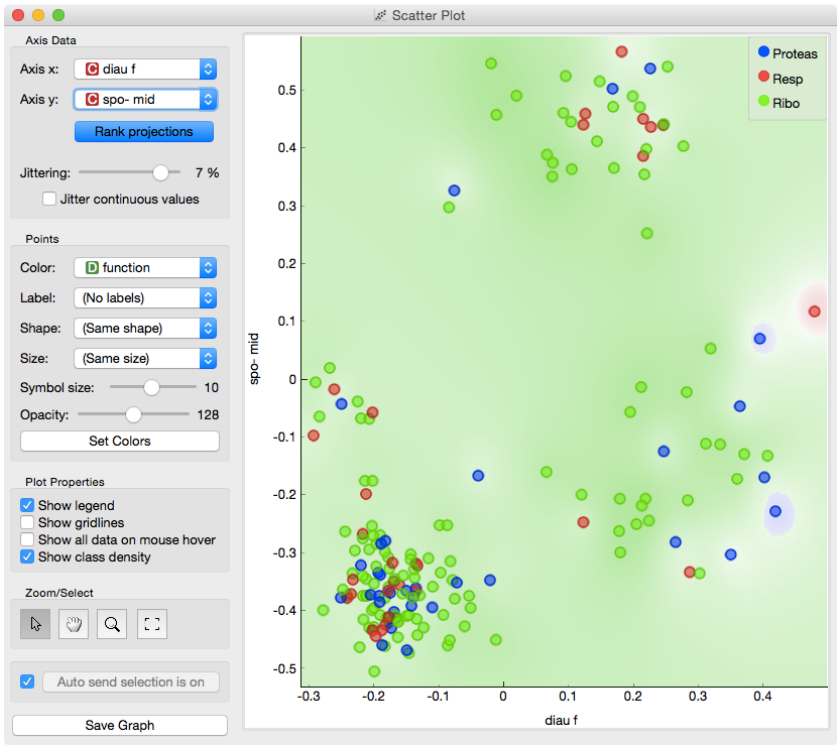


Randomize widget shuffles the column in the data table. It can shuffle the class column, columns with data features or columns with meta information. Shuffling the class column breaks any relation between features and the class, keeping the data points (genes profiles) intact.

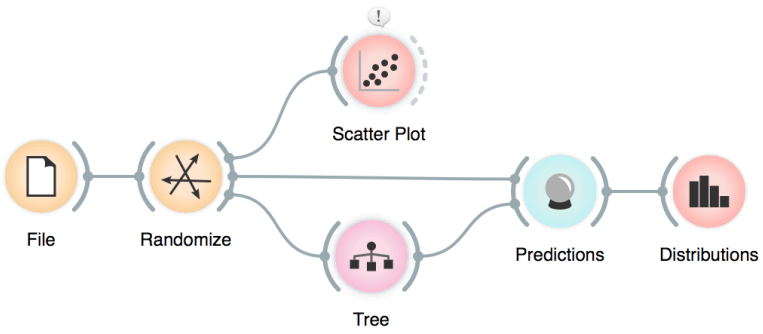
Why is the background in this scatter plot so green, and only green? Why have the other colors disappeared after the class randomization?

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!

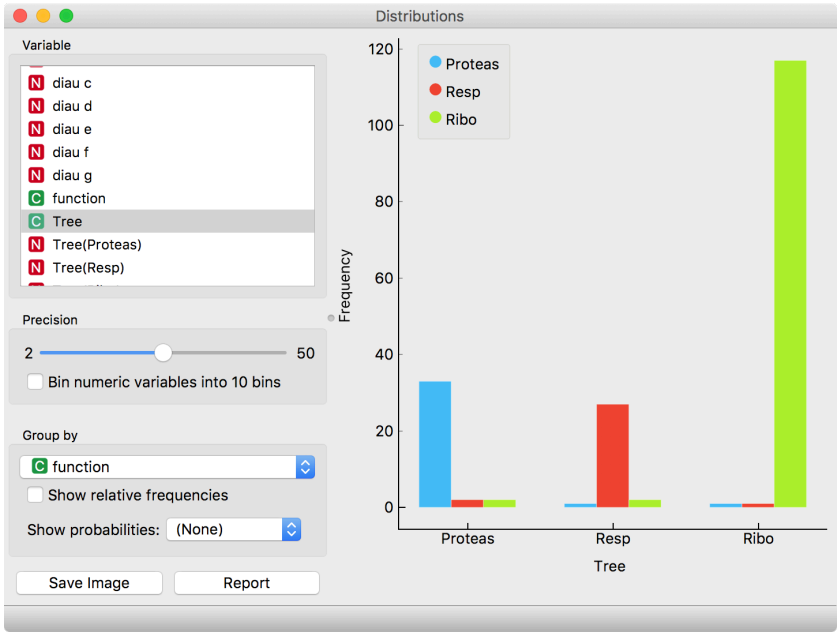


Fine. There can be no classifier that can model this mess, right? Let us test this. We will build classification tree and check its performance on the messed-up data set.



