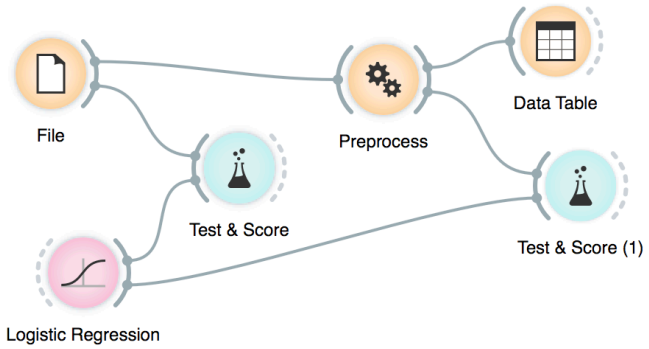
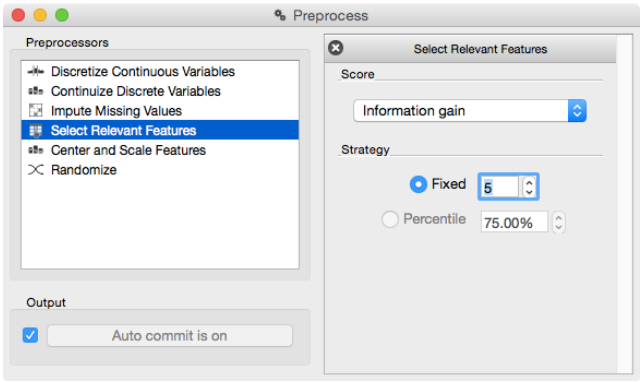
Lesson 21: More Cheating

Gene expression data set we will use was borrowed from Gene Expression Omnibus. There is a special widget in Orange bioinformatics add on that we could use to fetch this and similar data sets. Instead, we will here rely on GEO data set gds360 available at http://file.biolab.si/datasets/gds360.pkl.

Consider a typical gene expression data set where we have samples in rows and genes expressions in columns. These data sets are usually fat: there are many more genes than samples. Fat data sets are almost typical for systems biology. When samples are labeled with phenotype and our task is phenotype classification, many features (genes) will be irrelevant and most often only a few will be highly correlated with class. So why not simply first select a set of most informative features, 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 5 most informative features.



Observe the classification accuracy obtained on the original data set, and on the data set with five best selected features. What is happening? Why?

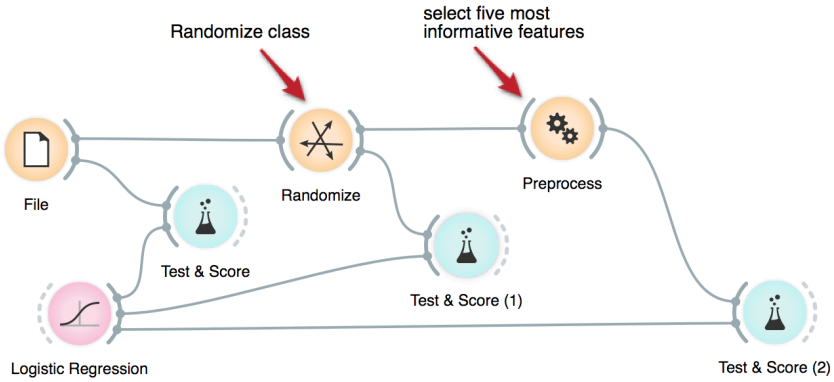
Lesson 22: Cheating Works Even on Randomized Data

We can push the example from our previous lesson to the extreme. We will randomize the classification data. That is, 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 do classification on this data set. We expect really low classification accuracy on randomized data set. Then, we will select five features that are most associated with the class. Even though we have 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 that some of them correlate 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 rather high? Would adding more "most informative features" improve or degrade the cross-validated performance on a randomized data set?

