An Unknown Signal

Symbols, Patterns and Signals

Jacob Daniel Halsey

March 2020

1 Least Squares Regression

The least squares calculations have been implemented in the `Segment` methods `lsr_polynomial()` and `lsr_fn()`. They both use the matrix formula: [1]

$$\boldsymbol{A} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{Y}$$

Where in the case of the polynomial regression:

$$\boldsymbol{X} = \begin{bmatrix} 1 & x_0 & (x_0)^2 & .. & (x_0)^k \\ 1 & x_1 & (x_1)^2 & .. & (x_1)^k \\ .. & .. & .. & .. & \\ 1 & x_k & (x_k)^2 & .. & (x_k)^k \end{bmatrix}$$

Or in the case of the arbitrary function (f) regression:

$$\boldsymbol{X} = \begin{bmatrix} 1 & f(x_0) \\ 1 & f(x_1) \\ .. & .. \\ 1 & f(x_k) \end{bmatrix}$$

$\boldsymbol{Y}$ is a vector containing the y values y_0 through to y_k

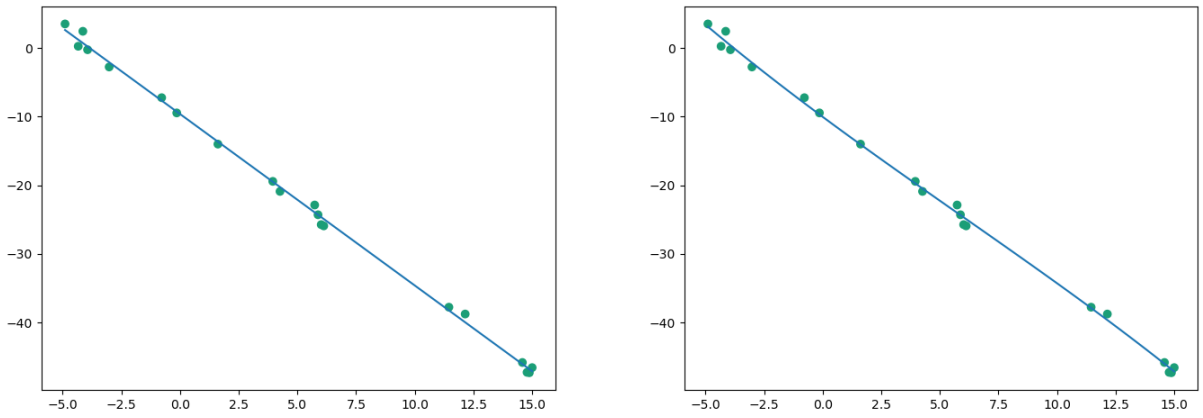
And the result $\boldsymbol{A}$ is the list of coefficients, in the order $a_0 + a_1x + .. + a_kx^k$ or $a_0 + a_1f(x)$.

Calculating the Error

The Sum Squared Error (SSE) is a method of measuring how well a fitted function (f) fits a dataset of n points, by calculating the difference between the predicted and actual data. [2]

$$SSE = \sum_{i=0}^{n}(y_i - f(x_i))^2$$

This has been implemented in the `ss_error()` function, which is called with an array of predicted y values, and array of actual y values.



(a) Minimum CVE: Linear

(b) Minimum SSE: Cubic

Figure 1: `noise_1.csv` fitted with different minimums

2 Overfitting

Whilst choosing the function with the lowest SSE will find the best fit for the given dataset, it does not prevent overfitting, a type of error where the model fits too closely to the noise in the data, and is therefore unable to accurately predict additional observations. In our case this can be caused by selecting a function class too complex, such as a higher order polynomial than the real function. Cross validation is a technique that can be used to detect this, first we split the data into training and validation sets, then fit a function to the training data, and finally measure the SSE of the validation data when using the fitted function, giving us the cross validation error [3].

K-Fold cross validation is a method of applying cross validation repeatedly across different combinations of training and validation data. To do so we split our data into K equally sized chunks, for $1..K$ each chunk is used as a validation set, and the other chunks form the training set [4]. This has been implemented by the `Segment split()` method which returns a list of unique training/validation pairs of length K. This in turn is called by the `cross_validated()` method which computes the mean validation error (CVE) for each of the pairs.

3 Method

The `compute()` function is used to find the best fitting function for a segment out of three choices: a linear function, a polynomial of given degree, or a specified custom function. To prevent overfitting it selects the function with the minimum k-fold cross validation. The importance of this is demonstrated by figure 1, which shows that if we just use the minimum SSE, a higher order polynomial will be incorrectly favoured instead of a linear function; due to the noise in the data.

4 Testing

To verify that the mathematical calculations have been implemented correctly, I have written tests using the `unittest` framework from the Python standard library [5]. I obtained test data for the regressions and sum square error using a *TI-Nspire* graphing calculator. There is complete test coverage of the `Segment` class and `compute()` function.

5 Training Data

The final part of the task was to establish which polynomial degree and which unknown function were used in the training data. To do so I have written the `evaluate_training_data()` function, which can be executed by running the script with the `--evaluate` switch. I have defined a range of polynomials from quadratic to sixth degree, and a range of candidates for the unknown function including *sin*, *cos*, *tan*, *reciprocal*, *exp*, *log*, *square* and *sqrt*. The evaluation iterates over all possible combinations of polynomial and unknown function, applying the `compute()` function to all training segments, totalling the cross validation error from each segment.

The results in ascending order were as follows:

```
Total CV Error: 1364.7933944444837, Polynomial Degree: 2, Function: sin
Total CV Error: 7417.386188226556, Polynomial Degree: 3, Function: sin
Total CV Error: 92417.36032612705, Polynomial Degree: 3, Function: cos
...
```

I therefore believe that the quadratic polynomial and *sin* function were used when generating the training data. I have defined them as constants which will now be used when script is run on a `csv` file.

References

[1] E. W. Weisstein, *Least squares fitting–polynomial*. [Online]. Available: `http://mathworld.wolfram.com/LeastSquaresFittingPolynomial.html`.

[2] ——, *Least squares fitting*. [Online]. Available: `https://mathworld.wolfram.com/LeastSquaresFitting.html`.

[3] L. Aitchison, *Lecture 2: Overfitting, regularisation and cross-validation*. [Online]. Available: `https://github.com/LaurenceA/intro_lectures/blob/master/Lecture_2.ipynb`.

[4] H. Tibshirani, *K-foldcross-validation*, Feb. 2009. [Online]. Available: `http://statweb.stanford.edu/~tibs/sta306bfiles/cvwrong.pdf`.

[5] T. P. S. Foundation, *Unittest — unit testing framework*. [Online]. Available: `https://docs.python.org/3/library/unittest.html`.