Data-Driven Computer Science Coursework

An Unknown Signal

Vlad Andrei Bucur (ot19588)

The goal of the project is given a set of points which follow an unknown signal, reconstruct the signal using only linear, polynomial with a fixed order or unknown nonlinear function and print the total reconstruction error.

Fitting Method

Maximum-likelihood / least squares regression was used to fit the functions.

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

1. **Linear function**

$$X = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ & \vdots \\ 1 & x_n \end{pmatrix} \qquad A = (a \quad b) \qquad y_i = a + b * x_i$$

2. **Polynomial function of degree 3**

$$X = \begin{pmatrix} 1 & x_1 & x_1^2 & x_1^3 \\ 1 & x_2 & x_2^2 & x_2^3 \\ & & \vdots & \\ 1 & x_n & x_n^2 & x_n^3 \end{pmatrix} \qquad A = (a\ b\ c\ d) \qquad y_i = a + b * x_i + c * x_i^2 + d * x_i^3$$

3. **Sin function**

$$X = \begin{pmatrix} 1 & \sin(x_1) \\ 1 & \sin(x_2) \\ & \vdots \\ 1 & \sin(x_n) \end{pmatrix} \qquad A = (a \quad b) \qquad y_i = a + b * \sin(x_i)$$

In order to avoid **overfitting,** I have used **leave-one-out cross-validation** (K-fold cross-validation with K = the number of data points in the set). LOOCV procedure it is considered a reliable and unbiased estimate of model performance.

This technique avoids randomness of the result – it is **deterministic** (unlike other K-fold CV which for this project would require setting a fixed random seed). It is appropriate when you have a **small data set** *(only 20 points)* and delivers an **accurate estimate** of the model performance*, using 19 points in the training data and 1 point in the test set*. However, it is **more expensive** than other K-fold CV or other train-test split - *LOOCV computes 20 times for each segment in this particular case.*

Other technique tried was the **Holdout method** which has a lower computational cost (it was training a single model), but it was very dependent on how I was choosing to split the data between the train set and test set, also training a single model would not be so reliable.

Determining the polynomial order and the unknown function

I started with analyzing which are the best fitting functions for the given data. I made use of all the files provided in the "*train_data*" folder and for each segment, using **the LOOCV procedure**, I computed **least squares regression** for the following functions:

- linear
- polynomial of order 2, order 3, order 4, order 5 (attempts for the polynomial fixed order function)
- sin, cos, e^x (attempts for the unknown function)

For each function from above I obtained 20 **SSE**s (due to K-fold) and I kept **the mean of them**. I selected the one with the smallest mean, because it was a providing a small overall error on all the folds.

- $SSE = \sum_i (\hat{y}_i - y_i)^2$
- $\min\,(mean(SSE1_{linear}, \ldots SSE20_{linear}), mean(SSE1_{poly2}, \ldots SSE20_{poly2}),\ mean(SSE1_{poly3}, \ldots SSE20_{poly3}), \ldots)$

Here is the choice of distribution of all the 25 segments provided:

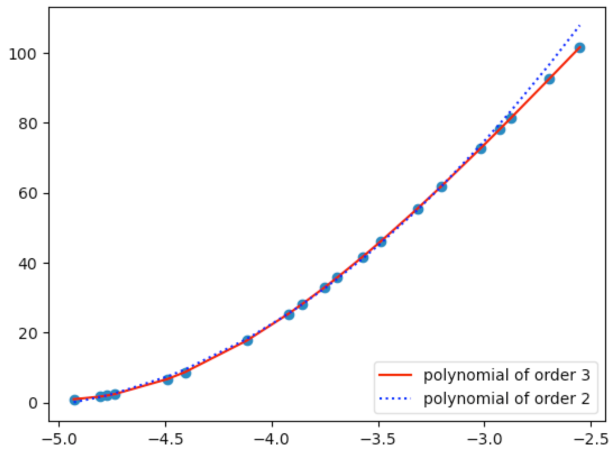
Linear	Polynomial order 2	Polynomial order 3	Polynomial order 4	Polynomial order 5	Sin	Cos	e^x
10	5	3	0	0	6	1	0

