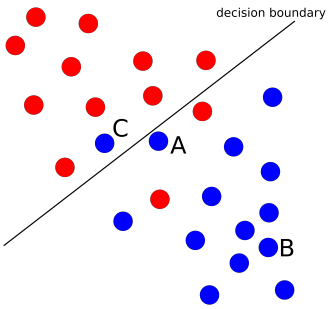
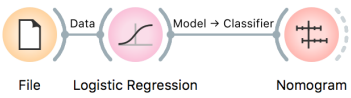
Logistic Regression



Can you guess what would the probability for belonging to the blue class be for A, B, and C?

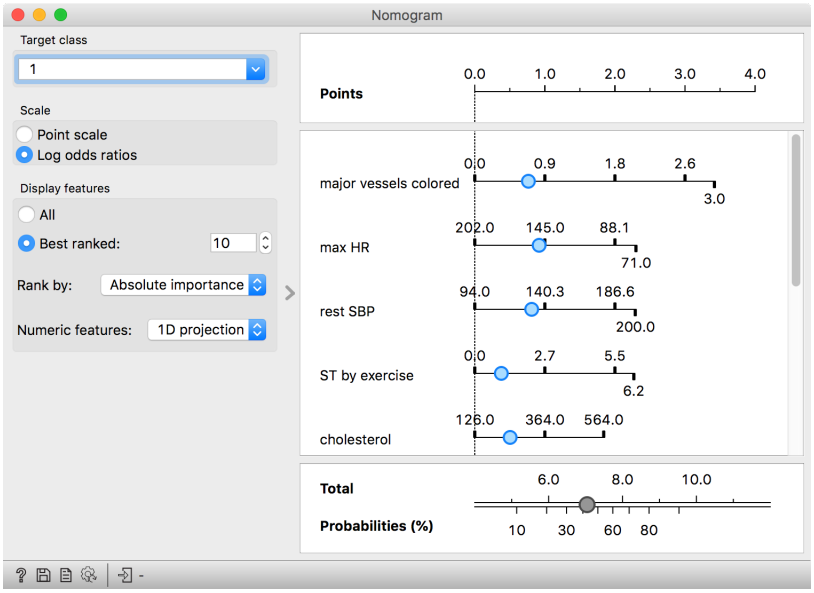


Logistic regression is one of the best-known classifiers. The model returns the probability of a class variable, based on input features. First, it computes probabilities with a one-versus-all approach, meaning that for a multiclass problem, it will take one target value and treat all the rest as "other", effectively transforming the problem to binary classification.

Second, it tries to find an optimal plane that separates instances with the target value from the rest. Then it uses logistic function to transform the distance to the plane into probabilities. The further away from the plane an instance will be, the higher the probability it belongs to the class on that side of the plane. The closer it is to the decision boundary (the plane), the more uncertain the prediction becomes (i.e. it gets close to 0.5).

Logistic regression tries to find such a plane that all points from one class are as far away from the boundary (in the correct direction) as possible.

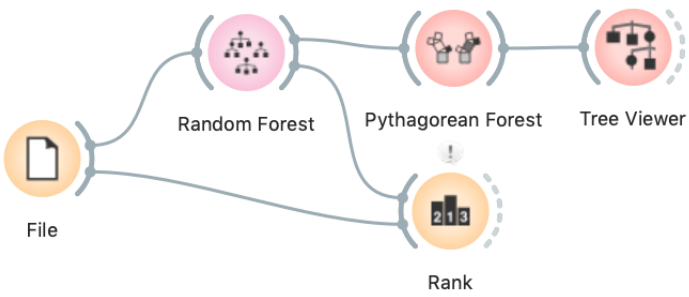
A great thing about *Logistic Regression* is that we can interpret it with a *Nomogram*. Nomogram shows the importance of variables for the model. The higher the variable is in the list, the greater its importance. Also, the longer the line, the greater the importance. The line corresponds to the coefficient of the variable, which is then mapped to the probability. You can drag the blue point on the line left or right, decreasing or increasing the probability of the target class. This will show you how different values affect the outcome of the model.



Another characteristic of logistic regression is that it observes all variables at once and takes the correlation into account. If some variables are correlated, their importance will be spread among them.

