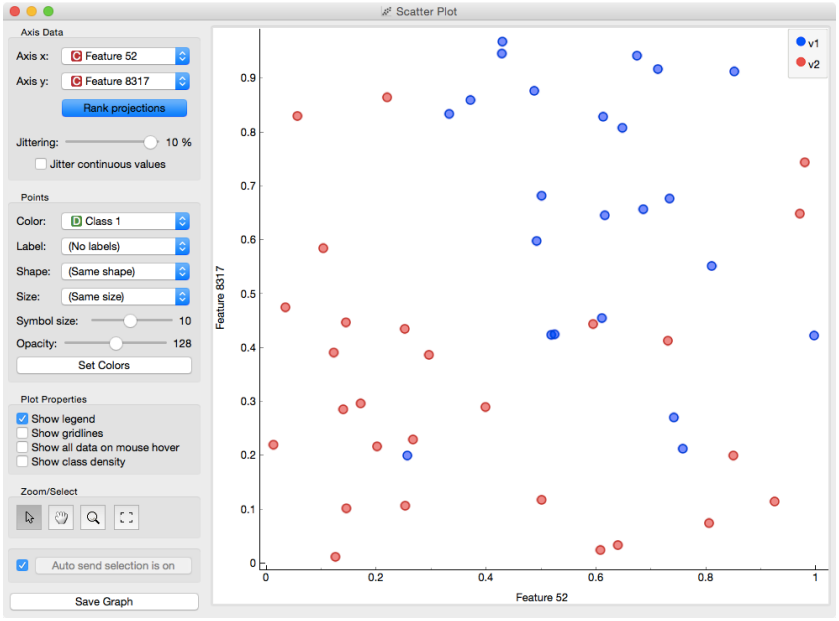
Lesson 17: Another Overfitting Demo

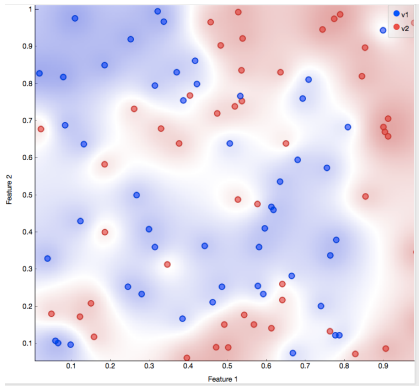
Download the file with a random data set from http://bit.ly/1EEB4FX

It is worth repeating: you should never select features or perform any other data preprocessing (binning, noise filtering, normalization...) on the data you will later use, in part, for testing. These operations are part of the analysis and can be carried out only on the training data set.

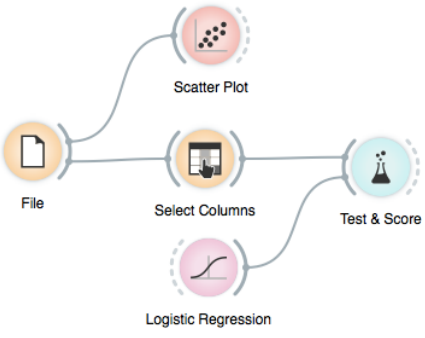
Here is a simple illustration why this is wrong. Consider a random data set with 100 instances, described by 10,000 features and assigned randomly to any of the two classes.



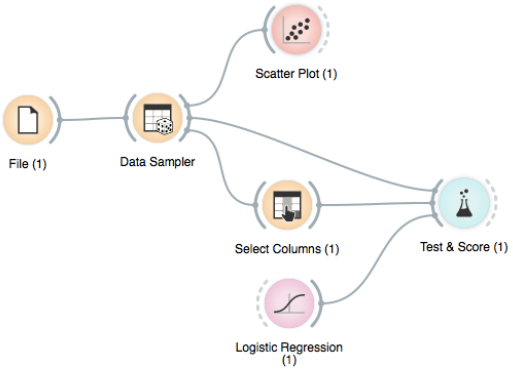
It is amazing, though, how quickly we spot random patterns. Adding a bit of color provides a "convincing argument" for the existence of "blue" and "red" regions in this plot.

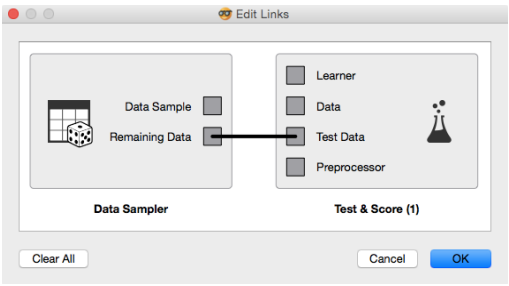


Although the data is random, there are enough features that some are randomly correlated with the class. We have previously used the Rank widget to find them. To make our illustration more illustrative, we will now use Rank projections to find a good a pair of features. Projection ranking discovers that, among few other feature pairs, features 52 and 8317 almost perfectly split the data into a red and blue region.



We use the Select column widget to remove all features but these two; the entire data set is now reduced to the above plot. We run, say, Logistic regression through 5-fold cross validation and get excellent results.





Consider what we've done. We used projection ranking to find a pattern in the random data. We reduced the whole data set to this pattern and then used 5-fold cross validation to verify if logistic regression is able to separate the classes. Of course it is! This is exactly why we have selected the two features in the first place.