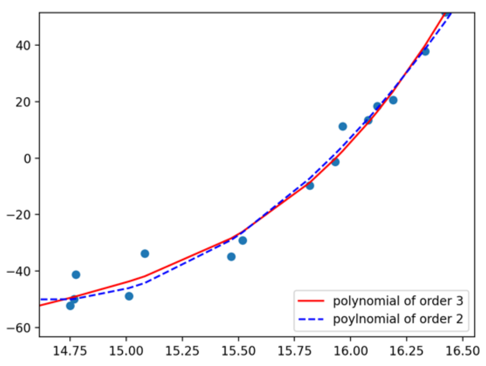
Choosing the polynomial order

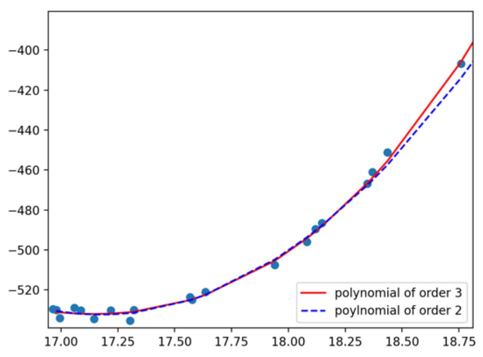
It is obvious that LOOCV is effective because it avoids choosing a higher polynomial order which can lead to overfitting. Polynomial of order 4 and 5 are not taken into consideration anymore, because they are not preferred at all by the given datasets (their error is bigger than the ones provided by order 2 or order 3).

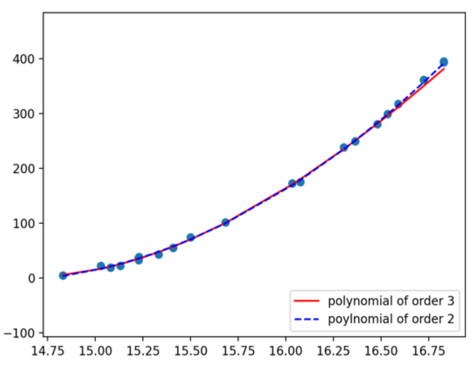
Now, we need to take a decision between polynomial of order 2 and order 3.

From "basic_3.csv", it is easy to predict by eyeballing it and comparing the reconstruction errors (for **polynomial order 2** –> *66.41*, for **polynomial order 3** -> *1.45 * 10^{-15}* – the order 3 error is way smaller!)









In the above examples we can see that both polynomial functions follow similar trends, so we need to look more in depth to come up with a decision. If we look at the mean errors for each order when the program decides the current segment is polynomial, we have the results presented below:

Segment Number	Mean error for polynomial of order 2	Mean error for polynomial of order 3
1	1.345718295333944	2.383661152127588e-18
2	0.0005642243641470097	4.188706169227719e-12
3	4.242579226503778	4.295889573640231
4	7.426333253000491	6.292348025293899
5	10.442567829930956	11.149971328412544
6	8.957520462861934	9.14484758298832
7	26.61421462447806	34.249881402127485
8	12.585299674326055	13.699244540483454

For segments 3 to 8 the ratio between the errors is **around 1** which means there is **not a big difference** between choosing the order 2 or order 3.

From segment 1 and 2, the ratio between the errors is very big. This leads to **a conclusion**: **the polynomial of order 3** is the function which **fits the best** based on the provided data. Moreover, the polynomial of order 2 is a subset of the polynomial of order 3 and it would be more convenient to choose the higher order as long as it does not overfit.

Choosing the unknown function

In the table with the choice of distribution of the segments for the unknown function provided on the second page the votes are in favor for the sin function.

