Assignment #1

Due: 7:59pm ET, February 4, 2022

Homework 1: Regression

Introduction

This homework is on different forms of linear regression and focuses on loss functions, optimizers, and regularization. Linear regression will be one of the few models that we see that has an analytical solution. These problems focus on deriving these solutions and exploring their properties.

If you find that you are having trouble with the first couple problems, we recommend going over the fundamentals of linear algebra and matrix calculus (see links on website). The relevant parts of the cs181-textbook notes are Sections 2.1 - 2.7. We strongly recommend reading the textbook before beginning the homework.

We also encourage you to first read the Bishop textbook, particularly: Section 2.3 (Properties of Gaussian Distributions), Section 3.1 (Linear Basis Regression), and Section 3.3 (Bayesian Linear Regression). (Note that our notation is slightly different but the underlying mathematics remains the same!).

Please type your solutions after the corresponding problems using this LaTeX template, and start each problem on a new page. You may find the following introductory resources on LaTeX useful: LaTeX Basics and LaTeX tutorial with exercises in Overleaf

Homeworks will be submitted through Gradescope. You will be added to the course Gradescope once you join the course Canvas page. If you haven't received an invitation, contact the course staff through Ed.

Please submit the writeup PDF to the Gradescope assignment 'HW1'. Remember to assign pages for each question.

Please submit your LaTeXfile and code files to the Gradescope assignment 'HW1 - Supplemental'. Your files should be named in the same way as we provide them in the repository, e.g. `T1_P1.py`, etc.

Problem 1 (Optimizing a Kernel, 15pts)

Kernel-based regression techniques are similar to nearest-neighbor regressors: rather than fit a parametric model, they predict values for new data points by interpolating values from existing points in the training set. In this problem, we will consider a kernel-based regressor of the form:

$$f(x^*) = \sum_n K(x_n, x^*) y_n$$

where (x_n, y_n) are the training data points, and $K(x, x')$ is a kernel function that defines the similarity between two inputs x and x'. Assume that each x_i is represented as a column vector, i.e. a D by 1 vector where D is the number of features for each data point. A popular choice of kernel is a function that decays as the distance between the two points increases, such as

$$K(x, x') = \exp\left(\frac{-||x - x'||_2^2}{\tau}\right) = \exp\left(\frac{-(x - x')^T (x - x')}{\tau}\right)$$

where τ represents the square of the lengthscale (a scalar value). In this problem, we will consider optimizing what that (squared) lengthscale should be.

1. Let $\{(x_n, y_n)\}_{n=1}^N$ be our training data set. Suppose we are interested in minimizing the residual sum of squares. Write down this loss over the training data $\mathcal{L}(W)$ as a function of τ.

   Important: When computing the prediction $f(x_i)$ for a point x_i in the training set, carefully consider for which points x' you should be including the term $K(x_i, x')$ in the sum.

2. Take the derivative of the loss function with respect to τ.

Problem 1 (cont.)

3. Consider the following data set:

```
x , y
0 , 0
1 , 0.5
2 , 1
3 , 2
4 , 1
6 , 1.5
8 , 0.5
```

   And the following lengthscales: $\tau = .01$, $\tau = 2$, and $\tau = 100$.

   Write some Python code to compute the loss with respect to each kernel for the dataset provided above. Which lengthscale does best? For this problem, you can use our staff **script to compare your code to a set of staff-written test cases.** This requires, however, that you use the structure of the starter code provided in `T1_P1.py`. More specific instructions can be found at the top of the file `T1_P1_Testcases.py`. You may run the test cases in the command-line using `python T1_P1_TestCases.py`. **Note that our set of test cases is not comprehensive: just because you pass does not mean your solution is correct! We strongly encourage you to write your own test cases and read more about ours in the comments of the Python script.**

4. Plot the function $(x^*, f(x^*))$ for each of the lengthscales above. You will plot x^* on the x-axis and the prediction $f(x^*)$ on the y-axis. For the test inputs x^*, you should use an even grid of spacing of 0.1 between $x^* = 0$ and $x^* = 12$. (Note: it is possible that a test input x^* lands right on top of one of the training inputs above. You can still use the formula!)

