CS181-S22

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 1.1:
To find the loss over the training data, we find the squared difference between the y_n's in the training data and the predicted values $\hat{y}_n = f(x_n)$:

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

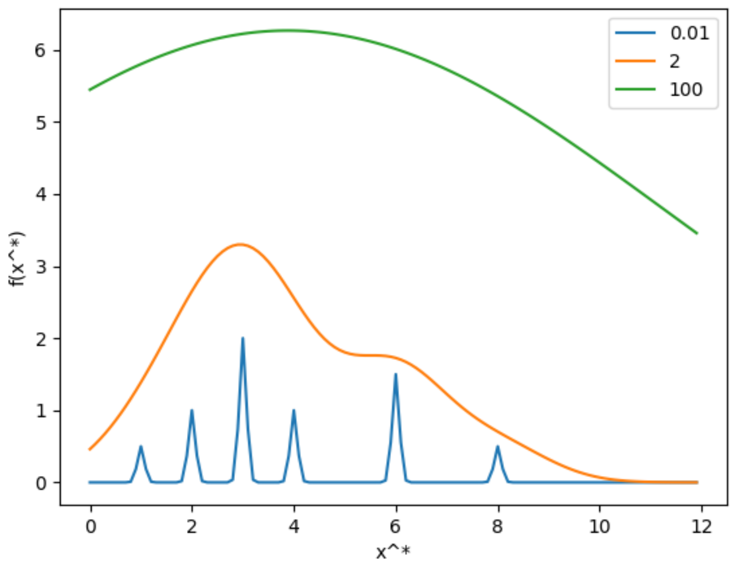
Solution 1.2:
The derivative of the loss function with respect to τ is found by differentiating $\mathcal{L}(W)$ found in 1.1.

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

Solution 1.3:
Computing the loss with respect to each kernel (with different values of τ) yields that $\tau = 2$ produces the lowest residual sum of squares, i.e. the least loss. Therefore, we can say that $\tau = 2$ is the lengthscale that does best with the dataset provided.

τ	$\mathcal{L}(W)$
0.01	8.75
2	3.305016494565789
100	120.35919342230957

Solution 1.4:



When $\tau = 0.01$, the model predicts the values of y for points in training data almost perfectly, but offers practically no insight on what y is at other values of x, defaulting to $y \approx 0$ at all other x values. This occurs because, when τ is small, $K(x, x')$ tends towards 0, unless $||x - x'||_2^2$ is close to 0 (i.e. you have two very nearby points), in which case $K(x, x')$ tends towards 1. Therefore, when predicting a point at an x^* that is equal to an x_i in the training data $K(x_i, x^*)$ tends towards 1 and $K(x \neq x_i, x^*)$ tends towards 0, so $f(x^*)$ tends towards y_i. Thus, when $\tau = 0.01$, the model greatly overfits the training data and is not generalizable, even though it had the second best loss metric.

When $\tau = 2$, the model produces a smooth curve over a set of y values that closely (albeit not perfectly) predict the points occurring in the testing data, giving this model a low loss and better generalizability. It also predicts y values that are too large because we did not normalize for the weights given to the y points used to predict $f(x^*)$ (see further explanation of this below when $\tau = 100$).

When $\tau = 100$, the model produces an even simpler smooth curve over a set of y values that predict the points occurring in the testing data less well than the previous models, i.e. underfitting the training data. It also predicts y values that are too large. This occurs because τ is effectively acting as a regularization parameter, penalizing large weights that are given to y's corresponding to x's that are close to x^*, and giving more weight (in comparison to previous models) to y's corresponding to x's that are further away from this x^*. Furthermore, in this model we did not normalize for the weights given to the y points used to predict $f(x^*)$, so when the weight given to x's that are further away from x^* do not decay to 0, the model ends up overestimating $f(x^*)$. Given the curve produced by this model is less expressive, it also makes it worse at interpolating and extrapolating from the training data, so ends up having the highest loss.

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 $\tau = 1$.