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?

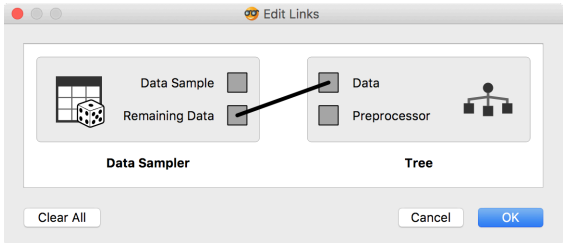
To find the answer to this riddle, open the Tree Viewer and check out the tree. How many nodes does it have? Are there many data instances in the leaf nodes?

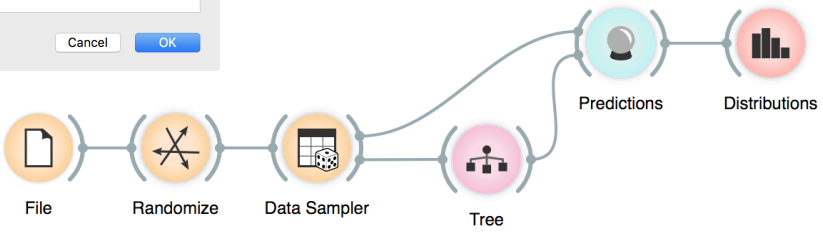
It looks like the tree just memorized every data instance from the data set. No wonder the predictions were right. The tree makes no sense, and it is complex because it simply remembered everything.

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.

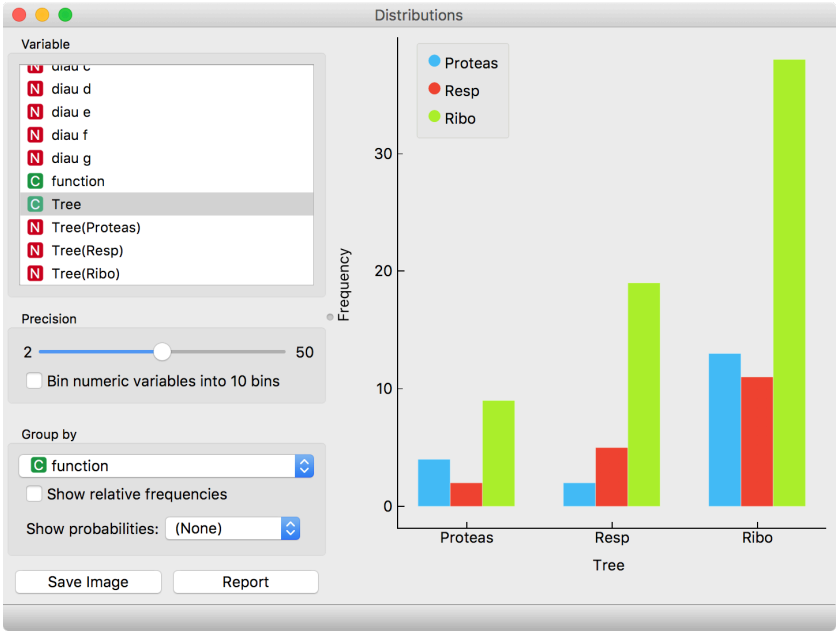
The signals from the Data Sampler widget have not been named in our workflow to save space. The Data Sampler splits the data to a sample and out-of-sample (so called remaining data). The sample was given to the Tree widget, while the remaining data was handed to the Predictions widget. Set the Data Sampler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





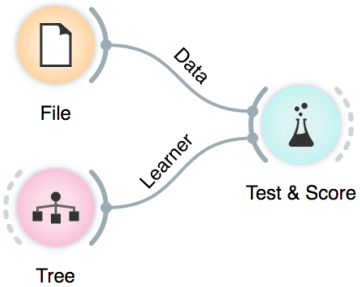
Let's check how the Distributions widget looks after testing the classifier on the test data.

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. 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.



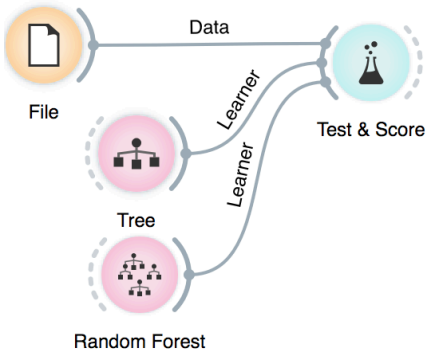
Learning is not only remembering. Rather, it is discovering patterns that govern the data and apply to new data as well. To estimate the accuracy of a classifier, we, therefore, need a separate test set. This assessment should not depend on just one division of the input data set to training and test 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.