   Initial impressions: Briefly describe what happens in each of the three cases. Is what you see consistent with the which lengthscale appeared to be numerically best above? Describe why or why not.

5. Bonus: Code up a gradient descent to optimize the kernel for the data set above. Start your gradient descent from $\tau = 2$. Report on what you find.

   Note: Gradient descent is discussed in Section 3.4 of the cs181-textbook notes and Section 5.2.4 of Bishop, and will be covered later in the course!

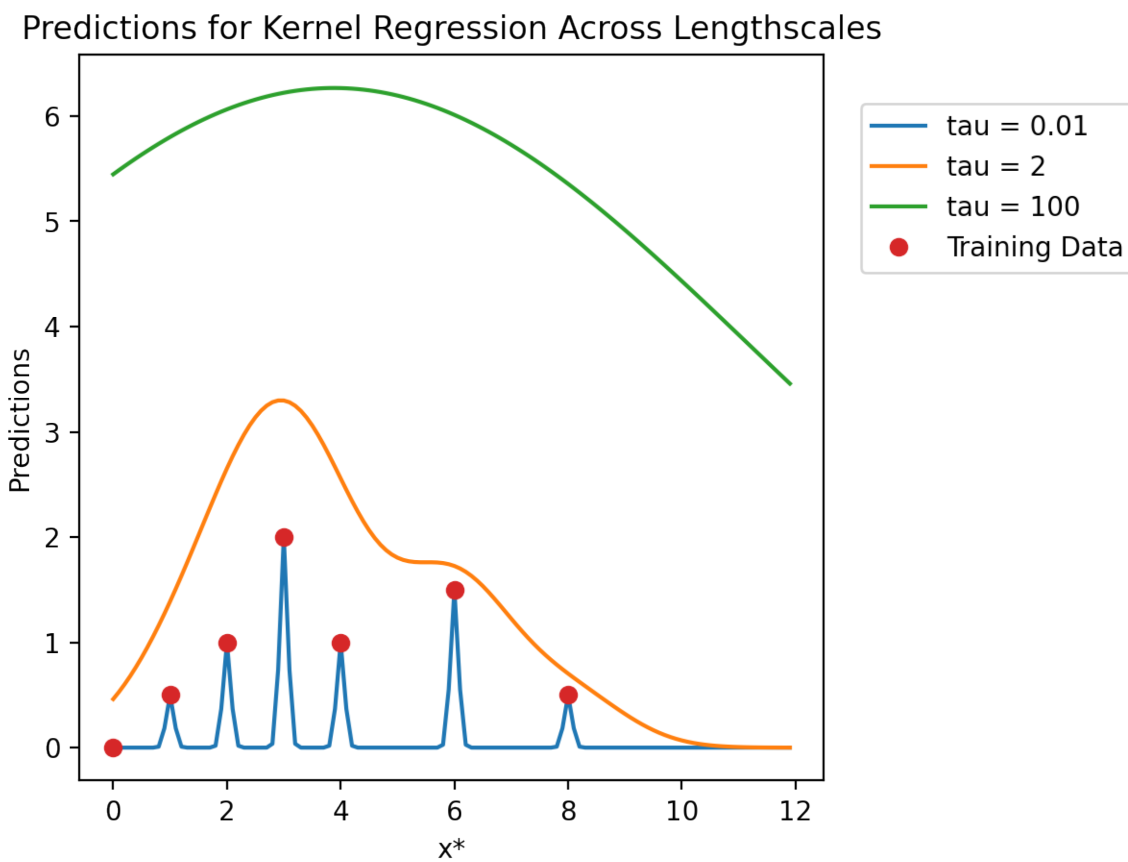
Solution

1. $\mathcal{L}(W) = \sum_{n=1}^{N}(y_n - \sum_{i\neq n} K(x_i, x_n)y_i)^2 = \sum_{n=1}^{N}(y_n - \sum_{i\neq n} \exp\left(\frac{-(x_n-x_i)^T(x_n-x_i)}{\tau}\right) y_i)^2$

