Statistical Machine Learning Notes 2

Overfitting, Model Selection, Cross Validation, Bias-Variance

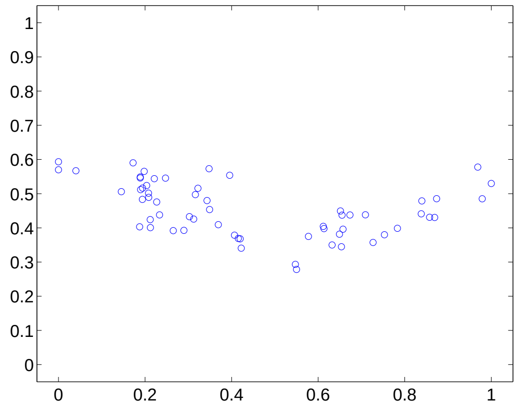
Instructor: Justin Domke

1 Motivation

Suppose we have some data

$$\text{TRAIN} = \{(x^1, y^1), (x^2, y^2), ..., (x^N, y^N)\}$$

that we want to fit a curve to:



Here, we want to fit a polynomial, of the form

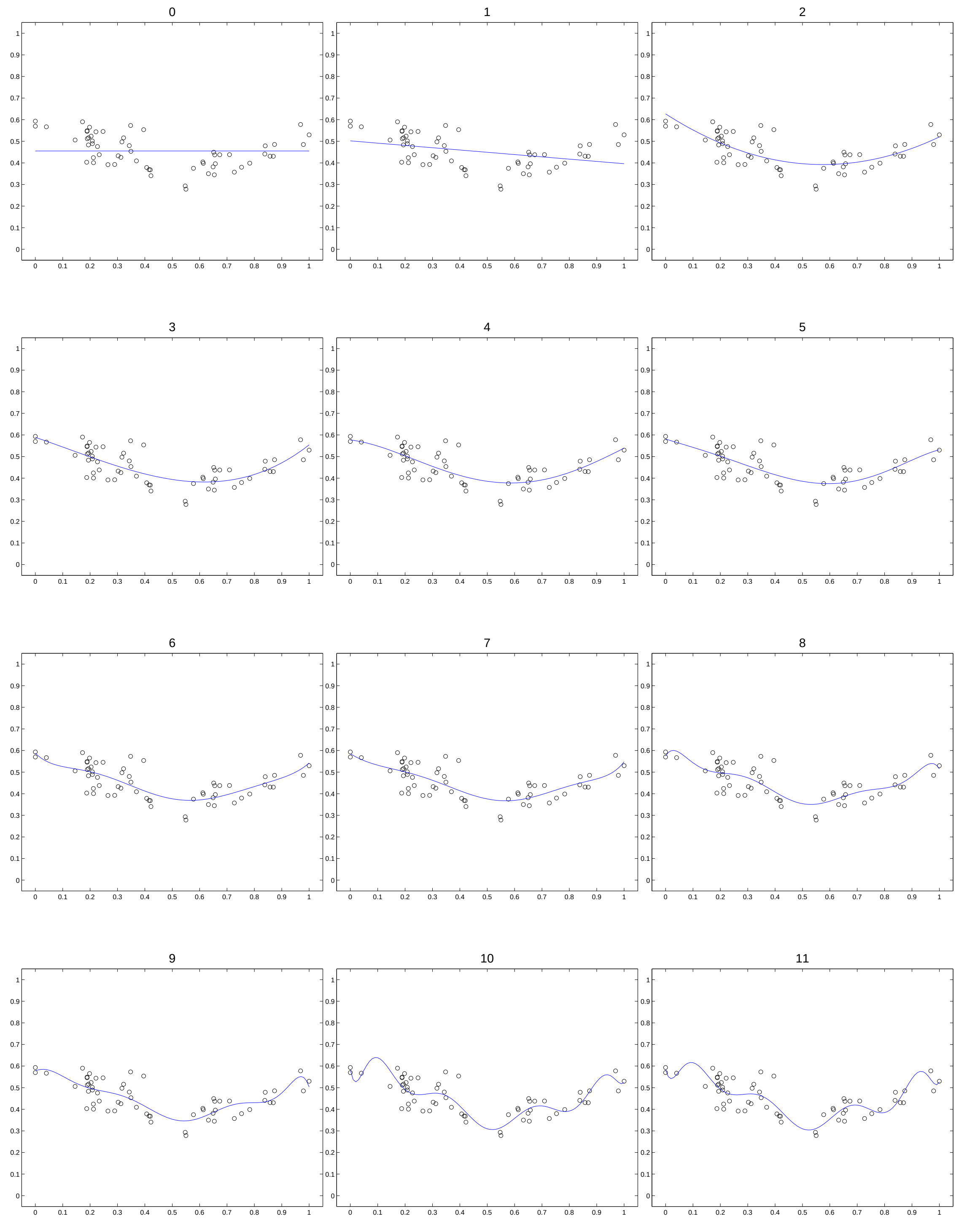
$$y = w_0 + w_1 x + w_2 x^2 + ... + w_p x^p.$$