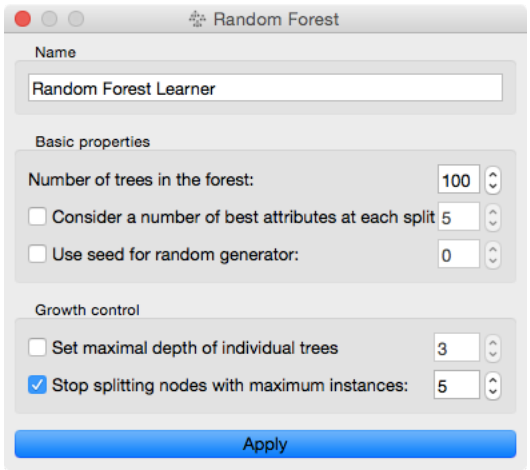
Lesson 19: A Few More Classifiers

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.

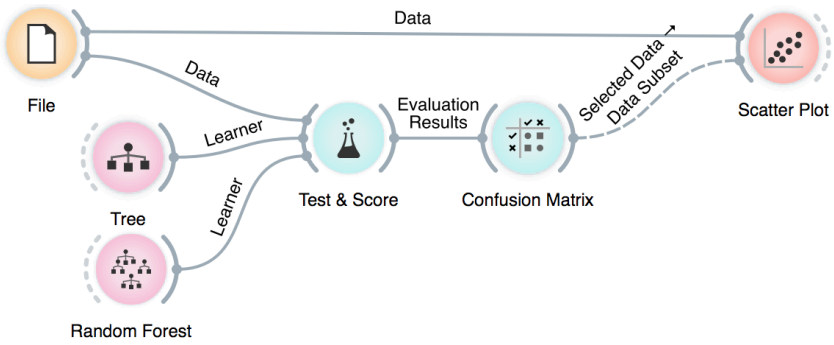


Let us see if this is really so. We give two learners to the Test Learners widget and check if cross-validated classification accuracy is indeed higher for random forest. Choose different classification data sets for this comparison, starting with those we already know (hearth disease, iris, brown selected).

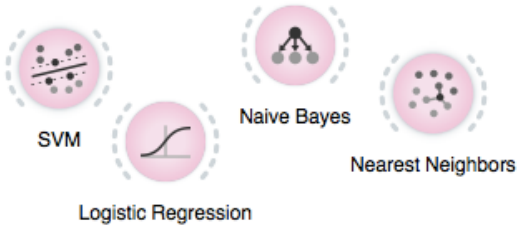


It may be interesting to compare where different classification methods make mistakes. We can use Confusion Matrix for this purpose, and then pass the signal from this widget to the Scatter Plot.

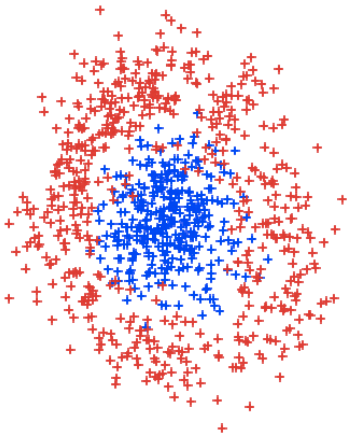
What kind of object is sent from the Test & Score widget to the Confusion Matrix widget? So far, we have used widgets that send data, or even learners. But what could the Test & Score widget communicate to other widgets?



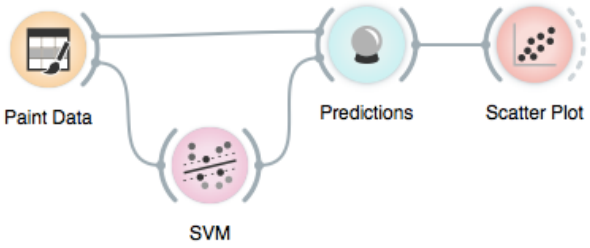
There are other classifiers we can try. We will briefly mention a few more, we won't dive to deep into how they work (we could spend a semester on this!).

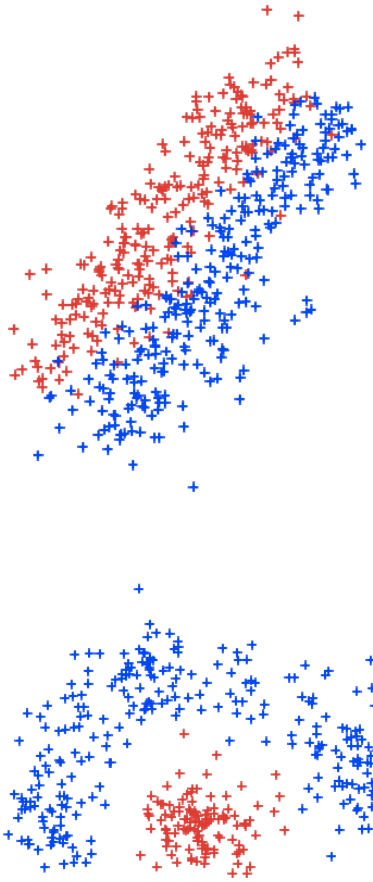


It would be nice if we could, at least on the intuitive level, understand the differences between all these methods and their variants (every method has some parameters). Remember, the classification tree finds hyperplanes orthogonal to the axis; those hyperplanes split the data space to regions with 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.