2. $\frac{\partial \mathcal{L}(W)}{\partial \tau} = 2\sum_{n=1}^{N}[y_n - \sum_{i\neq n} \exp\left(\frac{-(x_n-x_i)^T(x_n-x_i)}{\tau}\right) y_i][y_i \sum_{i\neq n} \exp\left(\frac{-(x_n-x_i)^T(x_n-x_i)}{\tau}\right)\left(\frac{-(x_n-x_i)^T(x_n-x_i)}{\tau^2}\right)]$

3. Code is uploaded as "T1_P1.py".
   The losses for when $\tau = 0.01, 2, 100$ are respectively 8.75, 3.31, and 120.36. It appears that 2 is the optimal lengthscale among our choices.

4. When the lengthscale is as small as 0.01, predictions turn to shrink to zero unless the test x is very close to the training data we have. On the other extreme, when tau is as big as 100, predictions tend to be more similar across different X's because the influence of distance between the test point and all the other training points is weakened by a large denominator. Therefore, tau = 2 is the best among our choices which is consistent with our loss computation above.
   It is worth noting that here it seems that the loss for when $\tau = 0.01$ should be very close to zero since we predict them nearly perfectly. When we calculated the loss, we ignored all points exactly located on training points while in predictions we included them.



Problem 2 (Kernels and kNN, 10pts)

Now, let us compare the kernel-based approach to an approach based on nearest-neighbors. Recall that kNN uses a predictor of the form

$$f(x^*) = \frac{1}{k} \sum_n y_n \mathbb{I}(x_n \texttt{ is one of k-closest to } x^*)$$

where $\mathbb{I}$ is an indicator variable. For this problem, you will use the **same dataset and kernel as in Problem 1**.

For this problem, you can use our staff **script to compare your code to a set of staff-written test cases.** This requires, however, that you use the structure of the starter code provided in `T1_P2.py`. More specific instructions can be found at the top of the file `T1_P2_Testcases.py`. You may run the test cases in the command-line using `python T1_P2_TestCases.py`. **Note that our set of test cases is not comprehensive: just because you pass does not mean your solution is correct! We strongly encourage you to write your own test cases and read more about ours in the comments of the Python script.**

Make sure to include all required plots in your PDF.

1. Implement kNN for $k = \{1, 3, N - 1\}$ where N is the size of the dataset, then plot the results for each k. To find the distance between points, use the kernel function from Problem 1 with lengthscale $\tau = 1$.

   As before, you will plot x^* on the x-axis and the prediction $f(x^*)$ on the y-axis. For the test inputs x^*, you should use an even grid of spacing of 0.1 between $x^* = 0$ and $x^* = 12$. (Like in Problem 1, if a test point lies on top of a training input, use the formula without excluding that training input.)

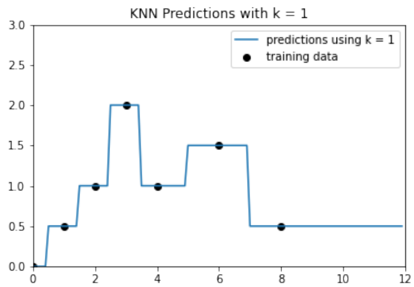
   You may choose to use some starter Python code to create your plots provided in `T1_P2.py`. Please **write your own implementation of kNN** for full credit. Do not use external libraries to find nearest neighbors.