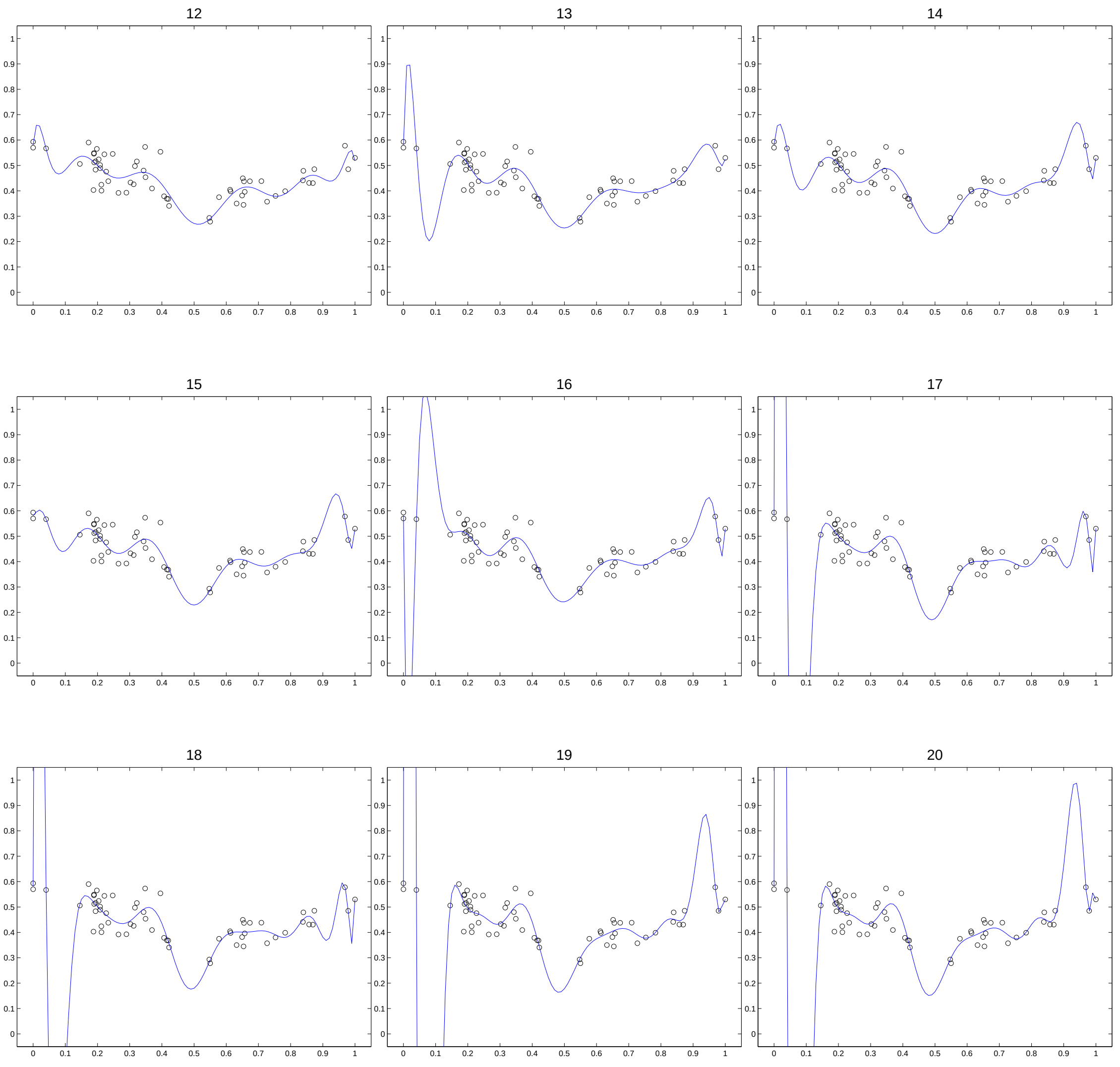
In these notes, we are interested in the question of how to choose p. For various p , we can find and plot the best polynomial, in terms of minimizing the squared distance to all the points $(\hat{y}, \hat{x})$

