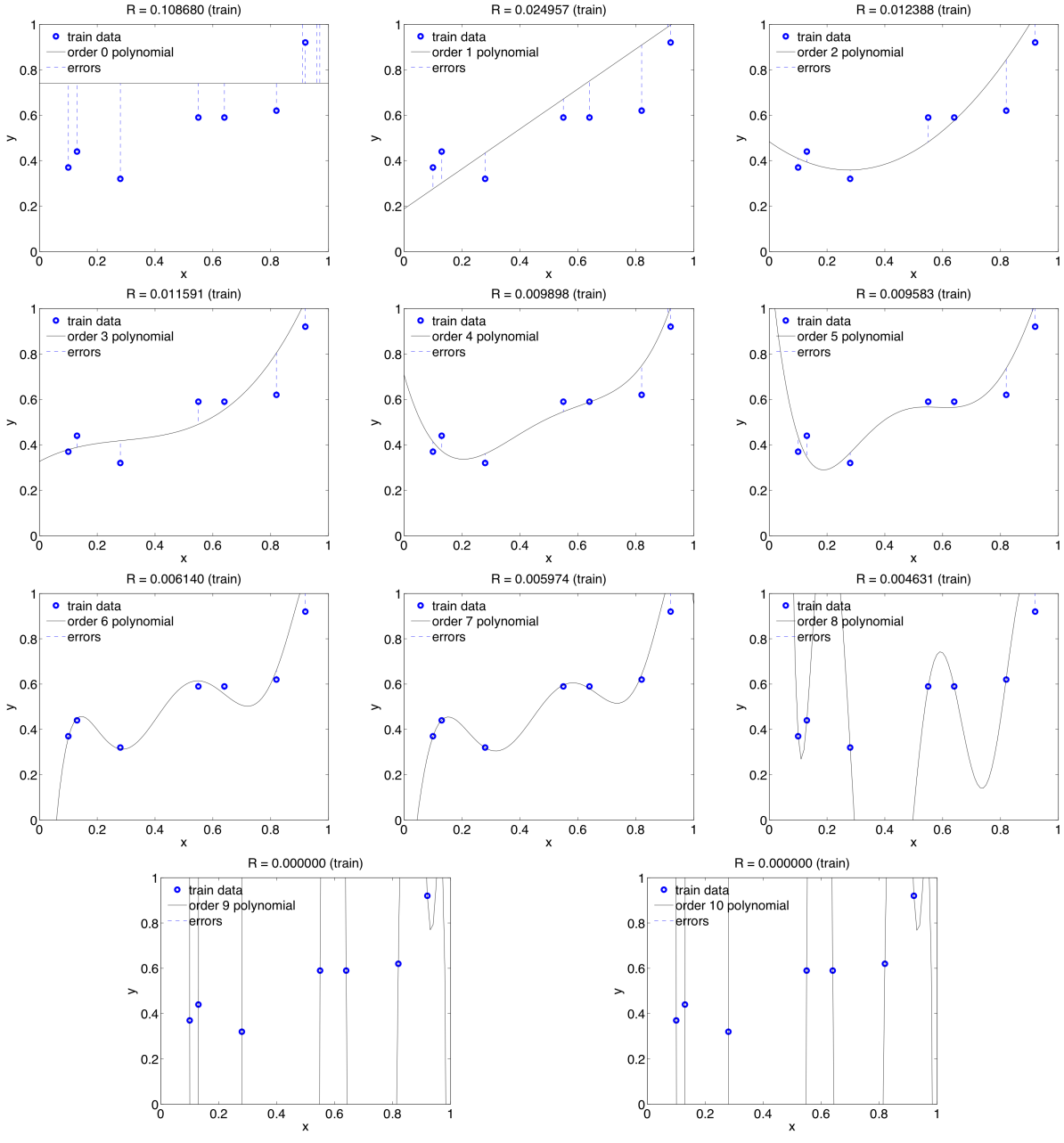
3 Overfitting

Remember our dataset from last time. We have a bunch of inputs x_i and corresponding outputs y_i. In the previous lecture notes, we considered how to fit polynomials of different orders, as