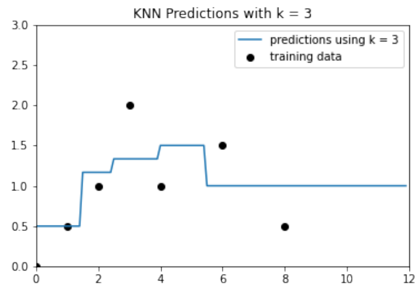
2. Describe what you see: What is the behavior of the functions in these three plots? How does it compare to the behavior of the functions in the three plots from Problem 1? Are there situations in which kNN and kernel-based regression interpolate similarly? Extrapolate similarly? Based on what you see, do you believe there exist some values of k and τ for which the kNN and kernel-based regressors produce the exact same classifier (ie. given *any* point x, the two regressors will produce the same prediction $f(x)$)? Explain your answer.

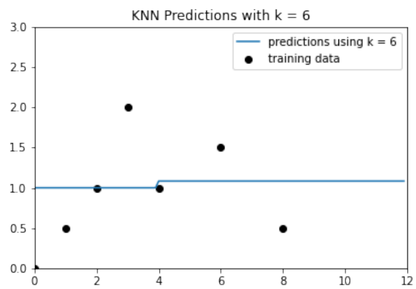
3. Why did we not vary τ for the kNN approach?

Solution

1. Code is uploaded as "T1_P1.py".







2. As k increases from 1 to 3 to 6, kNN predictions give more and more similar results across different x's, which resembles the three plots from Problem 1, where as tau goes from 0.01 to 2 to 100, the predictions are more and more similar, which is because the model is less and less flexible as k or tau increases and bias increases as variance decreases.

   kNN and kernel- based regressors interpolate similarly since k and τ control flexibility. However, they do not extrapolate similarly. For kNN, extrapolations outside of the range of training data give the same result on predicting on marginal data. While for kernel- based regressors, distance from the range matters since as the distance go larger, weight of all the training data go to zero.

   I don't think there exist some values of k and τ for which the kNN and kernel- based regressors produce the exact same classifier. kNN uses a discrete distance weight while kernel- based regressors uses a continuous distance weight, as can be shown from the two sets of plots above where predictions of kNNs are bumpy intervals while kernel- based regressors are continuous curves.

3. We do not change τ because here we need to find a way to quantify distance so as to determine nearest neighbours. By changing τ, we are playing with different quantification but they result in the same ranking and therefore the same nearest neighbours.

Problem 3 (Deriving Linear Regression, 10pts)

The solution for the least squares linear regressions "looks" kind of like a ratio of covariance and variance terms. In this problem, we will make that connection more explicit.

Let us assume that our data are tuples of scalars (x, y) that are described by some joint distribution $p(x, y)$. For clarification, the joint distribution $p(x, y)$ is just another way of saying the "joint PDF" $f(x, y)$, which may be more familiar to those who have taken Stat 110, or equivalent.

We will consider the process of fitting these data from this distribution with the best linear model possible, that is a linear model of the form $\hat{y} = wx$ that minimizes the expected squared loss $E_{x,y}[(y - \hat{y})^2]$.

Notes: The notation $E_{x,y}$ indicates an expectation taken over the joint distribution $p(x, y)$. Since x and y are scalars, w is also a scalar.

1. Derive an expression for the optimal w, that is, the w that minimizes the expected squared loss above. You should leave your answer in terms of moments of the distribution, e.g. terms like $E_x[x]$, $E_x[x^2]$, $E_y[y]$, $E_y[y^2]$, $E_{x,y}[xy]$ etc.
2. Provide unbiased and consistent formulas to estimate $E_{x,y}[yx]$ and $E_x[x^2]$ given observed data $\{(x_n, y_n)\}_{n=1}^N$.
3. In general, moment terms like $E_{x,y}[yx]$, $E_{x,y}[x^2]$, $E_{x,y}[yx^3]$, $E_{x,y}[\frac{x}{y}]$, etc. can easily be estimated from the data (like you did above). If you substitute in these empirical moments, how does your expression for the optimal w^* in this problem compare with the optimal w^* that we see in Section 2.6 of the cs181-textbook?
4. Many common probabilistic linear regression models assume that variables x and y are jointly Gaussian. Did any of your above derivations rely on the assumption that x and y are jointly Gaussian? Why or why not?

