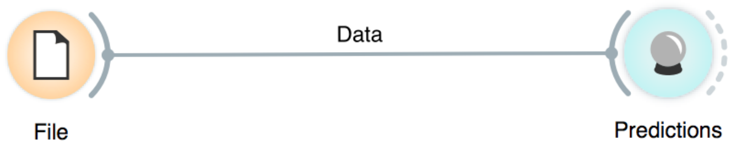
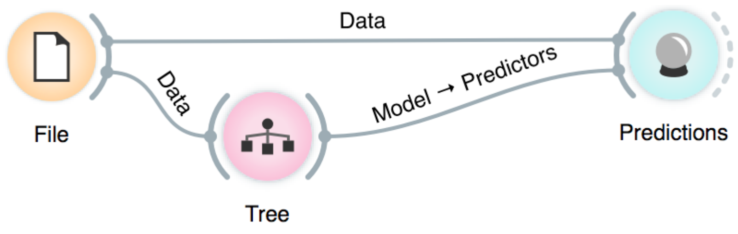
Lesson 15: Classification

Heart disease data was presented in notes, but we have not gone through it in the class. While you are encouraged to explore it, you can also use the workflows from this lesson on, say, iris data set.

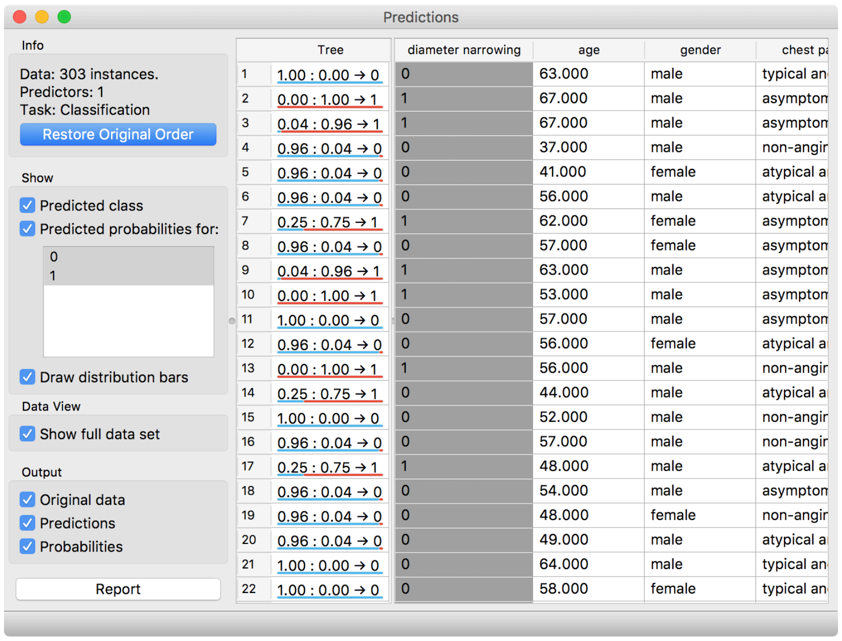
In one of the previous lessons, we explored the heart disease data. We wanted to predict which persons have clogged arteries — but we did not make any predictions. Let's try it now.



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



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



Data: 303 instances.
Predictors: 1
Task: Classification

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

✓ Show full data set

✓ Original data
✓ Predictions
✓ Probabilities

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

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

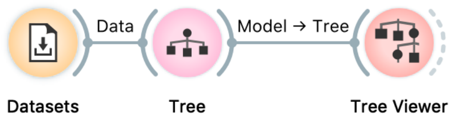
How correct are these predictions? Do we have a good model? How can we tell?

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

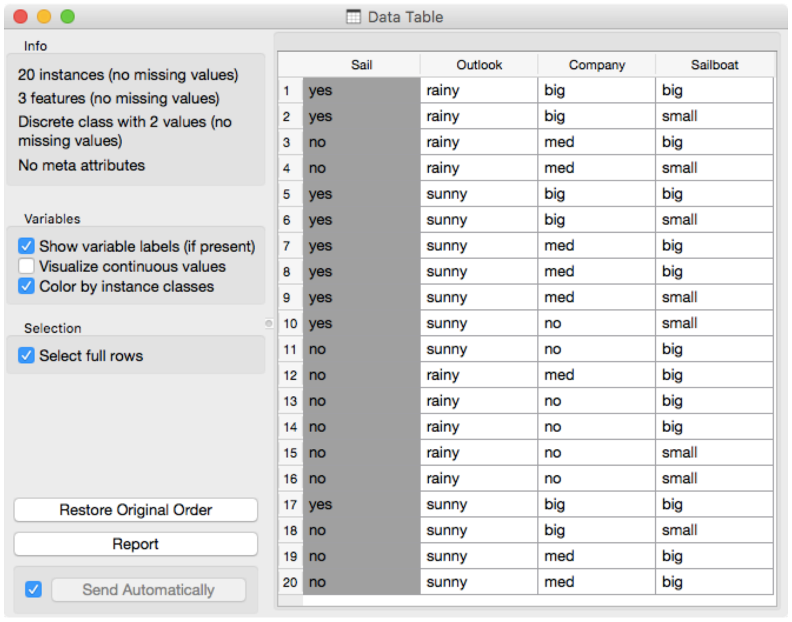
Lesson 16: Classification Trees

A classification tree is one of the oldest, but still popular, machine learning methods. We like it since the technique 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 use the data set on sailing from the Datasets widget that records the conditions under which a friend skipper went sailing, build a tree, 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?



Data Table

Info

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

Variables

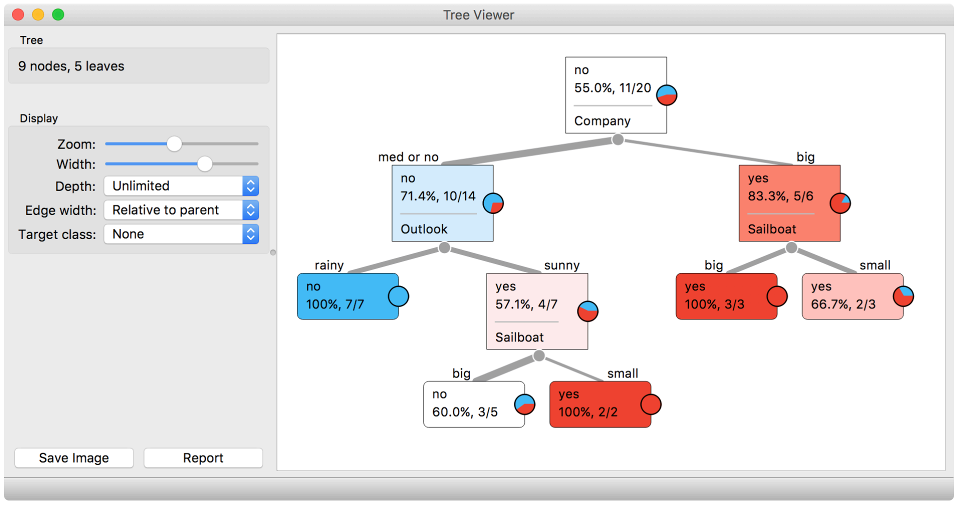
☑ Show variable labels (if present)
☐ Visualize continuous values
☑ Color by instance classes

Selection

☑ Select full rows

Restore Original Order
Report
☑ Send Automatically

	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



Tree Viewer

Tree

9 nodes, 5 leaves

Display

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

Save Image
Report

We read the tree from top to bottom. It looks like this skipper is a social person; as soon as there's a 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 large, there is no sailing either.

