Data Driven Computer Science Coursework Report

An Unknown Signal

Nisa Bayraktar(da19157)

1 Introduction

1.1 Least Squares Method

The Least Squares Method (LSM) is a method which calculates the best fitting line for a given dataset. The fitted line is called regression line and it helps us to estimate y, for a given value of x.

$$\hat{y} = a + \mathrm{b}x$$

To determine the most accurate line, we need to find the smallest Residual Sum of Squares (RSS). The error is determined by calculating the difference between the predicted and actual values of y ($\hat{y}$ and y respectively).

$$e = (y_i - \hat{y}_i)$$

Then, summing the squared errors (residuals) gives us the value of RSS.

$$R(a,b) = \sum(y_i - \hat{y}_i)^2 = \sum(y_i - (a + bx_i))^2.$$

1.2 LSM for Non-linear Functions

In this coursework, the LSM is extended to calculate non-linear functions. To do that, matrix form is used with a column vector Y for y coordinates, and matrix X for the x coordinates. X is a $nx2$ matrix which only contains 1s in the 1st column and x coordinates in the 2nd column. To calculate higher degrees for non-linear functions, it extends after the 2nd column and changes its shape to an nxm matrix.

$$Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}, X = \begin{bmatrix} 1 & x_1^1 & \dots & x_1^m \\ 1 & x_2^1 & \dots & x_2^m \\ \vdots & \vdots & & \vdots \\ 1 & x_n^1 & \dots & x_n^m \end{bmatrix}$$

Coefficient vector is calculated with,

$$A = (X^T x\, X)^{-1}\, x\;\; X^T\, x\, Y$$

The shape of A depends on the X's shape. When we multiply the matrix A by matrix X, we can calculate our non-linear line. (coefficients: a, b, c…m)

$$\hat{y} = a + bx + cx^2 + \cdots + mx^m$$

1.3 LSM for Unknown Function

For the unknown function we change matrix X with,

$$X = \begin{bmatrix} 1 & \sin(x_1) \\ 1 & \sin(x_2) \\ \vdots & \vdots \\ 1 & \sin(x_n) \end{bmatrix}$$

The coefficients are calculated with the same function as above and the regression line is going to be,

$$\hat{y} = a + b\sin(x)$$

2 Analysis

2.1 Implementation

Linear, non-linear, and unknown functions can be determined in the program. It uses functions to calculate extended X for polynomials (linear and non-linear), and for the unknown function. The total reconstruction error is calculated by summing RSS results for each segment together. To determine the function type and choosing the most accurate polynomial order the program calculates cross-validation error.

Dataset	Total Reconstruction Error
basic_1	4.65427934080397e-28
basic_2	3.1814760307563406e-27
basic_3	4.403509376325771e-19
basic_4	2.242927220844857e-11
basic_5	2.5364718002114325e-25
adv_1	198.23214726221235
adv_2	3.650864592182417
adv_3	979.0067252279254
noise_1	10.985055464509516
noise_2	797.916656822003
noise_3	477.6993381193355

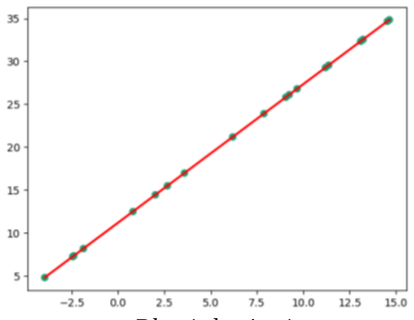
Table 1: Total Reconstruction Errors with linear(m=1) or polynomial order (m= 3)

Dataset	m=1	m=2	m=3	m=4
basic_1	1.00974e-29	5.1181e-27	2.5415e-23	5.7564e-23
basic_2(s:1)	4.29140e-29	1.8069e-26	2.3108e-24	7.7394e-22
basic_2(s:2)	1.00558e-23	4.8944e-17	3.54373e-11	415.84119
basic_3	509.680227	70.369517	2.7753e-13	1.465e-08
basic_4(s:1)	1.7749e-30	2.7738e-27	9.7014e-28	1.3503e-20
basic_4(s:2)	9.833488	0.056914	0.0041638	0.1499043

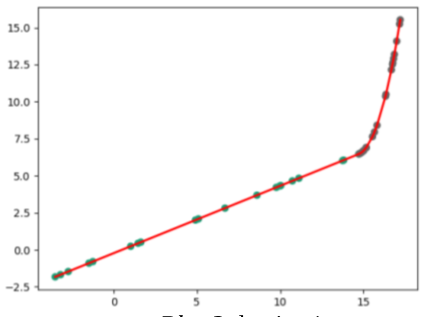
Table 2: Cross-validation error for each segment (m:order, s: segment)

2.2 Results

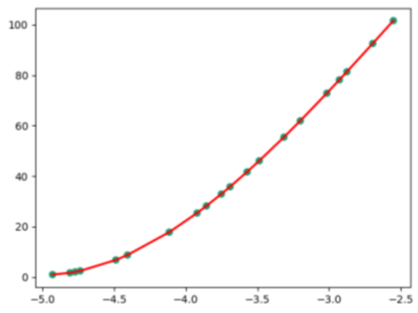
We can see in the *Table 1;* the *basic* datasets have the smallest error. So, it is obvious that they have almost no noise and have the most accurate results. To prove that, we can look at the results in *Table 2* and compare them with plots.



