(PSL) Coding Assignment 3

Contents

Part I: Optimal span for LOESS

Objective

Write your own function to employ LOO-CV and GCV in selecting the optimal span for LOESS. Definitions of LOO-CV and GCV can be found on page 33 of [lec_W5_NonlinearRegression.pdf]

Below, we'll use CV to refer to LOO-CV.

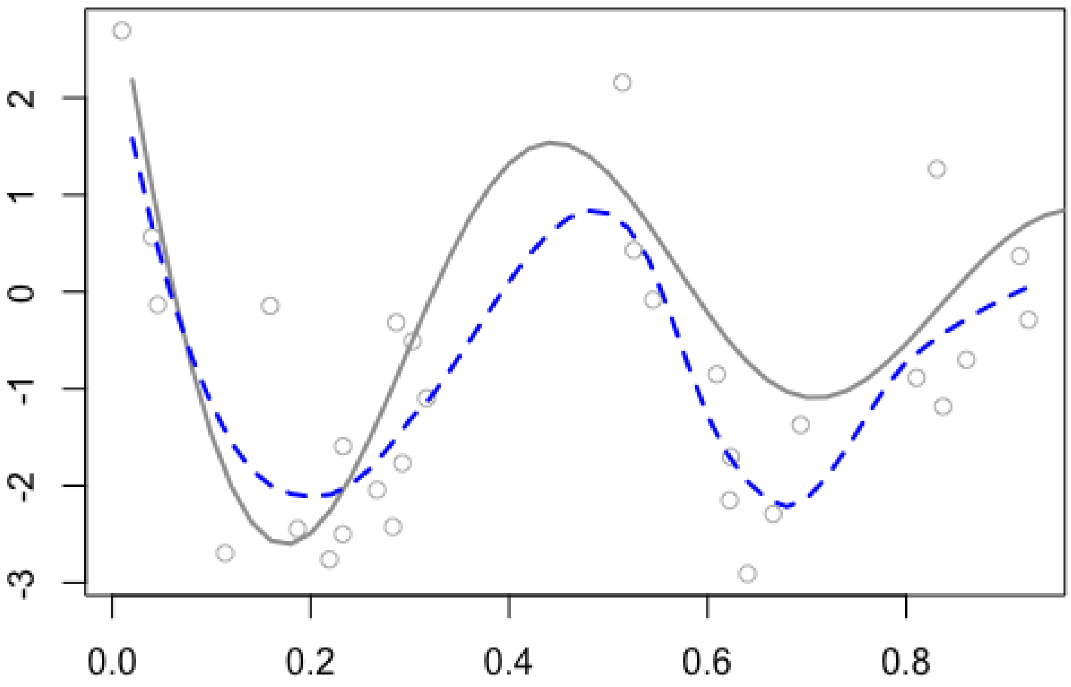
Tasks

1. Write a function to retrieve the **Diagonal of the Smoother Matrix**. We're only interested in the diagonal entries (which will be used in computing LOO-CV and GCV), so this function should return an n-by-1 vector.
   - **Inputs**: `x` (an n-by-1 feature vector) and `span` (a numerical value).
   - **Output**: n-by-1 vector representing the diagonal of the smoother matrix S.
   - If you're using **R**, please refer to the technique we used for computing the smoother matrix in smoothing spline models and adapt it for LOESS.
   - If you're using **Python**, you can either use the aforementioned technique or directly retrieve the diagonal entries from the output of `skmisc.loess` [Link]. Keep in mind that the input to this function should only be `x` and `span`, not `y`. When calling `skmisc.loess` to retrieve the diagonal entries, you can use a fake `y` vector, as the diagonal entries remain the same and do not depend on the response vector `y`.
2. Write a function to find the **Optimal Span(s)** based on CV and GCV.
   - Iterate over the specified span values.
   - For each span, calculate the CV and GCV values.
   - Return the CV and GCV values corresponding to each span.
   - Determine the best span(s) based on the CV and GCV results.
3. Test your code using the provided dataset [Coding3_Data.csv].
   - Report your CV and GCV for the following 15 span values: 0.20, 0.25, . . . , 0.90.
   - Determine the best span value based on the CV and GCV results. For this dataset, both methods recommend the same span.

- Fit a LOESS model over the entire dataset using the selected optimal span.
- Display the original data points and overlay them with the true curve and the fitted curve. Include a legend to distinguish between the two curves.
- The true curve is

$$f(x) = \frac{\sin(12(x+0.2))}{x+0.2}, \quad x \in [0,1]$$

- **Note**: To display a smooth curve, evaluate your function on a finer grid of points and then plot those points. Your plot may differ from the one shown below, and you are encouraged to use your own color schemes and styles.