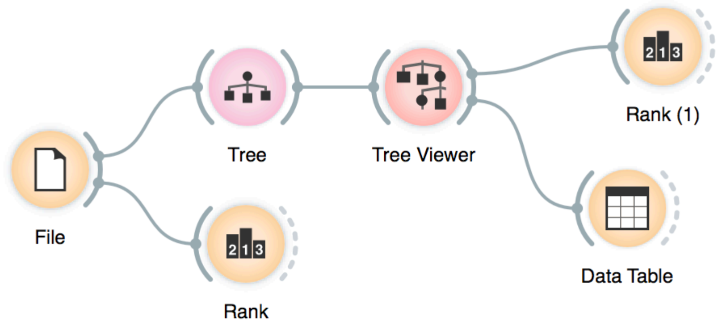
Classification trees were hugely popular in the early years of machine learning, when they were first independently proposed by the engineer Ross Quinlan (C4.5) and a group of statisticians (CART), including the father of random forests Leo Brieman.

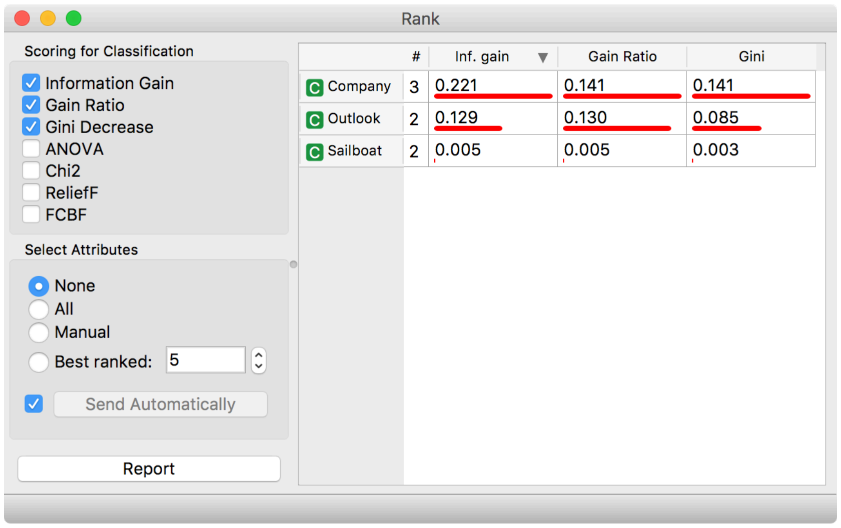
The Rank widget could be used on its own. Say, to figure out which genes are best predictors of the phenotype in some gene

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 amount of 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.





Scoring for Classification

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

Select Attributes: [● None] ○ All ○ Manual ○ Best ranked: 5

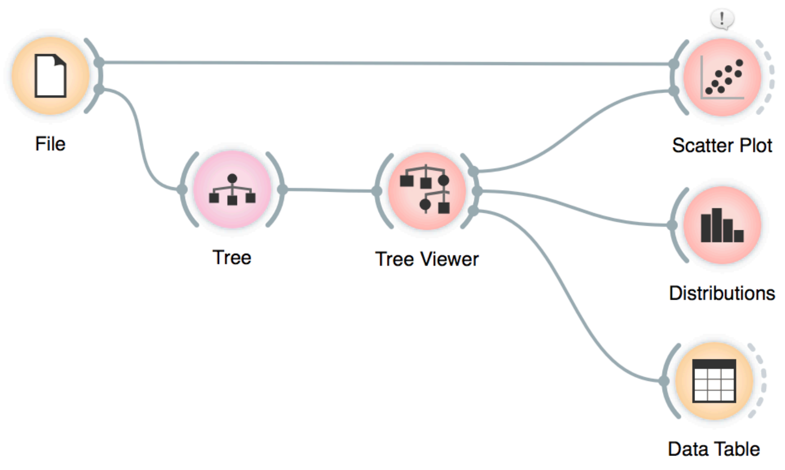
[✓] Send Automatically

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

Lesson 17: 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 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.

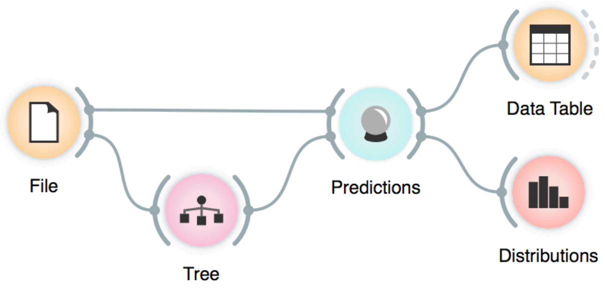


Just for fun, we have included a few other widgets in this workflow. In a way, the Tree Viewer widget behaves like the Select Rows widget, except that the rules used to filter the data are inferred from the data itself and optimized to obtain purer data subsets.

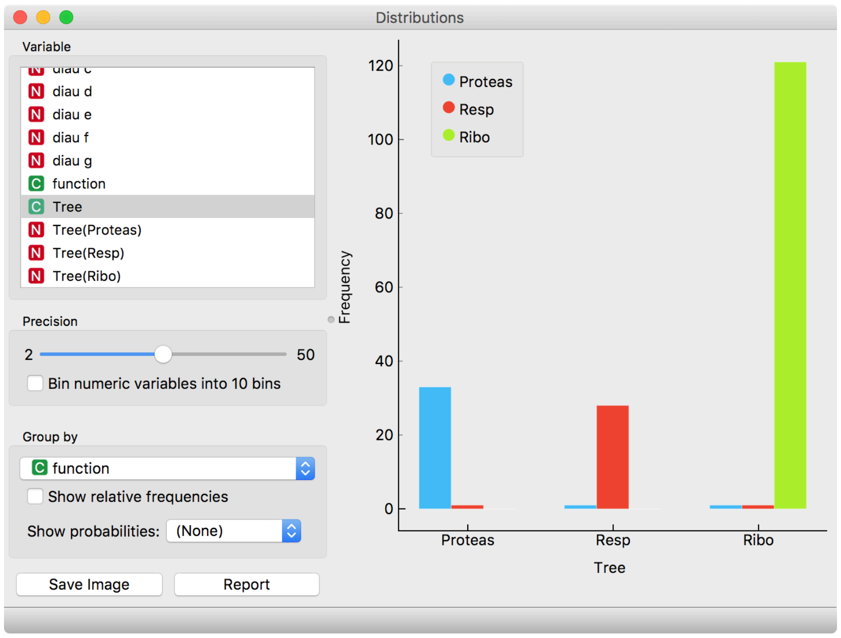
Lesson 18: Classification Accuracy

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

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



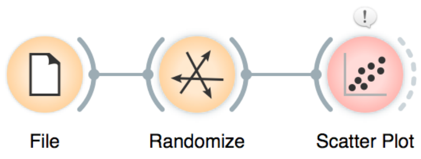
Let us try this schema with the brown-selected data set. The Predictions widget outputs a data table augmented with a column that includes predictions. In the Data Table widget, we can sort the data by any of these two columns, and manually select data instances where the values of these two features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