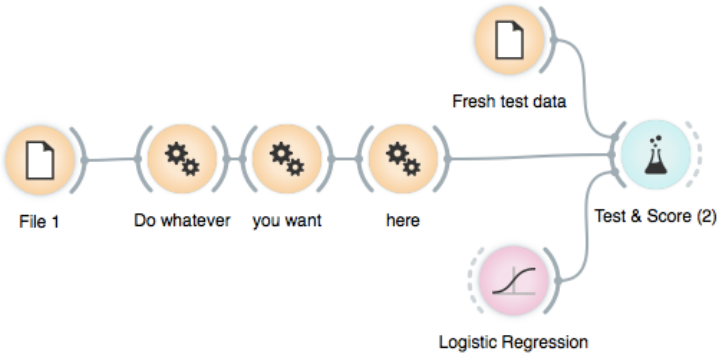
Now we will try to pull this off under the limitations of proper procedure: we first store away some data (say one fifth, 10 instances) for testing. What is left is the training data, on which we use projection rank to again find the best two features. This time, we get features 52 and 4870; as before, we select them in the Select Columns widget. We induce a classifier from this data. But - here's a difference: we don't test using cross validation on preprocessed data, but take the left out data instead. We use the Remaining data from the Data Sampler in the Test Data in Test & Score widget. We also have to choose the option Test on test data in the Test & Score widget.

Logistic regression no longer works as well. Since the data is random, the ten points that were left out and were not shown to projection ranking can (and do) lie just anywhere in the plot.

The bulk of the work for pattern finding in these examples is done by projection ranking, not by logistic regression.

In the second example, we have not used cross validation since it would require us to perform projection ranking and manually change the Select Columns' settings for each fold — our manual work would thus become part of the loop. In the example from the previous lesson, feature selection was automated, so we could include it within cross validation.

As a side note, having a separate test set — say a golden standard — is quite common. A typical schema that uses it would look like this.

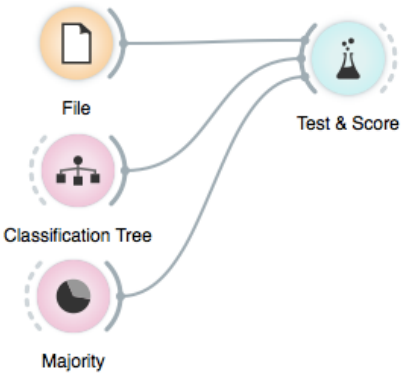


Lesson 18: Model Scoring

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.

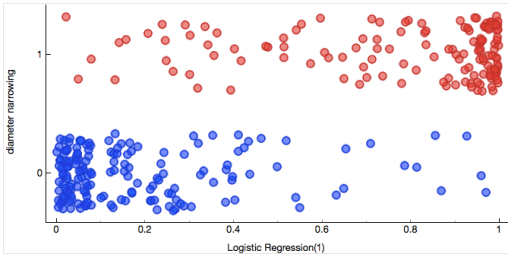


For instance, on GEO data set GDS 4182, the classification tree achieves 78% accuracy on cross validation, which may be reasonably good. Let us compare this with the Majority learner, which implements Dr. Smith's strategy by always predicting the majority. It gets 83%. Classification trees are not so good after all, are they?

On the other hand, their accuracy on GDS 3713 is 57%, which seems rather good in comparison with the 50% achieved by predicting the majority.

Method	AUC	CA	F1	Precision	Recall
Classification Tree	0.573	0.570	0.585	0.571	0.600
Majority	0.500	0.506	0.672	0.506	1.000

What do other columns represent? Keep reading!



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%?

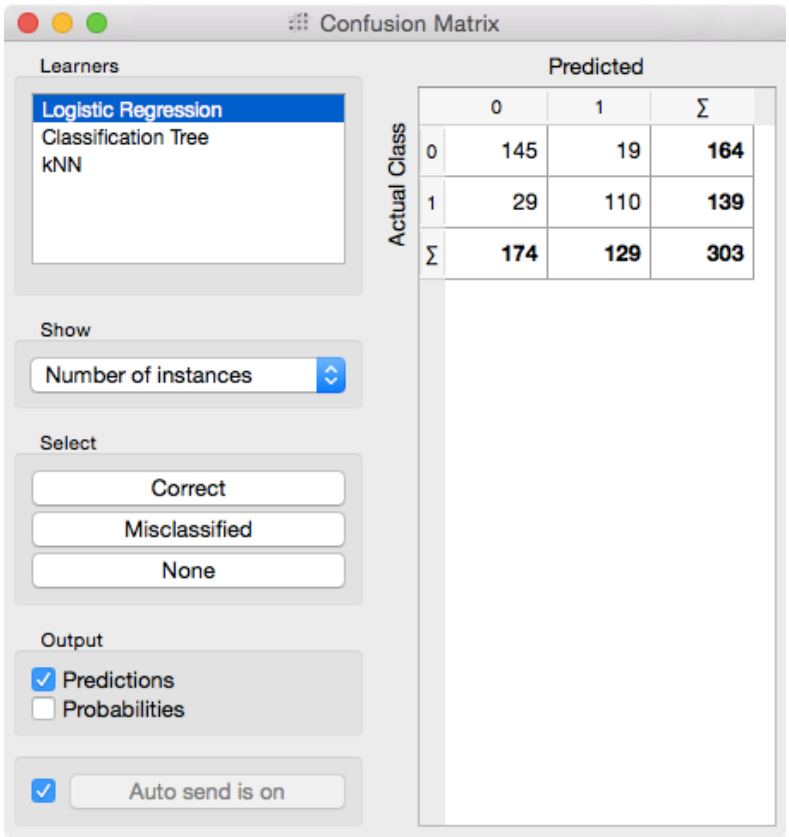
Say we decide that any leg with a 1% chance of being broken will be classified as broken. What will this do to our classification threshold? It is going to decrease badly — but we apparently do not care. What do we do care about then? What kind of "accuracy" is important?

