Homework 12

This homework will give you a peek into the expansive world of machine learning as implemented in the Python package `scikit-learn`. You won't emerge an expert by any means, but it will give you some exposure to a framework that's likely quite different from what you're used to. In the classic statistical / scientific modeling framework, it is common to spend your effort thinking through the how to craft your model based on your understanding of the phenomenon at hand; e.g. "I want to control for X, so I'll include that in the model", "I know incomes are right skewed, so I'll take a log-transformation".

In the machine learning framework, these concerns fall away and are replaced by a strong focus on predictive accuracy. Once you've settled on a specific measure of accuracy, it becomes an optimization problem: finding the particular model specification that maximized predictive accuracy. In this homework you'll use the tool of simulation to investigate the process of conducting model selection via cross-validation. Along the way you will:

- Generate data from a linear model,
- Fit several polynomial regression models,
- Create validation curves to select the optimal model, and
- Consider the effect of different measures of predictive accuracy.

Simulation

Generating data from the true model

The great advantage to studying models via simulation is that you're able to define the true model apriori and generate as much data as you like. You can then fit a wide range of candidate models and see which one does the best job of recovering the true model. The true model that you'll be working with is:

$$y = -1 - 6x + 5x^2 + 5x^3 - 5x^4 + x^5 + \epsilon; \quad \epsilon \sim \mathrm{N}(0, 1)$$

You can break the process of simulating data from this model in to four parts.

I. Specify parameters This model has 7 parameters: the intercept, the coefficients in front of each x, and the standard deviation of the error. Go ahead and code them in.

```
b_0 = -1
b_1 = -6
b_2 = 5
b_3 = 5
b_4 = -5
```

```
b_5 = 1
sigma = 1
```

II. Generate `x` An key part of any regression model is that it models the *conditional* distribution of the y given an x. Therefore we can choose any distribution for the x that we like. We do need to set `n`, however, which in a way acts like another parameter.

1. Set the value of `n` to 30 and use `np.random.RandomState()` to create a random state object. Use that object's appropriate method (call `dir()` on it or use tab-complete to see your options) to generate `n` random uniform numbers between 0 and 1. Rescale those numbers so that they are between 0 and 4, then subtract 1 from all of them.

III. Calculate $E(y)$ Now that you have your parameters specified and a set of `x` in hand, you can go ahead and calculate what your model says should be the expected value of y given each of those `x`. Another way to think of these are as the values of $\hat{y}$ for each `x`.

2. Calculate and save the array `Ey` as the linear combination of each of your parameters with `x`, `x^2`, etc.
3. Construct a scatterplot of `x` and `Ey`. You may run into an error that can be solved by calling the `.ravel()` method on `x` at the outset. What was the error caused by and how did `ravel()` fix it?

IV. Simulate `y` Everything up until this point is just laying the groundwork. The actual *random* part of the simulation is the generation of the error terms that make up your "observed" y values.

4. Create a new array `y` as `Ey` plus an array of random normal variables with mean 0 and standard deviation `sigma`.
5. Remake the scatterplot from question 3 but this time use `y` instead of `Ey`.

Fitting Models

With your data set of `x` and `y` in hand, you're now at the point that a data analysis normally begins: deciding which model to fit to a data set. Even though we know the correct model form (a quintic polynomial), normally we have no idea what the model form is. Start off by trying two models: a linear model and a 6th degree polynomial model. From here on out you'll be using `sklearn`, the interface to which can be found in the lecture notes.

6. Use the `LinearRegression` function within the `linear_model` module of `sklearn` to fit a linear model to your data.
7. Using the `PolynomialFeatures` function in the `preprocessing` module of `sklearn`, create a two dimensional array
8. Construct a scatterplot of your original data that has each of these models overlaid. Note that this requires creating a grid of x values that you then call each model's `.predict()` method on to get your predicted y-values. See the lecture notes and the textbook ch. 5 for reference.

9. Using the `metrics` module, calculate the *training* MSE on each of these models. Which model has better accuracy predicting back into the training set?

10. Using the `cross_validate()` function in the `model_selection` module, calculate the (negative) *testing* MSE for each model as estimated through 5-fold cross validation. Take the mean test MSE across all 5 folds to get a single estimate. Which model has the smaller testing error (higher negative MSE)?

Validation curves

The `cross_validate()` function is an efficient way to implement a procedure that would otherwise require an awful lot of error-prone typing to code up. It has allowed you to evaluate two potential models for your simulated data, and it should be clear that the 7th degree polynomial is preferred over the linear model.

To be thorough, though, we'd want to consider a wide range of models, say every polynomial between degree 1 and 20 and track the training and testing scores of each model. These results can be used to construct a *validation curve*, which can then be consulted to select the model with the highest testing score.

11. Adapt the code found in the section "Validation curves in Scikit-Learn" in Ch. 5 of the textbook to construct a validation curve for polynomial models of degree 1 through 20 using 7-fold cross-validation. Some notes on using this this function:

- Depending on the version of `sklearn` that you're using, the `validation_curve` function may be in the `model_selection` module.
- It is safer to name each of the arguments that you pass to the function rather than relying upon their position as the book does.
- You will need to make a judgment call on what appropriate y axis limits are to reveal the most important structure in the curves.
- You can specify that you're interested in the (negative) MSE by passing it as a string to the `scoring` argument. See the scoring document to see what other options are available (https://scikit-learn.org/stable/modules/model_evaluation.html#model-evaluation).

12. Which model does the validation curve suggest is the best one?

13. Why is the training score a strictly increasing function of the degree?

14. Change your scoring metric to the mean *absolute* error. Does that change which model is selected? If so, why do you think this particular metric would favor that particular model?