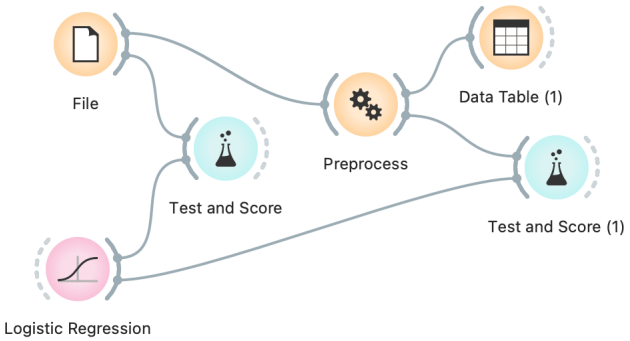
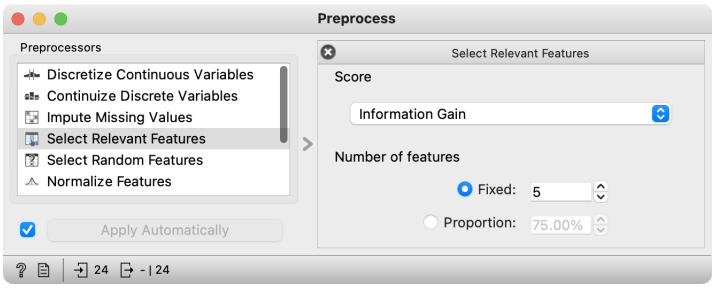
Cheating with Feature Subset Selection

We will borrow a gene expression data set from Gene Expression Omnibus for our example. There is a particular widget in the Orange bioinformatics add-on that we could use to fetch this and similar data sets. Instead, we will rely on the GEO data set gds360 available at `http://file.biolab.si/datasets/gds360.pkl`.

Consider a typical gene expression data set with samples in rows and genes expressions in columns. These data sets are usually fat; they include more genes than samples. Fat data sets are almost typical for systems biology. When we label the samples with phenotype, and our task is phenotype classification, many features (genes) will be irrelevant. Most often, only a few features correlate with class. So why not simply select a set of most informative features first and then do the whole analysis? At least cross-validation will then work much faster, as the model inference algorithms will deal with much smaller data tables. Cool. What a nice trick! Let's try it out in the following workflow.



The workflow above uses the data preprocessing widget, which we have configured to select the five most informative features.



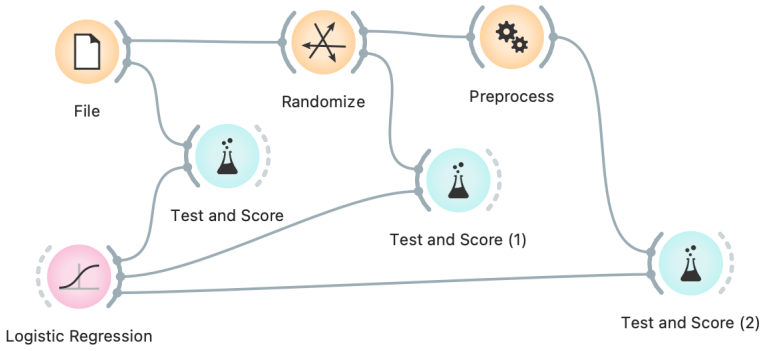
Observe the classification accuracy obtained on the original data set and the data set with the five best-selected features. What is happening? What is there such difference in classification accuracy?

Cheating Even Works on Randomized Data

We can push the example from our previous lesson to the extreme. We will randomize the classification data. We will take the column with the class values and randomly permute it. We will use the Randomize widget to do this.

Later, we will classify this data set. We expect low classification accuracy on randomized data set. Then, we will select five features that are most associated with the class. Even though we randomly permuted the classes, there have to be some features that are weakly correlated with the class. Simply because we have tens of thousands of features, and we have only a few samples. There are enough features to associate with class simply by chance. Finally, we will score a random forest on a randomized data set with selected features.

Compare the scores reported by cross-validation on different data sets in this pipeline. Why is the accuracy in the final one relatively high? Would adding more "most informative features" improve or degrade the cross-validated performance on a randomized data set?



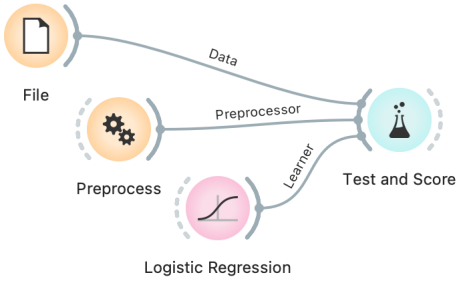
Instead of selecting the five most informative features, you can further reduce this number. Say, to two most informative features. What happens? Why does accuracy rise after this change?