Part II: Ridgeless and double descent

Objective

So far in our course, we've utilized the U-shaped bias-variance trade-off curve as a pivotal tool for model selection. This has aided us in methodologies such as ridge/lasso regression, tree pruning, and smoothing splines, among others.

A key observation is that when a model interpolates training data to the extent that the Residual Sum of Squares (RSS) equals zero, it's typically a red flag signaling overfitting. Such models are anticipated to perform inadequately when presented with new, unseen data.

> However, in modern practice, very rich models such as neural networks are trained to exactly fit (i.e., interpolate) the data. Classically, such models would be considered overfitted, and yet they often obtain high accuracy on test data. This apparent contradiction has raised questions about the mathematical foundations of machine learning and their relevance to practitioners. (Belkin et al. 2019)

In this assignment, we will use **Ridgeless** to illustrate the **double descent** phenomenon. Our setup is similar to, but not the same as, Section 8 in Hastie (2020).

Data

Remember the dataset used in Coding 2 Part I? It consisted of 506 rows (i.e., $n = 506$) and 14 columns: Y, X_1 through X_{13}.

Based on this dataset, we have formed Coding3_dataH.csv, which is structured as follows:

- It contains 506 rows, corresponding to $n = 506$.
- There are 241 columns in total. The first column represents Y. The subsequent 240 columns relate to the NCS basis functions for each of the 13 X variables. The number of knots are individually determined for each feature.

Task 1: Ridgeless Function

Your task is to write a function `ridgeless` to implement Ridgeless Least Squares, which is equivalent to Principal Component Regression (PCR) using all principal components, with the option `scale = FALSE`. This means that when computing the principal components (PCs), you center each column of the design matrix without scaling.

- **Input**: Train and test datasets. In both datasets, the first column represents the response vector Y, and the remaining columns represent the features X.
- **Output**: Training and test errors, where the error refers to the average squared error.
- You are allowed to use R/Python packages or built-in functions for PCA/SVD, but you are not permitted to use packages or functions specifically designed for linear regression, PCR, or ridge regression.
- Additionally, you are **not allowed to perform matrix inversion** in your code. After performing PCA/SVD, the resulting design matrix will have orthogonal columns. This allows you to compute the least squares coefficients through simple matrix multiplication, avoiding the need for matrix inversion.
- For numerical stability, you should exclude directions with extremely small eigenvalues (in PCA) or singular values (in SVD). For example, consider using a threshold of `eps = 1e-10` for singular values.
- Although training errors aren't required for our simulation, I recommend including them in the ridgeless output as a helpful debugging tool. Ideally, the training error should match the residual sum of squares (RSS) from a standard linear regression model.

Task 2: Simulation study

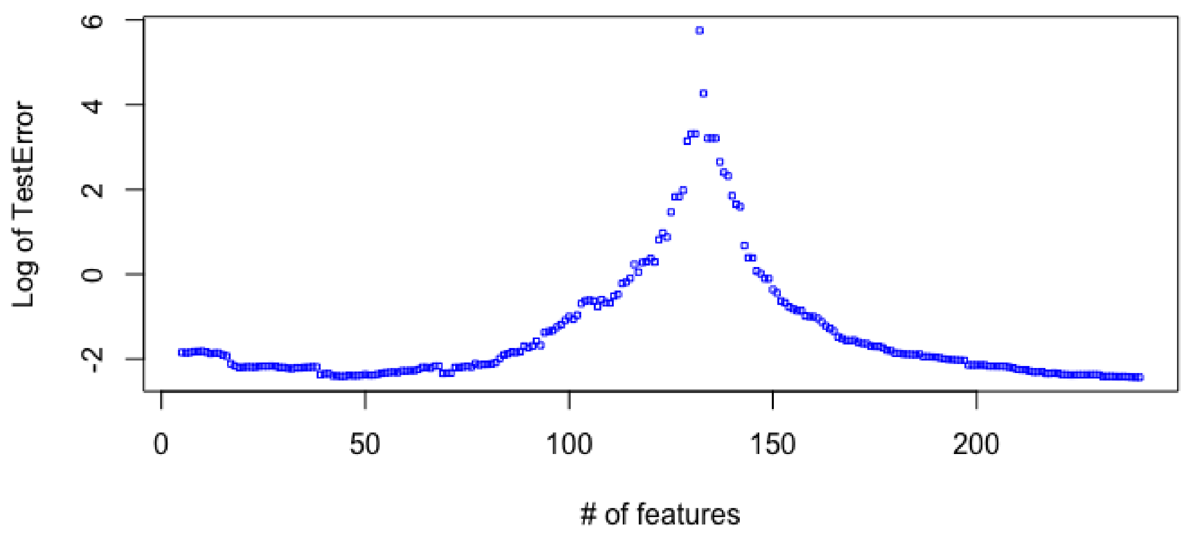
Execute the procedure below for $T = 30$ times.