$$\min_{\mathbf{w}} \hat{\mathbb{E}}_{\text{TRAIN}} \big(w_0 + w_1 X + w_2 X^2 + ... + w_p X^p - Y\big)^2. \tag{1.1}$$

We will refer to this mean-squared distance of squares distances as the **training error**.

$$\hat{\mathbb{E}}_{\text{TRAIN}}\left(w_0 + w_1 X + w_2 X^2 + \ldots + w_p X^p - Y\right)^2$$





The question we address in these notes is: what p do we expect to generalize best? Specifically, we will imagine that we will get some new data TEST from the same distribution. We want to pick p to minimize the same sum-of-squares difference we used for fitting to the training data. This is called the **test data**.

$$\hat{\mathbb{E}}_{\text{TEST}}\left(w_0 + w_1 X + w_2 X^2 + ... + w_p X^p - Y\right)^2$$

Here, we see a simple example of the basic game of machine learning. It works like this:

- Input some data TRAIN.
- Fit some model to TRAIN.

- Get some new data TEST.
- Test the model on the TEST.

In machine learning, one is generally considered to have "won" the game if one's performance on TEST is best. This is often in contrast to the attitude in statistics, where success is usually measured in terms of if the model is close to the true model.

Our assumption here is that the elements of TRAIN and TEST are both "drawn from the same distribution". What this means is that there is some "true" distribution $p_0(x,y)$. We don't know p_0, but we assume all the elements of TRAIN and TEST are both drawn from it independently.

An example should make this clear. Suppose that x is a person's height, and y is a person's age. Then $(x,y) \sim p_0(x,y)$ means that we got x and y by showing up at a random house on a random day, grabbing a random person, and measuring their height and age. If we got TRAIN by doing this in Rochester, and TEST by doing this in Buffalo, then we would be violating the assumption that TRAIN and TEST came from the same distribution.

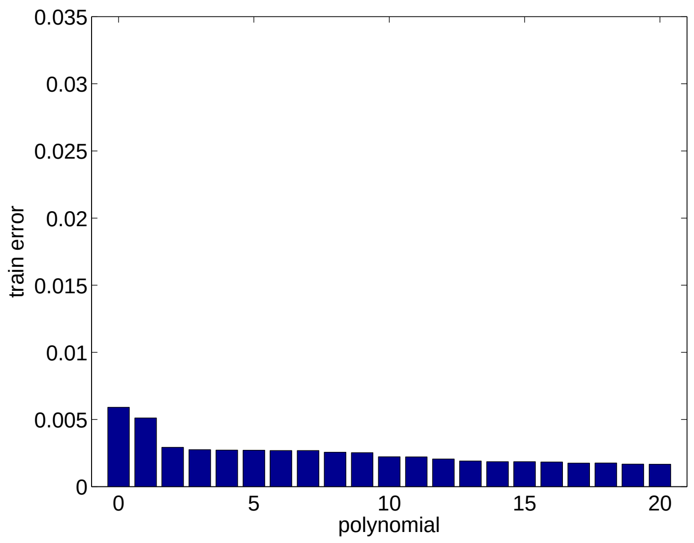
Now, return to our example of choosing p. What will happen to the training error as p gets bigger? Now, clearly, a larger p means more "power", and so the training error will strictly decrease. However, the high degree polynomials appear to display severe artifacts, that don't appear likely to capture real properties of the data. What is going wrong here?

