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!

Solutions

1. The loss function here will be:

$$\mathcal{L}(\tau) = \sum_{n=1}^{N}(y_n - \sum_{i=1, i\neq n}^{N} \exp\left(\frac{-||x_n - x_i||_2^2}{\tau}\right) y_i)^2$$

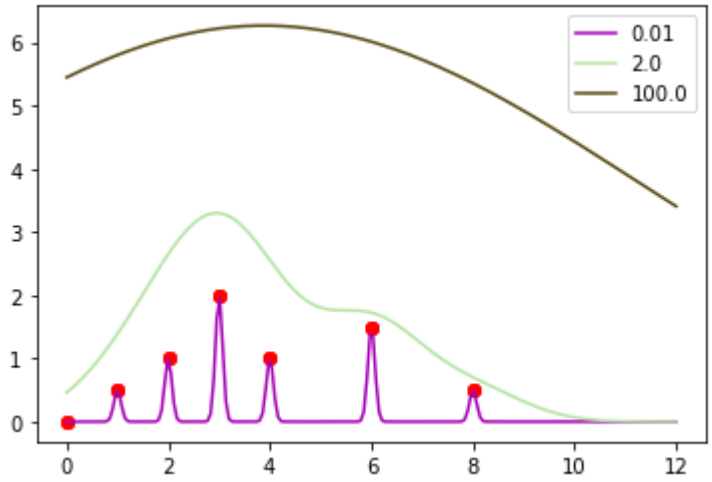
We don't want to include the $i = n$ case in the inner sum, as we want to include our best guess for y, and if y is already in the training set, it defeats the purpose for training on that point.

2. With $\mathcal{L}(\tau)$ as defined above:

$$\frac{\partial \mathcal{L}}{\partial \tau} = -2\sum_{n=1}^{N}((y_n - \sum_{i=1, i\neq n}^{N} \exp\left(\frac{-||x_n - x_i||_2^2}{\tau}\right) y_i)(\sum_{i=1, i\neq n}^{N} \frac{||x_n - x_i||_2^2}{\tau^2} \exp\left(\frac{-||x_n - x_i||_2^2}{\tau}\right) y_i))$$

3. The loss for $\tau = 0.01$ is 8.75, loss for $\tau = 2$: 3.305, and loss for $\tau = 100$: 120.359. It seems here that the value of $\tau = 2$ did the best, as the loss is minimized in this case.

4. The plots are below:



In general, for all of these cases, higher the τ, more we overestimate, because the method we use is not normalized.
We see that for the $\tau = 0.1$ case, there is a lot of "overfitting" happening, in that the model is extremely accurate for values close to the training data, but not otherwise. This is because the kernel decays exponentially in regards to error, so the points that are not close to the training data have extremely low values due to this.
In the $\tau = 2$ case, we see that this is a much better fit to the data, even though it is overestimated, as the τ used here is such that the decay function doesn't decay too fast or too slow as we move away from training data.
In the $\tau = 100.0$ case, there is a lot of smoothing, as the kernel weighs distant points too much. As such, it is not a good fit.

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.)

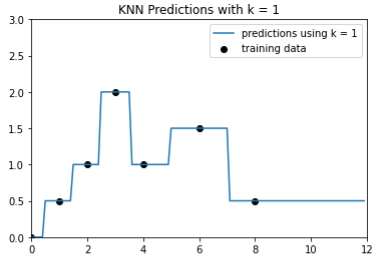
   You may choose to use some starter Python code to create your plots provided in `T1_P2.py`. Please **write your own implementation of kNN** for full credit. Do not use external libraries to find nearest neighbors.

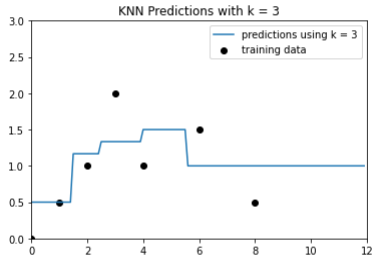
2. Describe what you see: What is the behavior of the functions in these three plots? How does it compare to the behavior of the functions in the three plots from Problem 1? Are there situations in which kNN and kernel-based regression interpolate similarly? Extrapolate similarly? Based on what you see, 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.

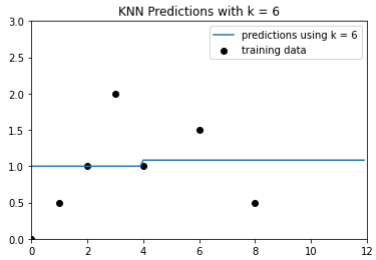
3. Why did we not vary τ for the kNN approach?

Solutions

1. The plots for all are below, with code in the supplemental.







2. For the $k = 1$ predictor, we see that it just fits the nearest point, so it doesn't average at all. For the $k = 3$ predictor, we can see that it averages better than the k=1 predictor, so it "stays between" the points more. Lastly, for the $k = N - 1$ predictor, it is at all times extremely close to the sample mean, as it averages all but one of the points.

