Report – An Unknown Signal

The general equation for linear regresion aiming to minimize the sum squared error and to determine the line of best fit for each data set is the following:

XE = (XT .X)-1. XT .Y, where XE is the extended matrix, X is the matrix with input points and Y is the matrix with true outputs. The creation of the extended matrix depends on the type of function we want to find the line of best fit for.

For a linear function, the estimated parameters that the least squares method should return are the gradient and the offset of the fitted line. Therefore, the extended matrix should be composed of a column consisting of ones and a column of x’s.

For the polynomial function, least squares method returns estimates for the coefficients of the polynomial. Because the order of the polynomial is unknown, I consider the order of the polynomial to be at least two (otherwise it would be a linear function). After adding the same columns as for the linear function, I keep on adding to the extended matrix columns consisting of X to the power of the current order of the polynomial. Least squares method returns a list of estimated coefficients for the fitted line of the polynomial.

The unknown functions that I have taken into consideration are sinus, exponential and cosinus. The extended matrices for these kind of functions consist of a column of ones, and a column of the function type (sin(x), cos(x), e^x). Only two parameters will be returned for the unknown functions.

In order to compute the output (the fitted line) based on the function type I used a limited sample of testing points in order to estimate how the model is expected to behave when making predictions on data that does not serve as training purposes. I evaluate the model on the test set and fit it on the training set.

The equations for the fitted lines are the following:

For the linear function it is: slope \* test\_data + offset, where slope and offset are the results computed by least squares method.

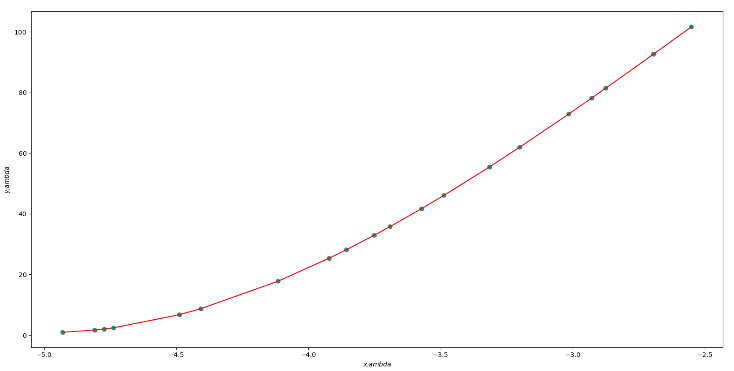
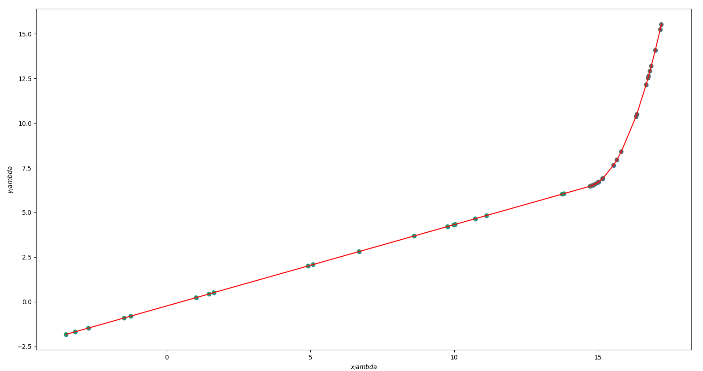
For the polynomial the equation is: ∑ coefficients[i] \* test\_data \*\* i, with i in range between 2 and the required degree of the polynomial determined by squared error and k-fold validation. “Coefficients” is the list of estimates computed via least squares for polynomial.

For the unknown functions, the equation is of the form: (first estimated parameter) + (second estimated parameter) \* (function type).

