Exercises 5.10 Four guys and four error plots

Eric, Stanley, Kyle, and Kenneth used hold out cross-validation to find the best degree polynomial fit to their respective datasets. Based on the error plots they have made (Fig. 5.21), what advice would you give to each of them as to what their next step should be.

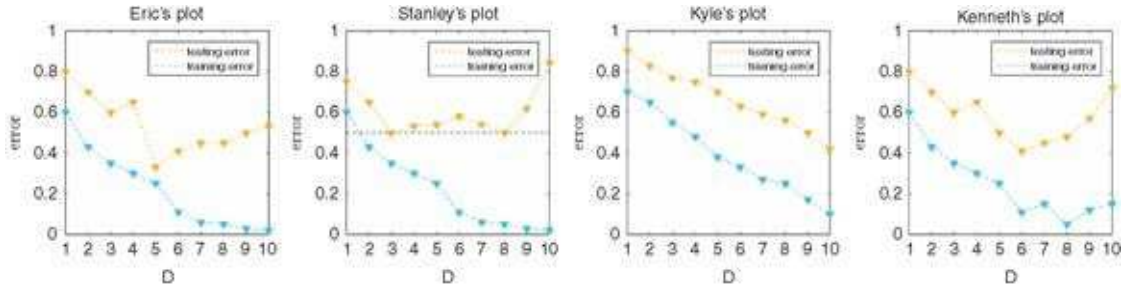

Fig. 5.21 Hold out cross-validation error plots for four different datasets.

Exercises 5.11 Practice the hold out method

In this exercise you will perform hold out cross-validation on the dataset shown in Fig. 5.16 as described in Example 5.5 of the text. Start by randomly splitting the dataset, located in *wavy_data.csv*, into $k = 3$ equal sized folds (keeping 2 folds as training, and 1 fold as testing data). Use the Fourier basis features M in the range $M = 2, 4, 6, \ldots, 16$ (or likewise D in the range $D = 1, 2, \ldots, 8$) and produce a graph showing the training and testing error for each D like the one shown in Fig. 5.16, as well as the best (i.e., the lowest test error) model fit to the data.

Note: your results may appear slightly different than those of the figure given that you will likely use a different random partition of the data. Note: you may find it very useful here to re-use code from previous exercises e.g., functions that compute Fourier features, plot curves, etc.

Exercises 5.12 Code up *k*-fold cross-validation

In this exercise you will perform k-fold cross-validation on the dataset shown in Fig. 5.19 as described in Example 5.7 of the text. Start by randomly splitting the dataset of $P = 6$ points, located in *galileo_ramp_data.csv*, into $k = 6$ equal sized folds (keeping 5 folds as training, and 1 fold as testing data during each round of cross-validation). The value of $k = P$ has been chosen in this instance due to the small size of the dataset (this is sometimes called “leave one out” cross-validation since each training set consists of all but one point from the original dataset).

Use the polynomial basis features and M in the range $M = 1, 2, \ldots, 6$ and produce a graph showing the average training and testing error for each M, as well as the best (i.e., the lowest average test error) model fit to the data.

Note: you may find it very useful here to re-use code from previous exercises, e.g., functions that split a dataset into k random parts, that compute training/testing errors, polynomial features, plot curves, etc.

Section 5.4 exercises

Exercises 5.13 Comparing all bases

In this exercise you will reproduce the k-fold cross-validation result shown in Fig. 5.12 using the wrapper *compare_maps_regression_hw* and the corresponding datasets shown in the figure. This wrapper performs k-fold cross-validation using the polynomial, Fourier, and single hidden layer tanh feature maps and produces the figure. It is virtually complete, i.e., the code necessary to generate the associated plot is already provided in the wrapper, save four modules you will need to include. Insert the data splitting module described in Exercise 5.11, as well as the polynomial, Fourier, and single hidden layer tanh modules for solving their corresponding Least Squares problems described in Exercises 5.7, 5.8, and 5.9 respectively. With these modules installed you should be able to run the wrapper.

Note that some of your results should appear different than those of the figure given that you will use a different random partition of the data in each instance.

5.7 Notes on continuous function approximation

Polynomials were the first provable universal approximators, this having been shown in 1885 via the so-called (Stone–) Weierstrass approximation theorem (see e.g., [71]). The Fourier basis (and its discrete derivatives) is an extremely popular function approximation tool, used particularly in physics, signal processing, and engineering fields. The convergence behavior of Fourier series has been studied for centuries, and much can be said about its convergence on larger classes of functions beyond $\mathcal{C}^N$ (see e.g., [61, 74] for a sample of results). The universal approximation properties of popular adjustable bases like the single-layer and multilayer neural networks were shown in the late 1980s and early 1990s [28, 38, 63]. Interestingly, an evolutionary step between fixed and adjustable bases, a random fixed basis where internal parameters of a given adjustable basis type are randomized, leaving only the external linear weights to be learned, has been shown to be a universal approximator more recently than deep architectures (see e.g., [69]).