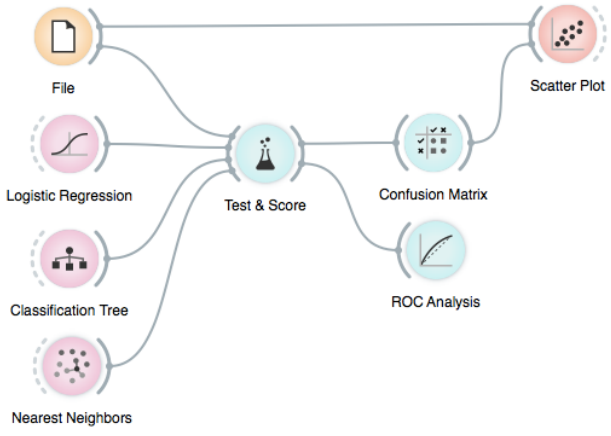
Not all mistakes are equal. We can summarize them in the Confusion Matrix. Here is one for logistic regression on the heart disease data.

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 145 healthy persons and 110 of the sick, the numbers on the diagonal. Classification accuracy is then 255 out of 303, which is 84.2%.

19 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? 145 out of 164, that is, 88%. 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

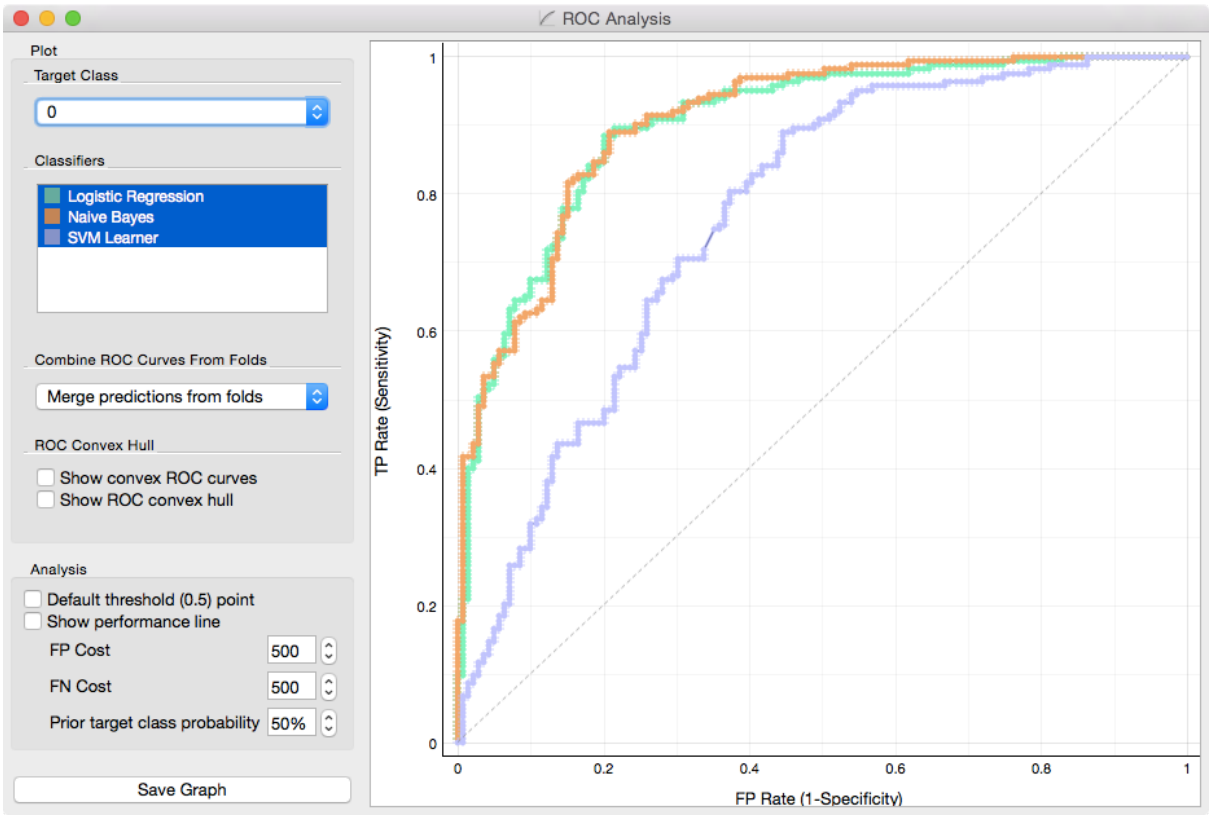
So, if you're classified as OK, you have a 88% chance of actually being OK? No, it's the other way around: 88% is the chance of being classified as OK, if you are OK. (Think about it, it's not as complicated as it sounds). If you're interested in your chance of being OK if the classifier tells you so, 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 19: 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 (another method that looks for the optimal hyperplane that separates the classes) 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 0.9 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. If the curve goes straight up and then right, the area is 1; this is optimal AUC and 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