Solution

1.

$$\begin{aligned} E_{x,y}[(y-\hat{y})^2] &= E_{x,y}[(y-wx)^2] \\ \frac{\partial}{\partial w} E_{x,y}[(y-wx)^2] &= E_{x,y}[-2x(y-wx)] \overset{set}{=} 0 \\ E_{x,y}[-2xy+2\hat{w}x^2] &= 0 \\ \hat{w} &= \frac{E_{x,y}(xy)}{E_x(x^2)} \end{aligned} \tag{1}$$

2. To estimate $E_{x,y}(yx)$ and $E_x(x^2)$, we can respectively use the formulas $\frac{1}{n}\sum_{i=1}^n x_i y_i$ and $\frac{1}{n}\sum_{i=1}^n x_i^2$.

3. $\hat{w} = \frac{E_{x,y}(xy)}{E_x(x^2)} = \frac{\sum_{i=1}^n x_i y_i}{\sum_{i=1}^n x_i^2}$
   In the textbook we see $\mathbf{w}^* = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$, which is just the matrix form of what we derived here.

4. None of the derivation above requires that x and y are jointly Gaussian. Here we are not interested in deriving uncertainty of the coefficients, therefore we do not need distribution assumptions in our calculations. In other words, the distribution of the error term does not influence the expected values of the coefficients.

   Typically, we assume $Y \sim N(\beta x, \sigma^2)$ but we do not assume the distribution of X.

Problem 4 (Modeling Changes in Republicans and Sunspots, 15pts)

The objective of this problem is to learn about linear regression with basis functions by modeling the number of Republicans in the Senate. The file `data/year-sunspots-republicans.csv` contains the data you will use for this problem. It has three columns. The first one is an integer that indicates the year. The second is the number of Sunspots observed in that year. The third is the number of Republicans in the Senate for that year. The data file looks like this:

```
Year,Sunspot_Count,Republican_Count
1960,112.3,36
1962,37.6,34
1964,10.2,32
1966,47.0,36
```

You can see scatterplots of the data in the figures below. The horizontal axis is the Year, and the vertical axis is the Number of Republicans and the Number of Sunspots, respectively.

(Data Source: http://www.realclimate.org/data/senators_sunspots.txt)

Make sure to include all required plots in your PDF.

1. In this problem you will implement ordinary least squares regression using 4 different basis functions for **Year (x-axis)** v. **Number of Republicans in the Senate (y-axis)**. Some starter Python code that implements simple linear regression is provided in `T1_P4.py`.

   Note: The numbers in the *Year* column are large (between 1960 and 2006), especially when raised to various powers. To avoid numerical instability due to ill-conditioned matrices in most numerical computing systems, we will scale the data first: specifically, we will scale all "year" inputs by subtracting 1960 and then dividing by 40. Similarly, to avoid numerical instability with numbers in the *Sunspot_Count* column, we will also scale the data first by dividing all "sunspot count" inputs by 20. Both of these scaling procedures have already been implemented in lines $65-69$ of the starter code in `T1_P4.py`. Please do *not* change these lines!

   First, plot the data and regression lines for each of the following sets of basis functions, and include the generated plot as an image in your submission PDF. You will therefore make 4 total plots:

   (a) $\phi_j(x) = x^j$ for $j = 1, \ldots, 5$
   ie, use basis $y = a_1x^1 + a_2x^2 + a_3x^3 + a_4x^4 + a_5x^5$ for some constants $\{a_1, ..., a_5\}$.