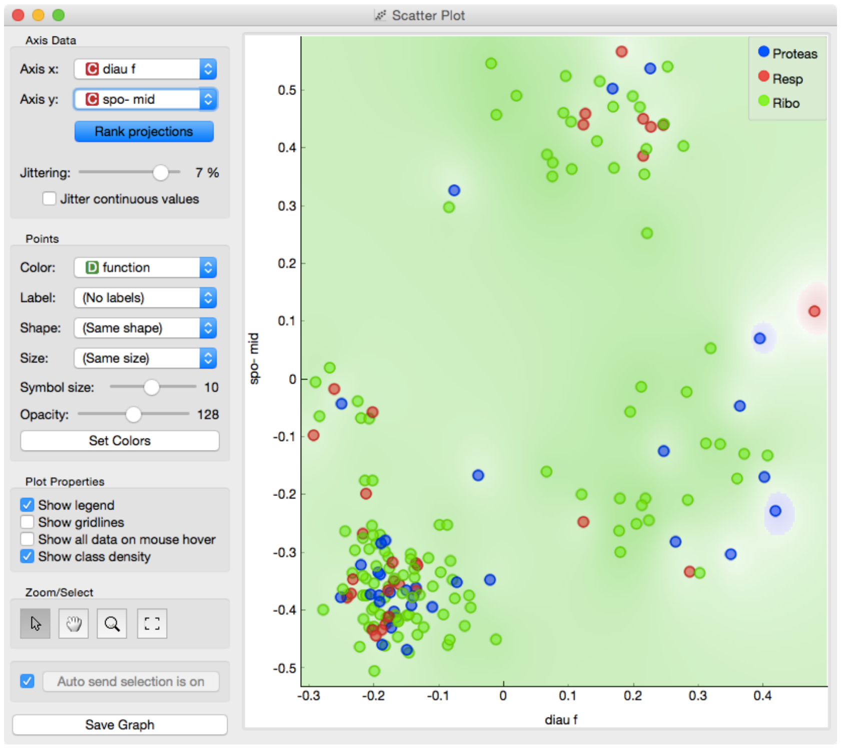
Lesson 19: How to Cheat

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

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

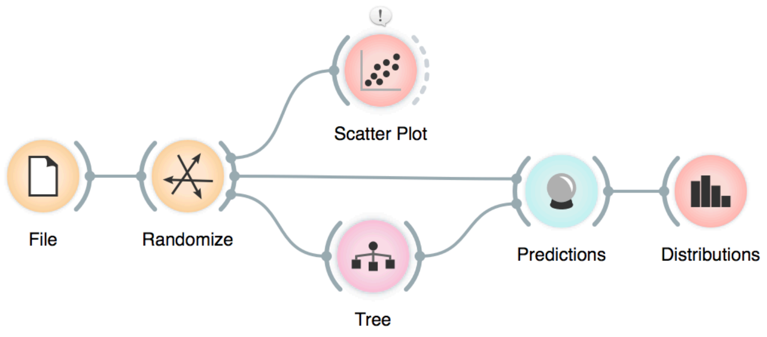


Randomize widget shuffles the column in the data table. It can shuffle the class column, columns with data features or columns with meta information. Shuffling the class column breaks any relation between features and the class, keeping the data points (genes profiles) intact.



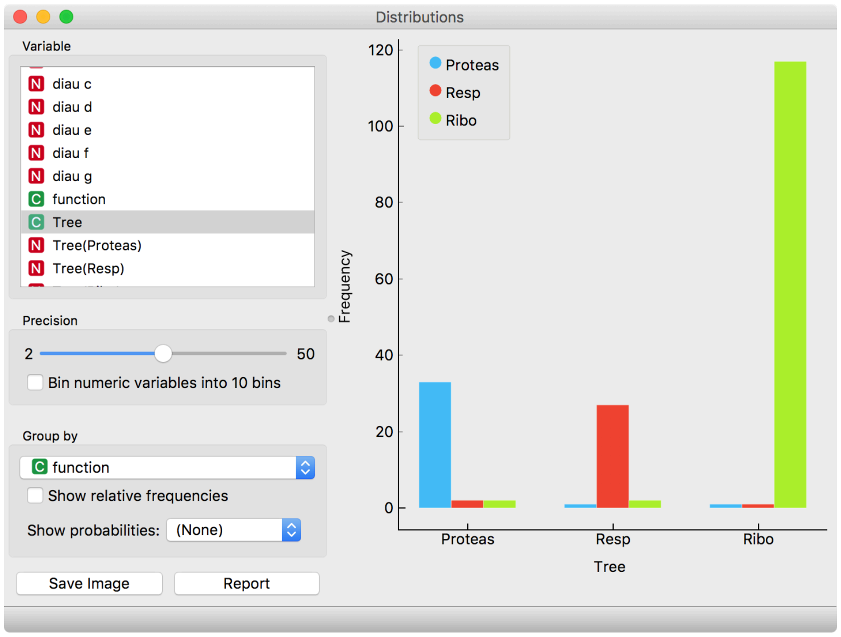
Why is the background in this scatter plot so green, and only green? Why have the other colors disappeared after the class randomization?

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



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!



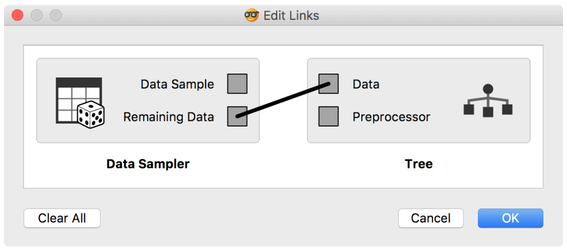
Most unusual. Almost no mistakes. How is this possible? On a class-randomized data set?

