Hertie School, Spring 2024
Machine Learning (Professor Dimmery)
Problem Set 2, Due Wednesday, April 3

In this problem set, we will use some of the tools you've learned in the first few lectures and labs. You should prepare your solutions in a Jupyter notebook, writing and running the appropriate code in the notebook (please no separate script files, as I want to see all of your work in a single document). Provide this document to me on Moodle in a single pdf.

You should prepare all of your writeup and code in this document, formatting it nicely: as if you were going to be sharing the results with your colleagues and boss. What does this mean?

- Clearly label your solutions.
- Solutions which require explanation should use complete sentences.
- If you use code or materials from anywhere else, that should be clearly labeled and cited.
- If you are asked to prepare graphs or tables of output, they should be labeled clearly. A good rule of thumb is that charts should be self-explanatory: they should not require reading the text in order to understand.

To export a Jupyter notebook to a pdf, the easiest solution is to run all of your code and then Print the page to a pdf. Depending on your computer's setup, you may be able to explicitly export using Jupyter's tools through the File - Download as - pdf via LaTeX (.pdf) menu (or one of the other exporters that ends in (.pdf). If you can't make this work (for further details, this blog has some additional instructions that might help), then just print it to pdf. This won't affect your grade, what I care about here is the content.

Note that each question in this problem set has a number of points associated with it. The maximum number of points on this assignment is 15 which, unsurprisingly, corresponds to the number of points on your final grade.

As a reminder, I don't mind if you work together with classmates when you get stuck, but the work you submit should be purely your own.

We will be working on a dataset today from a large randomized control trial run in Brazil. For more details, you can see the paper here (but you don't need to read the paper to do this exercise). The program was a financial literacy education, and students were assigned to receive it (the treatment) or not receive it (the control). To evaluate the effect of the program, the researchers gathered a set of background characteristics about students, assigned some students to receive treatment and then followed up a few months later to measure the outcome.

What we'll actually be doing in this exercise is fitting a model which could in theory be used to measure heterogeneous treatment effects, but we will just be treating it as a generic supervised learning model. (Specifically, we'll be fitting what's called an "S-learner"; it actually isn't a particularly smart way to do this if you care about getting heterogeneous

treatment effects right, but that doesn't matter for our purposes since we're not using it to do that.)

The data is available at this link, or you can use the following code to directly download and import it into your Python session:

```
url = "https://raw.githubusercontent.com/grf-labs/grf/master/r-package/grf/vignettes/data/bruhn2016.csv"
df = pd.read_csv(url)
X = np.hstack((
    df["treatment"].values.reshape(-1,1),
    df.iloc[:, 3:].values
))
y = df["outcome.test.score"].values
```

We're going to be focused on making predictions of the outcome, test scores (the column `outcome.test.score`), based on the other features available in the dataset. We will ignore the `school` variable, an identifier for schools, for now; you should drop it from your analysis as I did above. For this exercise, we'll treat *all other variables* as continuous.

The big picture for this homework is that you're going to do two things: cross-validate to find the optimal amount of regularization for a ridge regression, and then run a kernel ridge regression and tune its hyper-parameters the same way.

1. (1 point) Do some exploratory data analysis on the dataset. What do the variables look like? Is there missingness? What does the distribution of the outcome look like?

2. (1 point) Next, create your training and your test set. For this assignment, we're going to use almost the entire dataset as the **test** set: set aside 90% of the data for this test set. This will make your life easier as we iterate on models. By only training on a small fraction of the data, this process will be a lot faster[1]. Assume that the data is i.i.d. for the purposes of this assignment, so you don't need to do anything special in splitting the data.

   In the real world, you wouldn't want to have a split like this. It would be more common to have something like 30% of the data in the test set, but the size is usually determined by how accurately you need to know the error properties of your model.

3. (2 points) The next task is to define how we're going to (1) impute missing data, (2) standardize the data and (3) fit the model.

   Write a function that has just one argument: `regularization_strength`. This function should chain together your imputation strategy (`SimpleImputer`), your standardization strategy (`StandardScaler`) and the model class we want to estimate. In our case, that's `Ridge` regression[2]! The way you create this chain is through what scikit-learn refers to as a "Pipeline". In particular, you should pass each of these instantiated

[1] It is, in general, a good idea to do a lot of model development on a small sample of your data. You can work out the major bugs in your process before you scale up to datasets that make all of your models take substantially longer to run.