In each iteration,

- Randomly partition the data into training (25%) and test (75%).

- Calculate the log of the test error from the `ridgeless` function using the first d columns of the data, where d ranges from 6 to 241. Keep in mind that the number of regression parameters spans from 5 to 240 because the first column represents Y.
- This will result in recording 236 test errors per iteration. These errors are the averaged mean squared errors based on the test data. Store those test errors in a matrix of dimensions 30-by-236.

Graphical display: Plot the median of the test errors (collated over the 30 iterations) **in log scale** against the count of regression parameters, which spans from 5 to 240.



Part III: Clustering time series

Objective

Cluster time series data based on their fluctuation patterns using natural cubic splines.

Data

Download[Sales_Transactions_Dataset_Weekly.csv]. The original data is from from UCI Machine Learning Repository [Link]

This dataset contains the weekly purchased quantities of 811 products over 52 weeks, resulting in a time series of 52 data points for each product.

Center each time series by **removing its mean**, and store the resulting data in an 811-by-52 matrix $\mathbf{X}$.

Task 1: Fitting NCS

- Fit each time series with an NCS with `df = 10`. This corresponds to an NCS with 8 interior knots. Each row of $\mathbf{X}_{811\times 52}$ represents the response, with the 1-dimensional feature being the index from 1 to

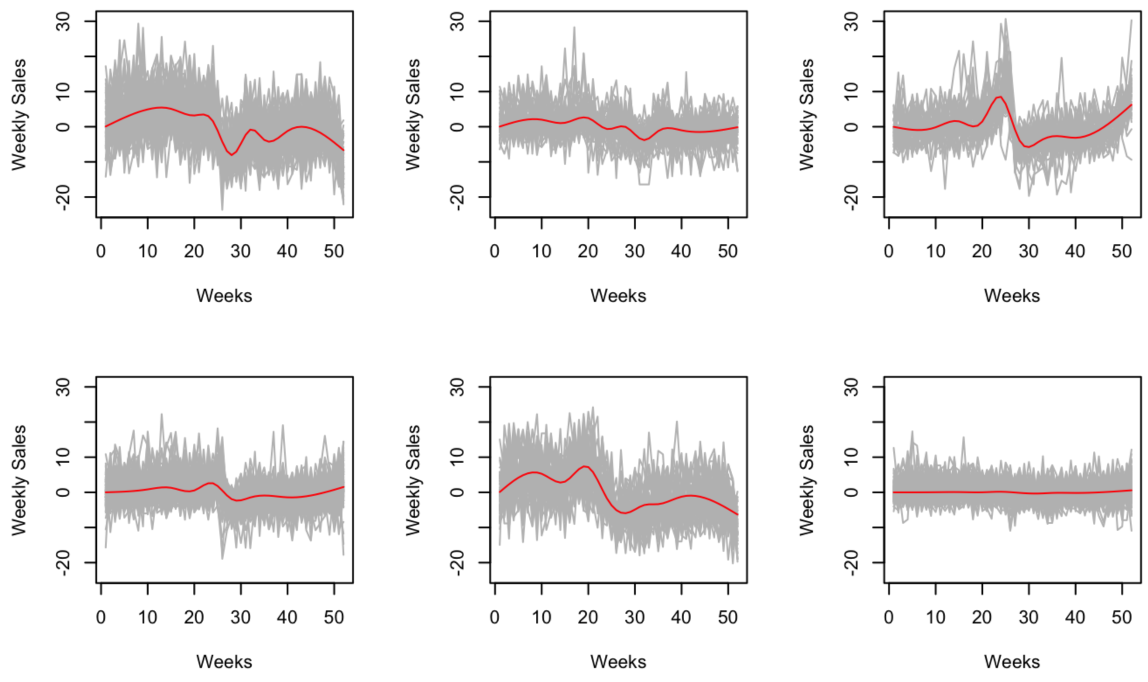
52.