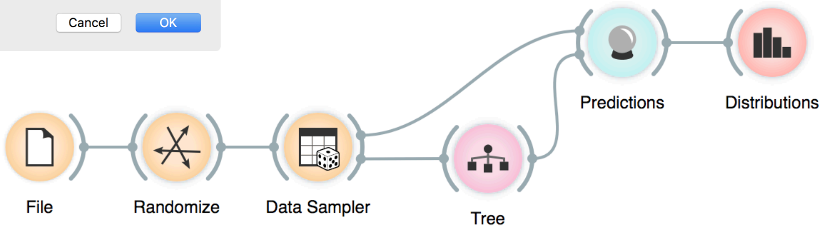
The signals from the Data Sampler widget have not been named in our workflow to save space. The Data Sampler splits the data to a sample and out-of-sample (so called remaining data). The sample was given to the Tree widget, while the remaining data was handed to the Predictions widget. Set the Data Sampler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

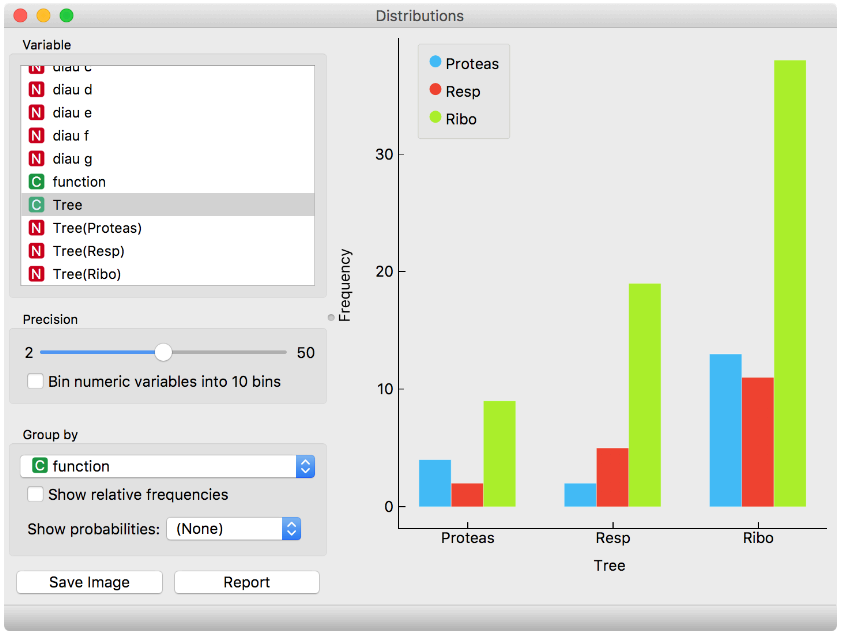
Ha, if this is so, that is, if a classifier remembers everything from a data set but without discovering any general patterns, it should perform miserably on any new data set. Let us check this out. We will split our data set into two sets, training and testing, train the classification tree on the training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





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

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



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

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

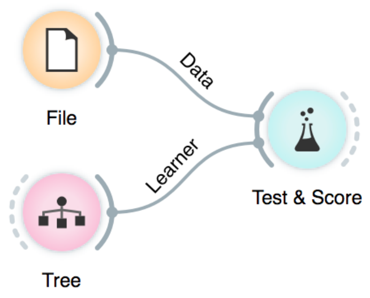
We needed to class-randomize only the training data set to fail in predictions. Try changing the workflow so that the classes are randomized only there, and not in the test set.

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

Lesson 20: Cross-Validation

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

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

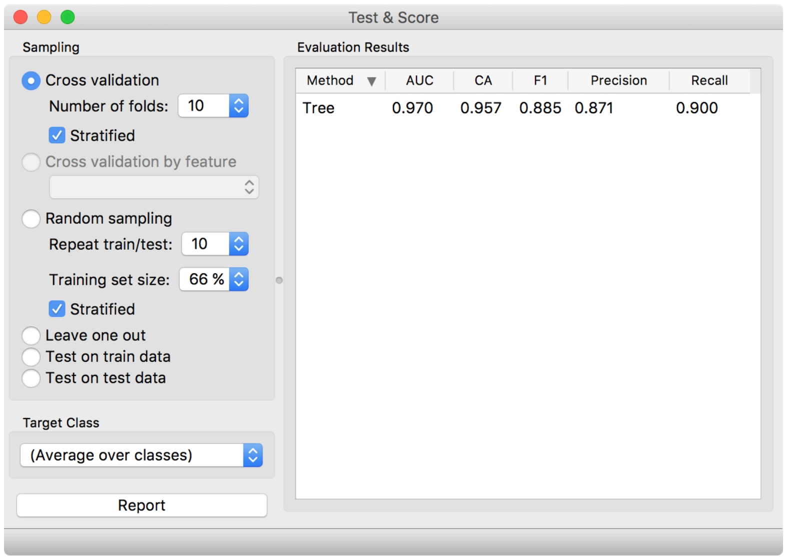


For geeks: a learner is an object that, given the data, outputs a classifier. Just what Test & Score needs.

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

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

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



Sampling

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

Target Class: (Average over classes)

Evaluation Results

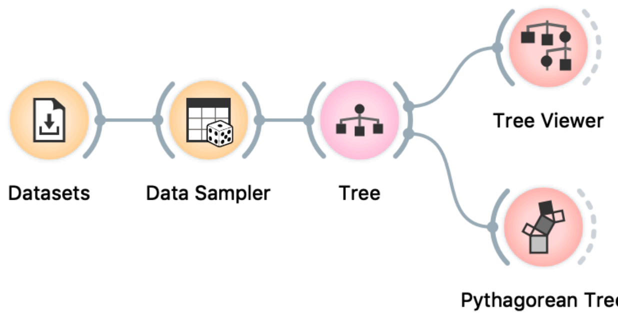
Method	AUC	CA	F1	Precision	Recall
Tree	0.970	0.957	0.885	0.871	0.900

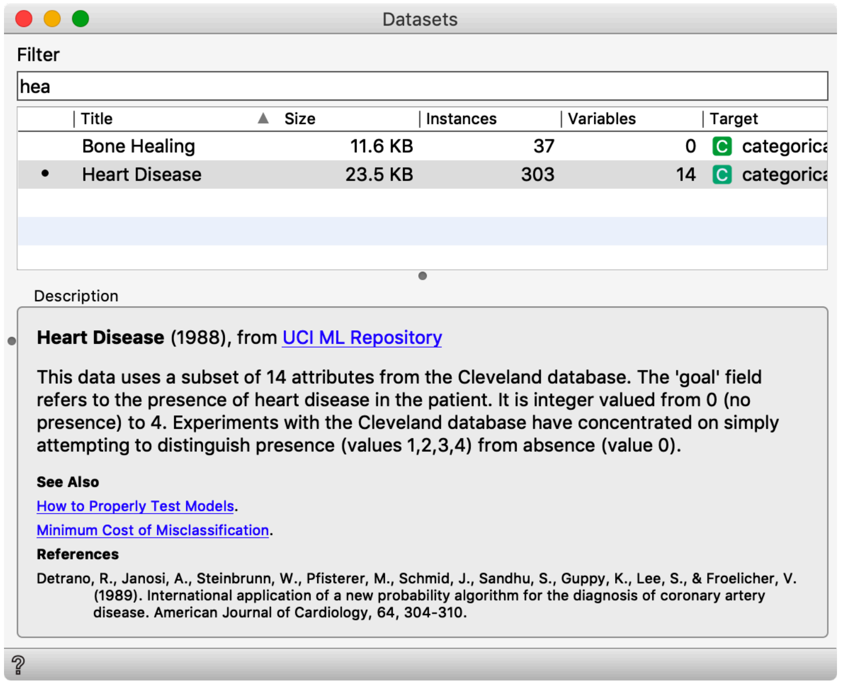
Lesson 21: Trees are not Stable

Classification trees may be sensitive to any changes in the input data. With a minor change in the data, the structure of the classification tree can drastically change. Data in biomedicine may include measurement or observation noise. Beautiful property of the trees is that they are interpretable; that is, we can "read" the model and explain it. But with the instability of the trees, if this is indeed the case, the interpretability does not make much sense.