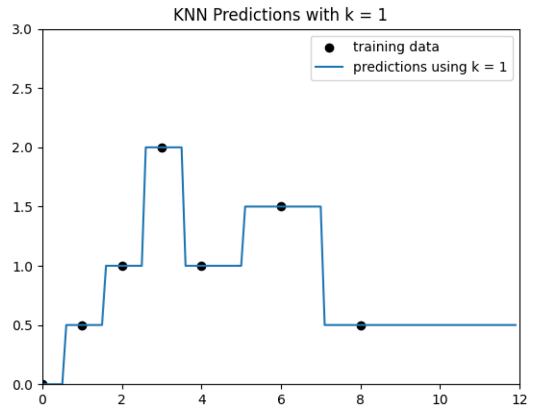
   As before, you will plot x^* on the x-axis and the prediction $f(x^*)$ on the y-axis. For the test inputs x^*, you should use an even grid of spacing of 0.1 between $x^* = 0$ and $x^* = 12$. (Like in Problem 1, if a test point lies on top of a training input, use the formula without excluding that training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

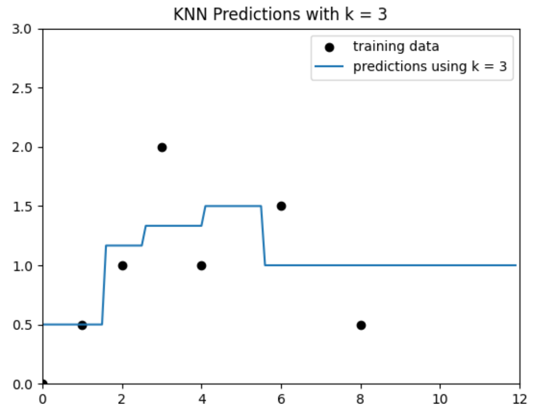
   You may choose to use some starter Python code to create your plots provided in `T1_P2.py`. Please **write your own implementation of kNN** for full credit. Do not use external libraries to find nearest neighbors.

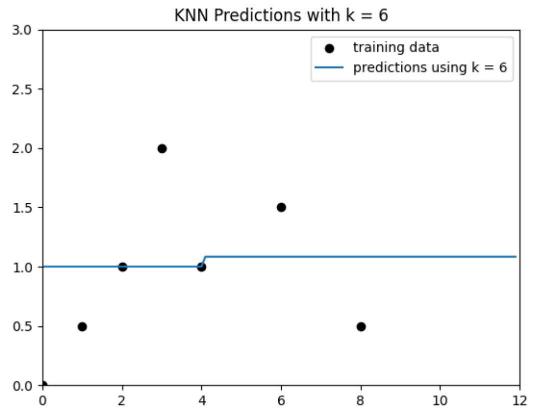
2. Describe what you see: What is the behavior of the functions in these three plots? How does it compare to the behavior of the functions in the three plots from Problem 1? Are there situations in which kNN and kernel-based regression interpolate similarly? Extrapolate similarly? Based on what you see, 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 2.1:


KNN Predictions with k = 1
training data
predictions using k = 1
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
2
4
6
8
10
12


KNN Predictions with k = 3
training data
predictions using k = 3
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
2
4
6
8
10
12


KNN Predictions with k = 6
training data
predictions using k = 6
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0
2
4
6
8
10
12

Solution 2.2:
When $k = 1$, the model predicts the values of y for points in training data perfectly, but clearly is overfitting to these data. This makes sense conceptually, as this model is finding the nearest (measured by the kernel) x_i in the training data to x^* and taking the corresponding y_i to be $f(x^*)$. For all x_i where $i \neq 0, N$, this y_i is given to be the predicted value for all x inputs on the range $[\frac{x_n+x_{n-1}}{2}, \frac{x_n+x_{n+1}}{2})$. Hence, we get what looks what a piecewise function that perfectly fits the training data. This is certainly similar to the plot with $\tau = 0.01$ from Problem 1, in terms of overfitting the training data.

