MATH 728D: Machine Learning Lab #16: Curve Fitting

John Burkardt

January 16, 2019

Can I draw a straight line through crooked data?

Science starts from observations and measurements of physical events and objects: the time the sun rises, the area of a circle; the resistance of a spring. A goal of such observation is to deduce a formula or theory that simplifies, predicts and explains the data. While the data is a sample of reality, a good theory replaces the data with a simple, usable mental model.

Physical events include natural variations, measurements involve small errors, and even the numeric storage and processing of measurements will add additional approximations. Thus, in the data, there is likely to be some noise as well as a meaningful signal. We will see that even relatively small amounts of noise can have serious consequences.

Interpolation and approximation are method of guessing an underlying, universal law that explains the observed data as well as all possible data. But since our data is "noisy", we need to be able to separate meaningful information from values that are meaningless, invalid, or outliers.

1 Fitting an Interpolating Polynomial to Data

Given a set of n data pairs (x, y), we may think that they result from some simple mathematical relationship. Two data points would determine a line; three would be enough to find a quadratic relationship; and so on. Since our data is inexact, we might expect that we would never find a perfect relationship, but that fairly soon we would have a formula that closely approximates a larger set of data.

Polynomial interpolation of degree n assumes a formula

$$p(x) = p_0 + p_1 x + p_2 x^2 ... + p_n x^n$$

and determines the coefficients p by evaluating $p(x)$ at $n + 1$ data points:

$$\begin{aligned} p(x_1) =& p_0 + p_1 x_1 + p_2 x_1^2 ... + p_n x_1^n = y_1 \\ p(x_2) =& p_0 + p_1 x_2 + p_2 x_2^2 ... + p_n x_2^n = y_2 \\ & ... \\ p(x_{n+1}) =& p_0 + p_1 x_{n+1} + p_2 x_{n+1}^2 ... + p_n x_{n+1}^n = y_{n+1} \end{aligned}$$

which is equivalent to the following *Vandermonde linear system:*

$$\begin{pmatrix} 1 & x_1 & x_1^2 & ... & x_1^n \\ 1 & x_2 & x_1^2 & ... & x_2^n \\ ... & ... & ... & ... & ... \\ 1 & x_{n+1} & x_{n+1}^2 & ... & x_{n+1}^n \end{pmatrix} * \begin{pmatrix} p_0 \\ p_1 \\ ... \\ p_n \end{pmatrix} = \begin{pmatrix} y_1 \\ y_2 \\ ... \\ y_{n+1} \end{pmatrix}$$

It is possible to set up and solve this linear system for the coefficients p, and then to compare the interpolating polynomial $p(x)$ with the data values.

To save you some time, we will instead use MATLAB's

```
p = polyfit ( x, y, d );
```

function, which produces the coefficients p of a polynomial of degree d that approximates a set of m data points (x,y). If $m = d+1$, then in fact, the approximating curve is actually an interpolating curve, and should match the data exactly. The polynomial can easily be evaluated at the set of points `xp` by the command:

```
yp = polyval ( p, xp );
```

Using this shortcut, we can quickly produce and display interpolant curves of degree 1 (a line), 2 (a quadratic) and so on.

Exercise 1:

1. Load the data file *sine_train.txt*;
2. Set `x` and `y` to columns 1 and 2 of this data;
3. Ask the `polyfit()` command for a first degree polynomial, and give it 2 data values:

```
degree = 1
p = polyfit ( x(1:degree+1), y(1:degree+1), degree )
xp = linspace ( 0.0, 1.0, 51 );
pp = polyval ( p, xp );
plot ( x, y, 'bo', xp, yp, 'r-' )
```

4. Repeat these commands for values of degree = 2, 5 and 9

You should observe that the interpolant of degree d does indeed match the first $d + 1$ data values, but doesn't seem able to come close to the later values; moreover, as d increases, the interpolant functions seem to become more oscillatory, which suggests they are not converging to some simple shape. We will see that approximation does a better job of finding the overall patterns in the data.

2 Fitting an Approximating Line to Data

Suppose we have a set of n pairs of data values (x, y), and we suspect there is a simple linear relationship of the form:

$$y = c\,x$$

How can we determine c? We may choose an objective function $f(c)$, and agree that the "best" c is the one that minimizes $f()$. For our example, the least squares objective function is

$$f(c) = \sum_{i=1}^{n} (y_i - c\,x_i)^2$$

The value c which minimizes $f(x)$ can be found by setting $f'(c)$ to zero and solving for c:

$$c = \frac{x'y}{x'x}$$

Given data y and a linear model $c\,x$ for that data, the R^2 or "goodness-of-fit" statistic is

$$rsq = 1 - \frac{\sum_{i=1}^{n} (y_i - c\,x_i)^2}{\sum_{i=1}^{n} (y_i - \bar{y})^2}$$

where $\bar{y}$ is the average value of y. For linear approximations like we are doing, `rsq` will be between 0 (bad) and 1 (perfect).

