Data Driven Computer Science CW: An Unknown Signal

Armand Cismaru

1 Linear regressions equations

The technique used for fitting the functions for all segments is the least squares regression:

$$W = (X^T.X)^{-1}.X^T.Y \tag{1}$$

The form of X changes depending on the fitted function, also changing the form of resulting matrix W. Thus we have:

1. Linear function:

$$y_i = a + b * x_i \qquad W = [a, b] \qquad X = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ 1 & x_3 \\ ... & ... \\ 1 & x_n \end{pmatrix} \tag{2}$$

2. Polynomial function of order 3:

$$y_i = a + b * x_i + c * x_i^2 + d * x_i^3 \qquad W = [a, b, c, d] \qquad X = \begin{pmatrix} 1 & x_1 & x_1^2 & x_1^3 \\ 1 & x_2 & x_2^2 & x_2^3 \\ 1 & x_3 & x_3^2 & x_3^3 \\ ... & ... & ... & ... \\ 1 & x_n & x_n^2 & x_n^3 \end{pmatrix} \tag{3}$$

3. Unknown function (sinus):

   Analogous to the linear function, instead of x_i there would be $sin(x_i)$.

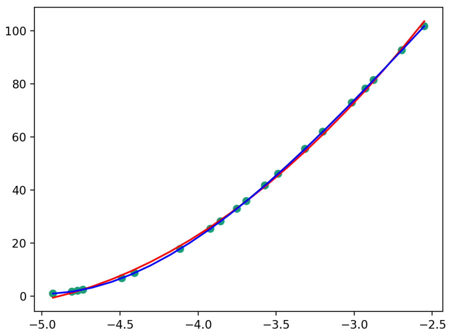
2 Choice of polynomial order

The main method chosen for finding the order of the polynomial function is k-fold cross-validation. The cross-validation error is obtained by using the mean of the squared-error function that calculates the distance between the ŷ fitted from the model and y_{test} points from the test set. The fitted model is calculated using the aforementioned least-squares method on segments chosen as polynomial functions.

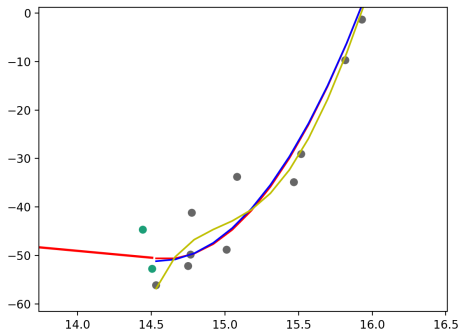
To obtain accurate data k is set to 10 and the seed used is constant. The tests have been run on different orders of the polynomial function. The average cross-validation errors across all test files on orders 2, 3, 4 and 5 are: 5.883, 5.011, 8.091 and 19.392. To rule out bias, data files containing a lot of noise were excluded from the tests. The general intuition is that very noisy data can lead to very similar values for higher-orders and thus being not very accurate. It is safe to rule out orders higher than 3, but the gap between orders 2 and 3 is narrow.

This method is not perfect, as some segments might show a smaller error for order 2 than for order 3, but with a ratio close to 1. Such an example is in adv.3, where the second segment has an cross-validation error of 12.831 for order 3, and an error of 11.280 for order 2, thus a ratio of 1.13.

To see a clear example of the difference between order 2 and 3 we need to look at a less noisy set of data. For basic.3, we get an error of 1.357 for order 2 *(Fig.a)* while the error for order 3 *(Fig.b)* is $1.039 * 10^{-18}$, which makes it more than clear that the function used to generate the data is a polynomial with an order of 3 across all files. Plot (a) below (basic.3) shows the difference of fit between the two orders, with order 3 clearly fitting the points better, safe of overfitting, as cross-validation ensures. Plot (b) captured in noise.2 illustrates the orders 2 and 3 being very close, as mentioned before, and a notable overfit of the order 5 polynomial.



(a) Order 3 in blue, Order 2 in red



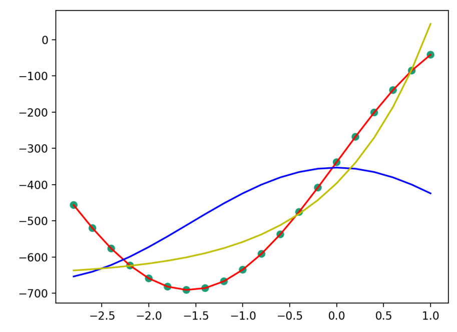
(b) Order 2, 3, 5 in red, blue and yellow

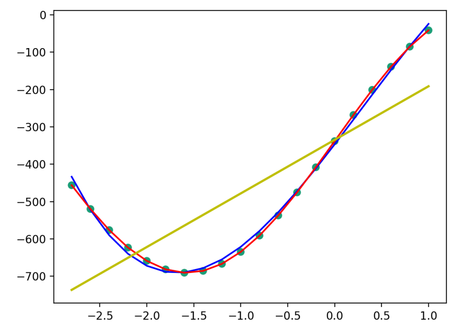
3 Choice of unknown order

The procedure for choosing the unknown function is based on k-fold cross-validation, similar to choosing the polynomial order. The tested models were fitted in the same way as the linear ones, with the single difference that instead of having x's on the second column of the X matrix, the sinus, cosinus and exponential functions were used instead.

The starting point was basic.5, having a very large reconstruction error for the polynomial of order 3 (already determined) - 2612.975, so it hinted to being the unknown function. Plot (b) below illustrates the misfit of the polynomial (blue) and linear (yellow) functions. The sinus (red) was added after determining it for the purpose of demonstration. Eyeballing the plot supports the assumption.