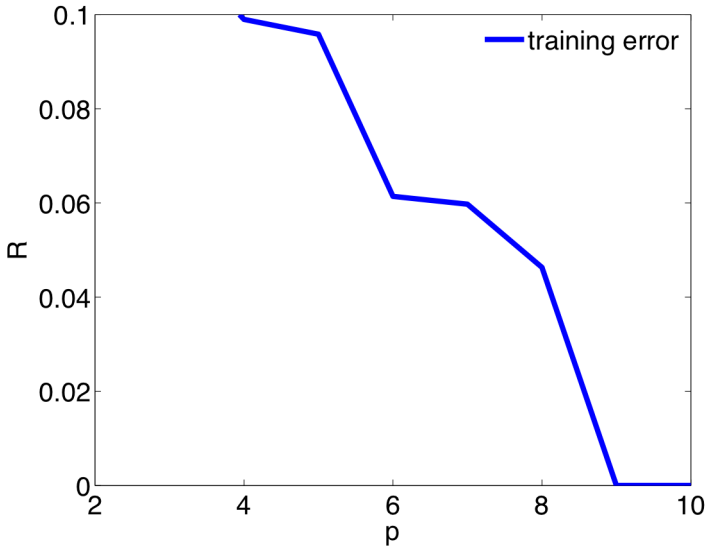
$$y = w_0 + w_1 x + w_2 x^2 + ... + w_p x^p.$$

Now, a natural question to ask is: what value of p should we use?

To start off, let's simply try fitting the above data by least squares with a range of p.

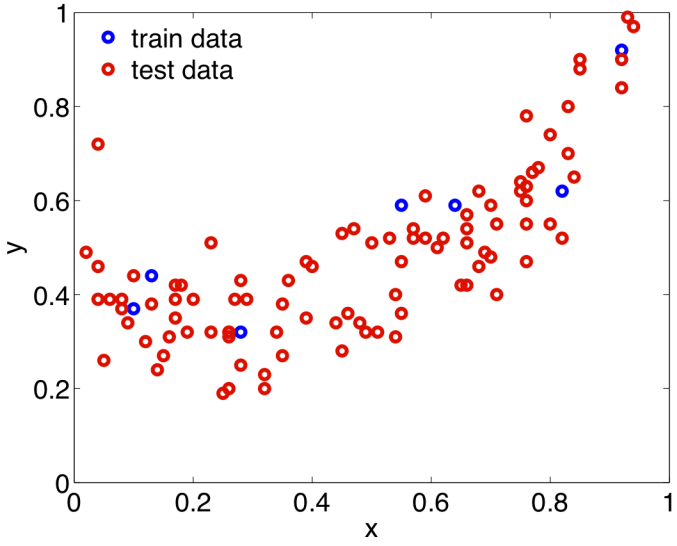

R = 0.108680 (train)
train data
order 0 polynomial
errors
R = 0.024957 (train)
train data
order 1 polynomial
errors
R = 0.012388 (train)
train data
order 2 polynomial
errors
R = 0.011591 (train)
train data
order 3 polynomial
errors
R = 0.009898 (train)
train data
order 4 polynomial
errors
R = 0.009583 (train)
train data
order 5 polynomial
errors
R = 0.006140 (train)
train data
order 6 polynomial
errors
R = 0.005974 (train)
train data
order 7 polynomial
errors
R = 0.004631 (train)
train data
order 8 polynomial
errors
R = 0.000000 (train)
train data
order 9 polynomial
errors
R = 0.000000 (train)
train data
order 10 polynomial
errors
x
y

If we plot the error R as a function of p , we see:

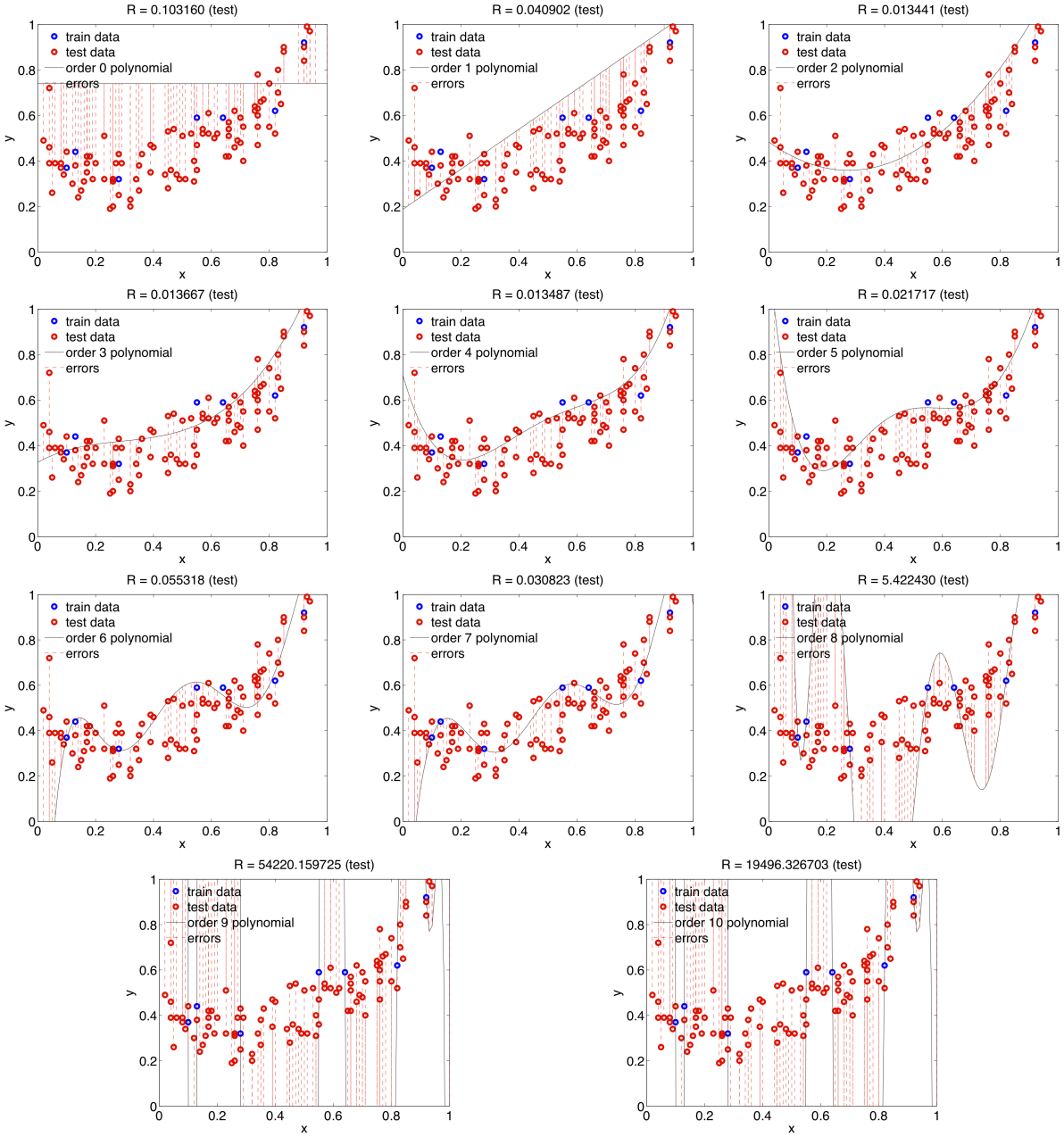


This doesn't seem satisfying. Should we always use the highest-order polynomial possible?

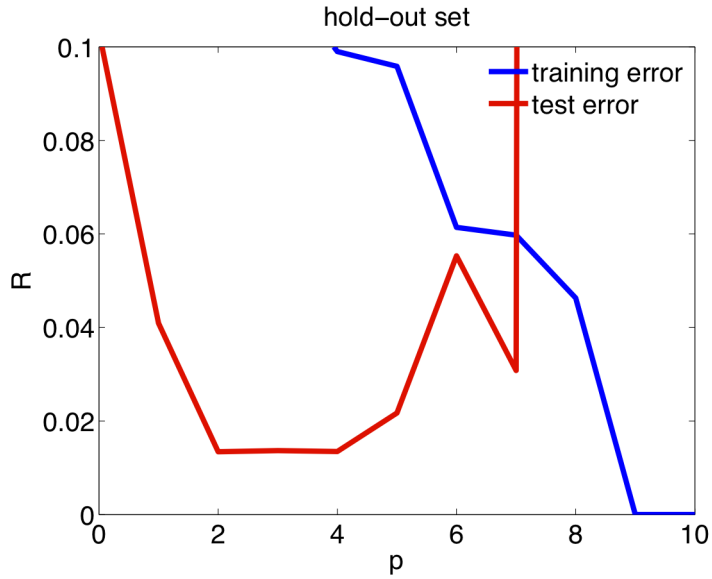
To think about this more, let's suppose for the moment that we happened to have an extra 100 point lying around in a "test set".



