Symbols, Patterns and Signals CW1 Report

Jakub Leszczynski (pt18419)

1. Introduction

This report focuses on the design and accuracy-related aspects of the “An Unknown Signal” coursework implementation. After receiving unknown points, my job as an operator for a nuclear early warning system, is to quickly and precisely reconstruct the signal that those points follow, along with overall error.

2. Methods & Materials

Installing Anaconda package was a straightforward way of downloading all relevant imports. All example points from csv files has been given as training data along with two utility functions, i.e. *load_points* and *view_point.* The goal of this coursework is to find the least-square regression of the data set and their associated residual error. This will be achieved by writing and implementing my own functions. The lines can be either linear, a fixed order polynomial, or of unknown type that I have to figure out.

3. Functionality and Design

Getting started

I added a legend to the plot to help visualise what the signal might be after being identified. The X and Y axes are arbitrary.

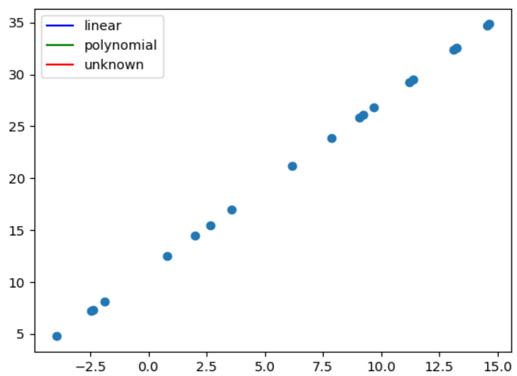

Figure 1. Plot of basic_1.csv with legend.

The description of the coursework specified that one line should only fit 20 points. However, the training files contains much longer sequences, therefore I realised that the first problem is to divide the given input points into chunks of 20 and consider them separately. I did that by implementing the *setify* function and was ready to start the core implementation. The description also pointed out that the residual error should be calculated over the whole sequence.

$$R(a, b) = \sum_{i=1}^{N} (y_i - (a + bx_i))^2$$

Figure 2. Formula for residual error [1].

Linear Function

The next step was to develop a logic for least squares linear regression. This regression minimises the sum of squared vertical offsets of the points from the line. I have used the matrix form of the formula, as it is easier to manipulate and expand.

$$\boxed{\mathbf{a}_{LS} = (\mathbf{X}^T\mathbf{X})^{-1}\,\mathbf{X}^T\,\mathbf{y}}$$

Figure 3. Formula for least squares regression in vector form [1].

In order to implement it, I have used Numpy arrays, because of their ease of use and many library functions. Due to the fact that matrix multiplication is not generally commutative, the main problem was getting the dimensions right. The calculated errors on the basic_1 and basic_2 example points are close to zero, which assured me that my implementation is correct.

```
def least_squares_linear(xs, ys):
    ones = np.ones(len(xs))
    xtrans = np.array((ones, xs))
    x = xtrans.T
    xxt = np.matmul(xtrans, x)
    a = np.linalg.inv(xxt) @ xtrans @ ys.T
    return a
```

Figure 4. My own function implementing linear regression.

Running the program on basic_2.csv with “--plot” argument creates a plot of example points with an almost perfectly fitted line, generated by my least_squares_linear function.

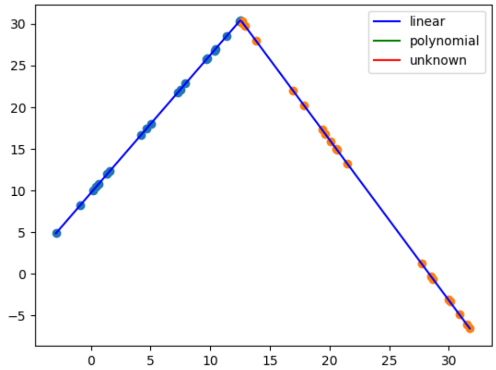

Figure 5. Plot of basic_2.csv.

Polynomial Function

At this stage, the crucial problem was to identify the correct order of the polynomial function, it was only certain from the description, that it was less than an order of 5. The implementation of the least-squares polynomial solution did not change compared to linear model, except in addition of new columns to X with higher orders.

$$\begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^p \\ 1 & x_2 & x_2^2 & \cdots & x_2^p \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_N & x_N^2 & \cdots & x_N^p \end{bmatrix}$$

Figure 6. Matrix form of X for p-polynomial least squares regression [1].