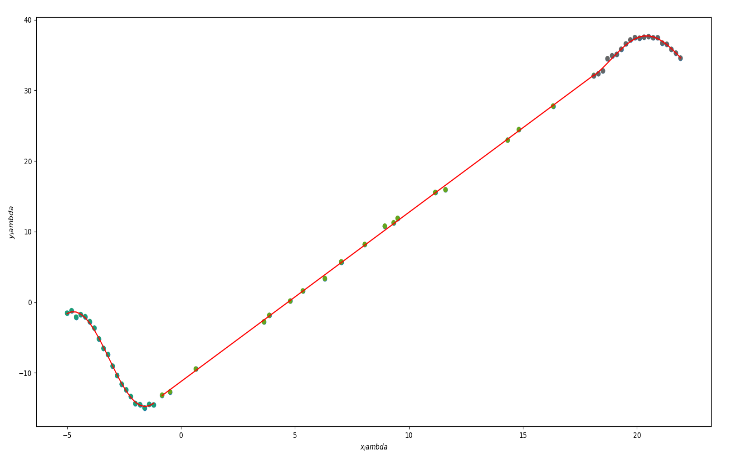
In order to find the order of the polynomial, in the section where I compute the errors for each chunk of 20 data points, I create a list of squared errors for each degree of the polynomial. The range of degrees for the polynomial that I considered is between 2 and 6. As a general observation, for only 20 input points, the degree of the polynomial has to be small, otherwise the amount of overfitting would be large. I traverse the list of errors and I pick the order of the polynomial to be the one corresponding to the index at which the smallest square error is found. The smallest errors for the polynomial I obtained over all data sets were for degrees two and three. The data sets that are the most relevant for choosing between order 2 and 3 are “basic\_3” and “basic\_4”. Here is a table picturing the errors for each order of the polynomial for these input data:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Inputs/Degree | 2 | 3 | 4 | 5 | 6 |
| basic\_3 | 6.43E+02 | 2.91E-16 | 4.57E-12 | 7.55E-08 | 3.77E-03 |
| basic\_4 | 2.53E-01 | 5.19E-10 | 8.46E-03 | 7.72E+07 | 7.95E+03 |
| Average | 321.554903 | 2.593E-10 | 0.0042293 | 38612760 | 3976.297 |

This clearly pictures that the suitable degree for the polynomial is three.

basic\_3 basic\_4

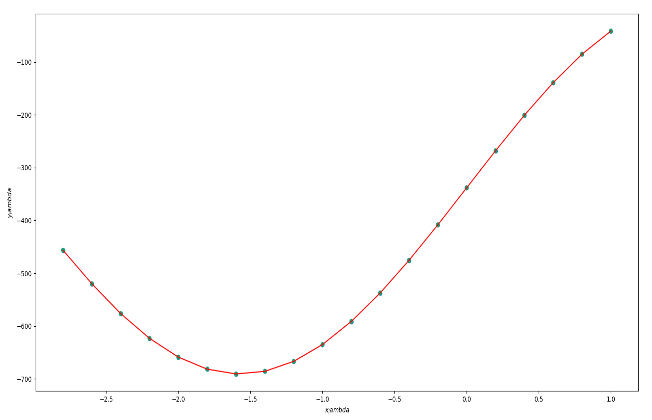
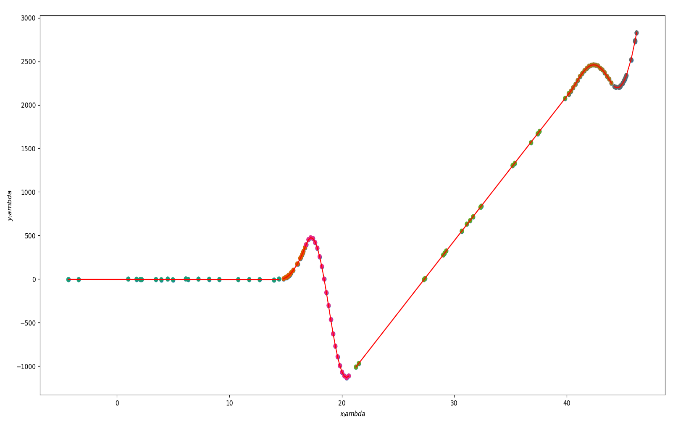


adv\_2

Here is a table envisaging the errors for the unknown functions I considered:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Inputs/ Unknown function | basic\_5 | adv\_2 - 1st function | adv\_2 - 3rd function | adv\_3 - 3rd function | adv\_3 - 5th function | noise\_3 - 3rd function | Average |
| sinus | 2.32E-23 | 32.1759644 | 41.08551391 | 6787.30893 | 4456.602882 | 6769.835732 | 3.01E+03 |
| cosinus | 22924486.26 | 14777.25172 | 1458.545311 | 265996516 | 5399252.636 | 2429313.131 | 4.95E+07 |
| exponential | 3600282.764 | 4687.564782 | 1682.468532 | 55653459.67 | 6323399.82 | 1975407.211 | 1.13E+07 |
| linear | 2956196.137 | 158.4634638 | 321.670487 | 2352516.235 | 1231403.059 | 645494.2405 | 1.20E+06 |
| polynomial | 14.45918428 | 51.22907002 | 47.11922057 | 68257.76247 | 65588.22029 | 12125.82316 | 2.43E+04 |