Let us observe the stability of the classification trees on an example. We will consider the data on the hearth disease with 14 attributes and a binary class variable that reports on the presence of the disease.

The trees inferred from heart disease dataset are large and do not fit well in Tree Viewer's visualization. Orange has a widget Pythagorean Tree with an alternative, more compact display.




Datasets

Filter

hea

	Title	Size	Instances	Variables	Target
	Bone Healing	11.6 KB	37	0	C categorica
•	Heart Disease	23.5 KB	303	14	C categorica

Description

Heart Disease (1988), from UCI ML Repository

This data uses a subset of 14 attributes from the Cleveland database. The 'goal' field refers to the presence of heart disease in the patient. It is integer valued from 0 (no presence) to 4. Experiments with the Cleveland database have concentrated on simply attempting to distinguish presence (values 1,2,3,4) from absence (value 0).

See Also

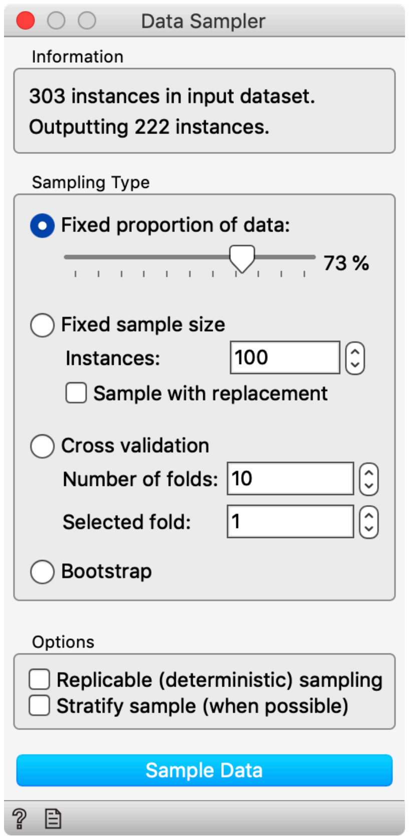
How to Properly Test Models.

Minimum Cost of Misclassification.

References

Detrano, R., Janosi, A., Steinbrunn, W., Pfisterer, M., Schmid, J., Sandhu, S., Guppy, K., Lee, S., & Froelicher, V. (1989). International application of a new probability algorithm for the diagnosis of coronary artery disease. American Journal of Cardiology, 64, 304-310.

In the workflow, we sample 70% of the data, develop a classification tree, and visualize it. Keeping both Data Sampler and


Data Sampler

Information

303 instances in input dataset.
Outputting 222 instances.

Sampling Type

[✓ Fixed proportion of data:] 73 %

Fixed sample size
Instances: 100
Sample with replacement

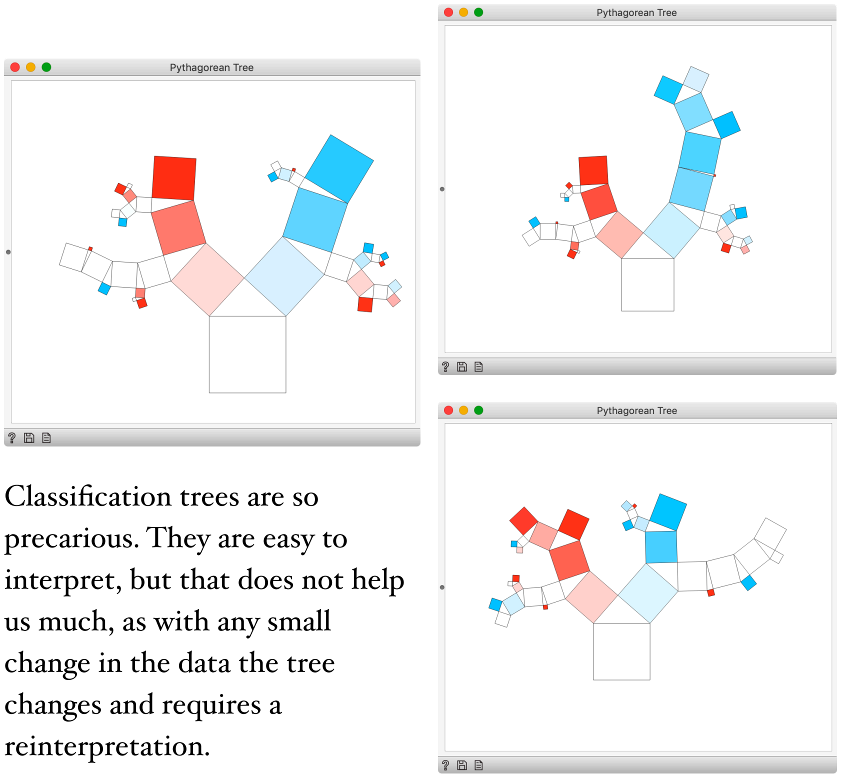
Cross validation
Number of folds: 10
Selected fold: 1

Bootstrap

Options

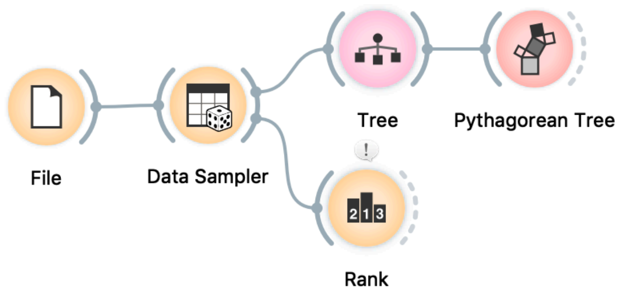
Replicable (deterministic) sampling
Stratify sample (when possible)

Pythagorean Tree widgets on the top, we can now create a new sample and see the changes in the tree structure. They are substantial, and it looks like that with every new sample, we obtain an entirely different tree.



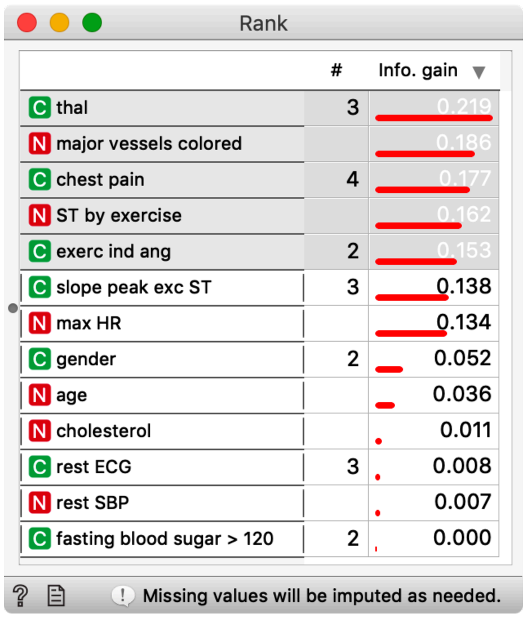
Classification trees are so precarious. They are easy to interpret, but that does not help us much, as with any small change in the data the tree changes and requires a reinterpretation.