When $k = 3$, the model predicts the values of y for points in training data slightly less well, and is not overfitting to these data as much. This makes sense conceptually, as this model is finding the nearest (measured by the kernel) 3 x_i's in the training data to x^* and taking the average of the corresponding 3 y_i's to be $f(x^*)$. This has a similar behaviour to the plot with $\tau = 2$ from Problem 1, in terms of producing a middle-ground between $k = 1$ and $k = N-1$ - neither overtly overfitting nor underfitting to the training data.

When $k = N - 1$, the model produces what looks like a piecewise function with two pieces: for $x^* < 4$, $f(x^*)$ is the average of the y_i's corresponding to $x_0, ..., x_{N-1}$; for $x^* > 4$, $f(x^*)$ is the average of the y_i's corresponding to $x_1, ..., x_N$. This has a similar behaviour to the plot with $\tau = 100$ from Problem 1, in terms of producing a line that is less expressive than the other models', and so underfits the data.

Given the behaviour of these plots is similar to the respective plots from Problem 1, it can be said that they interpolate similarly. However, they extrapolate quite differently. Since, kNN regressors produce what look like piecewise functions, for $x^* > x_N$, $f(x^*) = f(x_N)$, and the kNN regressors give no additional information when extrapolating outside of the domain of the data. By comparison, kernel-based regressors have some additional predictive power when extrapolating; since they generate smooth curves (when τ is sufficiently greater than 0.01), for $x^* > x_N$, $f(x^*) \neq f(x_N)$. Given such differences between the geometry of the plots, kNN and kernel-based regressors will not produce the exact same classifier (i.e. $f(x^*)$) for a given x^*.

Solution 2.3:
Given τ is a positive scalar, it simply re-scales the distances between x^* and its nearest neighbors but has no effect on the ranking of x^*'s nearest neighbors. This means that if x_1 is nearer to x^* than x_2 with τ equal to some c (where $c \in \mathbb{R}^+$), x_1 is nearer to x^* than x_2 for all possible values of c. Thus, varying τ would have no effect on the y_n's which we average to get $f(x^*)$, so we do not bother varying τ for the kNN approach.

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:
In this problem, we are given that the linear model has the form $\hat{y} = wx$. Substituting this into the expected squared loss gives:

$$E_{x,y}[(y-\hat{y})^2] = E_{x,y}[(y-wx)^2]$$

Expanding the brackets and applying linearity of expectation gives:

$$\begin{aligned} E_{x,y}[(y-\hat{y})^2] &= E_{x,y}[y^2 - 2wxy + w^2x^2] \\ &= E_{x,y}[y^2] - 2wE_{x,y}[xy] + w^2E_{x,y}[x^2] \end{aligned}$$

We can drop the y subscript for expectations involving only x and the x subscript for expectations involving only y:

$$E_{x,y}[(y-\hat{y})^2] = E_y[y^2] - 2wE_{x,y}[xy] + w^2E_x[x^2]$$

To find the optimal w that minimizes the expected square loss above, we take the derivative of this expression with respect to w, set equal to 0, and solve for w:

$$\begin{aligned} \frac{d}{dw}[E_y[y^2] - 2wE_{x,y}[xy] + w^2E_x[x^2]] &= 0 \\ \implies -2E_{x,y}[xy] + 2wE_x[x^2] &= 0 \\ \implies w &= \frac{E_{x,y}[xy]}{E_x[x^2]} \end{aligned}$$

Solution 3.2:
Unbiased and consistent formulas that estimate these expectations can be found using the Law of Large Numbers which states that for large data sets a sample mean $\bar{X}_n$ converges to the true mean pointwise, with probability 1, i.e. $P(\bar{X}_n \to \mu) = 1$. Thus, given observed data $\{(x_n, y_n)\}_{n=1}^N$, assuming N is sufficiently large:

$$\begin{aligned} E_{x,y}[xy] &= \frac{1}{N}\sum_{n=1}^{N} x_n y_n \text{ with probability 1} \\ E_x[x^2] &= \frac{1}{N}\sum_{n=1}^{N} x_n^2 \text{ with probability 1} \end{aligned}$$

