STAT 339: HOMEWORK 2 (LINEAR REGRESSION)

DUE VIA GITHUB BY 11:59 P.M. WEDNESDAY 10/27

Instructions. Create a directory called `hw2` in your `stat339` GitHub repo. Your main writeup should be called `hw2.pdf`, and any source code should either be in that directory, a subdirectory within it, or a "library" directory at the top level (in the case of files defining functions used in multiple assignments).

I suggest placing the definitions of any "helper" functions in a separate file which you load (in Python, `import`) from your main file.

I will access your work by cloning your repository; make sure that any file path information is written relative to your repo – don't use absolute paths on your machine, or the code won't run for me!

You may use any typesetting software to prepare your writeup, but the final document should be a PDF. LaTeXis highly encouraged.

All data files referred to in the problems below can be found at

`http://colindawson.net/data/<filename>.csv`

by replacing `<filename>` with the name of the dataset (for example, `womens100`).

1. **Implementing a Regression Solver From Scratch.**

   (a) (Adapted from FCML Ex. 1.2) Write a function that takes a dataset consisting of an $N \times (D \times 1)$ input matrix, $\mathbf{X}$ (you can assume that the column of 1s has already been added) and an $N \times 1$ target vector, $\mathbf{t}$, and returns a $(D+1) \times 1$ weight vector $\mathbf{w}$, consisting of the OLS coefficients for a linear model of the form $\hat{\mathbf{t}} = \mathbf{X}\mathbf{w}$.

   (b) (Adapted from FCML Ex. 1.6) Verify that your function works using the data in `womens100.csv` containing winning race times from the Olympic women's 100m event in various years. Plot the data and the resulting line and compare the coefficients returned by your model to the coefficients returned by a pre-packaged regression solver such as the `LinearRegression()` solver from `sklearn.linear_model` in Python (see also Sec. 1.2.1 of the textbook).

Date: Last Revised: October 21, 2021.

(c) The data ends with the 2008 olympics. Use your model to "predict" the winning times for the 2012 and 2016 olympics. Now look up the actual times; how does the model do? Report the squared prediction error for each year.

(d) Write a function that takes a single predictor column and a positive integer D, and creates a predictor matrix whose columns consist of powers of the entries in the original column from 0 to D

(e) Write a function that takes a target vector, a **single** predictor column, and a positive integer, D, and returns the coefficients of the OLS polynomial function of order D.

(f) (Adapted from FCML Ex. 1.9) Apply your function to the data in `synthdata2016.csv` using a cubic polynomial ($D = 3$). The data file as it is consists of an x column followed by a t column. Plot the data and the fitted cubic curve.

(g) Modify your solver to allow the user to specify a **regularization parameter**, λ, to do ridge regression using a loss function that penalizes the **standardized weights**. That is:

$$\mathcal{L}(\mathbf{w}) = (\mathbf{t} - \mathbf{QRw})^\mathsf{T}(\mathbf{t} - \mathbf{QRw}) + (\mathbf{Rw}^\mathsf{T})\mathbf{\Lambda Rw}$$

where $\mathbf{QR} = \mathbf{X}$ is the QR decomposition of $\mathbf{X}$ and $\mathbf{\Lambda}$ is a $((D+1) \times (D+1)$ matrix whose first diagonal entry and all off-diagonal entries are 0, and whose remaining diagonal entries are equal to the regularization parameter λ. Using $\mathbf{\Lambda}$ here in place of simply multiplying by the scalar λ exempts w_0 from any regularization penalty.

Recall that in QR decomposition, $\mathbf{Q}$ is an $N \times (D+1)$ matrix whose columns are standardized and uncorrelated (which means that $\mathbf{Q}^\mathsf{T}\mathbf{Q} = \mathbf{I}$), and whose first d columns are linear combinations of the first d columns of $\mathbf{X}$. That is, it contains the same information as $\mathbf{X}$ but expressed in different "coordinates". Meanwhile, $\mathbf{R}$ is the $(D+1) \times (D+1)$ matrix representing the linear translation from the weights as expressed in terms of the columns of $\mathbf{X}$ to weights expressed in terms of the columns of $\mathbf{Q}$. That is, $\mathbf{Xw} = \mathbf{Q}\boldsymbol{\theta}$ are the same prediction vector, where $\theta = \mathbf{Rw}$. In Python, $\mathbf{Q}$ and $\mathbf{R}$ can be found from $\mathbf{X}$ using `numpy.linalg.qr()`.

(h) Compare results for various choices of λ using a cubic model on the `synthdata2016.csv` data.

(i) Create a version of your solver that, instead of returning the coefficients, returns a **prediction function**. The resulting function should take a new predictor matrix, $\mathbf{X}_{\text{new}}$ as input and returns a vector $\hat{\mathbf{t}}$ of predictions (this will

be useful for the purposes of recycling cross-validation code you wrote for the previous problem set).

2. **Cross-Validation With Regression.**

   (a) Write a function that takes as input a target vector, $\mathbf{t}$, and a prediction vector, $\hat{\mathbf{t}}$, and returns the mean squared prediction error (MSE).

   (b) Write a function that takes the following inputs:

      (i) A target vector, $\mathbf{t}$

      (ii) A predictor matrix, $\mathbf{X}$

      (iii) A number of folds, J (default to 10)

      (iv) A random seed

      (v) A regularization parameter, λ (default to 0)

      (vi) A boolean (`True` or `False`) flag to govern whether to return training error (default to `False`).

   and performs J-fold cross-validation. Reuse as much code as possible to have this function split $\mathbf{X}$ into J random folds, fit a (ridge) regression model to each set of $K-1$ folds as the training set, compute the mean squared prediction error (MSE) on the remaining fold (as the validation set), and return the mean and standard deviation of the MSE across folds.