Instability of the trees emerges as, when developing the tree, at each internal node the algorithm chooses the most informative feature for the split of the node's training data. If there are several features with comparable correlation with the class, tiny changes in the data can change the ranking of the features. Changes in the feature order on the top reflects the change in the choice of the feature that will be selected to split the data at the root.



To explore how the changes in the data impact the ranking of the features in the tree's top node, that is, for the entire data set, we can use the Rank widget. Even for the whole dataset, when the set dataset is still reasonably large, different features win for the different data sample. When the feature selected for the root of the tree changes, the structure of the entire tree would change.

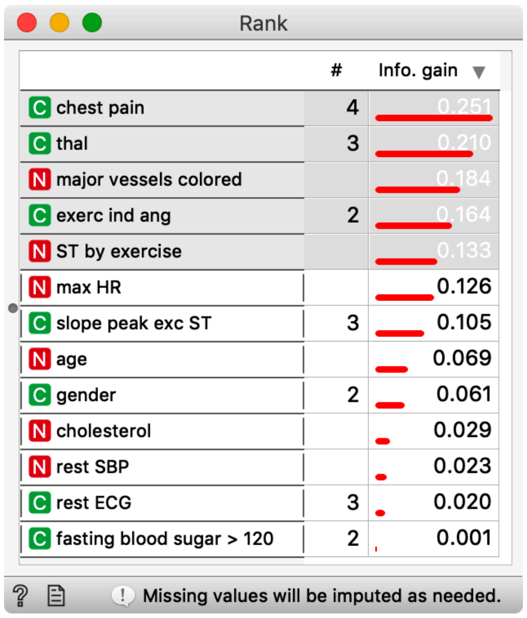
The instabilities of feature ranks are even more pronounced in the lower layers of the tree, where the datasets pertinent to each node becomes smaller.



Rank

	#	Info. gain
C thal	3	0.219
N major vessels colored		0.186
C chest pain	4	0.177
N ST by exercise		0.162
C exerc ind ang	2	0.153
C slope peak exc ST	3	0.138
N max HR		0.134
C gender	2	0.052
N age		0.036
N cholesterol		0.011
C rest ECG	3	0.008
N rest SBP		0.007
C fasting blood sugar > 120	2	0.000

Missing values will be imputed as needed.



Rank

	#	Info. gain
C chest pain	4	0.251
C thal	3	0.210
N major vessels colored		0.184
C exerc ind ang	2	0.164
N ST by exercise		0.133
N max HR		0.126
C slope peak exc ST	3	0.105
N age		0.069
C gender	2	0.061
N cholesterol		0.029
N rest SBP		0.023
C rest ECG	3	0.020
C fasting blood sugar > 120	2	0.001

Missing values will be imputed as needed.

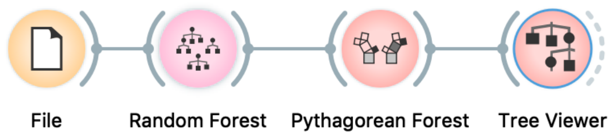
Feature ranking on entire sample of heart disease data set shows that the order of the features depends on the sample. Features most correlated with the class always appear on the top of the list, but their order changes. The tree induction algorithm would always pick the top-ranked feature for the split and the sole change in the root of the tree would result in entirely different trees.

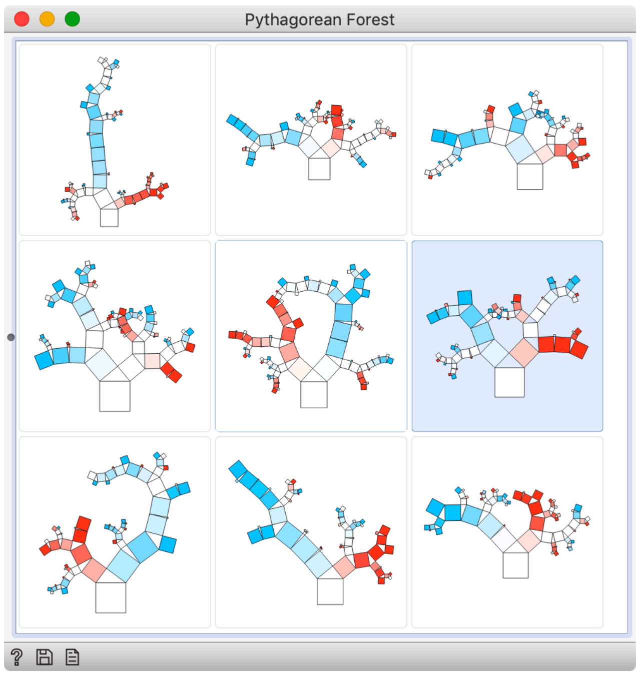
Lesson 22: Random Forests

With instability of trees, the interpretive advantage of the tree is gone. But constructing different, or better, entirely different trees from the same data set could have an advantage. Remember the TV show Who Wants to Be a Millionaire? In the show, the contestant has to answer consecutive multiple-choice questions of increasing difficulty. There is a lifeline called Ask the Audience, where the contestant asks the audience to answer the question and obtains the help by seeing the distribution of the answers. Most often, the answer that received the most votes from the audience is the right one. Such "wisdom of the crowd" approach, where the collective opinion of a group of individuals rather than that of a single expert, is today employed by social information sites. And is used in machine learning. Each of the tree, inferred from the sample of the training data, could be considered an individual. A collection of the tree would vote for the class, and a machine learning technique called random forests would then predict the class value that received the majority of votes.

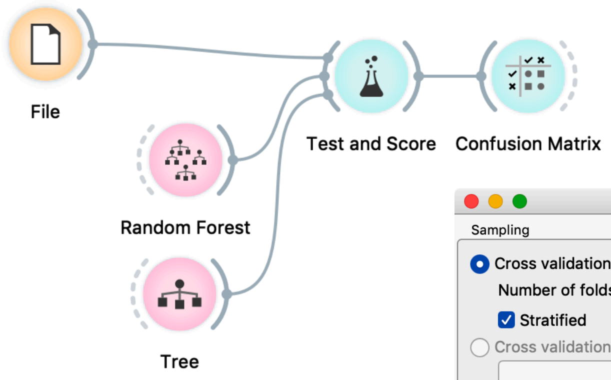
Random forest consist of a set of trees that can be in visualized in Orange using Pythagorean Forest widget. To observe the details of the tree, a selected tree from the forest can be sent to the Tree Viewer or to Pythagorean Tree for further inspection. Visualizing the trees in the forest serves only to assure us that the inferred trees are indeed different. Else, random forest are treated as black boxes.

