Introduction to Machine Learning (CS 135)
Assignment 02 (50 points)

Due on Gradescope by 11:59 PM, Monday, 26 September 2022

For this assignment, you will make modifications to a Python notebook (`hw02.ipynb`) that has been supplied. You will complete the various required sections outlined below in that notebook. When you are done, generate a PDF version of that notebook, with all results (figures, printed results, etc.) included, for submission to Gradescope. You will also submit the raw notebook source, and a `COLLABORATORS.txt` file, via a separate link, as described below.

Python Notebook PDF (45 points)

You have been given a data set (`data.csv`) consisting of some synthetically generated (input, output) (x, y) pairs, each represented by a single floating point value. You will write code that explores this data, using a number of different linear regression models.

1. (*15 pts.*) You fill first fit a sequence of models, starting from a simple (degree-1) linear regression, and then continuing for data that is augmented using an increasing sequence of higher-degree polynomials.

   1.1. (10 pts.) Fill in the `test_polynomials` function. This function should take in a list of positive integer values, corresponding to different degrees for polynomial models (so if the degree is 1, we get a purely linear model, if it is 2, we first transform the data so it is degree-2, and so on). Each model should be fit to the entire data-set, and then the model should be used to predict outputs for that data-set.

   The function will return two lists. One will consist of the predictions, i.e., it will be a list of length P, where P is the number of different polynomial degrees, and each element of the list consists of the N output values, where N is the size of the overall data-set). The other will consist of the P error values for the models, where error is calculated using the mean squared error metric (MSE).

   Once the function is completed, it should be called using the sequence of degrees $d \in \{1, 2, 3, 4, 5, 6, 10, 11, 12\}$. Once the function returns its two lists, the `plot_predictions` function (already written for you) will produce a plot containing each of the models' predictions, along with their MSE values (found in the title area of each subplot).

   1.2. (5 pts.) Discuss the plotted results. What do they show? What is the best model, based upon MSE? What models do particularly poorly? What does this tell you?

2. (*15 pts.*) You will now consider the same sequence of degrees as in the previous question. Instead of simply fitting the entire data-set, however, you will do 5-fold cross-validation, to allow an estimate of when the various models may be over-fitting to the data.

   We have supplied a `make_folds` function. This function takes a positive integer value, k, and divides the data-set (both x and y values) into k distinct sub-parts, where each part consists of the same number of elements (we will assume for the sake of this assignment that the data-set size divides evenly by the value k). The folds are returned in the form of two lists (for x and y), each of length k, where each element of a list consists of some consecutive sub-sequence of the original data, and each data-element is a member of exactly one fold (so that printing the contents of the k folds in order would give us back the exact same data-set). You will use this function in the rest of your code.

   2.1. (10 pts.) For each of the polynomial degrees previously considered, perform 5-fold cross-validation on the data. That is, for each degree, your code should build 5 separate models, training each time on a different 4/5 of the data and testing on the remaining 1/5. You should use the `make_folds` function that we provided to split the data into the 5 parts—it is then up to you to re-combine parts as needed in each iteration of the cross-validation.

   You should keep track of the average training and testing error (MSE) seen over the 5 models, for each degree. You will then plot these results. This should consist of a single plot with:

   - The distinct polynomial degrees along the x-axis, and average MSE along the y-axis.
   - Average error plotted as two separate lines, one for training data and one for testing data; the lines should be distinguishable, by color or other features.
   - Proper labels on all axes, an explanatory title, and a legend making clear which line is which.

   Also: Following the plot, there should be tabular text print-out of the data as well; this can be in any clear format.

   2.2. (5 pts.) Discuss the plotted results. What do they show? Where do we see the best results? Where is their underfitting, and why do you say that? Where is there overfitting, and why do you say that?

3. (*15 pts.*) You will now examine ways in which a basic linear/polynomial model, of a fixed degree, can be adjusted to help avoid overfitting.

   One way to do this is via ***ridge regression***. In this technique, rather than seek only to minimize error on the data-set, the regression model seeks to balance minimal error with weights that are smaller in magnitude (positive or negative)—that is, weights that are closer to 0. By doing this, there is often less chance that the model over-fits to one particular feature that is more characteristic of its training data than is representative of the overall truth.

   Discussion of ridge regression can be found in James et al. (section 6.2.1, pp. 215–219) [link] The model `sklearn.linear_model.Ridge` implements ridge regression. You will use this model to examine a range of regularization strengths, comparing how they affect performance on the entire data-set.[*]

   3.1. (10 pts.) Using whichever polynomial degree you found to give the best performance on the data-set (after cross-validation, in the second part of this assignment), generate a series of `Ridge` models:

   - Each model should use a different **regularization strength**; this value, set using the model parameter `alpha`, measures the penalty applied to large weights (so that a higher value means that the model tries even harder to keep weights small). You will generate a series of 50 distinct weights (and so 50 different models), distributed logarithmically over the interval $[0.01, 100]$, using the code snippet:

     ```
     np.logspace(-2, 2, base=10, num=50)
     ```

   - You will do 5-fold cross-validation for each strength-value, again keeping track of average training and test error. (Do this the same way as in the prior question, using the supplied function to generate the folds required.)
   - You will again produce a plot comparing these results. The plot should satisfy the guidelines of the one produced for the second part of the assignment, clearly showing the error results for each of the training and testing data. Again, a tabular version of the data should also be printed out.

   Note: Since we are using a logarithmic scale for the strength values, the graphical plot should also use such a scale, to be most readable. This can be achieved using:

   ```
   matplotlib.pyplot.xscale('log')
   ```

   3.2. (5 pts.) Discuss the plotted results. Where is the effect of increasing regularization strength helpful in avoiding overfitting, does it appear, and why do you say that? Where is the effect less useful, and why do you say that?

[*]See the documentation at:

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html

Code submission (5 points)

1. (*2.5 pts.*) Submit the source code (`hw02.ipynb`) to Gradescope.
2. (*2.5 pts.*) Along with your code, submit a completed version of the `COLLABORATORS.txt` file. An example has been provided, which you should edit appropriately to include:
   - Your name.
   - The time it took you to compete the assignment.
   - Any resources you used to complete the assignment, including discussions with the instructor, TA's, or fellow students, and any online or offline resources consulted. If you did not need to consult any outside resources, you can say so.
   - A brief description of what parts, if any, of the assignment caused you to seek help.

Submission details: Your work should satisfy the following requirements:

- Your code must work properly with the versions of Python and other libraries that are part of the 135 standard environment. The class Canvas site contains instructions for installing and using this environment (first Module).

  You can write the Python code using whatever tools you like, but you should ensure that the code executes properly in the course environment.
- Submit your work to the Gradescope site for the course.
- There are two Gradescope submission links for this assignment. You will upload the raw notebook file and `COLLABORATORS` file to one link, and the PDF of the notebook to the other. These will become available once the late deadline for the prior assignment has passed, to minimize confusion.
- When you upload the PDF, Gradescope will ask you to select the pages that correspond to each question. For full points, please do so. Your PDF should consist of multiple pages, for readability. If you're unsure how to select pages, see the Gradescope help pages.
- Note that you can upload multiple times to any part of the assignment—we will be grading just your final submission. Be sure to include all files in a single upload, if there are multiple files (either put them into a single ZIP file, or just drop all the files into the upload window).