After running polynomial regressions with order 2, 3 and 4 on the points in basic_3 and basic_4, the smallest error was given by the cubic function. Therefore, I discarded other polynomial regressions.

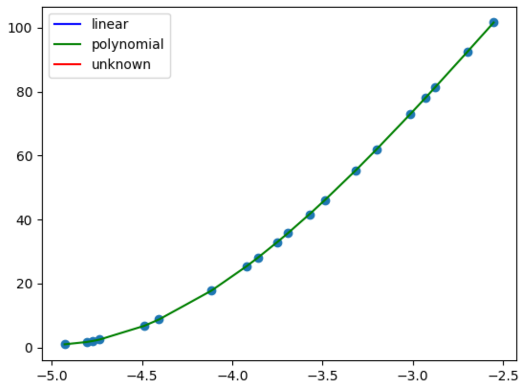

Figure 7. Plot of basic_3.csv.

Unknown Function

The last step was to find what the unknown function was. First, I considered an exponential function. However, by looking at the points in basic_5 I quickly ruled out that possibility, because the signal was changing direction. Then, I thought of trigonometric functions, as the signal seemed to follow synchronous changes. I adapted the code and tested both cosine and sine functions and ultimately sine gave better results, with the error for basic_5 being close to zero.

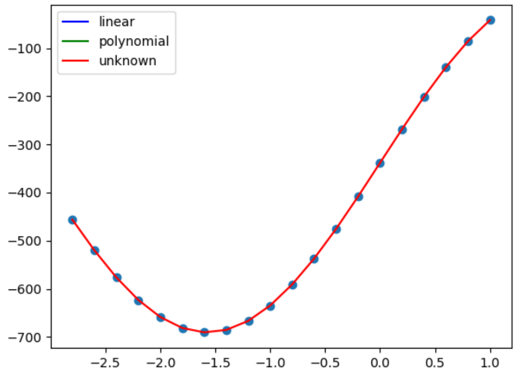

Figure 8. Plot of basic_5.csv.

Choosing between functions

At this point, deciding which function model to apply to the line segment, came down to calculating residual errors for all three of them and choosing the one with lowest residual. This was very prone to overfitting and because of that it was not an ideal solution and had to be improved.

4. Further Data Improvements and Analysis

Overfitting

Running the current solution on the noise_1 example points gives a highly overfit model that will generalise poorly to new data [2]. At this point, having all the functions worked out, the next step was to minimise the effect of noise. It became clear to me that I needed to use a hold-out method [3]. I would split the points into training and testing sets, and then fit a model on the training set to predict the values in the testing set.

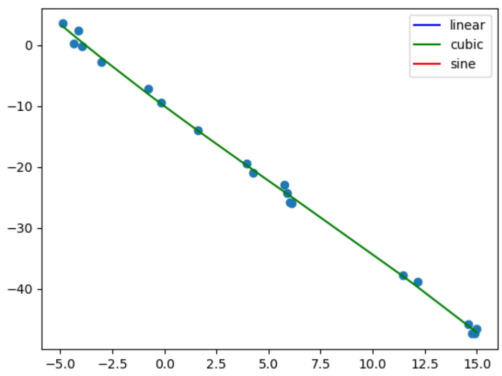

Figure 9. Plot of noise_1.csv before improvements.

However, my data consisted of sets of only 20 points, therefore my solution would depend heavily on how the points are chosen. I needed to minimise this arbitrariness, in order to get a well-fitted model [4].

K-fold Cross-Validation

This technique divides the input data set into k number of folds, and the holdout method is repeated k times. In each iteration, one of the folds is used as the test set, while the rest is used for training. This requires more computation but guarantees that every data point is used for testing [3]. The remaining question is how many folds would give best results. Fortunately, our data set consists of only 20 points, which is small enough to run Leave-one-out Cross-Validation, where the number of folds is simply equal to the number of data points, i.e. 20. This helped the model be more general to new data and less prone to wrong estimations from outliers.

```
cv_linear = 0
average_linear = np.array([0.0, 0.0])
for train_index, test_index in kf.split(xSets[i]):
    X_train, X_test = (xSets[i])[train_index], (xSets[i])[test_index]
    y_train, y_test = (ySets[i])[train_index], (ySets[i])[test_index]
    cv, a = cal_linear_cross(X_train, y_train, X_test, y_test)
    cv_linear += cv
    average_linear += a

average_linear = average_linear/number_of_splits
cv_linear = cv_linear/number_of_splits
```

Figure 10. Code implementation of K-fold Cross-Validation using sklearn library.