   Regarding extrapolation and interpolation, the kNN and Kernel Regressor models don't seem to be similar. For the former, see that in kNN, we predict either the averages of the last k points as x trends towards infinity, or the average of the first k points as x trends towards negative infinity. Unless the average of these points is 0, this prediction will not be zero. In the kernel regressor, as x trends towards infinity and negative infinity, we will trend towards zero, as the kernel function trends towards 0 as the distance from the input $x*$ to the training data x grows to infinity. In regards to interpolation, see that the kNN model averages out the high / low value data points, while the kernel formula "peaks" and favors these high / low value data points (as we see when the predicted input is a training input). I guess in some sense, the $\tau = 2$ and $k = 1$ case do look similar (except one is discrete, another is continuous, and we don't normalize in the kernel regressor), though the similarities are limited.

   I don't think that there exists some values of k, τ, such that the kNN and kernel based regressor produce the exact same classifier. First off, kNN classifiers aren't smooth, and have some

discrete steps in the prediction as we see in our graphs, unlike kernel regressors which are. In addition, as we said before, kNN averages, avoiding predictions that are extremely high or low, while kernel regressor predicts based on these peaks. One case I can think of where there could perhaps exist some value of k or τ such that the predictors are very, very similar, is if the data is basically defined for all $x \in \mathbb{R}$ - in this case, $k = 1$, and a very small τ might converge to the same plot, though I am not sure (this might require some real analysis).

3. The reason we didn't vary τ is because kNN only depends on the k-closest to x^*. If we look at the distance function we used, it is exponential, which is monotonically increasing. In addition, since the negative L2-norm is what we input to the distance function, as long as τ is positive, changing τ does not affect the relative closeness of points to other points, so the k closest neighbors for any input point will be the same regardless of τ.

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 3

1. The optimal $w = \frac{\mathbb{E}_{xy}(xy)}{\mathbb{E}_x(x^2)}$, as shown below:

$$\begin{aligned}\mathcal{L}(w) &= \mathbb{E}_{xy}(y - \hat{y})^2 \\ &= \mathbb{E}_{xy}(y - wx)^2 \\ &= \mathbb{E}_{xy}(y^2 - 2wxy - w^2x^2 \\ &= \mathbb{E}_y(y^2) + 2w\mathbb{E}_{xy}(xy) + w^2\mathbb{E}_x(x^2)\end{aligned}$$

   The last step is by linearity of expectations. Then, we minimize $\mathcal{L}(w)$ by differentiating in respect to w and setting to zero.

$$\begin{aligned}\frac{d\mathcal{L}}{dw} &= 2w\mathbb{E}_x(x^2) - 2\mathbb{E}_{xy}(xy) \\ 0 &= 2w * \mathbb{E}_x(x^2) - 2\mathbb{E}_{xy}(xy) \\ w* &= \frac{\mathbb{E}_{xy(}xy)}{\mathbb{E}_x(x^2)}\end{aligned}$$

   See that this is a minimum by the second derivative test, since $\mathbb{E}_x(x^2) \geq 0$

$$\frac{d^2\mathcal{L}}{dw^2} = 2\mathbb{E}_x(x^2)$$

2. We can estimate these by using the sample mean:

$$\begin{aligned}\mathbb{E}_{xy}(xy) &\approx \frac{1}{N}\sum_n x_n y_n \\ \mathbb{E}_x(x^2) &\approx \frac{1}{N}\sum_n x_n^2\end{aligned}$$

3. Plugging in our estimations above, and having X be the "design matrix" for x, and Y be the one dimensional vector of outputs y (one dimensional), we see that

$$\begin{aligned}w^* &= \frac{\mathbb{E}_{xy(}xy)}{\mathbb{E}_x(x^2)} \\ &\approx \frac{\frac{1}{N}\sum_n x_n y_n}{\frac{1}{N}\sum_n x_n^2} \\ = \frac{\sum_n x_n y_n}{\sum_n x_n^2} &= \frac{XY^T}{XX^T} = (X^TX)^{-1}(X^Ty)\end{aligned}$$

   This is the same as the optimal w^* we see in section 2.6!

4. None of the assumptions made above assume that variables x and y are jointly Gaussian, as we assumed that $p(x, y)$ is the pdf of any distribution.

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.

