46-927, Fall 2017: Homework #3

Due 5:20PM, Monday, February 5, 2017.

Homework should be submitted electronically on canvas before the deadline. Please submit your homework as a single file (pdf or html). We recommend using Rmarkdown or jupyter notebooks to prepare your homework.

Note: For this homework, try to do all problems in python. If you become too frustrated, you can use `smooth.spline` from `R` and `gam` from the `mgcv` library in `R`. As we continue on in the topics for this course, the support from python libraries will improve. Things like regularized regression and smoothing splines have strong enough ties to statistics that the `R` support tends to be better.

Please post any questions on the Piazza discussion board.

1. *You actually saw this data in Data Science as an example of kernel regression, but we will use it again to look at smoothing splines.*

   One approach to estimating the forward rate function (which can be used to calculate bond prices and yield curves) is to obtain many noisy point estimates of the forward rate at different times by comparing pairs of zero-coupon bonds, and them smooth them to get a more stable estimate of the curve.

   (a) Load `forward_rates.csv`. This contains a set of empirical forward rate estimates for a range of times (courtesy of Ruppert and Matteson, 2015, which you can also read – free on SpringerLink – for a better description of the data). Plot the empirical rates against time.

   You should see that the points make a very noisy estimate of a smooth curve. It should be clear that a linear function, or even a quadratic one, would not capture this shape very well.

   (b) Fit a smoothing spline to this data. In python, use LinearGAM like we did in class. In R, use `smooth.spline`. Cross-validate for a reasonable lambda. Plot your fit on the same plot as the points. Can you manage to get a believable, smooth fit?

   (c) Computing the yield curve from this forward rate function involves definite integrals of your estimated smooth function. Splines are particularly nice for

doing this. Think about the nature of cubic splines, and explain briefly why they can be integrated quickly and in essentially closed form (no numeric integration).

To think about: Compare this to nonparametric estimates from kernel regression, where you would need to evaluate the function on a fine grid of points and then carry out numerical integration.

2. *Kernel regression and varying smoothness.* Starter code for this problem is in `adaptivity_starter.py` (and alternatively in `adaptivity_starter.R`) on the course website. That code will generate a data set to be used for this problem, and will also provide a true mean function `mu(x)`. The resulting data frame has a `x` column (your predictor) and a `y` column (your response).

   Important: when you fit splines with LinearGAM for this problem, use `n_splines = 100` in your `LinearGAM` function. The default leads to very inflexible fits. When you cross-validate, I recommend

   ```
   lam = np.logspace(-5, 4, 100)
   ```

   (a) Plot y versus x. Overlay the true mean function `mu(x)`. What do you notice for $x \leq 4\pi$ and $x > 4\pi$?

   (b) Fit a smoothing spline to each of the following datasets, crossvalidating the lambda each time:

   i. Only those data points with $x \leq 4\pi$.
   ii. Only those data points with $x > 4\pi$.
   iii. All the data points

   For each of these regressions, what is the optimal lambda? How do they compare?

   (c) For each of the three selected lambda, make a plot showing:

   - The true mean $\mu(x)$.
   - The data points.
   - The smoothing spline predictions refitted on the *entire data set*, with the lambda specified to be the selected lambda. (Actually, use (selected lambda)/2 for the lambdas that were estimated on half the data set. When we change the amount of data, the meaning of lambda changes. This isn't the right correction, but it's closer.)

Note: Each plot should show *all* the data ($[0, 8\pi]$), but with the gam refitted to use the lambdas from the previous part (one lambda for each plot, on the whole signal now).

The result should be three plots, each tuned to one of the selected lambda. Give these plots clear titles to distinguish them. *Hint: to see the important details of your plot, you will probably need to make them reasonably large.*

(d) How do these three plots differ? In particular, how well do the regressions trained on the left and right halves do on each half of the data set? How well does the lambda fit on the overall data set do on each half? (Be specific about the types of problems that occur.) What lesson might this tell about functions of varying smoothness and smoothing splines, if any?

(Fun fact: you'll have the same problem with kernel regression and a fixed bandwidth.)

3. This problem explores additive models for prediction. You will consider a bike rental data set, and forecast utilization based on time and weather. This information could be useful for tasks like inventory management and maintenance; we will explore this impact in terms of loss later in the problem.

   Download `bike_train.csv` and `bike_test.csv` from canvas. These have daily data for two years. Load the data and get familiar with it.

   - `dayofyear`, `dayofweek` are the 0-indexed day of the year and of the week. I've pretended that 2012 is not a leap year.
   - `workday` is a binary indicator of whether it is a workday (1) or a weekend/holiday (0).
   - `weathertype` is a four level variable for type of weather, with higher numbers indicating worse weather: 1: Clear, Few clouds, Partly cloudy, Partly cloudy; 2: Mist + Cloudy, Mist + Broken clouds, Mist + Few clouds, Mist; 3: Light Snow, Light Rain + Thunderstorm + Scattered clouds, Light Rain + Scattered clouds; 4: Heavy Rain + Ice Pallets + Thunderstorm + Mist, Snow + Fog
   - `temp, humidity, windspeed` are all what they sound like.
   - `rentals` is the number of rentals, your target variable for prediction.

