DDCS Coursework: An Unknown Signal

Linear Least Squares Regression

In order to reconstruct the signals that generated the given points, I have used linear least squares regression as a method of finding the model parameters with maximum likelihood. I have assumed that the generating functions are of the form

$$y_\lambda = \vec{w} \cdot \vec{x}_\lambda + \varepsilon$$

Where y_λ is the output point, $\vec{x}_\lambda$is a vector of transformations to the input point (such as x^k, $\sin x$ *and* e^x), $\vec{w}$ is a vector of weights and ε is noise from a normal distribution. This assumption leads me to believe that each point may be modelled by the following distribution.

$$P(y_\lambda | \vec{x}_\lambda, \vec{w}) = \mathcal{N}(y_\lambda;\ \vec{x}_\lambda \cdot \vec{w}, \sigma^2)$$

Deriving the log likelihood from the distribution of all points gives rise to the following formula for the weights with maximum likelihood:

$$\widehat{\vec{w}} = (X^T X)^{-1} X^T \vec{y}$$

Where $\widehat{\vec{w}}$ is the vector of weights with maximum likelihood, X is the feature vector and $\vec{y}$ is the vector of output points. I assume that the feature vectors are of the following forms:

$$X_{linear} = \{1, x\} \quad , \quad X_{polynomial} = \{1, x, x^2, \dots, x^n\} \quad , \quad X_{unknown} = \{1, f(x)\}$$

Where $f\ (x)$ is a common basic function such as $\sin x$, e^x, $\frac{1}{x}$, etc.

Choosing the Polynomial and Unknown Functions

I used the exact same approach to choosing the unknown function class as with that of the polynomial.

Initial Observations

I first plotted the points as given in order to decide which functions to investigate.

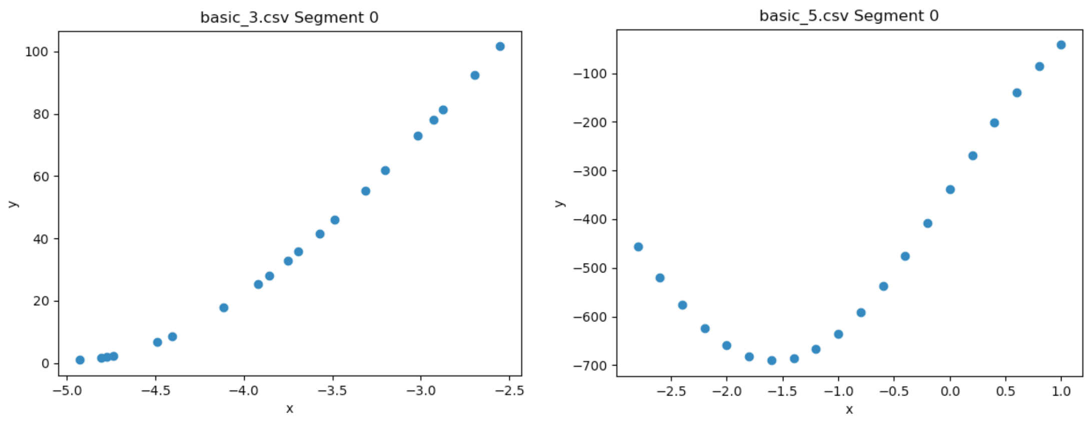


Figure 1 – Plots of the raw data for basic_3 (left) and basic_5 (right).

Some segments followed an upwards-curving path while others seemed to oscillate with at least two local maxima. I felt that I didn't have enough information to be able to decide which was the polynomial at this stage. The unknown functions I chose to consider were e^x and $\tan x$ for the curved segments and $\sin x$ for the oscillating segments.

Although I considered $\log x$, $\frac{1}{x}$, $\sqrt{x}$ and $\cos x$, visual inspection of the CSV files revealed that they contained x values that were incompatible with these functions' asymptotes (or local maxima in the case of $\cos x$). This rendered them impossible to consider while using linear least squares regression. Performing an automated test showed that there were no multiples of $\pm\frac{\pi}{2}$ among the x values so $\tan x$ was fine to investigate.