How to Correctly Perform Test and Score?

To put it simply: never, in any way, transform the data before cross-validation. Any transformation should happen within the cross-validation loop, first on the training set and, if required, on a test set. In a simple form: it's ok to transform the data, but we should change the data independently on the train, and the test set and the transformation on the test set should not use the information about the class value. Data imputation could be an example of such an operation, but it should be carried out separately for the train and test set and not consider classes.

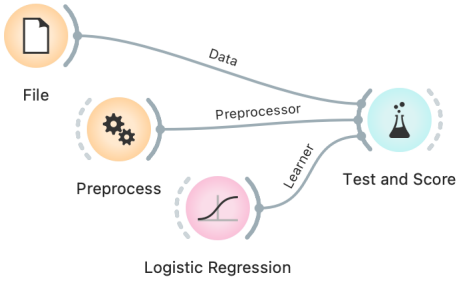
So how do we then correctly preprocess data in Orange? The idea of reducing the number of features before inferring a predictive model may still be appealing, now that we know we can use it on training data sets (leaving the test set alone). Following are two workflows that do this correctly.

The writing on the right looks straightforward. But actually one needs to be extremely careful not to succumb to overfitting when reporting results of cross-validation tests. The literature on systems biology is polluted with reporting on overly optimistic results, and high impact factors provide no guarantee that studies were carried out correctly (in fact, due to a lack of reviewers from the field of machine learning, mistakes likely stay overlooked).



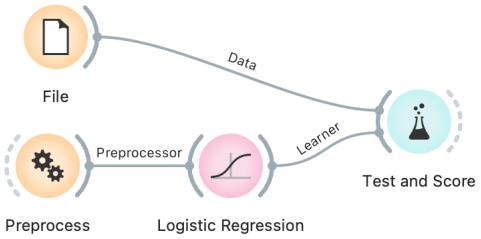
Simon et al. (2003) provides a great read on this topic. He found that many of the early papers in gene expression analysis reported high accuracy simply due to overfitting.

In this first workflow, we gave the *Test and Score* widget a preprocessor – we used feature selection in this example. The *Test and Score* uses it correctly only on the training sets. This type of workflow is preferred if we would like to test the effect of preprocessing on several different learning algorithms.



Alternatively, we can include a preprocessor in a learning method. The workflow now calls the preprocessor on the training data set just

before this learner performs inference of the predictive model.



The Preprocess widget does not necessary require a data set on its input. An alternative use of this widget is to output a method for data preprocessing, which we can then pass to either a learning method or to a widget for cross validation.

Can you extend this workflow to such an extent that you can test both a learner with preprocessing by feature subset selection and the same learner without this preprocessing? How does the number of selected features affect the cross-validated accuracies? Does the success of this particular combination of machine learning techniques depend on the input data set? Does it work better for some machine learning algorithms? Try its performance on k-nearest neighbors learner (warning: use small data sets, this classifier could be very slow).

This is not the first time we have used a widget that instead of a data passes forward a computation method. All the learners, like Random Forest, do so. A learner could get data on its input and pass a classifier to its output, or simple pass an instance of itself, that is, pass a learning algorithm to whichever widget could use it. For instance, to the Test and Score widget.

Somehow, in a shy way, we have also introduced a technique for feature selection and pointed to its possible utility for classification problems. Feature subset selection, or FSS in short, was and still is, to some extent, an essential topic in machine learning. Modern classification algorithms, though, perform it implicitly and can deal with many features without the help of external procedures for their advanced selection. Random forest is one such technique.

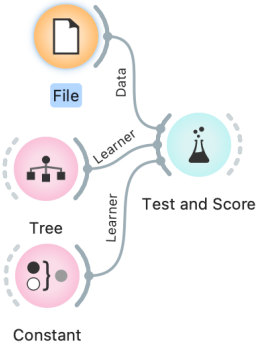
Classification Model Scoring

In multiple-choice exams, they grade you according to the number of correct answers. The same goes for classifiers: the more correct predictions they make, the better they are. Nothing could make more sense. Right?

Maybe not. Consider Dr. Smith. He is a specialist for specific diseases, and his diagnosis is correct in 98% of the cases. Would you consider visiting him if you have some symptoms related to his specialty?

Not necessarily. Dr. Smith's specialty is rare diseases. Two out of a hundred of his patients have it, and, being lazy, he always dismisses everybody as healthy. His predictions are worthless — although highly accurate. Classification accuracy is not an absolute measure, which we can judge out of context. At the very least, we have to compare it with the frequency of the majority class, which is, in the case of rare diseases, quite, hm, substantial.