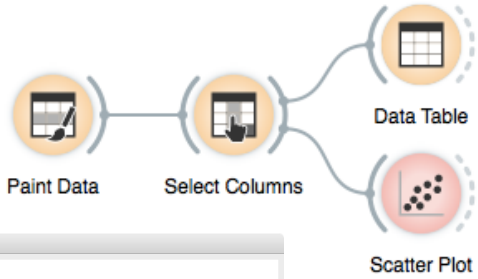
AUC also has a nice probabilistic interpretation. Say that we are given two data instances and we are told that one is positive and the other is negative. We use the classifier to estimate the probabilities of being positive for each instance, and decide that the one with the highest probability is positive. 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.

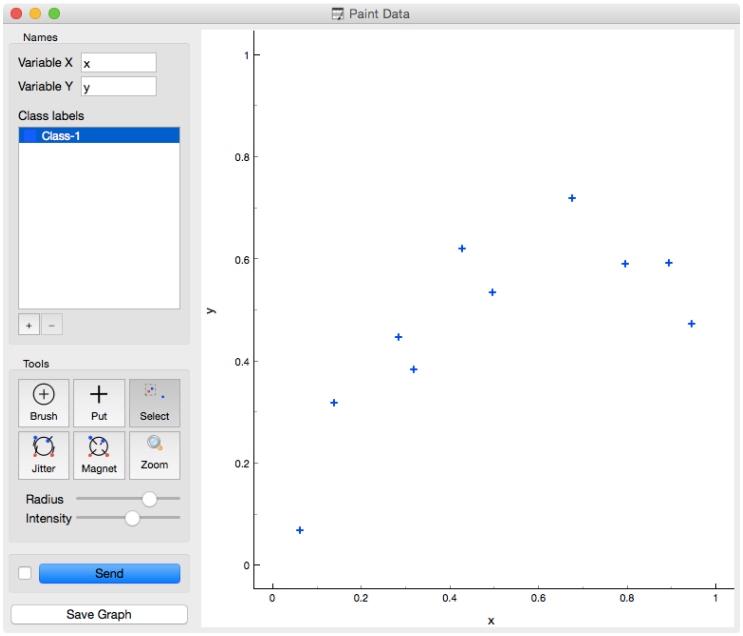
From another perspective: if we use a classifier to rank data instances, then AUC of 1 signifies a perfect ranking, an AUC of 0.5 a random ranking and an AUC of 0 a perfect reversed ranking.

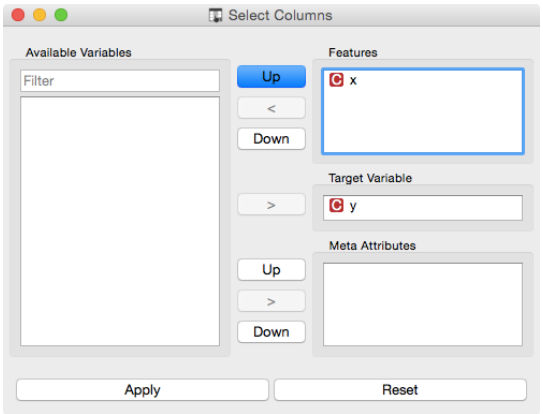
Lesson 20: Linear Regression

In the Paint Data widget, remove the Class-2 label from the list. If you have accidentally left it while painting, don't despair. The class variable will appear in the Select Columns widget, but you can "remove" it by dragging it into the Available Variables list.

For a start, let us construct a very simple data set. It will contain a just one continuous input feature (let's call it x) and a continuous class (let's call it y). We will use Paint Data, and then reassign one of the features to be a class by using Select Column and moving the feature y from the list of "Features" to a field with a target variable. It is always good to check the results, so we are including Data Table and Scatter Plot in the workflow at this stage. We will be modest this time and only paint 10 points and will use Put instead of the Brush tool.