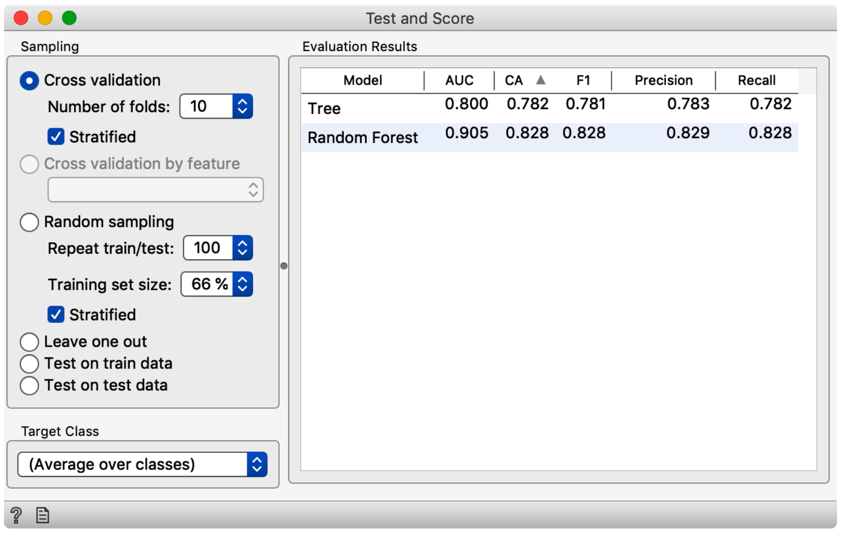
Random forest develop trees from so-called bootstrap sample, a sample of the training data that is of the same size but draws randomly from the training set with replication. To increase the diversity of the trees, the features on which the tree nodes split the data are randomly drawn from the top-ranked features.





Random forests are impossible to interpret. Their only gain is in accuracy. And this could be substantial. Let us measure this through cross-validation on a heart disease data.





Test and Score

Sampling
- [✓ Cross validation]
  - Number of folds: 10
  - ✓ Stratified
- Cross validation by feature
- Random sampling
  - Repeat train/test: 100
  - Training set size: 66 %
  - ✓ Stratified
- Leave one out
- Test on train data
- Test on test data

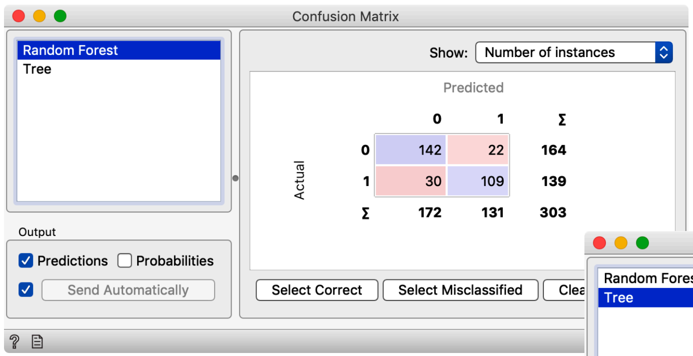
Target Class: (Average over classes)

Evaluation Results

Model	AUC	CA	F1	Precision	Recall
Tree	0.800	0.782	0.781	0.783	0.782
Random Forest	0.905	0.828	0.828	0.829	0.828

A core parameter of the random forest machine learning method is the number of trees. We have set this parameter to 100 in our experiment. Most often, a few hundred trees in the forest are sufficient for optimized accuracy, and increasing the number of trees above 500 does not yield any further gains.

The gain of the forest over the individual tree is substantial. Not just in classification accuracy (CA), but also in every other statistics that we will present in detail in one of the following lessons.

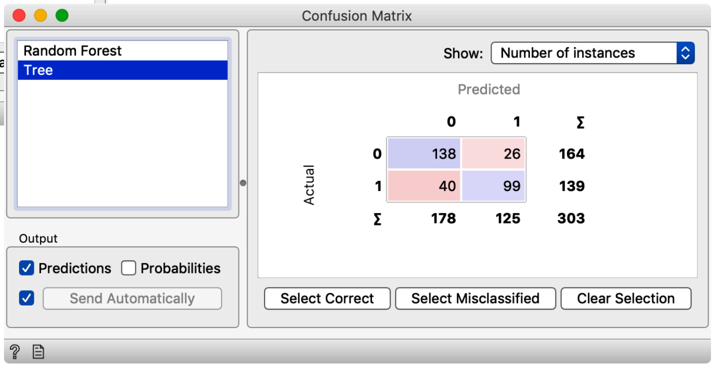


Confusion Matrix — Random Forest (selected), Tree

Show: Number of instances

Actual \ Predicted	0	1	Σ
0	142	22	164
1	30	109	139
Σ	172	131	303

Output: ✓ Predictions, Probabilities; ✓ Send Automatically



Confusion Matrix — Random Forest, Tree (selected)

Show: Number of instances

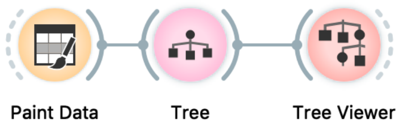
Actual \ Predicted	0	1	Σ
0	138	26	164
1	40	99	139
Σ	178	125	303

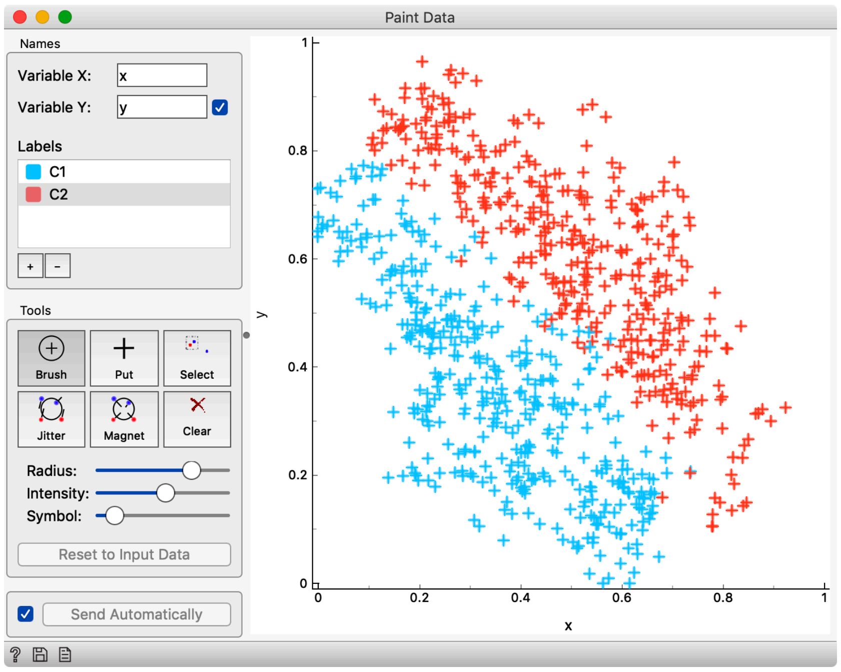
Output: ✓ Predictions, Probabilities; ✓ Send Automatically

With classifications, it is always wise to check the confusion matrix. Here, we see that out 303 data instances, the trees misclassified 66, and random forests misclassified 52, with a most significant reduction of misclassification taking place for the subjects with disease.

Lesson 23: Logistic Regression

Trees “cut” the feature space according to the value of a single feature, that is, they discover “boxes” in the feature space with a prevailing class value. The method struggles with more complex shapes of decision boundaries. We can paint the data set where the classification tree would struggle. For instance, for the painted data set as depicted below, the inferred tree is quite complicated and contains 22 internal nodes and 11 laves.