For instance, on the GEO data set GDS 4182, the classification tree achieves 79% cross-validation accuracy, which may be reasonably good. Let us compare this with the Constant model, which implements Dr. Smith's strategy by always predicting the majority. It gets 83%. Classification trees are not so good after all, are they?

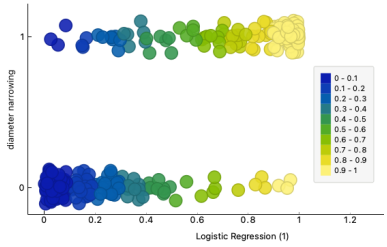


What do other columns in Test and Score widget represent? Keep reading!

Model	AUC	CA	F1	Precision	Recall
Tree	0.711	0.792	0.792	0.792	0.792
Constant	0.425	0.833	0.758	0.694	0.833

On the other hand, the accuracy of classification trees on GDS 3713 is about 71%, which seems rather good compared to the 50% achieved by predicting the majority.

Model	AUC	CA	F1	Precision	Recall
Tree	0.701	0.709	0.709	0.709	0.709
Constant	0.488	0.506	0.340	0.256	0.506



Classes versus probabilities as estimated by logistic regression. Can you replicate this image?

The problem with classification accuracy goes deeper, though. Classifiers usually make predictions based on the probabilities they compute. If a data instance belongs to class *healthy* with a probability of 80% and *disseased* with a probability of 20%, we classified it as *healthy*. Such classification makes sense, right?

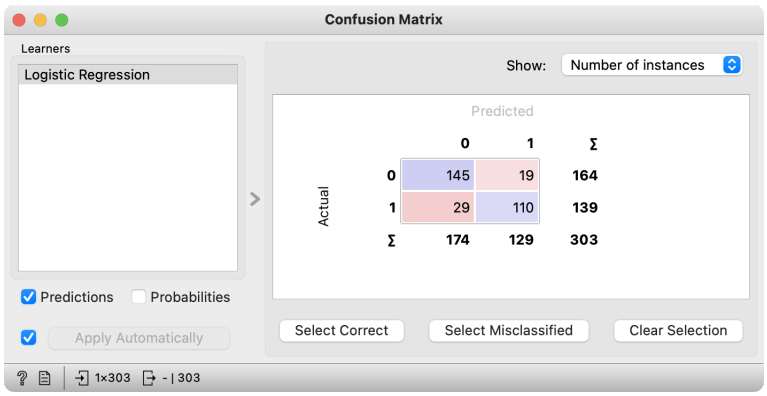
Maybe not, again. Say you fall down the stairs, and your leg hurts. You open Orange, enter some data into your favorite model and compute a 20% of having your leg broken. So you assume your leg is all right, and you take an aspirin. Or perhaps not?

What if the chance of a broken leg was 10%? 5%? 0.1%? Say we

decide that any leg with a 1% chance of being broken will be classified as broken. What will this do to our classification threshold? It is going to decrease badly — but we do not care. What do we do care about then? What kind of "accuracy" is essential?

Not all mistakes are equal. We can summarize them in the Confusion Matrix. Here is one for logistic regression on the heart disease data.

The numbers in the Confusion Matrix have names. We can classify a data instance as positive or negative; imagine this as being positive or negative when tested for some medical condition. This classification can be true or false. So there are four options, true positive (TP), false positive (FP), true negative (TN), and false negative (FN). Identify them in the table!



Logistic regression correctly classifies 145 healthy persons and 110 of the sick, the numbers on the diagonal. Classification accuracy is then 255 out of 303, which is about 84%.

Use the output from Confusion Matrix as a subset for Scatter plot to explore the data instances that were misclassified in a certain way.

Nineteen healthy people were unnecessarily scared. The opposite error is worse: the heart problems of 29 persons went undetected. We need to distinguish between these two kinds of mistakes.

We are interested in the probability that a person who has some problem will be correctly diagnosed. There were 139 such cases, and we correctly discovered 110. The proportion is $110/139 = 0.79$. This measure is called *sensitivity* or *recall* or *true positive rate* (TPR).

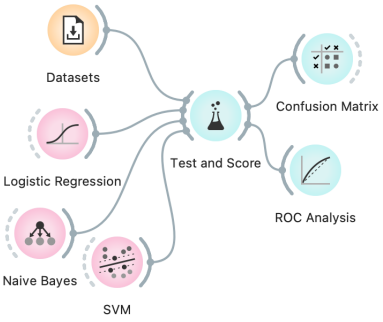
If you were interested only in sensitivity, though, here's Dr. Smith's associate partner — wanting to be on the safe side, she considers everybody ill, so she has a perfect sensitivity of 1.0.