   Important: We will be fitting on data from 2011 and testing on data from 2012. However, this service became much more popular in 2012. To give your methods a fighting chance, you get the additional information that your overall ridership is up

about 1.6 times. That is, *whenever you make a prediction on the test data, multiply your prediction by 1.6.*

Note: You will be fitting a few different models. Store each of them as you go, since we will evaluate them in different ways later in the problem.

(a) Suppose that you want to predict the number of rentals (`rentals`) based on `dayofyear, dayofweek, workday, weathertype, temp, humidity, windspeed`. Fit a *l*inear model to make this prediction.

Make predictions on your test data. Remember to scale your guesses by 1.6 to account for overall growth. What is your MSE on the test data?

(b) Now fit an additive model. Note that smooth fits make sense for some of your variables, but not others. However, pygam's defaults handle this fine so you don't need to worry.

If you're using MGCV in R, you should wrap the following terms in `s()` in your formula to make them smooth: `dayofyear, temp, humidity, windspeed`.

Make predictions on your test data (again scaling by 1.6). What is your MSE now?

(c) Plot the functions for each of your smooth covariates. What nonlinear trends do you see that a linear model would miss? Give a very brief interpretation for each of your smooth fits.

(d) It turns out that MSE is a strange measure for our inventory management. Consider the following more realistic loss function (though also idealized): For every unit that we are understocked, we lose \$5 because we miss out on a rental. For every unit that we overstock, we lose \$1 because we maintain a surplus inventory:

$$\mathcal{L}(Y, \widehat{Y}) = \begin{cases} 5(Y - \widehat{Y}) & \text{if } Y > \widehat{Y} \\ \widehat{Y} - Y & \text{if } Y < \widehat{Y} \end{cases}$$

The following function computes this new loss:

```
def real_loss(true, guess):
    return(np.mean(np.where(true>guess, 5*(true-guess), guess-true)))
```

What is this loss for your linear and your spline model?

(e) As you saw on your last homework, when the loss is asymmetric, the mean is not a great predictor. One solution would be to optimize the correct loss (in this case, the corresponding method is called *quantile regression*). Instead, let's consider adjusting our estimates with a hack.

Since it hurts more to undershoot than overshoot, we want to skew our estimates upward to improve our loss. It turns out that a reasonable adjustment would be to add the 83.333% (5/6) quantile of our residual distribution to our estimates. (We will discuss this further in class.) Because of the weird scaling across years and to simplify this homework, we'll use the test data to calculate this quantile. Other approaches are possible.

Using the `test` data, calculate the 5/6th quantile of the residuals $Y - \widehat{Y}$ for each predictor (where you have scaled $\widehat{Y}$ by 1.6 already). In python, you can use `np.percentile`; in `R`, you can use `quantile`. Now build new predictors for both your linear method and your additive model by adding the corresponding quantile to your predictions. That is:

$$\widehat{Y}_{new} = \widehat{Y} + \text{quantile}$$

Evaluate these new predictions on the test set in terms of the asymmetric loss function. How does the performance compare to the unadjusted estimator from the previous part? How does the additive model compare to the linear model now?

What would be your final choice of estimator?

4. *Understanding how splines are fit.* This problem is intended to walk you through a subset of the claims I made in class. Consider a smoothing spline fit to n points, $(x_1, y_1), \ldots, (x_n, y_n)$. For a fixed λ, we solve

$$\min_f \sum_{i=1}^{n} (y_i - f(x_i))^2 + \lambda \int \left(f''(x)\right)^2 dx$$

Suppose that you know:

- The solution to such a problem must be a natural cubic spline with knots at the x_i.
- Natural cubic splines can be represented in terms of a basis of known functions of x:

$$f(x) = \beta_0 + \sum_{k=1}^{K} \beta_k b_k(x)$$

Here the $b_k(x)$ are the known basis functions.

This means that to fit your f, you only need to figure out the β_k.

(a) Since we know each of the $b_k(x)$ functions, we could evaluate them at all of our training coordinates x_i. Show that you can rewrite the first part of your optimization objective, $\sum_{i=1}^{n}(y_i - f(x_i))^2$ entirely in terms of the y_i, $b_k(x_i)$, and β_k. Write the term entirely in matrix/vector notation, as we do in regression. You will need to define a matrix of your $b_k(x_i)$ values.

(b) Similarly, rewrite your penalty term, $\int (f''(x))^2 \, dx$ in terms of your functions $b_k(x)$ (and/or their derivatives), β_k. Convert this to a matrix/vector expression.
Hint: You can assume you know all the derivatives of your $b_k(x)$. Your matrices are allowed to have elements that depend on integrals of your $b_k(x)$ functions (and their derivatives). It may help to first think about what $f''(x)$ looks like in terms of the $b_k(x)$ functions, and then move along to squaring and integrating it.

(c) Combining these two pieces, you should now have an objective that is written entirely in terms of matrices and vectors. The only unknowns will be your vector of β coefficients. If everything has gone right, your objective will now remind you very strongly of ridge regression.
Thinking back to your homework from last week, find a solution closed form solution for β (in terms of the matrices you defined above).

The result from this problem means that you can calculate your coefficient β as a linear operator on your vector of Y. Once you have this β, your spline would just be $\beta_0 + \sum_{k=1}^{K} \beta_k b_k(x)$.