2 The Bias-Variance Tradeoff

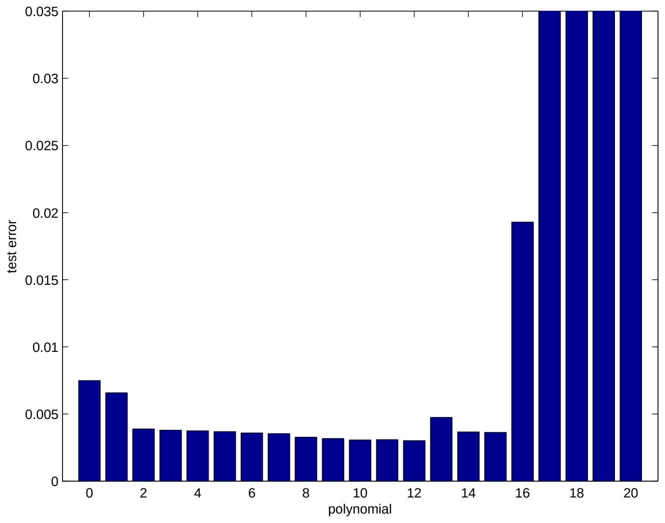
Let us return to our initial problem of trying to pick the right degree p for our polynomial. The first idea that springs to mind is to pick that p that fits the data best. If we plot

$$\min_{\mathbf{w}} \hat{\mathbb{E}}_{\text{TRAIN}} \big(w_0 + w_1 X + w_2 X^2 + ... + w_p X^p - Y\big)^2.$$

for each p, we see something disturbing:



Clearly this is not what we want. Instead, suppose we had a giant pile of 100,000 extra point drawn from the same distribution as the training error. We will call this **test data**, and the average error on it the **test error**.



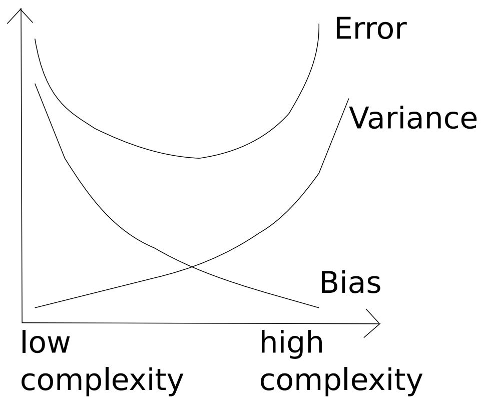
Things blow up after $p = 13$. (This is intuitively plausible if you look at the plots of the polynomials above.)

There are two things that are happening here:

- For very low p, the model is very simple, and so can't capture the full complexities of the data. This is called **bias**.

- For very high p, the model is complex, and so tends to "overfit" to spurious properties of the data. This is called **variance**.

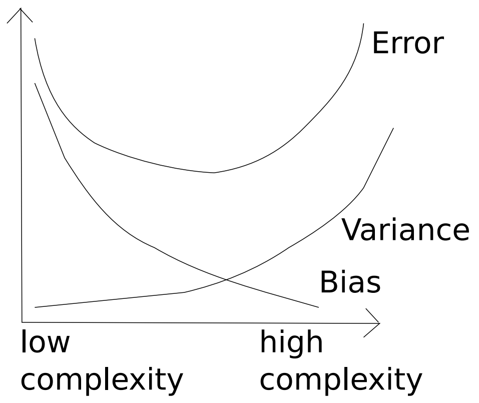
Don't think about the names "bias" and "variance" too much. The names derive from technical concepts in fitting least-squares regression models, but we will use them more informally for all types of models. Now, when doing model selection, one often sees a **bias-variance tradeoff**. This is commonly thought of by drawing a picture like this:



This shows the bias, variance, and error of a range of different learning methods, for a given amount of training data. It shows the common situation in practice that (1) for simple models, the bias increases very quickly, while (2) for complex models, the variance increases very quickly. Since the riskiness is additive in the two, the optimal complexity is somewhere in the middle. Note, however, that these properties do *not* follow from the bias-variance decomposition, and need not even be true.

Question: Suppose the amount of training data is increased. How would the above curves change?

Answer: The new curves would look something like:



The variance is reduced since there is more data, and so a slightly more complex model minimizes the expected error.