To counterbalance the sensitivity, we compute the opposite: what is the proportion of correctly classified *negative* instances? 145 out of 164, that is, about 90%. This score is called *specificity* or *true negative* rate.

If you are interested in a complete list of statistics, see the Wikipedia page on Receiver operating characteristic, https://en.wikipedia.org/wiki/Receiver_operating_characteristic.

So, if the model classifies you as healthy, do you have a 90% chance of actually being OK? No, it's the other way around: 90% is the chance of being classified as OK if you are OK. (Think about it, it's not as complicated as it sounds). If you're interested in your chance of being OK if the classifier tells you so, you look for the *negative predictive value*. Then there's also *precision*, the probability of being positive if you are classified as such. And the *fall-out* and *negative likelihood ratio*. There is a list of other indistinguishable fancy names for various statistics, each useful for some purpose.

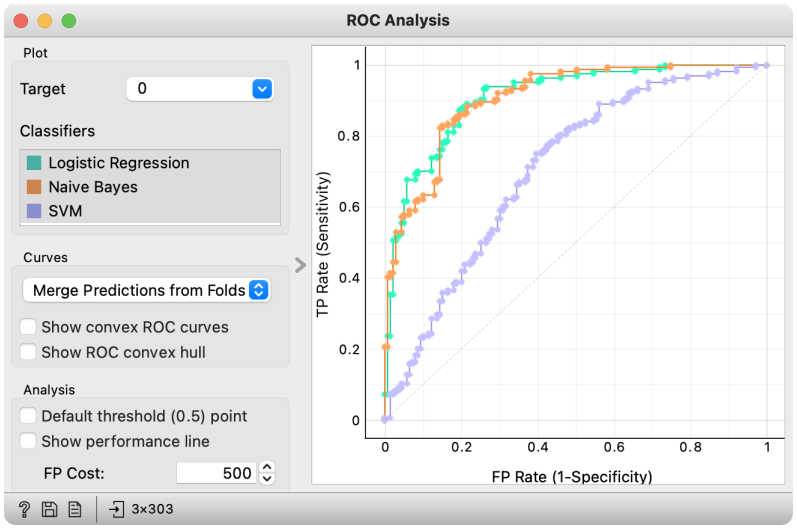
Choosing the Decision Threshold



The common property of scores from the previous lesson is that they depend on the threshold for classifying an instance as positive. We can balance between them by adjusting the threshold to find the required sensitivity at an acceptable specificity. We can even assign costs (monetary or not) to different kinds of mistakes and find the threshold with the minimal expected cost.

A valuable tool to investigate the effects of different thresholds is the Receiver-Operating Characteristic curve. Don't mind the (historical) meaning of the name and just call it the ROC curve.

Here are the curves for logistic regression, SVM with linear kernels, and naive Bayesian classifier on the same ROC plot.



Sounds complicated? If it helps: perhaps you remember the term parametric curve from some of your math classes. ROC is a parametric curve where x and y (the sensitivity and 1 - specificity) are a function of the same parameter, the decision threshold.

The curves show how the sensitivity (y-axis) and specificity (x-axis, but from right to left) change with different thresholds.

There exists, for instance, a threshold for logistic regression (the green curve) that gives us 0.65 sensitivity at the specificity of 0.95 (the curve shows 1 - specificity). Or 0.9 sensitivity with a specificity of 0.8. Or a sensitivity of (almost) 1 with a specificity of somewhere around 0.3.

The optimal point would be at the top left. The diagonal represents the behavior of a random guessing classifier.

Which of the three classifiers is the best? It depends on the specificity and sensitivity we want; at some points, we prefer logistic regression and, at other points, the naive bayesian classifier. SVM doesn't cut it anywhere.

A popular score derived from the ROC curve is called an *area under curve*, or AUC for short. It measures, well, the area under the curve. ROC curve. If the curve goes straight up and then right, the area is

1; we can not reach optimal AUC in practice. If the classifier guesses at random, the curve follows the diagonal, and AUC is 0.5. Anything below that is equivalent to guessing or bad luck.

AUC has a kind of absolute scale. As a rule of thumb: 0.6 is bad, 0.7 is bearable, 0.8 is publishable, and anything above 0.95 is suspiciously good.

AUC also has an excellent probabilistic interpretation. Say that we are given two data instances, and we are told that one is positive and the other is negative. We use the classifier to estimate the probabilities of being positive for each instance and decide that the one with the highest probability is positive. It turns out that the probability that such a decision is correct equals the AUC of this classifier. Hence, AUC measures how well the classifier discriminates between the positive and negative instances.

From another perspective: if we use a classifier to rank data instances, then AUC of 1 signifies a perfect ranking, an AUC of 0.5 a random ranking, and an AUC of 0 an ideal reversed ranking.