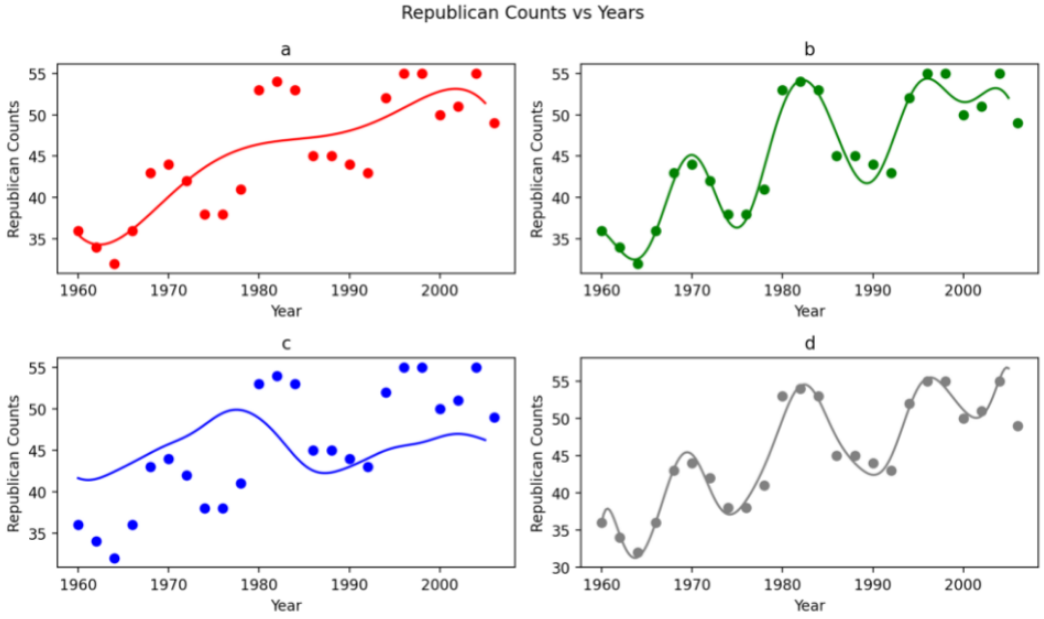
   Which of the three bases (a, c, d) provided the "best" fit? **Choose one**, and keep in mind the generalizability of the model.

   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)?

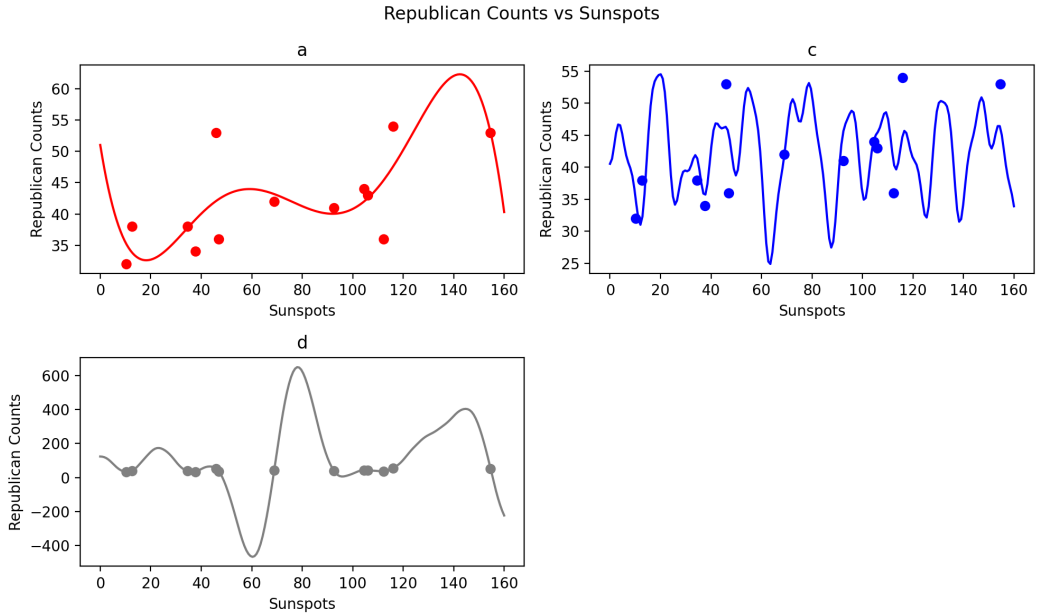
Solution 4

1. Below are both the data and regression lines for each of the sets of basis functions, and the residual sum of squares error.

```
[('a', 394.9803839890881),
 ('b', 54.273096616719556),
 ('c', 1082.8088559867188),
 ('d', 39.001103560787676)]
```



2. Below are the data and regression lines for each of the set of basis functions for the second part, as well as squared error.




```
[('a', 351.227935774181),
 ('c', 375.106757781674),
 ('d', 1.4063485566756565e-21)]
```

The best "fit" (lowest squared error) came from the model fit with basis d. However, it is not generalizable - it is impossible that at around 60 sunspots, we'd predict -400 as the count of Republicans. It is likely that this is the best fit because we overfit with 25 dimensions for w. For basis c, this is clearly not generalizable as well, given the abrupt spikes between data points. As such, we are left with basis a, which doesn't do the best in terms of loss, but at least doesn't jump to extremes, and is relatively smooth in between the domain of sunspots that we've seen.

It is important to note that we are talking about interpolation here, as extrapolation wise, none of these functions seems to do a great job (a would diverge, c and d would be periodic with the same issues that interpolation brought). Given the quality of the fit, I am not convinced that the number of spots control the number of Republicans in the Senate. Intuitively, that doesn't make any sense. Qualitatively, looking at the graph, it seems like the correlation is near zero, and what is there is just noise - the curve we fit seems pretty arbitrary, but "good" enough that it looks like it fits.

Name

John Wang

Collaborators and Resources

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

Calibration

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