Bias-variance tradeoffs are seen very frequently, in all sorts of problems. We can very often understand the differences in performance between different algorithms as trading off between bias and variance. Usually, if we take some algorithm, and change it to reduce bias, we will also increase variance. This doesn't **have** to happen, though. If you work hard, you can change an algorithm in such a bad way as to increase both bias and variance.

3 Cross Validation

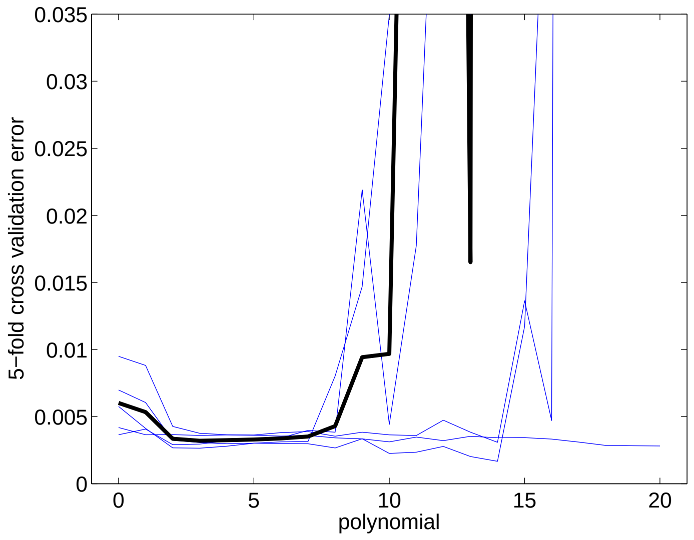
Let's return to the issue of picking p. We saw above that if we had 100,000 extra data, we could just fit the model with each p, and pick the p that does best on the validation data. What would you do if you don't happen to have a spare 100,000 extra data sitting around? The first idea that comes to mind is to "hold out" some of our original training data.

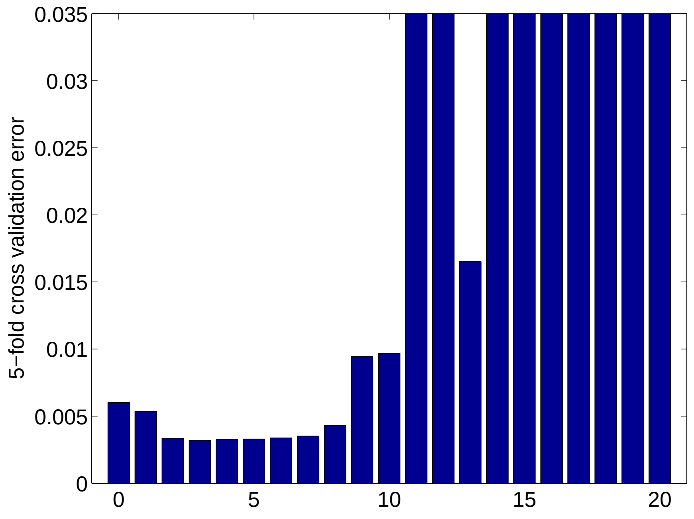
1. Split TRAIN into $\text{TRAIN}_{\text{fit}}$ and $\text{TRAIN}_{\text{validation}}$. (e.g. half in each)
2. Fit each model to $\text{TRAIN}_{\text{train}}$, and evaluate how well it does on $\text{TRAIN}_{\text{validation}}$.
3. Output the model that has the best score on $\text{TRAIN}_{\text{validation}}$.

This can work reasonably well, but it "wastes" the data by only training on half, and only testing on half. We can make better use of the data by making several different splits of the data. Each datum is used once for testing, and the other times for training. This algorithm is called **K-fold cross validation**.

1. Split TRAIN into K chunks
2. For $k = 1, 2, ..., K$:
   (a) Set $\text{TRAIN}_{\text{validation}}$ to be the kth chunk of data, and $\text{TRAIN}_{\text{fit}}$ to be the other $K - 1$ chunks.
   (b) Fit each model to $\text{TRAIN}_{\text{fit}}$ and evaluate how well it does on $\text{TRAIN}_{\text{validation}}$.
3. Pick the model that has the best average test score.
4. Retrain that model on all of TRAIN, and output that.