- Store the NCS coefficients (excluding the intercept) in an 811-by-9 matrix $\mathbf{B}_{811\times 9}$.
- Matrix $\mathbf{B}$ can be derived as follows:
  - $\mathbf{F}$ is a 52-by-9 design matrix without the intercept. For instance, this can be obtained using the `ns` command with `df = 9` and `intercept = FALSE`.
  - **Remove the column mean from F** to disregard the intercept.
  - Then compute $\mathbf{B}$ via
$$\mathbf{B}^t = (\mathbf{F}^t\mathbf{F})^{-1}\mathbf{F}^t\mathbf{X}^t.$$
  - The formula above is given for the transpose of $\mathbf{B}$, since it is equivalent to fitting 811 linear regression models: the design matrix stays the same (i.e., $\mathbf{F}$) but the response vector — there are 811 response vectors corresponding to the 811 rows of $\mathbf{X}$ — would vary; each nine dimensional regression coefficient vector corresponds to a row in $\mathbf{B}$ (or equivalently, column in $\mathbf{B}^t$).
  - If you do not want to remove the column mean from $\mathbf{F}$, add the intercept column to $\mathbf{F}$ and denote the resulting 52-by-10 matrix as $\tilde{\mathbf{F}}$. Then compute the 811-by-10 matrix $\tilde{\mathbf{B}}$:
$$\tilde{\mathbf{B}}^t = (\tilde{\mathbf{F}}^t\tilde{\mathbf{F}})^{-1}\tilde{\mathbf{F}}^t\mathbf{X}^t.$$

    Next, obtain $\mathbf{B}$ by dropping the first column of $\tilde{\mathbf{B}}$, as this column corresponds to the intercepts from the 811 regression models.

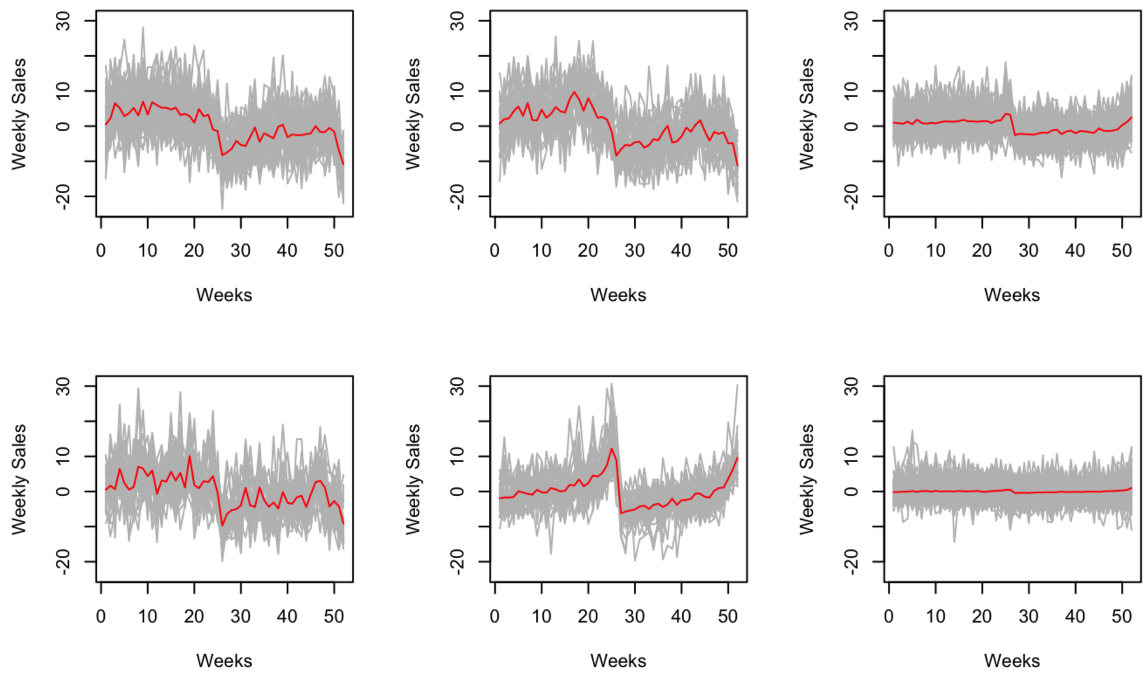
Task 2: Cluster Matrix B

- Run the k-means algorithm on matrix $\mathbf{B}$ to cluster the 811 products into 6 clusters.
- Visualize the **centered** time series (i.e., rows of $\mathbf{X}$), colored in grey, of products grouped by their clusters. Overlay the plots with the cluster centers (colored in red). Arrange the visualizations in a 2-by-3 format.
- **Note**: When using matrix $\mathbf{B}$ for clustering, the centers are the average of the rows of $\mathbf{B}$ within each cluster. To get the corresponding time series from a cluster center $\mathbf{b}$, use the matrix product $\mathbf{Fb}$.



Task 3: Cluster Matrix X

- Run the k-means algorithm on matrix **X** to cluster the 811 products into 6 clusters.
- Similarly, visualize the **centered** time series of products grouped by their clusters, accompanied by their respective cluster centers. Arrange the visualizations in a 2-by-3 format.


Weekly Sales
Weeks