An Unknown Signal

Joshua Hall - qn19325

Equations for Linear Regression -

The method I used to perform linear regression was the matrix form of least squares. This method comes from the following equation:

$$\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{pmatrix} \begin{pmatrix} \beta_1 \\ \beta_2 \end{pmatrix} + \begin{pmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_n \end{pmatrix}$$

This can be written as $Y = X\beta + \epsilon$. In this equation Y is a vector of the y values from a data set, X is matrix to be manipulated to be representative of various function types, β is a vector of parameters that we will be calculating to give estimates for coefficients and ϵ represents an error vector. From this equation we can obtain a method for calculating good estimates for β. The resulting equation is $\hat{\beta} = (X^T X)^{-1} X^T Y$.
In order to derive estimates for different function types we have only to manipulate the X matrix and sub it into the previous formula. In this coursework the functions I used required the X matrix to be of these shapes:

$$linear = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{pmatrix}, polynomial = \begin{pmatrix} 1 & x_1 & \dots & x_1^n \\ 1 & x_2 & \dots & x_2^n \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & \dots & x_n^n \end{pmatrix}, sin = \begin{pmatrix} 1 & sin(x_1) \\ 1 & sin(x_2) \\ \vdots & \vdots \\ 1 & sin(x_n) \end{pmatrix}$$

The general layout of X is first, a column of 1's to represent constants in a function and then additional columns representing x values applied to variables in a function.

Justification of Polynomial Order -

The polynomial order I have selected is third order (cubic function). I initially thought that this was the order as a result of plotting the various line segments from the 'basic' and 'adv' data sets and then eyeballing the shape of the curves. I used these sets as they have very little noise which makes it easier to see trends of the plots. When doing this I realised that the greatest number of extrema any of the data segments had was 2; third order is the lowest order that can have 2 extrema so I thought this would be a good order to start at.

Order	CV Error
2	1.3457182953120328
3	4.403496031576045e-19
4	4.0926073035860174e-14
5	9.51420563858535e-10

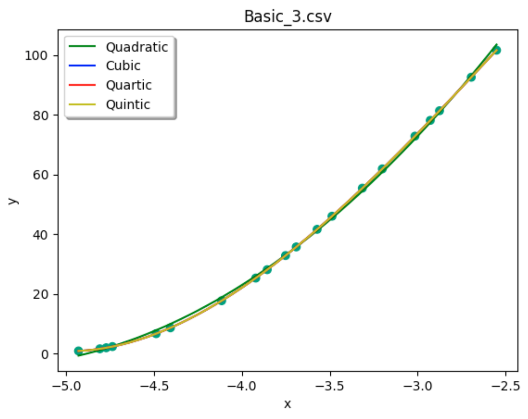

Figure 1

Figure 1 shows the plots and resulting errors of different order polynomial functions fitted to data from the 'basic_3.csv'. Note I am only showing up to 5th order as with only 20 data points in each segment higher orders are very likely to overfit. I used least squares to determine the closest estimates of the coefficient values for each function. Once this was done I evaluated models produced by different orders through cross validation to determine which fitted the data the best. As seen in the results third order does indeed produce the smallest error when doing cross validation. However, the errors of orders 4 and 5 are also all very small. Thus the results of CV alone did not give enough evidence to convince me that the order was 3.

Order	CV Error
2	12.585299678426177
3	13.699245238024412
4	16.121057122050953
5	21.867328815143914

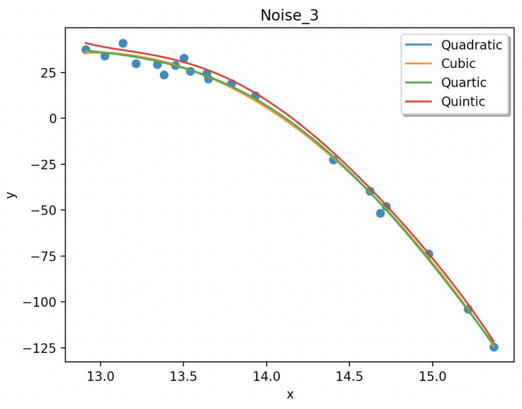

Figure 2

This is where I again went back to looking at the plots, but this time at plots with noise. This enabled me to see the effects of overfitting more clearly. Figure 2 shows the plots of various orders from the data set 'Noise_3.csv' along with the CV errors each of them produces. As the order increases so does the CV error, this is because higher order functions overfit the data. To summarise, from Figure 1 I can see that quadratic still produces a relatively high CV error when there is no noise suggesting it is not the order the function was fitted from, also the cubic function produces the lowest error. From the second I can see that the higher the order the more overfitting occurs. Hence why I selected cubic (3rd order) as the function type.

Justification of Unknown Function -

The unknown function I have selected is the sine function. I started by fitting all the noise-free 'basic' line segments with either a linear or cubic function and then looked to see which of the segments did not fit particularly well. 'Basic_5.csv' produced a higher cross validation error than I would expect from a data set with effectively 0 noise.