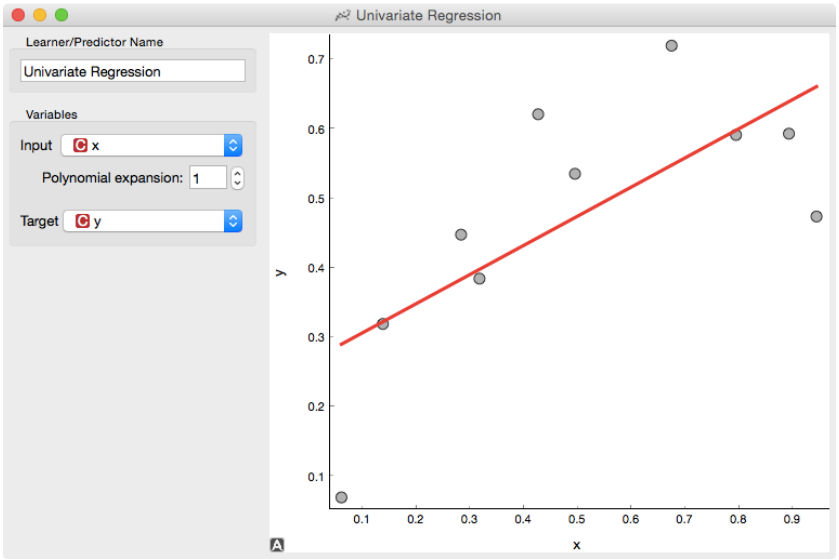




We would like to build a model that predicts the value of class y from the feature x. Say that we would like our model to be linear, to mathematically express it as $h(x)=\theta_0+\theta_1 x$. Oh, this is the equation of a line. So we would like to draw a line through our data points. The θ_0 is then an intercept, and θ_1 is a slope. But there are many different lines we could draw. Which one is the best one? Which one is the one that fits our data the most?

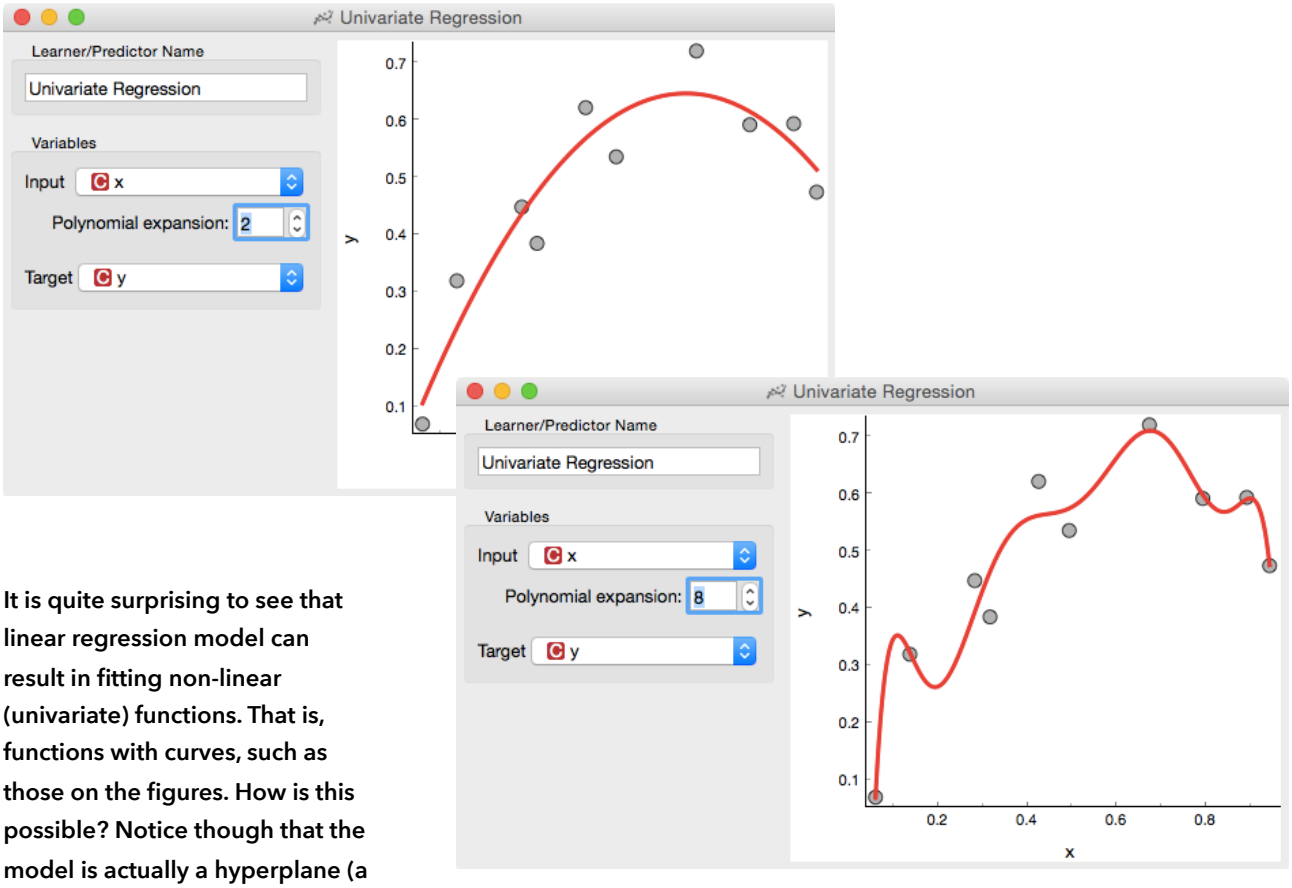
The question above requires us to define what a good fit is. Say, this could be the error the fitted model (the line) makes when it predicts the value of y for a given data point (value of x). The prediction is h(x), so the error is h(x) - y. We should treat the negative and positive errors equally, plus, let us agree, we would prefer punishing larger errors more severely than smaller ones. Therefore, it is perfectly ok if we square the errors for each data point and then sum them up. We got our objective function! Turns out that there is only one line that minimizes this function. The procedure that finds it is called linear regression. For cases where we have only one input feature, Orange has a special widget in the educational add-on called Polynomial Regression.