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



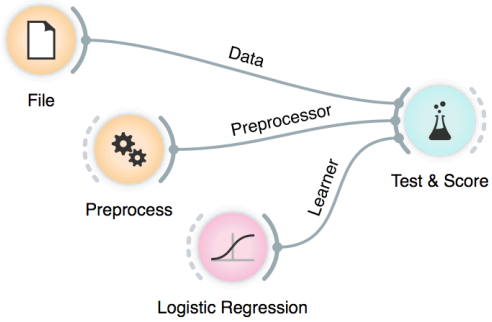
Lesson 23: How to Correctly Perform Test and Score

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.

To put it simply: never, in any way, transform the data prior to cross-validation. Any transformation should happen within cross-validation loop, first on the training set, and then, if required, on a test set. In a relaxed form: it's ok to transform the data, but the transformation should be done independently on the train and the test set and the transformation on the test set should in no way use the information about the class value. Data imputation could be an example of such operation, but again it should be carried out separately for the train and test set and should not consider classes.

But how do we then correctly apply preprocessing in Orange? The idea of reducing the number of features prior to inferring a predictive model may be still 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.

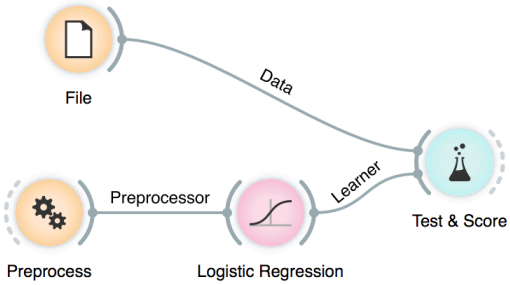


In this first workflow, we gave the Test & Score widget a preprocessor (feature selection was used in this example). The Test & Score widget uses it correctly only on the training sets. This type of workflow is preferred if we would like to test the effect of preprocessing on a number of different learning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

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.

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 & Score widget.

Alternatively, we can include a preprocessor in a learning method. The preprocessor is now called on the training data set just before this learner performs inference of the predictive model.



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 technique 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).

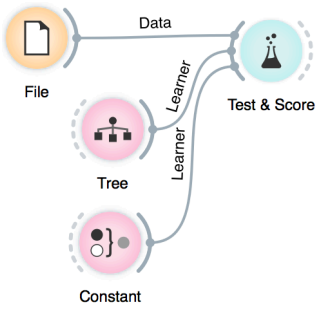
Somehow, in a shy way, we have also introduced a technique for feature selection, and pointed to its possible utility for classification problems. Feature subset selection, or FSS in short, was and still is, to some extent, an important topic in machine learning. Modern classification algorithms, though,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

Lesson 24: Model Scoring

In multiple choice exams, you are graded 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. Dr. Smith is a specialist of a type and his diagnosis is correct in 98% of the cases. Would you consider visiting him if you have some symptoms related to his speciality?

Not necessarily. His specialty, in fact, are rare diseases (2 out of 100 of his patients have it) and, being lazy, he always dismisses everybody as healthy. His predictions are worthless — although extremely accurate. Classification accuracy is not an absolute measure, which can be judged out of context. At the very least, it has to be compared with the frequency of the majority class, which is, in case of rare diseases, quite ... major.

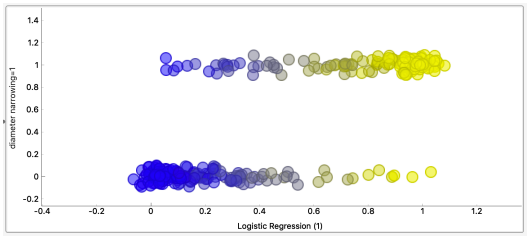


For instance, on GEO data set GDS 4182, the classification tree achieves 79% accuracy on cross validation, which may be reasonably good. Let us compare this with the Constant model, which implements Dr. Smith's strategy by always predicting the majority. It gets 83%. Classification trees are not so good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

On the other hand, their accuracy on GDS 3713 is 72%, which seems rather good in comparison with the 50% achieved by predicting the majority.

What do other columns represent? Keep reading!

Model	AUC	CA	F1	Precision	Recall
Tree	0.703	0.722	0.720	0.726	0.722
Constant	0.488	0.506	0.340	0.256	0.506



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

The problem with classification accuracy goes deeper, though.