[2] Note that scikit-learn refers to regularization strength in ridge regression as the parameter `alpha`. This is exactly the same as what we've referred to as λ in lectures.

objects into a call to `make_pipeline`. A function that just standardizes features and runs OLS would look like the following:

```
def create_ols_pipeline():
    return make_pipeline(
        StandardScaler(),
        LinearRegression()
    )
```

You may be asking why we're going to the trouble to construct this complicated object instead of just doing it step-by-step by hand. The reason is that we're going to be doing cross-validation, and this vastly simplifies the amount of code you'll need to write, while ensuring sure that you always respect the training/validation split.

4. (2 points) Using the pipeline you created, fit a ridge regression where the regularization parameter is `alpha` $= 0.1$ on the (entire) training set.

   Fitting and predicting with this pipeline works exactly the same as if you had directly instantiated a `Ridge` model directly (i.e. the `fit()` and `predict()` methods work just the same.

5. (1 point) Use the `KFold` class to construct a set of 10 folds to be used for cross-validation.

   We create this ahead of time because we want to use the exact same folds for every value of the hyperparameters (and every model) that we test so that we compare them, apples to apples.

6. (2 points) Using the training data, use cross-validation to choose a good value of the regularization parameter. You should:

   - Try a range of different values (e.g. 20 values between 10^{-6} and 10^{6}) and inspect the estimated mean squared error. You'll want to choose values in a log-spaced grid (i.e. powers of 10). You can do this using numpy's `logspace` function, e.g. `np.logspace(-6, 6, num = 10, base = 10)`.
   - Plot the MSE for each value, along with standard errors.

   It might help you to use the `cross_val_score` function. The following code shows you how to use this function on a model / pipeline named "`model`", for features called "`X`" and labels called "`y`", with a cross-validation object (e.g. created by `KFold`) called "`cv`" and defining a "scoring" function from the mean squared error:

```
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error, make_scorer
cross_val_score(model, X, y, cv=cv, scoring=make_scorer(
   mean_squared_error))
```

   This will return a numpy array with one error estimate for each fold of CV. You can then calculate the average (to get the overall CV estimate of MSE) or the standard error. Since you have 10 folds of CV, you will have 10 MSE estimates for each value of the regularization parameter you pick. Calculate the standard error as `np.std(mses) / np.sqrt(mses.shape[0])`.

7. (1 point) Point out which value of the regularization parameter (and therefore which model) you would choose to use based on these results, using the one-standard-error rule we discussed in class.

8. (1 point) Next, create a new version of your function from Question 3 which replaces ridge regression with `KernelRidge`. You will need to add a second argument to the function as well to indicate the "lengthscale". This is the parameter that determines the width of the region near a test point which should have positive weight. To instantiate a Kernel Ridge regression in scikit-learn, you'd do the following:

```
KernelRidge(alpha = 0.01, kernel = "rbf", gamma = 1)
```

   where the `gamma` denotes the lengthscale. For more context: scikit-learn uses some weird terminology, but it's all the same mechanics that we've talked about in class. An 'rbf' kernel (radial basis function kernel) is the same idea as a Gaussian kernel. The developers of scikit-learn define this RBF kernel to be $\exp\{-\gamma\|x - x'\|^2\}$. This is identical to a Gaussian kernel when $\gamma = \frac{1}{2\sigma^2}$. This difference in how they encode the kernel means that a larger `gamma` parameter corresponds to a smaller region in which two points are considered similar. Contrast this to σ^2, in which larger σ^2 corresponds to a larger region of "nearby" points. The difference in terminology is confusing, but the main point is that there is a one to one mapping between values of `gamma` and values of σ^2. We're going to search over a range of values of `gamma` to see what works best regardless, so it's primarily an issue of interpretation.

9. (2 points) Find the cross-validated MSE for a range of values of `gamma`, e.g. 10 values between 10^{-7} and 10^{-1} (again, on a log-spaced grid). Do this for one fixed level of the regularization parameter. You can set `alpha` $= 0.1$ or whatever you found to be optimal for ridge regression in Question 7.

10. (1 point) Calculate the error in the test set for both your best `Ridge` and `KernelRidge` model. Which is better?

11. (1 point) It turns out that the way that students were assigned to treatment was not actually i.i.d. Rather, the *high school* was assigned into treatment. This was done out of a concern that the outcomes between students might be correlated within schools.[3]

    Using this fact, and what we talked about at the beginning of lecture 4 about reasons we might want to split data into folds for cross-validation differently than a naive, completely random split, what might you do differently in the cross-validation process to account for this dependency structure? You don't need to actually do it, just describe what you *would* do differently.

[3]There are a number of reasons for this: students talk with each other, they have the same teachers, there are similar socioeconomic factors which affect students in a given school, and you can probably think of other reasons as well.