This emphasizes the fact that the unknown function is sinus.

basic\_5 adv\_3

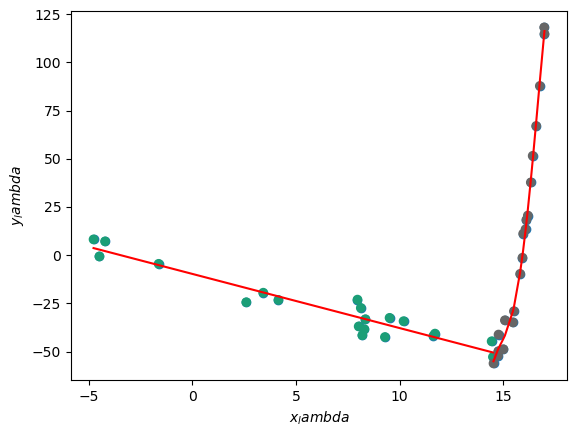
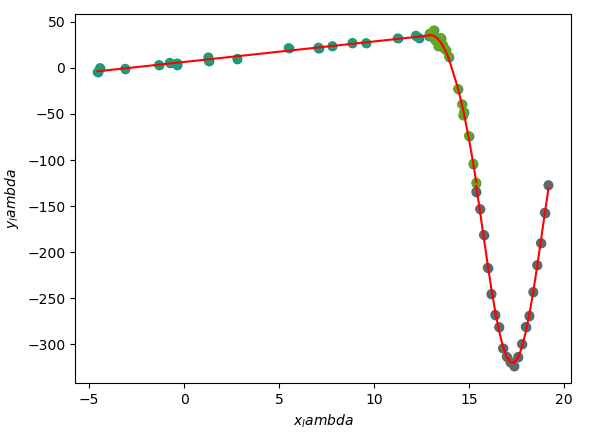
In order to estimate the skill of the model for each data set, I acquired test and data points for an implementation of a k-fold validation. I considered 5 points of test and 15 of train, my choice being an implementation of 5-fold cross validation. The parameter “step” is increasing by 5 at each iteration for each portion of 20 points (each function). As a consequence, for the first iteration, the first 5 points are test points and the last 15, train points. For the second iteration, points starting from the fifth up to the tenth will be the test points and the others, the train points and so on. Therefore, the error for each type of function and for each degree of the polynomial is computed and summed four times for each choice of test and train points in turn. This is why I divide each error by four before deciding what function describes each chunk of data.

After splitting the datasets into chunks of 20 data points to delimit each function I shuffle these datasets randomly in order to avoid choosing the points for test and respectively training in a biased manner. The function I used to do this is “np.random.permutation()” as it keeps an association between input and output points. Because I shuffle the x and y coordinates for the k-fold validation, the errors will end up being slightly different and, in the end, before choosing the suitable function for the 20 points data set, I take the mean of these errors. The reason for this approach is to compensate for the randomness.

The errors are calculated using the square error formula which is a sum of offsets / residuals from the plotted curve that need to be minimized. These are minimized thanks to the least squares method. The lines are fitted based on the estimated values for the parameters for each function type. Based on these errors, and on the errors for each order of the polynomial, I can decide the type of the function that best fits the current chunk of data. The smallest error corresponds to the suitable function. Then, I can fit the line according to the function type. The fitted line is computed using the test points.

I have also tested the model using 10 points as test (a version of 10-fold cross validation). The values for the total reconstruction error were very similar. I fitted the line using 10-fold cross validation as well to check if a bias-variance trade-off discussion would be appropriate. The data set for which I obtained different reconstruction errors is “noise\_2”.

|  |  |  |
| --- | --- | --- |
| K-fold cross validation | 5-fold cross validation | 10-fold cross validation |
| Reconstruction error for "noise\_2" | 782.2539795 | 850.9013646 |

noise\_2 noise\_3

The following table illustrates the total reconstruction error for each dataset:

|  |  |
| --- | --- |
| Input | Total reconstruction error |
| basic\_1 | 1.68E-27 |
| basic\_2 | 6.47E-27 |
| basic\_3 | 2.92E-18 |
| basic\_4 | 2.76E-13 |
| basic\_5 | 2.50E-25 |
| adv\_1 | 244.1104051 |
| adv\_2 | 3.68513205 |
| adv\_3 | 1014.409779 |
| noise\_1 | 12.20746014 |
| noise\_2 | 876.9205194 |
| noise\_3 | 483.0638008 |

We can see that for noisy data sets, inspite of having few functions that these data points describe, the reconstruction error is quite large.