Do not worry about the strange name of the widget Polynomial Regression, we will get there in a moment.



Looks ok. Except that these data points do not appear exactly on the line. We could say that the linear model is perhaps too simple for our data sets. Here is a trick: besides column x, the widget Univariate Regression can add columns x^2, x^3... x^n to our data set. The number n is a degree of polynomial expansion the widget performs. Try setting this number to higher values, say to 2, and then 3, and then, say, to 9. With the degree of 3, we are then fitting the data to a linear function $h(x) = \theta_0 + \theta_1 x + \theta_1 x^2 + \theta_1 x^3$.

The trick we have just performed (adding the higher order features to the data table and then performing linear regression) is called Polynomial Regression. Hence the name of the widget. We get something reasonable with polynomials of degree 2 or 3, but then the results get really wild. With higher degree polynomials, we totally overfit our data.



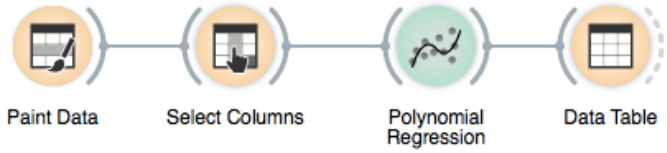
It is quite surprising to see that linear regression model can result in fitting non-linear (univariate) functions. That is, functions with curves, such as those on the figures. How is this possible? Notice though that the model is actually a hyperplane (a flat surface) in the space of many features (columns) that are powers of x. So for the degree 2, $\mathbf{h}(x)=\theta_0+\theta_1 x+\theta_1 x^2$ is a (flat) hyperplane. The visualization gets curvy only once we plot $\mathbf{h}(x)$ as a function of x.

Overfitting is related to the complexity of the model. In polynomial regression, the models are defined through parameters θ. The more parameters, the more complex is the model.

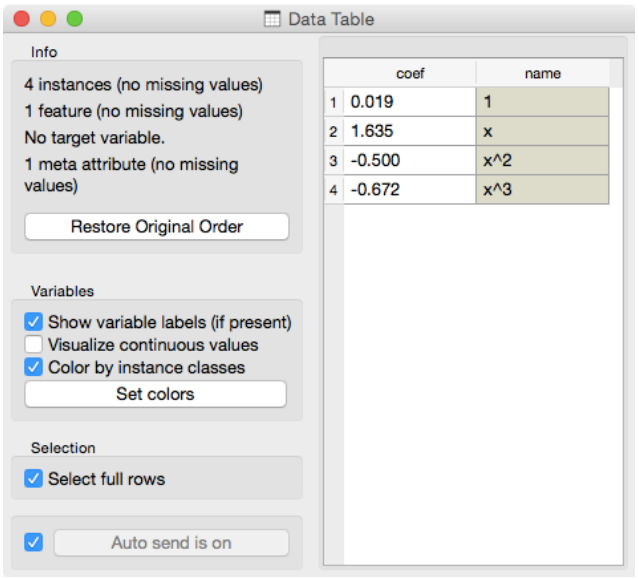
Obviously, the simplest model has just one parameter (an intercept), ordinary linear regression has two (an intercept and a slope), and polynomial regression models have as many parameters as is the degree of the polynomial. It is easier to overfit with a more complex model, as this can adjust to the data better. But is the overfitted model really discovering the true data patterns? Which of the two models depicted in the figures above would you trust more?

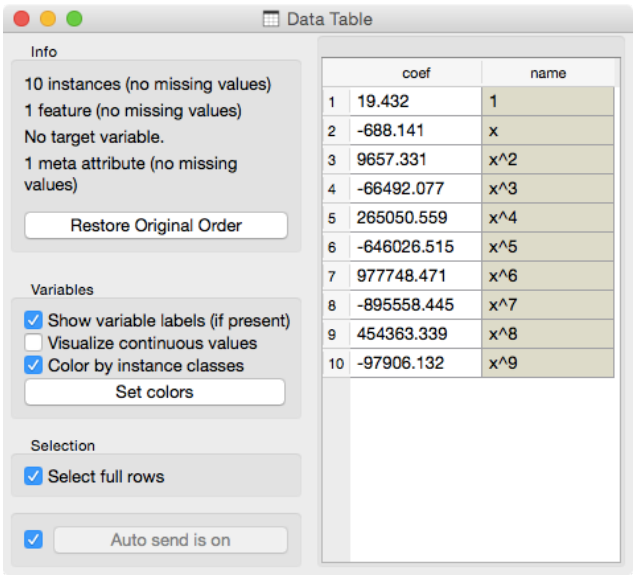
Lesson 21: Regularization

There has to be some cure for the overfitting. Something that helps us control it. To find it, let's check what the values of the parameters θ under different degrees of polynomials actually are



With smaller degree polynomials values of θ stay small, but then as the degree goes up, the numbers get really large.