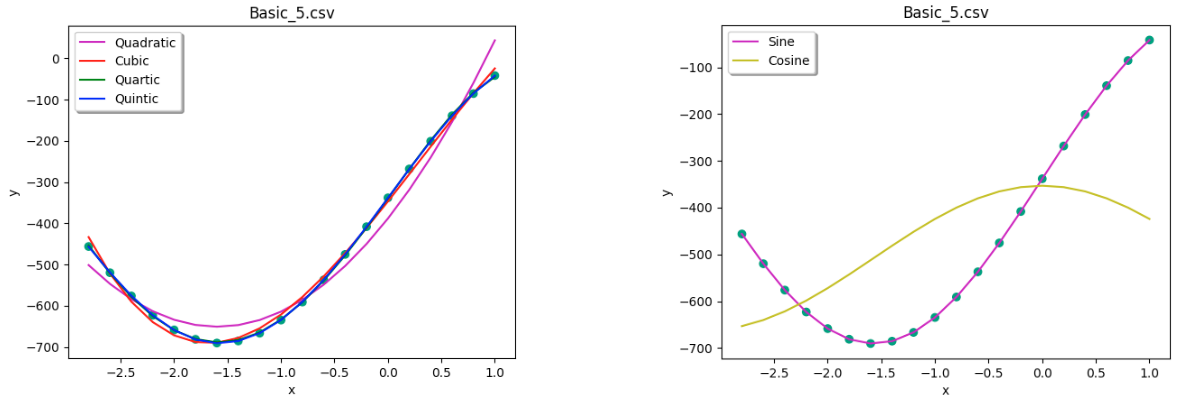

Figure 3

Next I fitted a series of polynomials to the data (left, Figure 3). I found from this although I could get a very low CV error score (<0.001) this was only from functions with higher orders and as we only had 20 data points I believed high orders would produce a lot of overfitting. Therefore I decided that this line segment was formed from the unknown function and that it was not polynomial. From the shape it looked to me as if it could be a sinusoid, this was because the curve is smooth and has potential to be periodic. Thus I decided to fit the sine and cosine functions to it (right, Figure 3).

The sine function fits the curve almost perfectly with a CV error = 8.764560201367529e-27. As this error is so low and from plotting the rest of the line segments from the data and seeing how well the sine function fit some of them I decided that the unknown function was the sine function and it is plotted by calculating A and B in Asin(x) + B.

Selecting Between Functions -

Originally I implemented a function for calculating the sum squared error and then simply chose the function that had the lowest error. Of course this is a far too basic method as it does not account for overfitting. Overfitting occurs when a model learns the trends/noise of the set it is trained on and adjusts to fit this. However, when unseen data is added to the data set it is not guaranteed to be affected by the traits observed in the training set, thus the error between the predicted values from the model and the actual data will begin to increase (showing a decrease in the skill of the model).

Therefore I needed a method that mitigates overfitting, cross validation (CV) was my solution. CV is primarily used to judge the skill of a model on unseen data (thus effectively calculating how much overfitting has occured). The general procedure for cross validation is to start by shuffling the data, in order to try and reduce bias/patterns in the data set. Next we split the data into k groups (folds). For each unique group you take the current group as the validation set and a training set is made from all the other elements of the

data set. Next develop a model from the training set and calculate the sum squared error between this fit and the actual values in the validation set. Finally sum the errors up from each fold and calculate the mean to get the average CV error. The function with the smallest CV error is then selected as the function to fit the segment.

The first variation of CV I used was k-fold. The main advantage of k-fold over basic CV is that it performs several iterations of CV, producing k models that are all tested for their skill. We can then average these errors and get a more accurate, valid representation of the overall skill of the model that does not have a high variation between results of each CV. This is because the effect of bias', outliers etc that may be present in some training data sets are reduced as each model has less affect on the actual result. Another advantage of k-fold CV is that it uses all the data at some point to test the fits as opposed to single iteration where only one group is used for validation, this means that final error calculations are fair and representative.

The first disadvantage CV is that splitting the data gives us less data points to train a model on, meaning the model will be more easily influenced by bias' and outliers. This issue is particularly relevant in this project as data segments only contain 20 elements, not alot to fit a model from. Another disadvantage of k-fold is that it requires a lot more computation than single iteration. However, as mentioned previously we only have to consider 20 data points for each segment so each fold doesn't require much computation meaning it is not an issue if we have many folds.

After considering these disadvantages the method of CV I ended up using was leave one out cross validation. The general CV process is the same but the value of k is set to the size of the data set. Meaning a model is trained by all the elements in the data set bar the current element, this is done for every element and overall error is average of all errors produced. The first advantage is that each element in the data set at one point represents the entirety of the test data set is that the estimate of the skill of the model is as representative/valid as it can be from the method of CV. The other advantage is that we use as many data points as possible to fit the model giving its maximum accuracy. The disadvantage of this method is that it has a very high computational cost so it would not be feasible with large data sets or models that take a lot of computation to fit. (As mentioned before this is not an issue for us)

The final functionality I considered adding to help with selection between function types was a case to choose a linear function instead of a polynomial function if the coefficients of x^3 and x^2 variables were 'suitably' low. I implemented this as when I originally had a value of k=5 there was some variation in the results from CV. This meant that at times a polynomial fit was chosen over a linear fit even when the coefficients of the third and second order variables were extremely low. When the coefficients are this small the function is effectively a linear graph with an inflexion point so small it is insignificant. Therefore I added a case where if this occurred the simpler linear function would be selected, again attempting to mitigate overfitting. However, when I changed to leave one out cross validation the issue with variation in results was removed thus I removed this functionality.