Overfitting

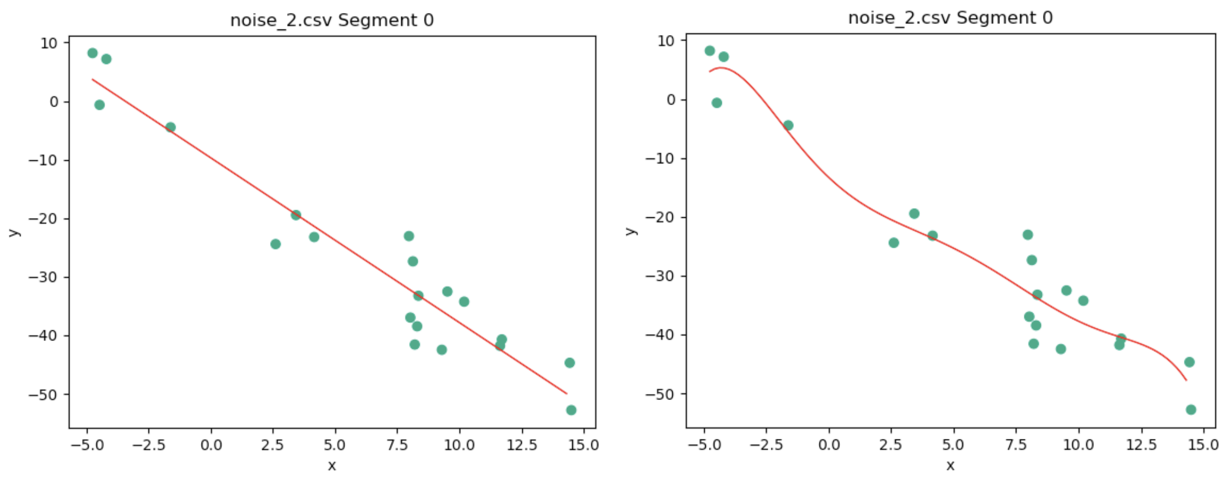


Figure 2 - The plotted points for the first segment of the noise_2 file along with the line of the fitted linear model (left) and with the line of the fitted polynomial of order 6 (right). The sum-squared errors were 471.9 and 411.2 respectively to 4dp.

As illustrated by figure 2, overfitting is evident in certain files such as noise_2. It would appear by eye that the points follow a linear trend. However, fitting a higher order polynomial to the data produces a lower sum squared error. This is likely due to the function-class of the model being overly complex and using its additional parameters to fit more closely to the given data. The problem is that if the trained model were then used on an unseen data point, it would probably be highly inaccurate.

Another potential cause of overfitting is the small sample size of the data. Although less of a problem for noise free files, the advanced and noise files may suffer from not having enough data to provide a strong enough ratio of signal to noise. As shown in figure 2, the polynomial of order 6 appears to be strongly influenced by the noise when there is little data—especially for the first four data points.

Cross-Validation

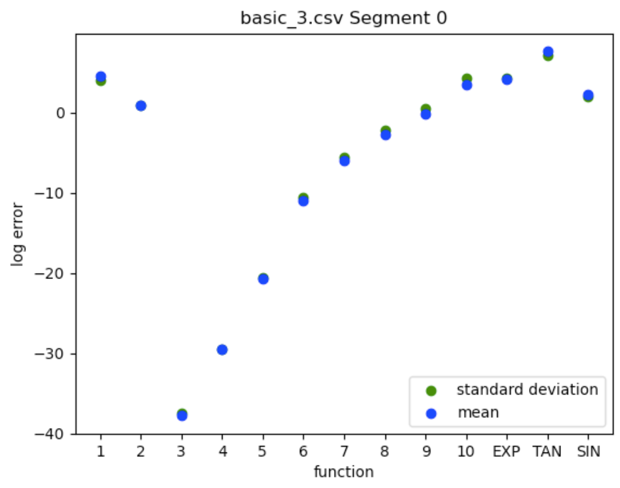
In order to combat the problem of overfitting I chose to use 10-fold cross-validation. This involved splitting the data into train and test sets so that the model could be trained on one set of data and then be tested on an unseen set. This would allow me to identify if overfitting by function-class was occurring as explained above.

The reason I opted to use a version of k-fold rather than a simpler approach was due to the limited amount of data. A leave-one-out approach would result in small test sets that don't accurately represent the sample while a holdout approach could introduce bias. I used a value of k=10 in order to have a low bias towards any particular subset, making the training subsets more reflective of the overall data. Using a factor of the sample size (20) meant that each fold would be more easily comparable since they are of equal size. The size of the test set is also more representative of the data at 10% than a leave-one-out approach at 5%. The downside to using a large value of k is a slightly larger variance of results.

