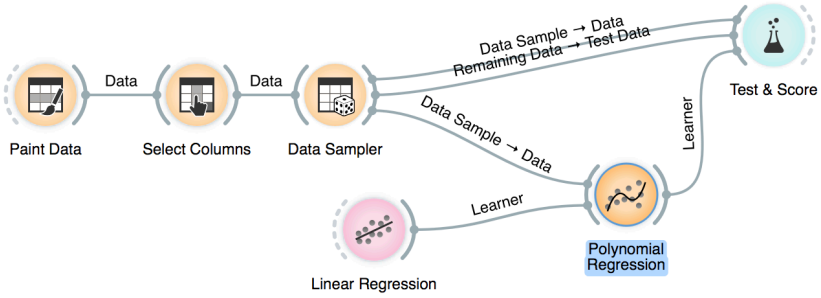
Lesson 22: Regularization and Accuracy on Test Set

Overfitting hurts. Overfit models fit the training data well, but can perform miserably on new data. Let us observe this effect in regression. We will use hand-painted data set, split it into the training (50%) and test (50%) data set, polynomially expand the training data set to enable overfitting, build a model on it, and test the model on both the (seen) training data and the (unseen) held-out data:

Paint about 20 to 30 data instances. Use attribute y as target variable in Select Columns. Split the data 50:50 in Data Sampler. Cycle between test on train or test data in Test & Score. Use ridge regression to build linear regression model.



Now we can vary the regularization strength in Linear Regression and observe the accuracy in Test & Score. For accuracy scoring, we will use RMSE, root mean squared error, which is computed by observing the error for each data point, squaring it, averaging this across all the data instances, and taking a square root.

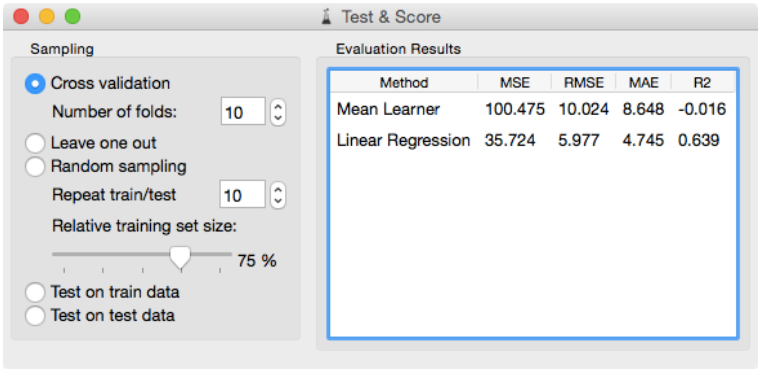
The core of this lesson is to compare the error on the training and test set while varying the level of regularization. Remember, regularization controls overfitting - the more we regularize, the less tightly we fit the model to the training data. So for the training set, we expect the error to drop with less regularization and more overfitting, and to increase with more regularization and less fitting. No surprises expected there. But how does this play out on the test set? Which sides minimizes the test-set error? Or is the optimal level of regularization somewhere in between? How do we estimate this level of regularization from the training data alone?

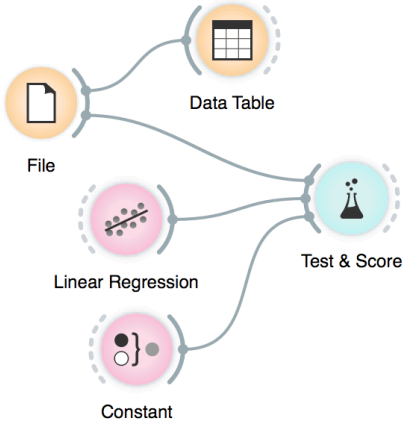
Orange is currently not equipped with parameter fitting and we need to find the optimal level of regularization manually. At this stage, it suffices to say that parameters must be found on the training data set without touching the test data.

Lesson 23: Prediction of Tissue Age from Level of Methylation

Download the methylation data set from http://file.biolab.si/files/methylation.tab. 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 as 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 the age from the methylation profile - and if we can do this better than by just predicting the average age of subjects in the training set.





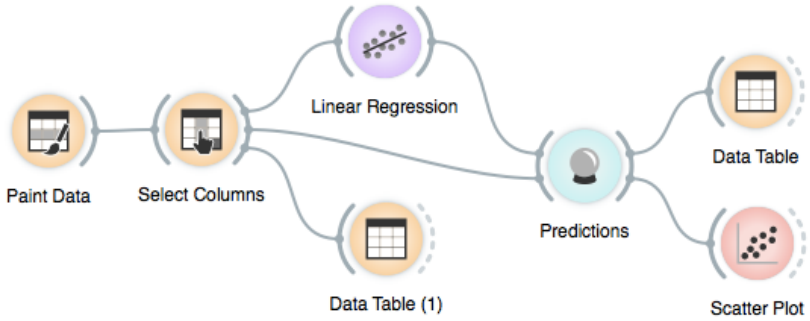
Using other learners, like random forests, takes a while on this data set. But you may try to sample the features, obtain a smaller data set, and try various regression learners.