   Implementation Tip: Be careful when randomly splitting the data into folds that you keep rows of $\mathbf{X}$ inputs together with the corresponding entry in $\mathbf{t}$. You may want to represent your folds using a partition of indices rather than randomly splitting $\mathbf{X}$ directly; alternatively, you could do the split into folds before splitting the data into the $\mathbf{X}$ and $\mathbf{t}$ components.

   (c) Write a function to select the best polynomial order using cross-validation. It should take as inputs:

      (i) A **target** vector, $\mathbf{t}$

      (ii) A (single column) **predictor** matrix, $\mathbf{X}$ (which will need to get expanded to form the desired polynomial basis)

      (iii) A positive integer $D_{\max}$, representing the **maximum order of polynomial** to consider (default to $D_{\max} = N$, the number of data points)

(iv) An integer J (at least 2), representing the number of **folds** to use

(v) A **random seed**

(vi) A **boolean flag** to govern whether to return training error (default to False).

Your function should perform cross-validation for each polynomial order D ranging from 0 to $D_{\max}$, and return the mean and standard deviation across folds of the MSE for each order. If the option to return training error is turned on, it should also return the mean and standard deviation across folds of the training error for each polynomial order. In addition, it should find and return the polynomial order with the lowest average cross-validation MSE. (You may want to split this up into modular functions that call each other)

(d) Apply the function you wrote in 2c to both the `synthdata2016.csv` and the `womens100.csv` datasets to identify the optimal polynomial order. Try both $J = 10$ and $J = N$ (i.e., leave-one-out) cross-validation. Note that during cross-validation, if $\lambda = 0$ (corresponding to the OLS solution), the size of $D_{\max}$ is limited by the size of the smallest training set, because the $\mathbf{X}^\mathsf{T}\mathbf{X}$ matrix will not be invertible if $D + 1 > N_{\text{train}}$.

Plot the mean training and validation errors for each polynomial order, as well as error bars 1 standard deviation in either direction, and comment on what the plots show.

(e) For the `womens100.csv` dataset, we can use the 2012 and 2016 times as a true test set. Compute squared prediction errors for each of the polynomial degree models (fit on the full dataset) for 2012 and 2016. Did cross validation give the best model on this metric?

(f) Now use 10-fold cross-validation and a grid search to find a good combination of values of λ and D for a ridge regression model. For λ use logarithmic spacing from $\log_{10}(\lambda) = -6$ to $\log_{10}(\lambda) = 2$. How does generalization performance (on the true test set) for the chosen model and regularization parameter compare to that for the best OLS regression fit?

3. **Product Rule of Matrix Differentiation.** (No coding here)

(a) Prove that the product rule shown in class is valid. That is, show that for functions f and g that take vectors in $\mathbb{R}^N$ and return vectors in $\mathbb{R}^M$, N-dimensional vector $\mathbf{v}$, and scalar function

$$a(\mathbf{v}) = f(\mathbf{v})^\mathsf{T} g(\mathbf{v})$$

the following "product rule" applies

$$\frac{da(\mathbf{v})}{d\mathbf{v}} = f(\mathbf{v})^\mathsf{T} \frac{dg(\mathbf{v})}{d\mathbf{v}} + g(\mathbf{v})^\mathsf{T} \frac{df(\mathbf{v})}{d\mathbf{v}}$$

(**Hint:** To make it easier to remember that $f(\mathbf{v})$ and $g(\mathbf{v})$ are M-dimensional vectors, denote the entries of $f_1(\mathbf{v})$ through $f_M(\mathbf{v})$ as the scalar coordinates of $f(\mathbf{v})$, and similarly for $g(\mathbf{v})$.)

(b) Apply the product rule to the special case where $M = N$, f is the identity function, and $g(\mathbf{v}) = \mathbf{A}\mathbf{v}$ for some **symmetric** matrix $\mathbf{A}$ that does not depend on $\mathbf{v}$ to show that

$$\frac{d}{d\mathbf{v}} \mathbf{v}^\mathsf{T} \mathbf{A} \mathbf{v} = 2\mathbf{v}^\mathsf{T} \mathbf{A}$$

4. **Deriving a Weighted Least Squares Regression Fit.** (adapted from FCML Ex. 1.11) (No coding here) Consider a generalization of the OLS loss function that applies (known) weights to each observation, given by:

$$\mathcal{L}(\mathbf{w}; \mathbf{x}, \mathbf{t}) = \frac{1}{N} \sum_{n=1}^{N} \alpha_n (t_n - \mathbf{w}^\mathsf{T} \mathbf{x}_n)^2$$

(or, equivalently, for the whole dataset by

$$\mathcal{L}(\mathbf{w}; \mathbf{X}, \mathbf{t}) = \frac{1}{N} (\mathbf{t} - \mathbf{X}\mathbf{w})^\mathsf{T} \mathbf{A} (\mathbf{t} - \mathbf{X}\mathbf{w})$$

where $\mathbf{X}$ is an input data matrix whose nth row consists of the input vector $\mathbf{x}_n$, and $\mathbf{A}$ is a diagonal matrix whose nth diagonal entry is α_n (and whose off-diagonal entries are all zero, which is part of the definition of a "diagonal" matrix).

Derive an expression for the estimated weight vector $\hat{\mathbf{w}}$, represented in terms of $\mathbf{x}$, $\mathbf{t}$ and $\mathbf{A}$, that minimizes this loss.

(**Hint:** Make use of the propositions for matrix differentiation that we proved in class and that you proved in the previous problem)