The algorithm I applied is as follows:

```
1. Randomly split the segment into 10 partitions
2. For each model:
3.   For each partition:
4.     Let this partition be the test set, the rest training
5.     Use least squares regression on the training set to find the parameters with maximum likelihood
6.     Calculate the sum squared error of the test set against the trained model
7.   Return the mean and standard deviation of the errors for this model
8. Return the mean and standard deviation of errors for every model
```

Comparing Results



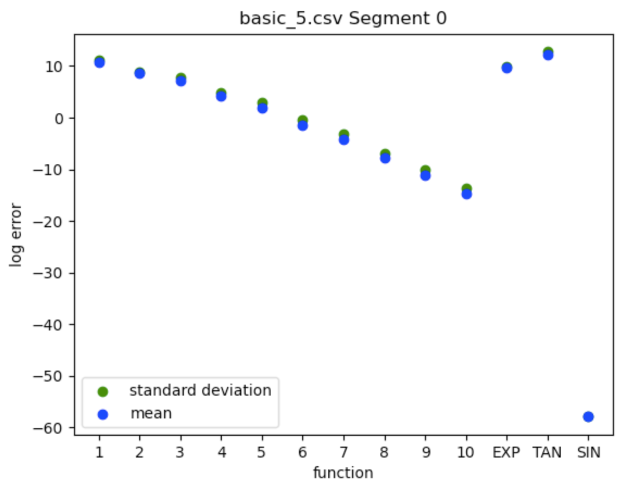


Figure 3 - The mean and standard deviations of the cross-validation errors for each investigated function on the first segment of basic_3 (left) and basic_5 (right).

I plotted the logs of the performance values returned from cross-validation for each file. Among the basic series, there were three functions that clearly performed best: order 1, order 3 and sine. Order 1 was expected since the task description stated there would be a linear function. Although I did test polynomials up to order 20, I have only included up to order 10 so that the graphs would be clear in this report. These higher order polynomials gave high cross-validation errors in all the basic and most of the advanced files.

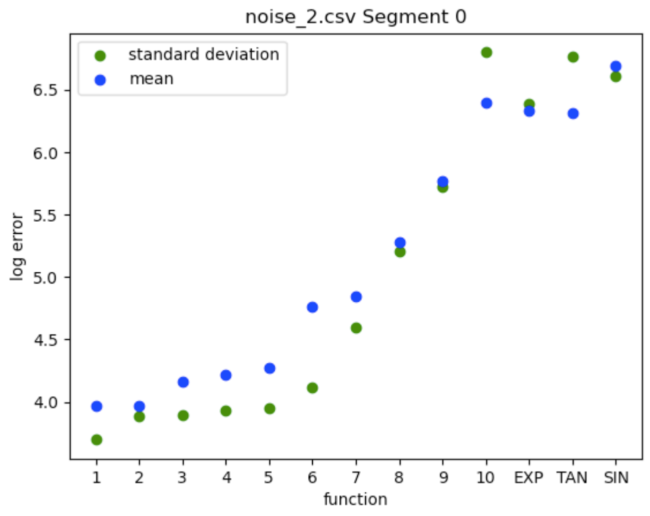


Figure 4 - Visualising the performance of the investigated functions on the first segment of noise_2. The performance is plotted as the log of the mean and standard deviation of the cross-validation errors for each model.

In the first segment of the noise_2 file, the polynomial of order 6 had a lower sum-squared error and a closer-fitting line than the linear function (see figure 2). However, both the mean and standard deviation of cross-validation errors was much higher for order 6 than linear (see figure 4), suggesting that cross-validation has been successful in overcoming the problem of overfitting.

The low mean error value of the linear model implies that it fits unseen data well. Although the mean errors are quite similar for linear and polynomial of order 2, the linear model had a much lower standard deviation of errors. This suggests that there was less variation of errors and it is therefore less likely to have overfitted to some folds more than others.

Ultimately I chose order 3 for the polynomial and sine for the unknown function.

Selecting Between the Chosen Three Models

Method

Having decided which three functions to use, I then adapted my cross-validation method to select between them at runtime. I made two significant changes to the method. The first was that I only supplied those three models to the cross-validation procedure. The second was that I used a leave-2-out cross-validation procedure instead of 10-fold.

Leave-p-out cross-validation essentially follows the same algorithm as k-fold. However in the k-fold procedure the data is randomly split into k folds, each of which is used as the test set. On the other hand, the leave-p-out procedure uses every permutation of folds of size p as the test set.