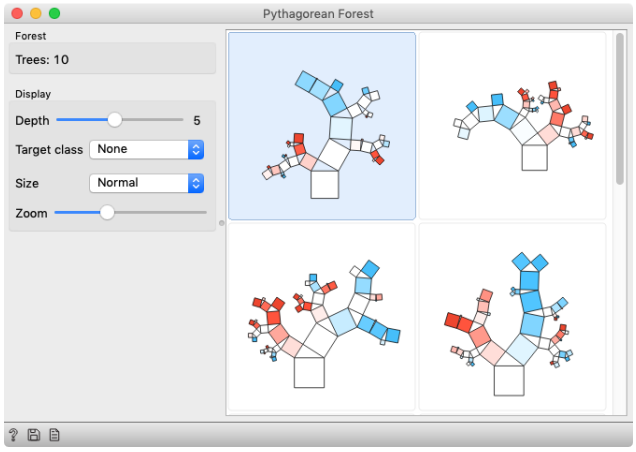
A not so great thing about logistic regression is that it operates with planes, meaning that the model won't work when the data cannot be separated in such a way. Can you think of such a data set?

Random Forests



Random forests, a modeling technique introduced in 2001, is still one of the best performing classification and regression techniques. Instead of building a tree by always choosing the one feature that seems to separate best at that time, it builds many trees in slightly random ways. Therefore the induced trees are different. For the final prediction the trees vote for the best class.

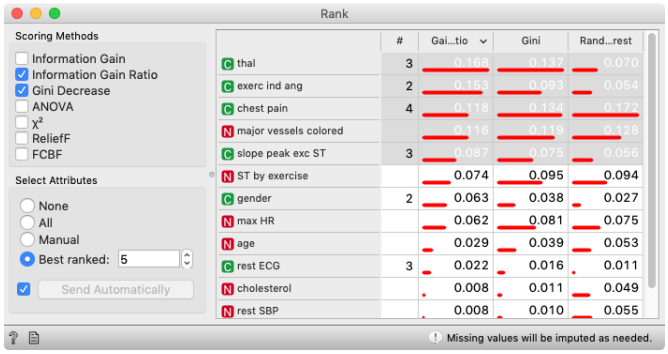
The *Pythagorean Forest* widget shows us how random the trees are. If we select a tree, we can observe it in a *Tree Viewer*.



There are two sources of randomness: (1) training data is sampled with replacement, and (2) the best feature for a split is chosen among a subset of randomly chosen features.

Which features are the most important? The creators of random forests also defined a procedure for computing feature importances from random forests. In Orange, you can use it with the *Rank* widget.

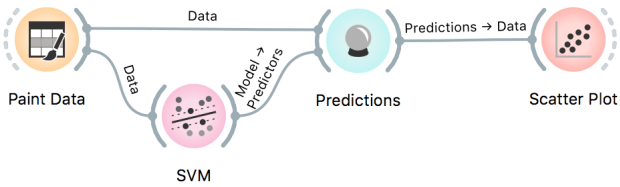
Feature importance according to two univariate measures (gain ratio and gini index) and random forests. Random forests also consider combinations of features when evaluating their importance.