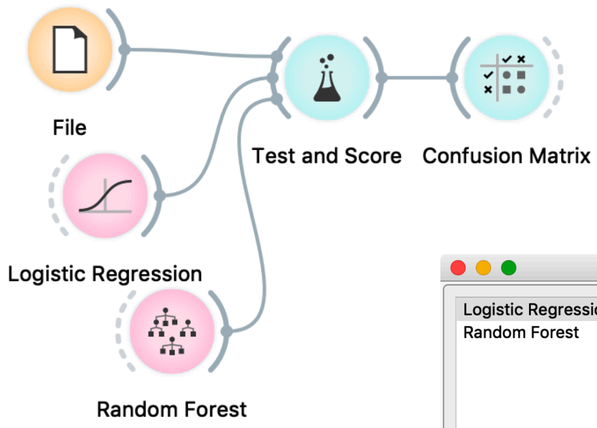


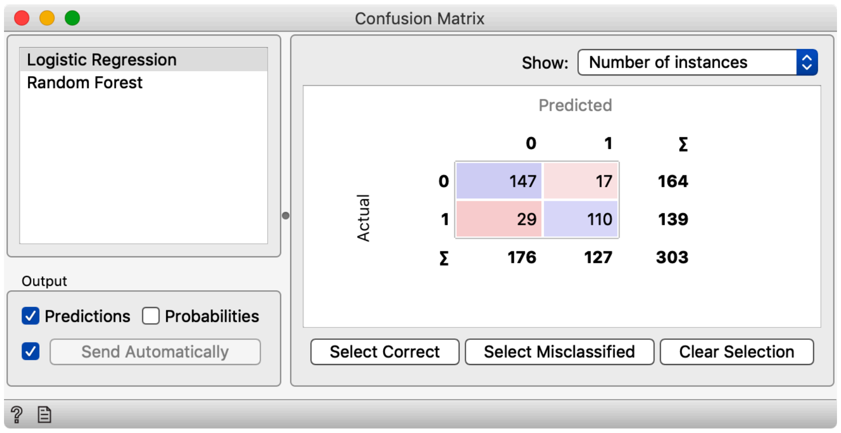


Variable X: x
Variable Y: y
C1
C2

Possibly the best model for the data shown above is a line that splits blue and red points. We could also assign the class probabilities according to this separation line: the further away from the line a data point, the more likely it belongs to one or the other class. The data points on the separation line are those for which we cannot decide on a class, and where the class probabilities are equal, that is. Notice that a line could separate the two classes in two-dimensional feature space, whereas for three or higher dimensional spaces we need a plane or a hyperplane.

The model that follows our reasoning above, and from the data infers the planes that separate the two classes is called logistic regression. The “language” of this model is linear as the planes are flat. Surprisingly, though, this model behaves rather well and is often comparable to the random forest. On the heart disease data set, the logistic regression even slightly beats the forest, as demonstrated in the now-familiar workflow with *Test & Score* widget.





Show: Number of instances

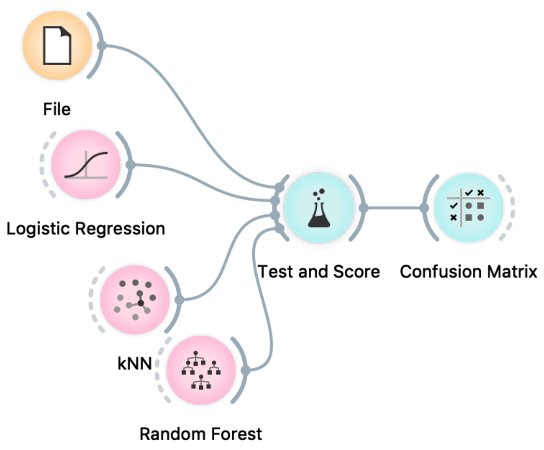
Predicted / Actual

	0	1	Σ
0	147	17	164
1	29	110	139
Σ	176	127	303

There are, of course, cases where logistic regression fails miserably, and it is not hard to, for instance, paint a data set where the performance of this model would score much lower than random forests. Because of the simplicity and robustness that comes with it, logistic regression is an often-used technique. Logistic regression is also a core building block in neural networks, and we will address this briefly towards the end of this course.

Lesson 24: *k*-Nearest Neighbors

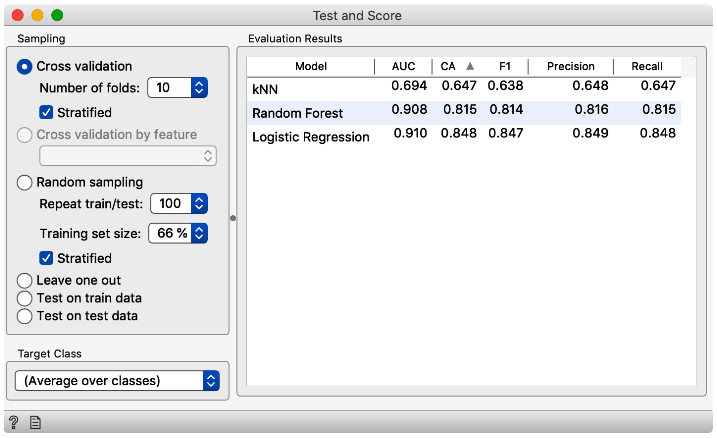
Here is one more technique before we finish with methods for classification. It is perhaps the most intuitive technique of them all, but surprisingly imprecise. We mention it here solely because of the concept, and not as a method you should use when developing a predictive model.



The method is called *k*-nearest neighbors, and it does what the name says. It classifies the new data instance according to its k nearest neighbors from the training set. For example, in the space with two continuous features, the data instance marked with A will be classified as blue, and instance B as red. We have used 3 for the value of *k*, that is, for every new instance, we found five closest data instances in the training set.

The parameter for k-nearest neighbors method is *k*, which is by default set to 10, and the distance function, where the default is Euclidean distance. In general, *k*-NN could use any other approach to distance scoring.

The performance of the nearest neighbor classification on heart disease data set is not stellar. Nearest neighbors perform substantially worse than logistic regression. In cases, where data would reside in two-dimensions, *k*-NN would do something similar to what we would be doing manually. Thus, this approach is at least conceptually interesting.



Sampling:
- (●) Cross validation
  - Number of folds: 10
  - ✓ Stratified
- ( ) Cross validation by feature
- ( ) Random sampling
  - Repeat train/test: 100
  - Training set size: 66 %
  - ✓ Stratified
- ( ) Leave one out
- ( ) Test on train data
- ( ) Test on test data

Target Class: (Average over classes)

Model	AUC	CA	F1	Precision	Recall
kNN	0.694	0.647	0.638	0.648	0.647
Random Forest	0.908	0.815	0.814	0.816	0.815
Logistic Regression	0.910	0.848	0.847	0.849	0.848

Exercise 4: Projections and Embeddings

The exercise for this lesson is available at https://forms.gle/3peayyakphX5MPsa6 .