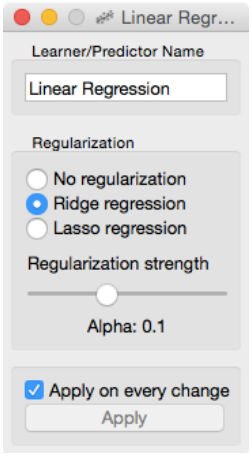


More complex models can fit the training data better. The fitted curve can wiggle sharply. The derivatives of such functions are high, and so need to be the coefficients θ. If only we could force the linear regression to infer models with a small value of coefficients. Oh, but we can. Remember, we have started with the optimization function the linear regression minimizes, the sum of squared errors. We could simply add to this a sum of all θ squared. And ask the linear regression to minimize both terms. Perhaps we should weigh the part with θ squared, say, we some coefficient λ, just to control the level of regularization.

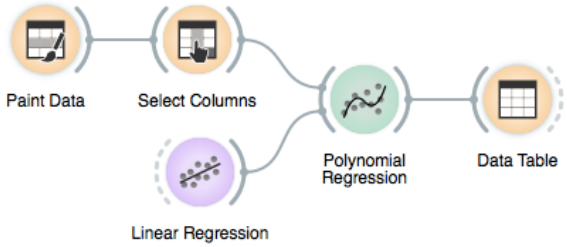
Which inference of linear model would overfit more, the one with high λ or the one with low λ? What should the value of λ be to cancel regularization? What if the value of λ is really high, say 1000?

Here we go: we just reinvented regularization, a procedure that helps machine learning models not to overfit the training data.

Internally, if no learner is present on its input, the Polynomial Regression widget would use just its ordinary, non-regularized linear regression.



To observe the effects of the regularization, we can give Polynomial Regression our own learner, which supports these kind of settings.



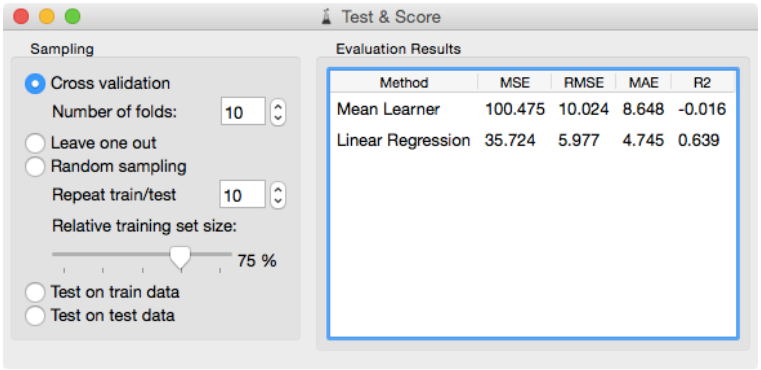
The Linear Regression widget provides two types of regularization. Ridge regression is the one we have talked about and minimizes the sum of squared coefficients θ. Lasso regression minimizes the sum of absolute value of coefficients. Although the difference may seem negligible, the consequences are that lasso regression may result in a large proportion of coefficients θ being zero, in this way performing feature subset selection.

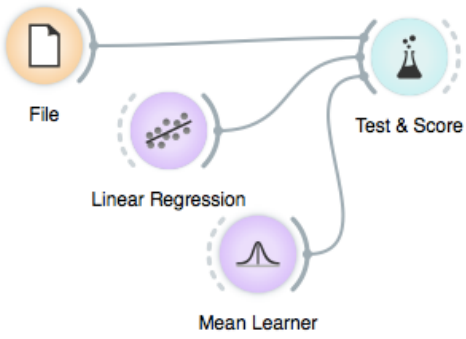
Now for the test. Increase the degree of polynomial to the max. Use Ridge Regression. Does the inferred model overfit the data? How does degree of overfitting depend on regularization strength?

Lesson 22: Prediction of Tissue Age from Level of Methylation

Download the methylation data set from http://bit.ly/2c4X3fK. Predictions of age from methylation profile were investigated by Horvath (2013) Genome Biology 14:R115.

Enough painting. Now for the real data. We will use a data set that includes human tissues from subjects at different age. The tissues were profiled by measurements of DNA methylation, a mechanism for cells to regulate the gene expression. Methylation of DNA is scarce when we are young, and gets more abundant when we age. We have prepared a data set where the degree of methylation was expressed per each gene. Let us test if we can predict age from the methylation profile, and if we can do this better than just predicting the average age of subjects in the training set.