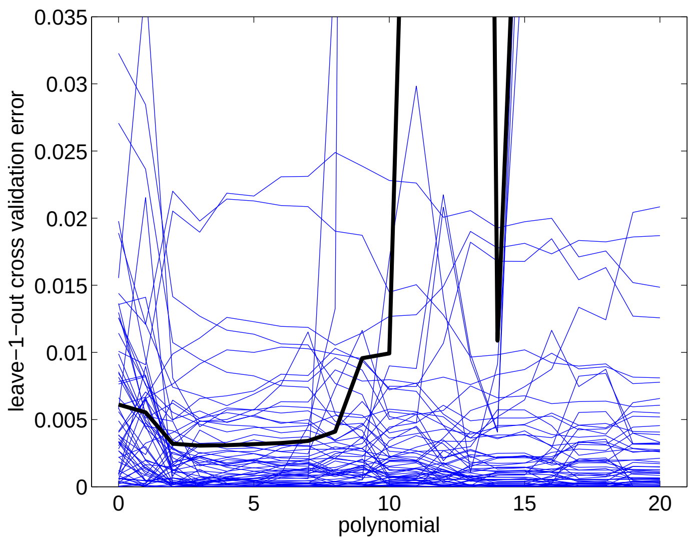
If we do 5-fold cross validation on our original set of 60 points, we get:

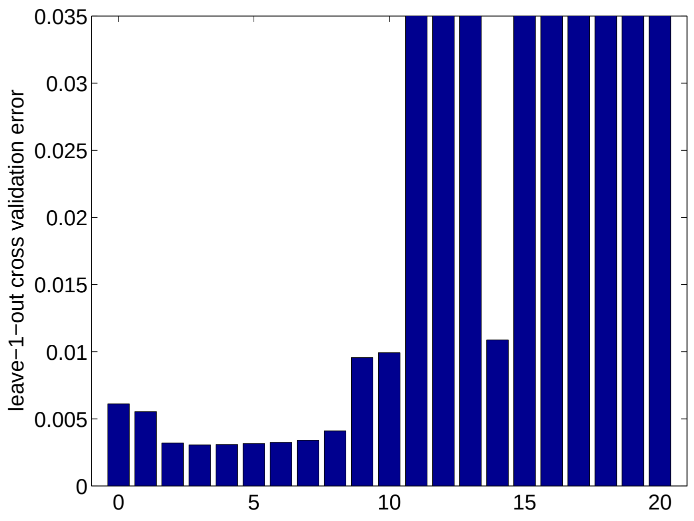




We can see that this picks $p = 3$, rather than the optimal $p = 12$. Clearly, cross validation is no substitute for a large test set. However, if we only have a limited training set, it is often the best option available.

In this case, cross-validation selected a simpler model than optimal. What do we expect to happen on average? Notice that with 60 points, 5-fold cross validation effectively tries to pick the polynomial that makes the best bias-variance tradeoff for 48 $(60 \cdot \frac{4}{5})$ points. If we had done 10-fold cross validation, it would instead try to pick the best polynomial for 54 $(60 \cdot \frac{9}{10})$ points. Thus, cross validation *biases towards simpler models*. We could reduce this to the minimum possible by doing 60-fold cross validation. This has a special name: **leave-one-out cross validation**. In complex models, this can be expensive, though there has been research on clever methods for doing leave-one-out cross validation with out needing to recompute the full model. In practice, we usually don't see too much benefit for doing more then 5 or 10 fold cross validation. In this particular example, 60-fold cross validation still selects $p = 3$.





Cross validation is a good technique, but it doesn't work miracles: there is only so much information in a small dataset.

People often ask questions like:

- "If I do clustering, how many clusters should I use?"
- "If I fit a polynomial, what should its order be?"
- "If I fit a neural network, how many hidden units should it have?"

- "If I fit a mixture of Gaussians, how many components should it have?"
- “How do I set my regularization constant?”
- “I want to fit a Support Vector Machine classifier to this dataset. Should I use a Gaussian, polynomial, or linear kernel?”

A reasonable answer to all of these is cross-validation.

Though cross validation is extremely widely used, there has been difficulty proving that it actually works better than just using a single hold-out set of $\frac{1}{K}$th of the data! There is even a $500 bounty. See: http://hunch.net/?p=29 http://hunch.net/?p=320