Classifiers usually make predictions based on probabilities they compute. If a data instance belongs to class A with a probability of 80% and to B with a probability of 20%, it is classified as A. This 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 not broken and you take an aspirin. Or perhaps not?

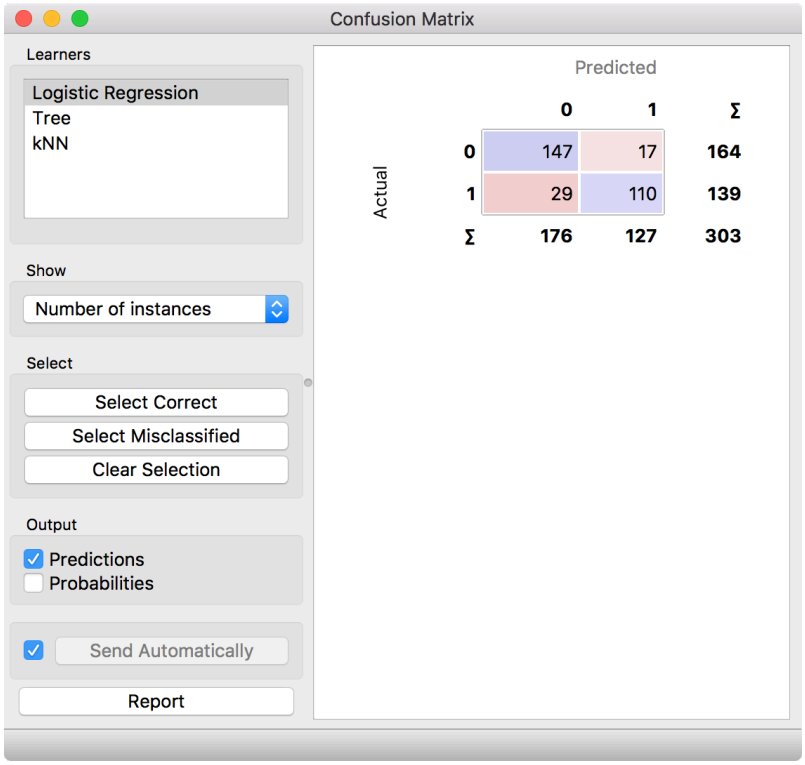
What if the chance of a broken leg was just 10%? 5%? 0.1%?

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 apparently do not care. What do we do care about then? What kind of "accuracy" is important?

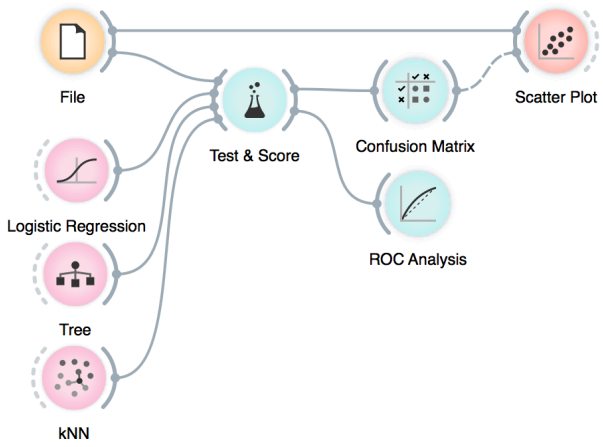
Not all mistakes are equal. We can summarize them in the Confusion Matrix. Here is one for logistic regression on the heart disease data.

These numbers in the Confusion Matrix have names. An instance can be classified as positive or negative; imagine this as being positive or negative when being 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!



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



Logistic regression correctly classifies 147 healthy persons and 110 of the sick, the numbers on the diagonal. Classification accuracy is then 257 out of 303, which is 85%.

17 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 110 were discovered. 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? 147 out of 164, that is, 90%. This is called *specificity* or *true negative rate*.