Final Solution

At this point I was content with the extent of my implementation, as it fulfilled all the requirements outlined in the coursework description. Now, the model applied to the given segment will be the function with the lowest cross validation error, which is a mean of differences between the predicted output values and the actual values in the test set.

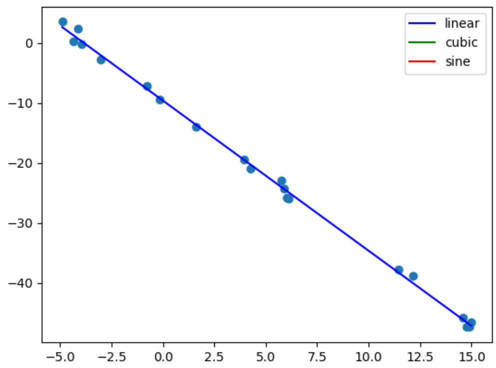

Figure 11. Plot of noise_1.csv after improvements.

Combining least squares regression with k-fold cross validation also gives acceptable results on the advanced example points with reasonable residual errors.

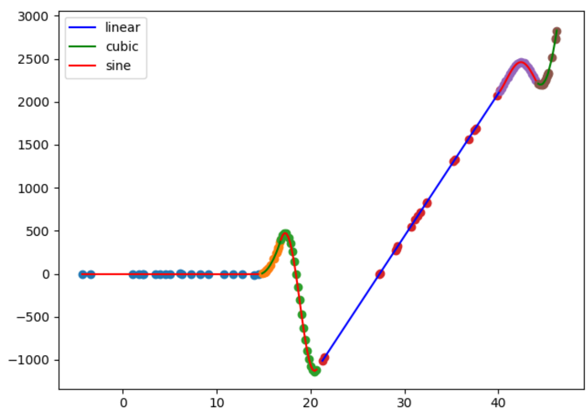

Figure 12. Final plot of adv_3.csv.

Possible future optimisations

One of possible next steps could be adding regularisation [5]. This would further improve accuracy in noisy data, by penalising the weights of outliers [5].

5. Conclusion

The goal of the coursework was to use least-squares regression to recreate signals from example points. Using mathematical methods described in this report I managed to do that, along with improved accuracy, achieved through K-fold Cross-Validation. The lines outputted by the program are models approximating the original functions that created given sets of points. This can help me, as the operator of the nuclear early warning system, make my decision on whether this is a serious threat or not.

References:

1. Costa RP, Damen D. Deterministic Data Models [Internet]. COMS21202: Symbols, Patterns and Signals; 2020 [cited 2020 March 24]. Available from: https://uob-coms21202.github.io/COMS21202.github.io/.

2. Al-Masri Anas. What Are Overfitting and Underfitting in Machine Learning? [Internet]. Towards Data Science. 2019 [cited 2020 Mar 24]. Available from: https://towardsdatascience.com/what-are-overfitting-and-underfitting-in-machine-learning-a96b30864690

3. Schneider J. Cross Validation. In: A Locally Weighted Learning Tutorial using Vizier; 1997 [cited 2020 March 24] [Internet]. Available from: https://www.cs.cmu.edu/~schneide/tut5/node42.html.

4. Reitermanov´a Z. Data Splitting. In: WDS 2010 - Proceedings of Contributed Papers: Part I - Mathematics and Computer Sciences [Internet]. Charles University, Faculty of Mathematics and Physics, Prague, Czech Republic: MATFYZPRESS; 2010 [cited 2020 Mar 24]. p. 31–6. Available from: https://www.mff.cuni.cz/veda/konference/wds/proc/pdf10/WDS10_105_i1_Reitermanova.pdf

5. Gupta P. Regularization in Machine Learning [Internet]. Towards Data Science. 2017 [cited 2020 Mar 24]. Available from: https://towardsdatascience.com/regularization-in-machine-learning-76441ddcf99a