This workflow 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 the mean age in the training set.

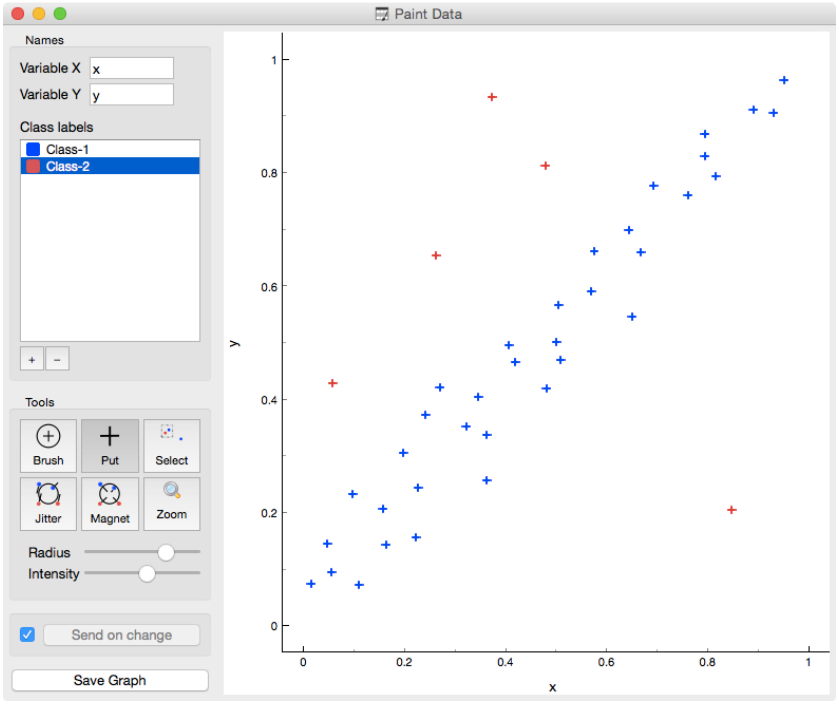
Lesson 24: Evaluating Regression

The last lessons 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!

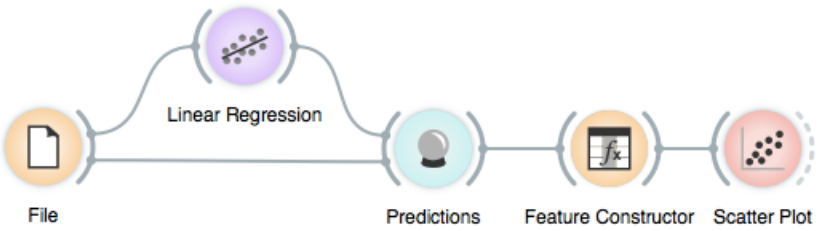
This workflow visualizes the predictions that were performed on the training data. How would you change the widget to use a separate test set? Hint: The Sample widget can help.

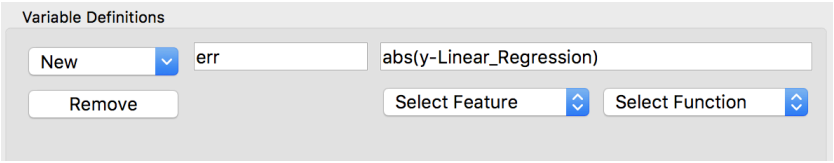


We can play around with this workflow by painting the data such that the regression would perform well on blue data point and fail on the red outliers. In the scatter plot we can check if the difference between the predicted and true class was indeed what we have expected.