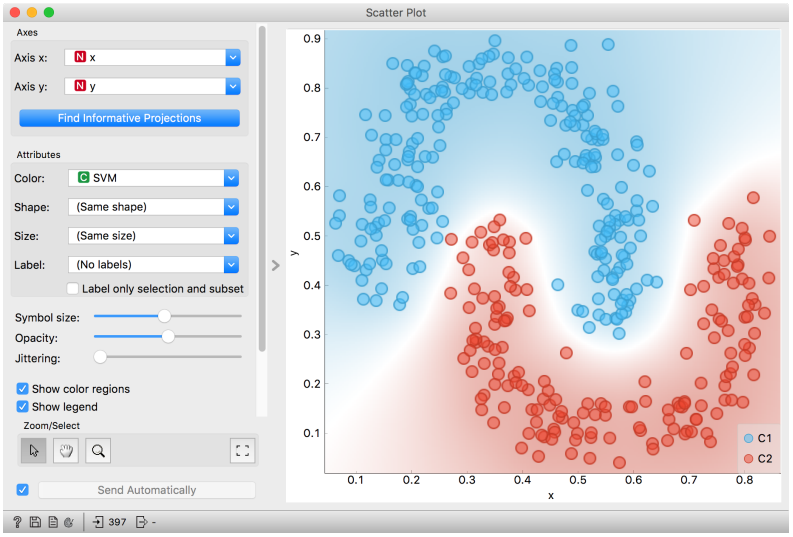
Support Vector Machines

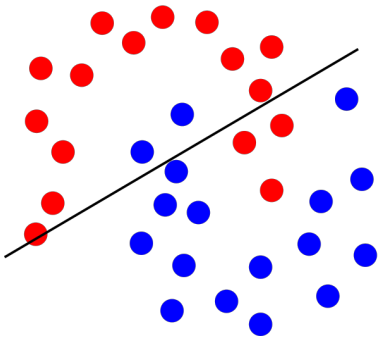
Support vector machines (SVM) are another example of linear classifiers, similar to logistic or linear regression. However, SVM can overcome splitting the data by a plane by using the so-called *kernel trick*. This means the hyperplane (decision boundary) can be transformed to a higher-dimensional space, which can fit the data nicely. In such a way, SVM becomes a non-linear classifier and can fit more complex data sets.

The magic of SVM (and other methods that can use kernels, and are thus called kernel methods) is that they will implicitly find a transformation into a (usually infinite-dimensional) space, in which the distances between objects are such as prescribed by the kernel, and draw a hyperplane in this space.

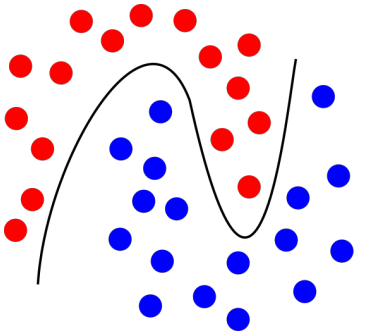


Abstract talking aside, SVM with different kernels can split the data not by ordinary hyperplanes, but with more complex curves. The complexity of the curve is decided by the kernel type and by the arguments given to the algorithm, like the degree and coefficients, and the penalty for misclassifications.



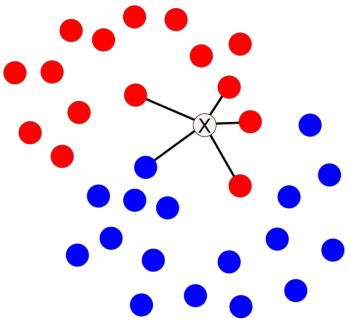


Decision boundary of a linear regression classifier.



Decision boundary of a support vector machine classifier with an RBF kernel.

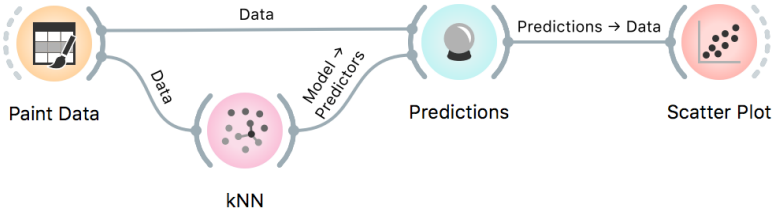
k-Nearest Neighbors



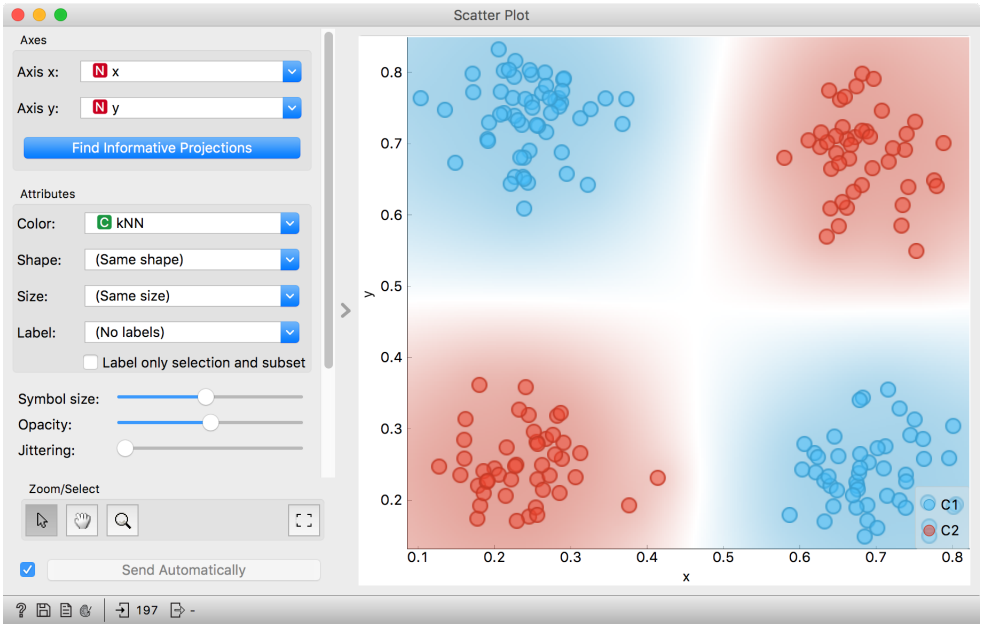
kNN classifier looks at k nearest neighbors, say 5, of instance X. 4 neighbors belong to the red class and 1 to the blue class. X will thus be classified as red with 80% probability.