Testing the potential unknown functions between each other in different files has led to cross-validation errors being outstandingly in favour of the sinus function, as plot (a) below illustrates (captured in basic.3; sin - red; exp - yellow; cos - blue). Based on the empirical evidence and from the observation of the plots it was confident to assume sinus as being the unknown function.


(a) sin, cos ,exp


(b) linear, polynomial of order 3, sin

4 Procedure of selecting between functions

4.1 General idea

In order to mitigate overfitting, the k-fold cross validation procedure was chosen. Of course, for choosing between the linear, polynomial or unknown function you can just look at the reconstruction error, which calculated as the squared sum of the differences between the fitted model and the initial points on the y axis, but the data would be inaccurate due to overfitting. So, the choice of function type relies solely on k-fold cross-validation.

The procedure is the following: A k is chosen that will dictate how many folds will be used for training/testing the data. The choice of k will be discussed in the following paragraphs. The data sets are shuffled, and then each data set will be trained/tested in the typical k-fold CV technique. The train/tests sets will then be fitted against each of the linear/polynomial/unknown functions using the least-squares method and will return the mean squared error between the ŷ fitted from the model and y_{test} points from the test set.

After finishing all the folds, the program will have the mean of the cross-validation error for each of the 3 functions. The type of function is decided by the minimum of these errors, for example if the error for linear fitting is the lowest, the segment will be labeled as so. This method works really well for noisy data, even though it has its flaws.

4.2 Fixed vs Randomness

The first attempt to mitigate overfitting was by holding-out 20 points as 17:3 training/testing. Whilst the results were better than the reconstruction error, the main issue of the hold-out method was failing to differentiate between clearly linear functions (such as basic.1 or noise.1) and the polynomial function on some test cases. So the general intuition demanded an improved approach, to accurately detect and mitigate overfitting, especially for noisy data.

So, k-fold CV was brought to help. The data sets are first shuffled in a pseudo-random way using a seed, and then split into k groups. This ensures that each group gets the chance to be both used for training and testing. Averaging the CV errors using the mean sum-squared function gave considerably better results compared with the the hold-out method where choosing a fixed set of points might fails to mitigate bias and noise.

Choosing a seed means impacting the results. Even if a seed is just a way of shuffling the data, changing it is very sensible. So, the seed used needs to be constant across all observations, otherwise the program will give a random one, thus making the data volatile and inaccurate.

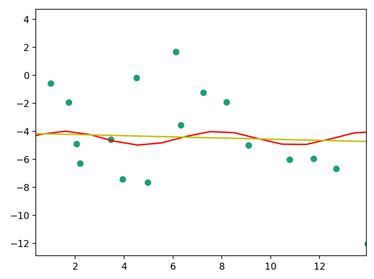
4.3 Accuracy vs Efficiency

The line between being accurate and being efficient at the same time is very thin, especially on small sets of data. The general paradigm is that the more data you use for training, the more accurate the analysis will be. The first round of cross-validation was done using k = 5. This is an efficient approach for larger sets of data, while still being accurate. But testing with k = 10 made it clear that trading efficiency for accuracy is worth it, at least for small sets.

To illustrate a clear case, in adv.3, for the fifth segment, the CV error for the unknown function with $k = 5$ was 9.578 while for $k = 10$ it went down to 8.884. Another big difference was spotted for the fourth segment, a linear, but for the polynomial fit: from 65.636 to 7.960. But, for the average, the empirical evidence shows error gaps closing on bigger k's.

One more way to test is leave-one-out k-fold cross-validation, but it comes at a big efficiency cost, making the time price/accuracy ratio not worth it for very big data sets. Still, it brings positive outcomes for smaller data sets, but not very different from having k = 10, that's why the choice was made for 10 folds, even if it can mean that certain more accurate values are lost. On very noisy data sets, testing on only one point per fold may be subject to bias, thus diminishing the efficiency of the procedure. For example, in noise.1 (linear segment), the leave-one-out k-fold produced an error of 0.76 compared to 0.73 for $k = 10$. However neither are perfect, and choosing one in account for the other doesn't affect the ability to choose between the functions (with small exceptions), but some values might be extremely close (some polynomials vs unknown or linear vs polynomials).

4.4 Not everything is perfect



Even though the results from using k-fold cross-validation are satisfactory, there is the case of certain slip-ups. Due to the relative nature of the shuffling, results can vary quite significantly depending on the seed. Or, as mentioned above, using the leave-one-out method might alter the results. But still, they produce accurate models. On the other hand, in basic.2, using hold-out identifies the first segment as polynomial whilst using k-fold sees it at linear, which is correct. In the plot to the right (adv.3), there is one interesting case where all methods deem the segment to be sinus, even though it appears to be linear. The noise makes the points very dispersed and the k-fold method interprets them as fluctuating in a periodic manner. The difference between the CV errors for the linear and sinus functions is very slim, the first being 13.859 and the latter 13.383. Even though failures exist, the success/failure ratio of the procedure is very high, thus minimising the importance of the flaws.

4.5 Fine tuning

While the used method ensures that the correct function is chosen, it is not applied to the final fitting. The models on which the reconstruction error is calculated using the sum squared function are fitted from the whole set. For very high reconstruction accuracy, cross-validation can be used to give the best fitting parameters of the function. The program can be modified such that you get the model with the lowest cross-validation error, after gathering data from the k-folds. In cases with little data (as here) this doesn't make a big difference (or any at all).