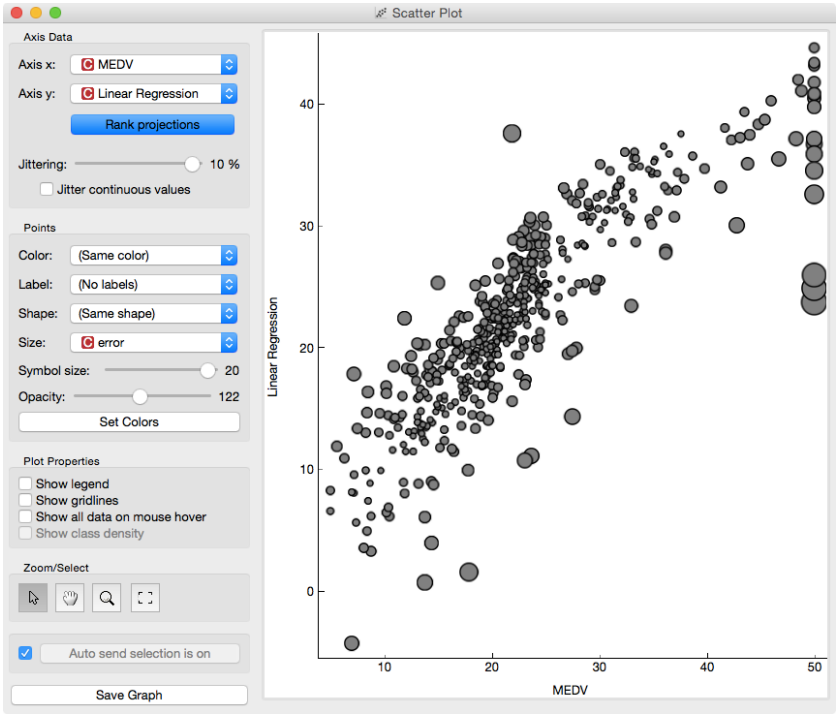
A similar workflow would work for any data set. Take, for instance, the housing data set (from Orange distribution). Say, just like above, we would like to plot the relation between true and predicted continuous class, but would like to add information on the absolute error the predictor makes. Where 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).



Lesson 25: Feature Scoring and Selection

For this lesson, load the data from imports-85.tab using the File widget and Browse documentation data sets.

Linear regression infers a model that estimate the class, a real-valued feature, as a sum of products of input features and their weights. Consider the data on prices of imported cars in 1985.

Data Table (1)

	height	curb-weight	engine-type	num-of-cylinders	engine-size	fuel-system	bore	stroke
1	48.800	2548.000	dohc	four	130.000	mpfi	3.470	2.680
2	48.800	2548.000	dohc	four	130.000	mpfi	3.470	2.680
3	52.400	2823.000	ohcv	six	152.000	mpfi	2.680	3.470
4	54.300	2337.000	ohc	four	109.000	mpfi	3.190	3.400
5	54.300	2824.000	ohc	five	136.000	mpfi	3.190	3.400
6	53.100	2507.000	ohc	five	136.000	mpfi	3.190	3.400
7	55.700	2844.000	ohc	five	136.000	mpfi	3.190	3.400
8	55.700	2954.000	ohc	five	136.000	mpfi	3.190	3.400
9	55.900	3086.000	ohc	five	131.000	mpfi	3.130	3.400
10	52.000	3053.000	ohc	five	131.000	mpfi	3.130	3.400
11	54.300	2395.000	ohc	four	108.000	mpfi	3.500	2.800
12	54.300	2395.000	ohc	four	108.000	mpfi	3.500	2.800
13	54.300	2710.000	ohc	six	164.000	mpfi	3.310	3.190
14	54.300	2765.000	ohc	six	164.000	mpfi	3.310	3.190
15	55.700	3055.000	ohc	six	164.000	mpfi	3.310	3.190
16	55.700	3230.000	ohc	six	209.000	mpfi	3.620	3.390
17	53.700	3380.000	ohc	six	209.000	mpfi	3.620	3.390
18	56.300	3505.000	ohc	six	209.000	mpfi	3.620	3.390

Inspecting this data set in a Data Table, it shows that some features, like fuel-system, engine-type and many others, are discrete. Linear regression only works with numbers. In Orange, linear regression will automatically convert all discrete values to numbers, most often using several features to represent a single discrete features. We also do this conversion manually by using Continuize widget.

Continuize

Categorical Features
- (•) Target or first value as base
- ( ) Most frequent value as base
- ( ) One attribute per value
- ( ) Ignore multinomial attributes
- ( ) Remove categorical attributes
- ( ) Treat as ordinal
- ( ) Divide by number of values

Numeric Features
- ( ) Leave them as they are
- ( ) Normalize by span
- (•) Normalize by standard deviation

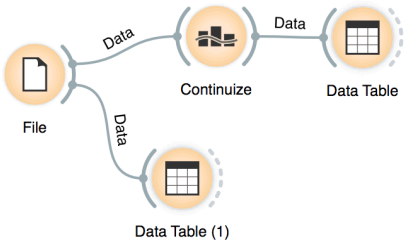
Categorical Outcomes
- (•) Leave it as it is
- ( ) Treat as ordinal
- ( ) Divide by number of values
- ( ) One class per value

Value Range
- ( ) From -1 to 1
- (•) From 0 to 1

Report

[✓] Apply Automatically

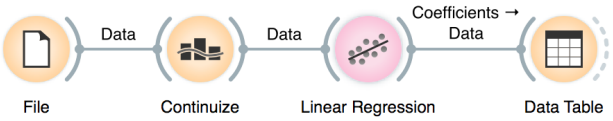
Before we continue, you should check what Continuize actually does and how it converts the nominal features into real-valued features. The table below should provide sufficient illustration.