The idea of k-nearest neighbors is simple - find k instances that are the most similar to each data instance. We make the prediction or estimate probabilities based on the classes of these k instances. For classification, the final label is the majority label of k nearest instances. For regression, the final value is the average value of k nearest instances.

Unlike most other algorithms, kNN does not construct a model but just stores the data. This kind of learning is called *lazy learning*.



The advantage of kNN algorithm is that it can successfully model the data, where classes are not linearly separably. It can also be retrained quickly, because new data instances effect model only locally. However, the first training is can be slow for large data sets, as the model has to estimate k distances for data instance.



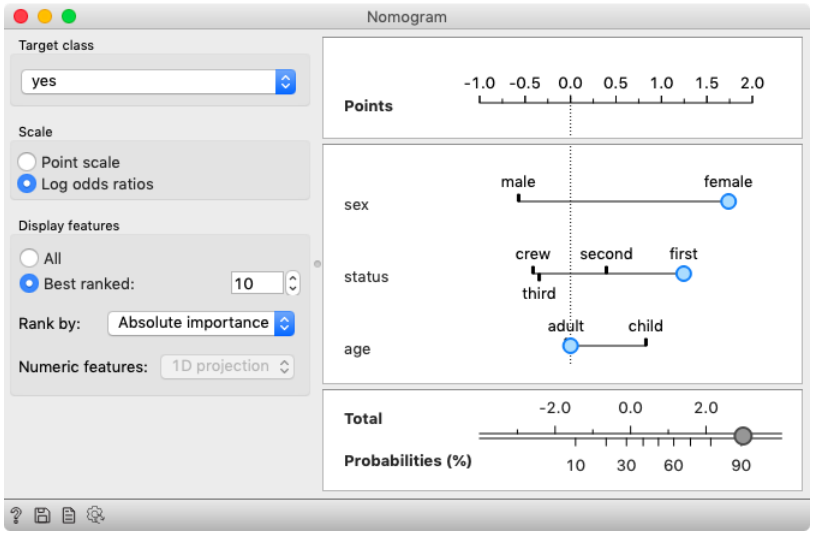
Naive Bayes

Naive Bayes is also a classification method. To see how naive Bayes works, we will use a data set on passengers' survival in the Titanic disaster of 1912. The *Titanic* data set describes 2201 passengers, with their tickets (first, second, thirds class or crew), age and gender.

Naive Bayes assumes class-wise independent features. For a data set where features would actually be independent, which rarely happens in practice, the naive Bayes would be the ideal classifier.

We inspect naive Bayes models with the *Nomogram* widget. There, we see a scale 'Points' and scales for each feature. Below we can see probabilities. Note the 'Target class' in upper left corner. If it is set to 'yes', the widget will show the probability that a passenger survived.

The nomogram shows that gender was the most important feature for survival. If we move the blue dot to 'female', the survival probability increases to 73%. Furthermore, if that woman also travelled in the first class, she survived with probability of 90%. The bottom scales show the conversion from feature contributions to probability.



According to the probability theory individual contributions should be multiplied. Nomograms get around this by working in a log-space: a sum in the log-space is equivalent to multiplication in the original space. Therefore nomograms sum contributions (in the log-space) of all feature values and then convert them back to probability.