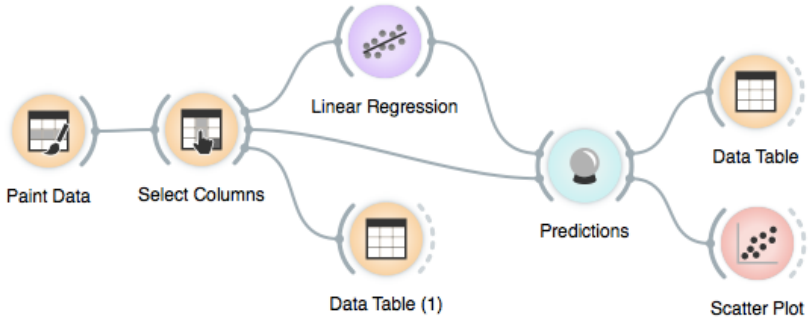


This schema looks familiar and is similar to those for classification problems. The Test & Score widget reports on statistics we have not seen before. MAE, for one, is the mean average error. Just like for classification, we have used cross-validation, so MAE was computed only on the test data instances and averaged across 10 runs of cross validation. The results indicate that our modeling technique misses the age by about 5 years, which is a much better result than predicting by a mean age in the training set.

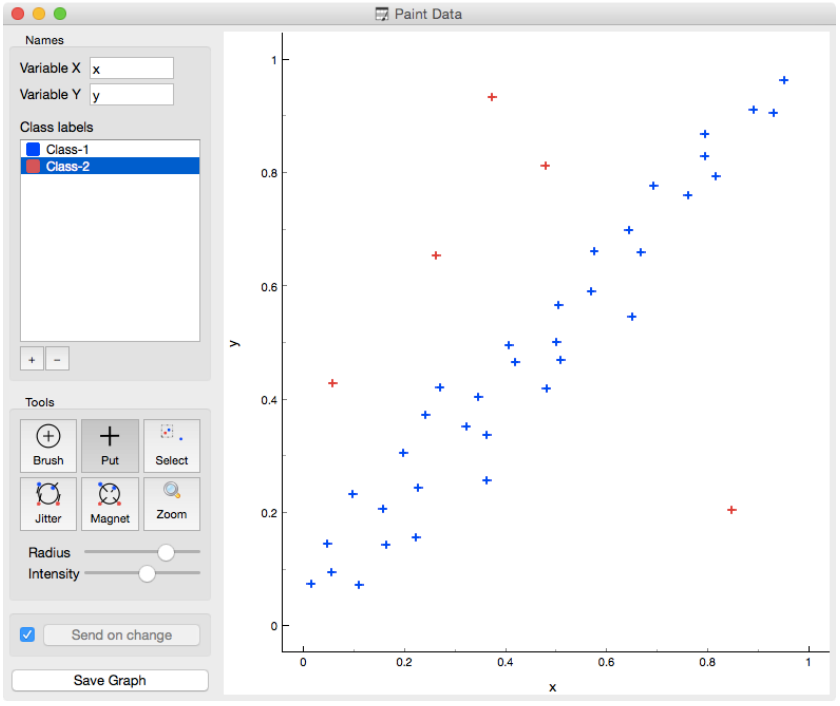
Lesson 22: Evaluating Regression

The last lesson quickly introduced scoring for regression, and important measures such as RMSE and MAE. In classification, a nice addition to find misclassified data instances was the confusion matrix. But the confusion matrix could only be applied to discrete classes. Before Orange gets some similar for regression, one way to find misclassified data instances is through scatter plot!

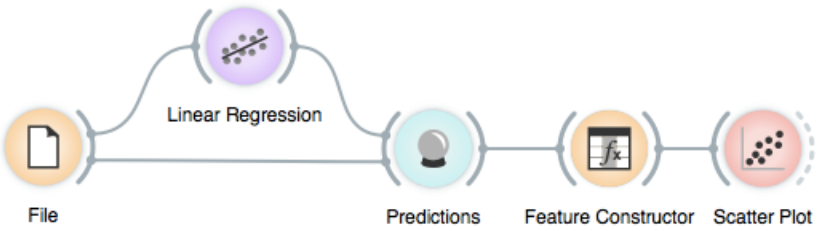
This workflow visualizes the predictions that were performed on the training data. How would you change the widget to use a separate test set? Hint: The Sample widget can help.

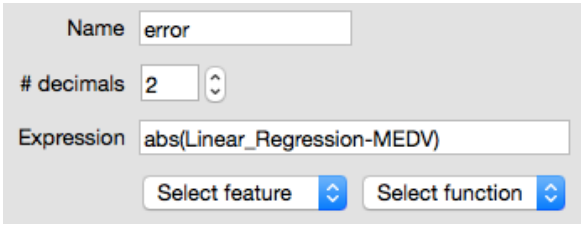


We can play around with this workflow by painting the data such that the regression would perform well on blue data point and fail on the red outliers. In the scatter plot we can check if the difference between the predicted and true class was indeed what we have expected.



A similar workflow would work for any data set. Take, for instance, the housing data set (from Orange distribution). Say, just like above, we would like to plot the relation between true and predicted continuous class, but would like to add information on the absolute error the predictor makes. Where is the error coming from? We need a new column. The Feature Constructor widget (albeit being a bit geekish) comes to the rescue.





In the Scatter Plot widget, we can now select the data where the predictor erred substantially and explore the results further.

We could, in principle, also mine the errors to see if we can identify data instances for which this was high. But then, if this is so, we could have improved predictions at such regions. Like, construct predictors that predict the error. This is weird. Could we then also construct a predictor, that predicts the error of the predictor that predicts the error? Strangely enough, such ideas have recently led to something called Gradient Boosted Trees, which are nowadays among the best regressors (and are coming to Orange soon).