   (b) $\phi_j(x) = \exp\frac{-(x-\mu_j)^2}{25}$ for $\mu_j = 1960, 1965, 1970, 1975, \ldots 2010$

   (c) $\phi_j(x) = \cos(x/j)$ for $j = 1, \ldots, 5$

   (d) $\phi_j(x) = \cos(x/j)$ for $j = 1, \ldots, 25$

   * Note: Please make sure to add a bias term for all your basis functions above in your implementation of the `make_basis` function in `T1_P4.py`.

   Second, for each plot include the residual sum of squares error. Submit the generated plot and residual sum-of-squares error for each basis in your LaTeX write-up.

Problem 4 (cont.)

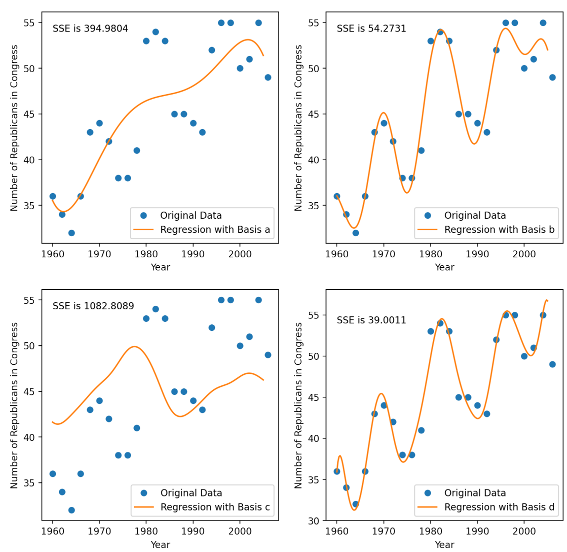
2. Repeat the same exact process as above but for **Number of Sunspots (x-axis)** v. **Number of Republicans in the Senate (y-axis)**. Now, however, only use data from before 1985, and only use basis functions (a), (c), and (d) – ignore basis (b). You will therefore make 3 total plots. For each plot make sure to also include the residual sum of squares error.

   Which of the three bases (a, c, d) provided the "best" fit? **Choose one**, and keep in mind the generalizability of the model.

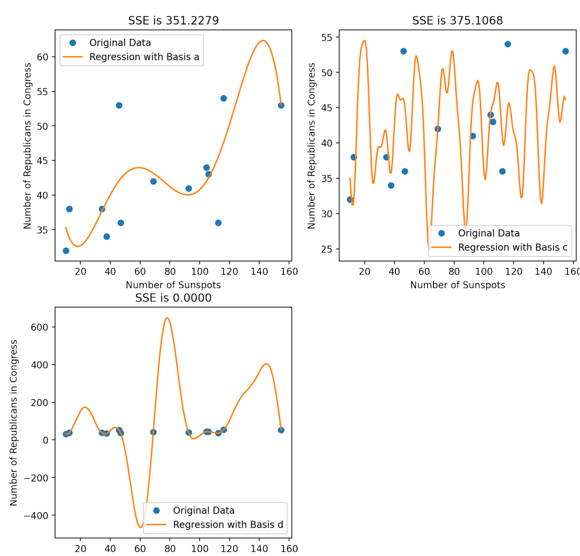
   Given the quality of this fit, do you believe that the number of sunspots controls the number of Republicans in the senate (Yes or No)?

Solution

1. Code is uploaded as "T1_P4.py".



2. It appears that base a provided the best fit. Bases b and d are examples of over-fitting. I do not believe the number of sunspots controls the number of Republicans in the senate because association is not causality. Further causal approaches are needed to determine causality.



Name

Collaborators and Resources

Whom did you work with, and did you use any resources beyond cs181-textbook and your notes?

Zhaoxun Hou, Shuyi Chen, and Zechen Liu.

Calibration

Approximately how long did this homework take you to complete (in hours)?

About 8 hours, including about 2 hours coding this up in LATEX.