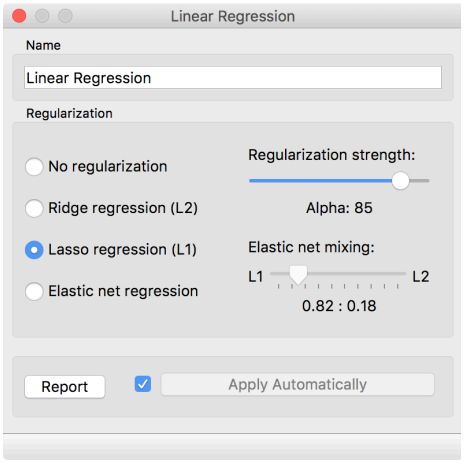


Data Table

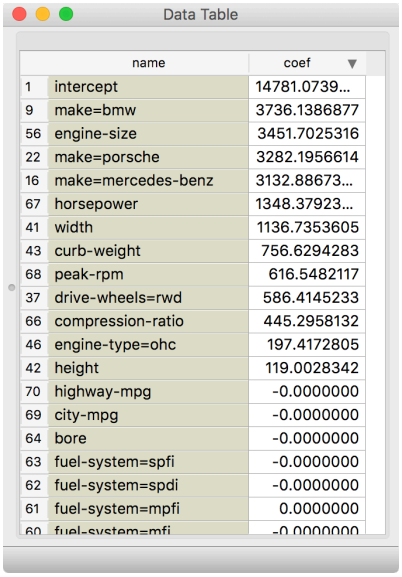
	symboling=3	ormalized-losses	make=audi	make=bmw	make=chevrolet	make=dodge
1	1.000	?	0.000	0.000	0.000	0.000
2	1.000	?	0.000	0.000	0.000	0.000
3	0.000	?	0.000	0.000	0.000	0.000
4	0.000	1.189	1.000	0.000	0.000	0.000
5	0.000	1.189	1.000	0.000	0.000	0.000
6	0.000	?	1.000	0.000	0.000	0.000
7	0.000	1.019	1.000	0.000	0.000	0.000
8	0.000	?	1.000	0.000	0.000	0.000
9	0.000	1.019	1.000	0.000	0.000	0.000
10	0.000	?	1.000	0.000	0.000	0.000
11	0.000	1.981	0.000	1.000	0.000	0.000
12	0.000	1.981	0.000	1.000	0.000	0.000
13	0.000	1.868	0.000	1.000	0.000	0.000
14	0.000	1.868	0.000	1.000	0.000	0.000
15	0.000	?	0.000	1.000	0.000	0.000
16	0.000	?	0.000	1.000	0.000	0.000
17	0.000	?	0.000	1.000	0.000	0.000
18	0.000	?	0.000	1.000	0.000	0.000
19	0.000	-0.028	0.000	0.000	1.000	0.000
20	0.000	-0.679	0.000	0.000	1.000	0.000

Now to the core of this lesson. Our workflow reads the data, coninuizes it such that we also normalize all the features to bring them the to equal scale, then we load the data into Linear Regression widget and check out the feature coefficients in the Data Table.





In Linear Regression, we will use L1 regularization. Compared to L2 regularization, which aims to minimize the sum of squared weights, L1 regularization is more rough and minimizes the sum of absolute values of the weights. The result of this "roughness" is that many of the feature will get zero weights.


Data Table

	name	coef
1	intercept	14781.0739...
9	make=bmw	3736.1386877
56	engine-size	3451.7025316
22	make=porsche	3282.1956614
16	make=mercedes-benz	3132.88673...
67	horsepower	1348.37923...
41	width	1136.7353605
43	curb-weight	756.6294283
68	peak-rpm	616.5482117
37	drive-wheels=rwd	586.4145233
66	compression-ratio	445.2958132
46	engine-type=ohc	197.4172805
42	height	119.0028342
70	highway-mpg	-0.0000000
69	city-mpg	-0.0000000
64	bore	-0.0000000
63	fuel-system=spfi	-0.0000000
62	fuel-system=spdi	-0.0000000
61	fuel-system=mpfi	0.0000000
60	fuel-system=mfi	-0.0000000

But this may be also exactly what we want. We want to select only the most important features, and want to see how the model that uses only a smaller subset of features actually behaves. Also, this smaller set of features is ranked. Engine size is a huge factor in pricing of our cars, and so is the make, where Porsche, Mercedes and BMW cost more than other cars (ok, no news here).

We should notice that the number of features with non-zero weights varies with regularization strength. Stronger regularization would result in fewer features with non-zero weights.