We can see the errors on this test set like so


R = 0.103160 (test)
train data
test data
order 0 polynomial
errors
R = 0.040902 (test)
train data
test data
order 1 polynomial
errors
R = 0.013441 (test)
train data
test data
order 2 polynomial
errors
R = 0.013667 (test)
train data
test data
order 3 polynomial
errors
R = 0.013487 (test)
train data
test data
order 4 polynomial
errors
R = 0.021717 (test)
train data
test data
order 5 polynomial
errors
R = 0.055318 (test)
train data
test data
order 6 polynomial
errors
R = 0.030823 (test)
train data
test data
order 7 polynomial
errors
R = 5.422430 (test)
train data
test data
order 8 polynomial
errors
R = 54220.159725 (test)
train data
test data
order 9 polynomial
errors
R = 19496.326703 (test)
train data
test data
order 10 polynomial
errors
x
y

If we plot the error R as a function of p , we see the best test error is found at $p = 2$, followed by a rapid increase.



So, this is all well and good. The problem, of course, is that we generally won't have a gigantic extra amount of data sitting around. And if we did, do we want to use it for training or for testing? In reality, we have one dataset, which we will, ourselves, split into a training set and a test set. We might formalize this as a type of algorithm.

Algorithm: Polynomial selection using a test set

- Input some data.
- Take some fraction (e.g. 30%) of the data as the test set, keep the rest as the training set.
- For $p = 0, 1, ..., P$
  - Fit a p-order polynomial to the training data.
  - Evaluate the error $R_{\text{test}}(p)$ on the test set
- Find $p^* = \arg\min_p R_{\text{test}}(p)$.
- Re-fit a polynomial of order p^* to the entire dataset, and output it.

Note, of course, that this kind of algorithm is not specific to polynomials. In any kind of application where one is considering different classes of predictors, some of which are more "powerful" than others, this kind of strategy can be used.

The previous algorithm is all well and good. However, can we think of any weaknesses of it?

- How to split between training and test set? A 70/30 split is not obviously optimal.
- Will, "on average" pick a p^* that is too small. (Suppose we have 100 data. If we use 70% of the data as the training set, it intuitively tries to find the p to generalize best from 70 data, not 100.)

3.1 Error estimation using cross-validation

Let's start from the beginning, and try to build up approaches to estimate errors. For example, take our original dataset. Suppose that we want to estimate the test error of an affine function we fit to it. The simplest way to do this would be a **hold-out set**. This means we

1. Take some subset of the data as a test set.
2. Take the rest as the training set.
3. Fit a function to the training set.
4. Evaluate the error on the test set.

Exercise 21. Make a procedure to estimate errors of an affine function using the simple dataset, holding out the first 2 points. Verify that you compute a training error of $.0513$ and a test-error of $.0899$.

This is not a particularly good estimate. Notice that, above, using a large test set, we have a test error of .0245, while here we happen to have a much larger estimate. Notice also that (as we would expect) the training error is below the true test error.

Now, of course, the particular choice of using the first two points as a hold-out set was totally arbitrary. Why not the first two? A key idea of cross-validation is that we can use *multiple test sets*, and average the results. The procedure works as follows.

Algorithm: K-fold cross validation

1. Divide the data into K chunks
2. For $k = 1, 2, ..., K$
   (a) Take the k-th chunk of the data as a test set.

(b) Take the other $K-1$ chunks as the training set.

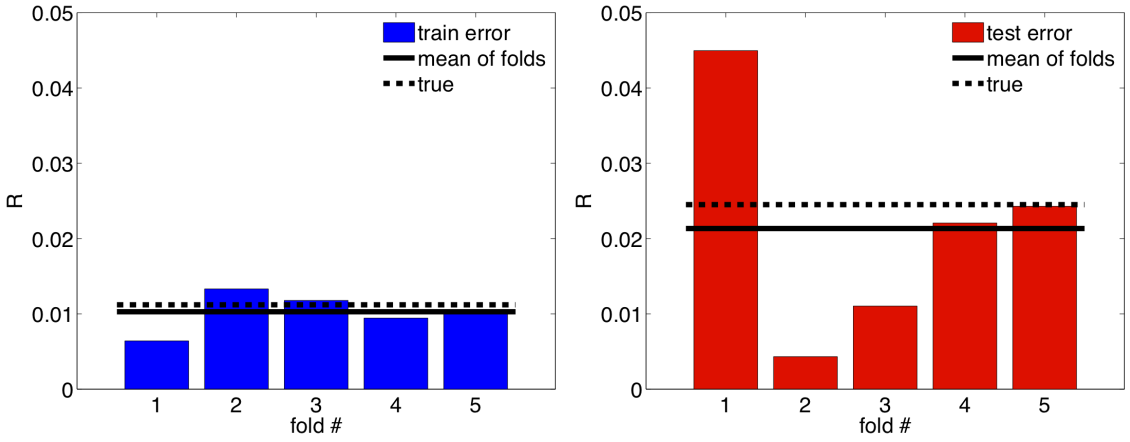
(c) Fit a function to the training set.