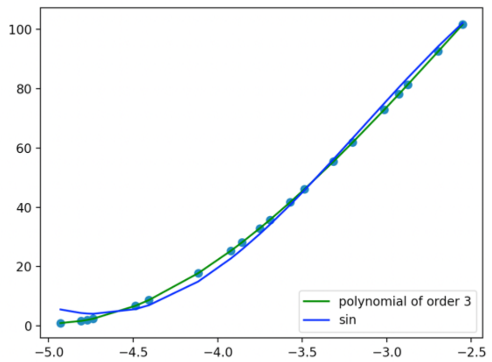


Figure 1 - basic_3.csv

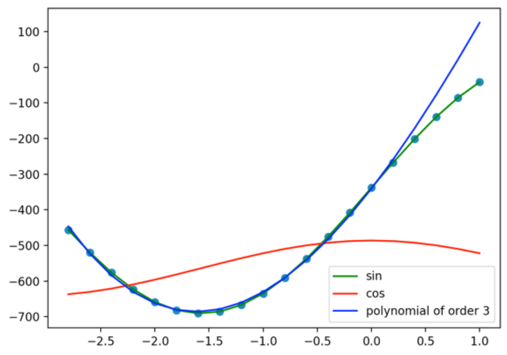


Figure 2 - basic_5.csv

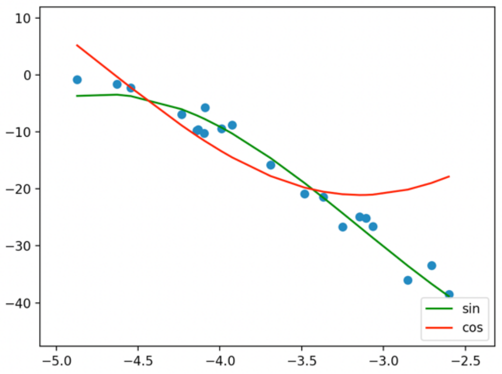
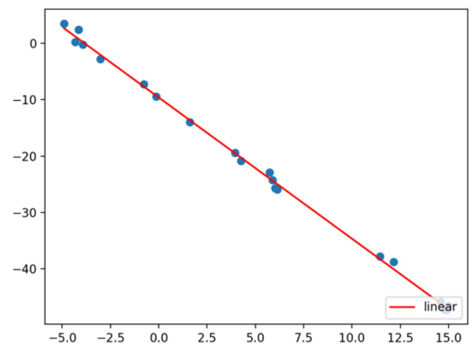


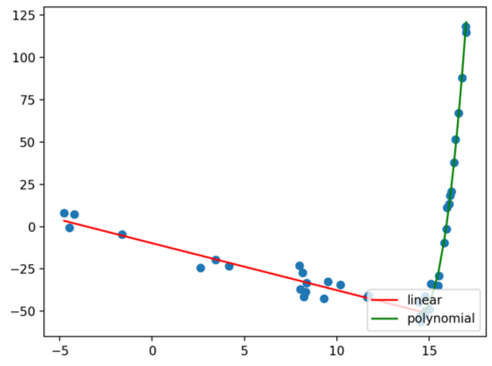
Figure 3 - adv_1.csv

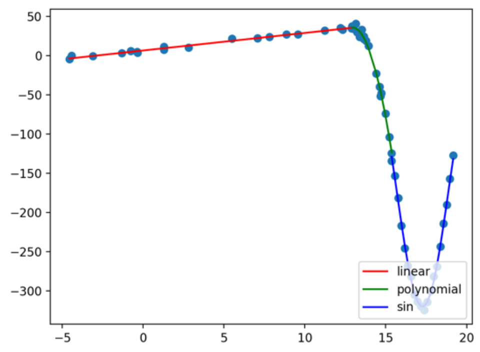
In **the first graph** the reconstruction error for polynomial is *$1.46 * 10^{-15}$* and for sin is *88*. In **the second graph** the difference is similar, so it is clear that the sin function is not confused with the polynomial of the chosen order. Moreover, we can predict again by eyeballing that there is a sinusoidal function *(basic_5.csv*).

In **the second and third figure** it can be observed that sin is the function which **fits better** than cos, obtaining smaller errors than the cos function.

Selecting between linear / polynomial of order 3 / sin function







On each segment the procedure used is **leave-one-out cross-validation** which has good results on **small data sets**, 20 points in our case. I choose the type of function which gives the **smallest mean of errors** on CV.

After that, I am training the model on all the 20 points using the corresponding least square formula. It can be observed that it handled well **noisy data sets**, the plots from above are: noise_1.csv, noise_2.csv, noise_3.csv.

Even if this technique can be **computationally expensive** (in our case it will not affect us too much because we have a small data set), it has demonstrated to provide good results on the existing "*train_data*".