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

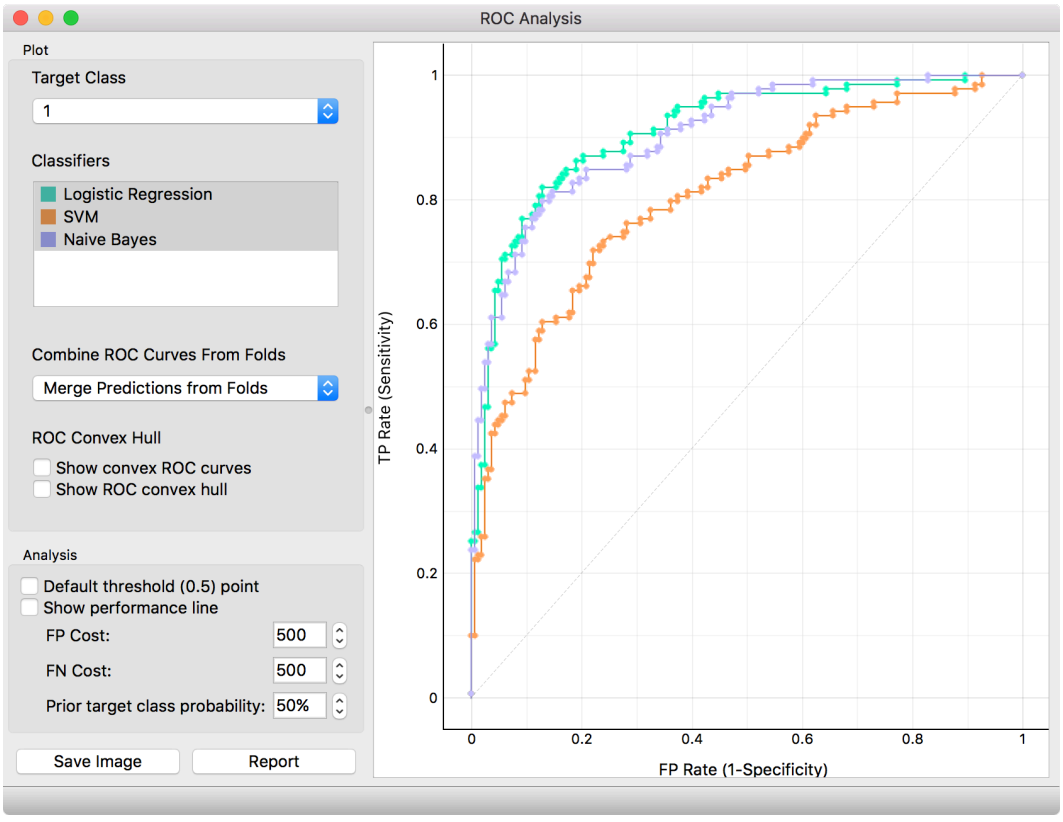
So, if you're classified as OK, you have a 90% chance of actually being OK? No, it's the other way around: 90% is the chance of being classified as OK, if you are OK. (Think about it, it's not as complicated as it sounds). If you're interested in your chance of being OK if the classifier tells you so,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

Lesson 25: Choosing the Decision Threshold

The common property of scores from the previous lesson is that they depend on the threshold we choose for classifying an instance as positive. By adjusting it, we can balance between them and find, say, the threshold that gives us 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 useful tool for this is the Receiver-Operating Characteristic curve. Don't mind the meaning of the name, 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.



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

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.

There exists, for instance, a threshold for logistic regression (the green curve) that gives us 0.65 sensitivity at 0.95 specificity (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 top left. The diagonal represents the behavior of a random guessing classifier.

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

There is a popular score derived from the ROC curve, called Area under curve, AUC. It measures, well, the area under the curve. This curve. If the curve goes straight up and then right, the area is 1; such an optimal AUC is not reached 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 + bad luck.

AUC has a kind of absolute scale. As a rule of a thumb: 0.6 is bad, 0.7 is bearable, 0.8 is publishable and 0.9 is suspicious.

ROC curves and AUC are fascinating tools. To learn more, read T. Fawcett: ROC Graphs: Notes and Practical Considerations for Researchers

AUC also has a nice 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 a perfect reversed ranking.