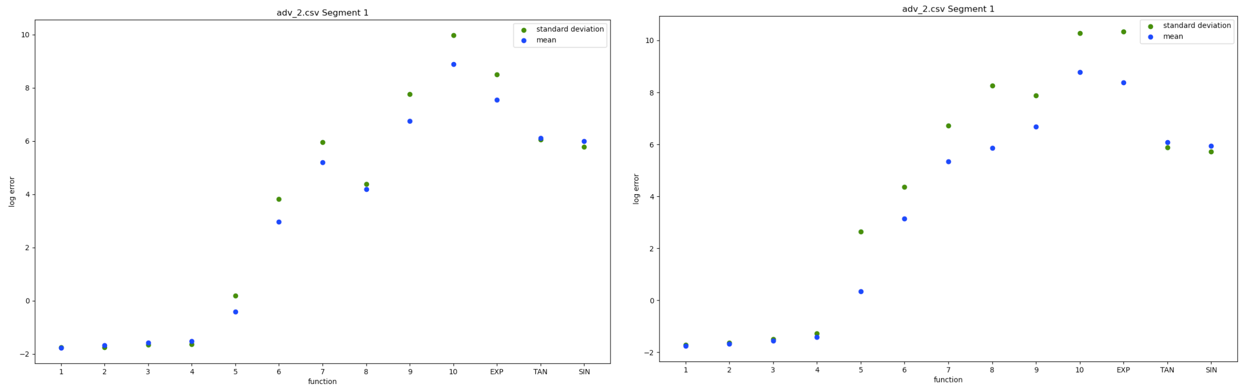


Figure 5 - Performance analysis of the functions on the same file and segment (adv_2, second segment). It compares 10-fold cross-validation (left) with leave-2-out cross-validation (right).

A leave-2-out approach seemed to offer the same advantages against cross-validation as 10-fold, but with more accuracy. As can be seen by the two graphs in figure 5, the general trend is very similar across the two methods since the 10-fold folds are a random subset of the leave-2-out permutations. The main trade-off with the leave-2-out method is that it drastically increases the time complexity of the solution since the number of folds increases from 10 in the 10-fold approach to $20C2 = 190$ in the leave-2-out approach. Because the size of each segment is so small, the increase in runtime is only a few seconds at worst. I decided that the benefit of a higher accuracy and minimal overfitting outweighed this slightly longer runtime.

After running the leave-2-out cross-validation procedure on each of the three models, the one with the lowest mean error is selected. This model is then trained on all the data using linear least squares regression and the parameters with maximum likelihood are used to fit a line to the data points.

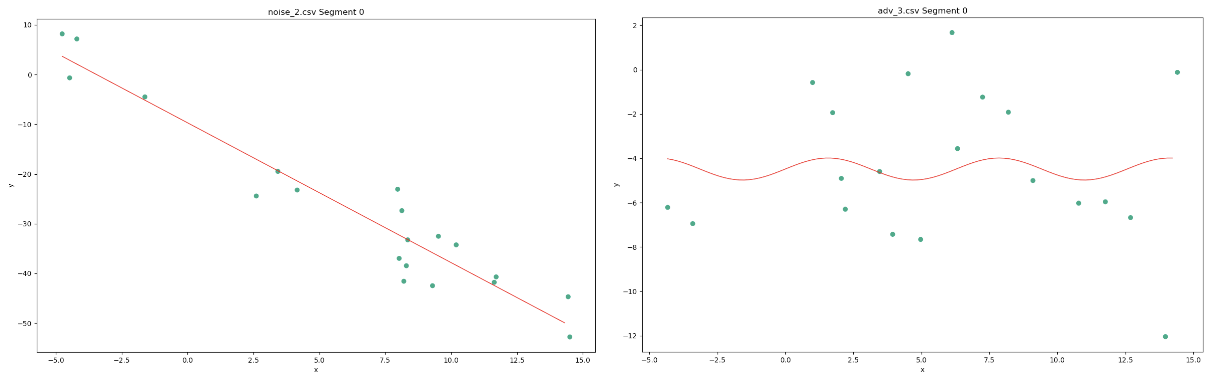


Figure 6 - Fitted lines to the first segment of noise_2 (left) and advanced_3 (right).

As shown in figure 6, the selection process was successful in choosing the linear model for the first segment of noise_3 rather than the higher order polynomial option. Although there isn't enough space to include plots for all other segments in this report, all but one plot was fitted as expected. The only unusual plot was the first segment of advanced 3 (see figure 6) where the unknown function was selected rather than linear. This is understandable due to the large amount of noise and small amount of data.