Exercise 2:

1. The following data measures the deflection y of a spring under various loads x;
2. Load the data file *spring.txt*;
3. Set `x` and `y` to columns 1 and 2 of this data;
4. Set `hold on` so your next four plots are overlaid;
5. Plot `x` versus `y` as red points, 'ro';
6. Compute the least squares linear coefficient c for this data;
7. Plot `x` versus `c*x` as a red line, 'r-';
8. Report the goodness of fit for this line to the data;
9. Set `x2` and `y2` to the first 14 entries in `x` and `y`;
10. Plot `x2` versus `y2` as blue points, 'bo';
11. Compute the least squares linear coefficient $c2$ for this data;
12. Plot `x2` versus `c*x2` as a blue line, 'r-';
13. Report the goodness of fit for this line to the data;

We observe that the last six data values did not follow the linear pattern at all; hence, it is inappropriate to try to model them with a linear relationship. By dropping them from the data set, we were able to improve the goodness of fit statistic, and produce a line that better matches the remaining data.

In this case, we had to separate the information (linear relationship) from two kinds of noise - small data variations, and the fact that the linear relationship breaks down for large values of x.

3 Fitting an Approximating Polynomial to Data

Suppose we are sampling data from the sine function, and our measurements include both information and noise. An example of a formula that would have this property is

$$y = \sin(2 * pi * x) + 0.1 * randn();$$

where MATLAB's `randn()` returns a normally distributed random value.

Using only the sample data, we look for a formula that approximates the data in a "reasonable" way, and which might allow us to see the information or signal that is somewhat hidden within our data.

Exercise 3:

1. Load the data file *sine_train.txt*;
2. Set `x` and `y` to columns 1 and 2 of this data;
3. On a single plot, show `(x,y)` as blue points 'bo', and (x,sin(2*pi*x)) as a red line 'r-';
4. Determine the best linear approximant to all your data:

```
degree = 1
p = polyfit ( x, y, degree )
```

5. Compare your data and your linear approximation:

```
xp = linspace ( 0.0, 1.0, 101 );
yp = polyval ( p, xp );
plot ( x, y, 'bo', xp, yp, 'r-' )
e = sqrt ( sum ( ( y - polyval(p,x) ).^2 ) )
```

6. What happens to the approximation error as we increase degree by steps from 1 to 9?

```
p = polyfit ( x, y, degree );
e = sqrt ( sum ( ( y - polyval(p,x) ).^2 ) )
```

7. Does the degree 9 approximation match the data? Does it look "reasonable"?

```
degree = 9;
p = polyfit ( x, y, degree );
xp = linspace ( 0.0, 1.0, 101 );
yp = polyval ( p, xp );
plot ( x, y, 'bo', xp, yp, 'r-' )
```

Your experience with this data should suggest that you can approximate a given set of data with polynomials. As you increase the degree of approximation, the data is matched better and better, but the approximating polynomial may become so oscillatory that it seems a bad model for the underlying process we are trying to uncover!

4 Avoiding Overfitting Using Training and Testing Data

Although we can fit a curve to any data; however, we probably want the curve not just to summarize the data we've seen, but also to *predict* the behavior of new values.

Suppose that we have split our data into two sets, the training data, used to generate the model (in this case, the polynomial curves), and the testing data, which we can compare against "predictions" made by our training-data curve.

Looking at the prediction errors will help us to decide whether a given model has managed to get just the information from the data, or has been weakened because it also modeled the meaningless noise.

Exercise 4:

1. Load the data file *sine_train.txt*;
2. Set `x` and `y` to columns 1 and 2 of this data;
3. Load the data file *sine_test.txt*;
4. Set `x2` and `y2` to columns 1 and 2 of this data;
5. For degrees 1 through 9, compute and print the approximation error e, and plot degree versus error:

```
for i = 1 : 9
  degree(i) = i;
  p = polyfit ( x, y, degree(i) );
  e(i) = sqrt ( sum ( ( y2 - polyval(p,x2) ).^2 ) )
  fprintf ( 1, ' %d, %g\n', degree(i), e(i) );
end
plot ( degree, e, 'ro-' )
```

For what degree do you see the biggest drop in error? For what degree is the error the smallest?