(d) Evaluate the error on the test set.

3. Return the average of all the errors calculated in step 2(e).

Exercise 22. Make a procedure to estimate errors of estimating an affine function using the simple dataset, using 5 different folds. Check that the mean training error is 0.0822 and the mean test error is 0.0427.

We can look at how much variability there is the different folds in the following graphs. These show the "true" training and test errors using the full set of 10 points for training and the 100 points for testing.



3.2 K-Fold Cross-Validation

Now, we have a good method for estimating the errors. How can we use this to find the best polynomial? The idea is simple:

Algorithm: Polynomial selection using K-fold cross validation

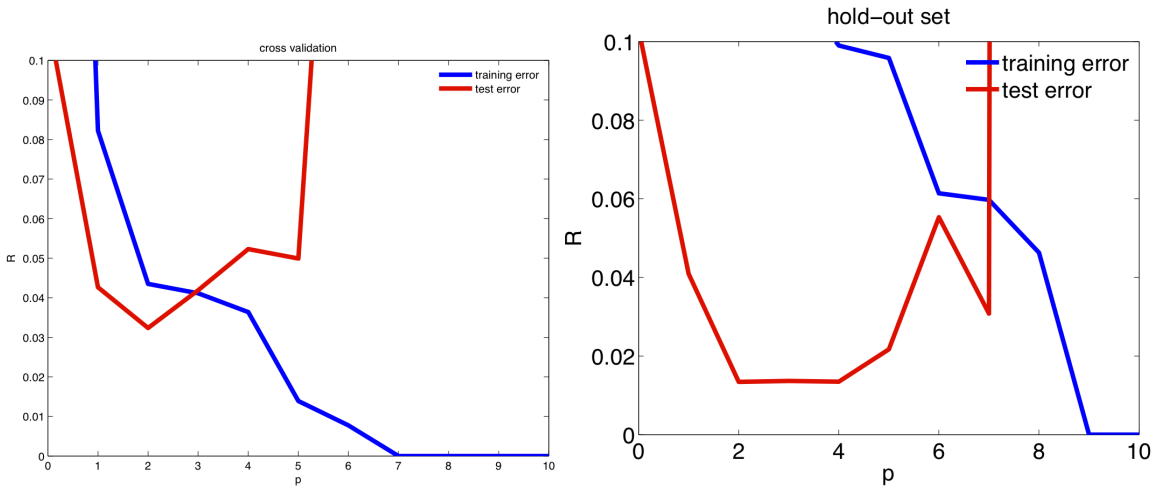
- For each p, estimate the test error of fitting a p-th order polynomial using cross validation.
- Pick the polynomial order p^* that has the smallest estimated test error.
- Re-fit a polynomial of order p^* to the entire dataset, and output it.

Doing N-fold cross-validation on a dataset with N points has the special name of "leave-one-out" cross-validation.

Exercise 23. On the above dataset, for $p = 0, 1, ..., 10$ estimate the test error using 5-fold cross validation. Verify that you get estimated test errors as in the following table:

order	train error	test error
0	0.4155	0.1107
1	0.0822	0.0427
2	0.0435	0.0324
3	0.0411	0.0419
4	0.0364	0.0523
5	0.0139	0.0499
6	0.0077	0.2376
7	0.0000	13.6933
8	0.0000	25.3749
9	0.0000	39.2409
10	0.0000	49.9065

We can visualize this with the following plot. Compare to our earlier plot using the large test set:



Cross-validation does a reasonable job. (In particular, in this example, both methods would pick $p = 2$.) However, it does not miracles. As practitioners tend to say, more data beats a clever algorithm.

3.3 Discussion

The best way of doing these types of things remains to some degree an active research problem. For more information, see Andrew Moore's
Cross-validation fordetecting and preventing overfitting.