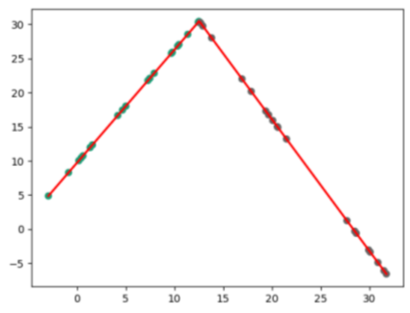
Plot 1: basic_1



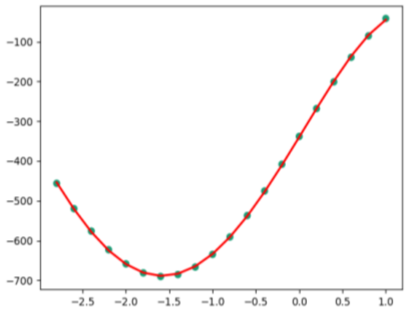
Plot 2: basic_4



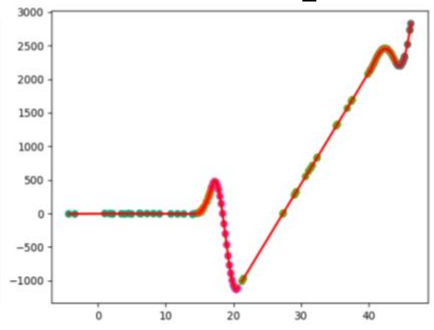
Plot 3: basic_3



Plot 4: basic_2



Plot 5: basic_5.



Plot 6: adv_3

The highlighted numbers in *Table 2* show the smallest cross-validation error for dataset segments. We can see that *basic_1* has the smallest error in order 1 and *Plot 1* has a linear line, same with *basic_2* for both segments. So, results from errors match with plots.

2.3 Choosing the Unknown Function

Basic_5(Plot 5), helped us to choose unknown function. Because the errors that we get by using linear and polynomial orders for *Basic_5* are very large and *basic* files are the most accurate files to determine the functions. After tyring different types of function (sin, cos, log, exp etc.), the smallest error that we get for *basic_5* is ***sine*** function. By calculating cross-validation error of sine function, in the last segments of the datasets adv_2, adv_3 and noise_3 get smaller error then before (calculating the error with polynomial order) and the program chooses the unknown function. This shows that, our unknown function is correct.

3 Overfitting and Cross-validation

3.1 Implementation of Cross-validation

To find the most accurate results and avoid overfitting, the program uses cross-validation. Data segments (total 20 datapoints in each segment) are divided into 2 parts as training (10 datapoints) and validation set (10 datapoints). Cross-validation error is calculated with using Mean Squared Error (MSE) which takes the average of squared errors. The error is defined for both unknown function and polynomial function. The program chooses the smallest error by comparing the unknown and polynomial function errors. So that, we can determine the function type for each segment of the dataset.

3.2 Finding the Fixed Polynomial Order

As all the polynomial orders are the same in all datasets, we can determine the most accurate fixed polynomial order by using the *basic* datasets that contains polynomial function and have almost no noise. The cross-validation error gives the smallest error with ***order 3*** in the datasets *basic_3* and the 2nd segment of *basic_4*(*Table 2*). When we plot the *basic_3* and *basic_4* we can easily tell that cross-validation error results are accurate and order 3 is the fixed order. Because *basic_3* (*Plot 3*) plots a cubic function with a fitted line, and basic_4(*Plot 2)* plots a linear line in the 1st segment and cubic in the 2nd segment and has the smallest error in order 1 and 3 respectively.

	seg 1	**seg 2**	**seg 3**	**seg 4**	**seg 5**	**seg 6**
m = 1 (linear)	16.6	14987.3	260654.8	13.1	106745.2	79622.4
m = 3 (cubic)	46.4	2617.0	17111237.2	59357.1	40891.1	77397.0
u(sine)	14.2	13161.6	20.82	2329321.7	11.1	98668.2

Table 3: Cross-validation errors for adv_3 in each segment (seg: segment, m: order, u: unknown function)

	seg 1	**seg 2**	**seg 3**	**seg 4**	**seg 5**	**seg 6**
m = 1 (linear)	212.0	8753.8	356857.4	128.6	156842.0	90675.3
m = 3 (cubic)	184.8	167.43	321940.6	116.0	2589.7	130.1
u(sine)	210.3	12748.3	235.16	14258828	142.4	654045.1

Table 4: RSS errors for adv_3 in each segment (seg: segment, m: order, u: unknown function)

3.3 Explaining the Overfitting

Before implementing the cross-validation in the program. As it shown in the highlighted points of the *Table 4* the smallest errors that we get with RSS for each segment in *adv_3* refers to ***cubic,cubic,sine,cubic,sine,cubic (***per segment in order) function types. But it is obvious that *adv_3* has a linear function type in segment 4, we can see that in *Plot 6*. This means that we were overfitting these datasets. When cross-validation error is calculated, the function types change to ***sine,cubic,sine,linear,sine,linear*** (per segment in order)according to new error results. These results prove the accuracy of the cross-validation and how cross-validation prevents overfitting.

4 Possible Improvements

The program uses very basic cross-validation implementation known as train-test split. It divides the segments in 2 parts (10:10) to train the model on training set and the use test set for validation error. With train-test split method, training set doesn't go through all the data, it misses some information about the unused data which might cause high bias. We can prevent this with using k-fold cross-validation. K-fold cross validation splits the data into k folds, then trains the model with k-1 folds and uses remaining folds for testing. It iterates the number of folds by changing the testing and the training points each time and take average of the errors. Iterating the points and changing them each time give a chance to every datapoint to appear in the testing and training sets. Therefore, it has a less biased model compared to other cross-validation types.

5 Conclusion

In summary, after acknowledging the program can be improved, we can say that, by using Least Squares method, the correct function types can be found (linear, cubic, sine) to determine the best fitting model and using cross-validation is a good way to handle overfitting.