Solution 3.3:
Substituting in these moment terms from 3.2 into the expression for the optimal w from 3.1 gives:

$$\begin{aligned} w^* &= \frac{\frac{1}{N}\sum_{n=1}^N x_n y_n}{\frac{1}{N}\sum_{n=1}^N x_n^2} \\ &= \frac{\sum_{n=1}^N x_n y_n}{\sum_{n=1}^N x_n^2} \end{aligned}$$

This expression is the 1-dimensional version of the Moore-Penrose pseudo inverse seen in section 2.6 of the textbook, $(\mathbf{X}^\top\mathbf{X})^{-1}\mathbf{X}^\top\mathbf{y}$. $\mathbf{X}$ is a Nx1 vector containing all the x_i's, so $(\mathbf{X}^\top\mathbf{X})^{-1}$ is the inverse/reciprocal of a scalar equal to $\frac{1}{\sum_{n=1}^N x_n^2}$. Likewise $\mathbf{y}$ is a Nx1 vector containing all the y_i's, so $\mathbf{X}^\top\mathbf{y}$ is equal to the scalar $\sum_{n=1}^N x_n y_n$. Thus, in the 1-dimensional case, where $\mathbf{X}$ and $\mathbf{y}$ are Nx1 vectors, w^* is given by the expression above.

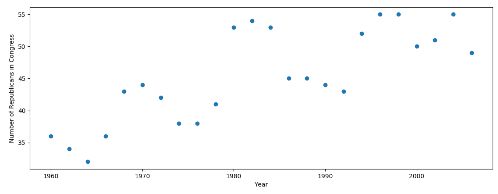
Solution 3.4:
No. None of the steps in the derivations above (linearity of expectation, differentiation, Law of Large Numbers) relied on an assumption that variables x and y are jointly Gaussian. Furthermore, the cs181-textbook shows that solving for w^* by minimizing the loss function, as we did in this problem, is equivalent to maximizing the probability under the assumption of a linear model with Gaussian noise. Thus, we can clearly use this process for models in which x and y are not jointly Gaussian.

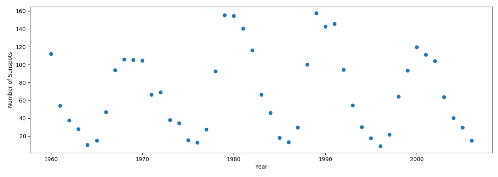
Problem 4 (Modeling Changes in Republicans and Sunspots, 15pts)

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.

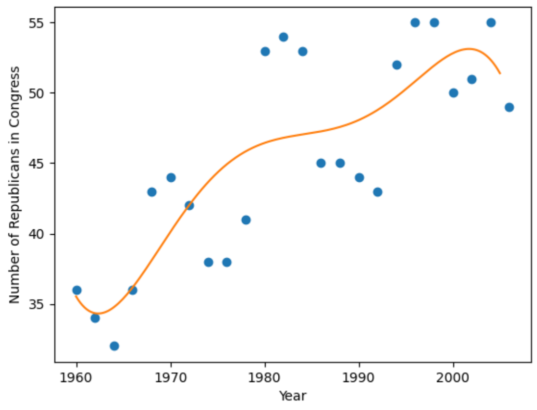
Problem 4 (cont.)

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.

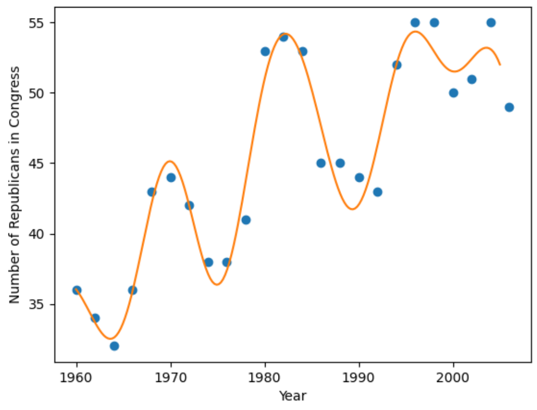
   Which of the three bases (a, c, d) provided the "best" fit? **Choose one**, and keep in mind the generalizability of the model.

   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)?

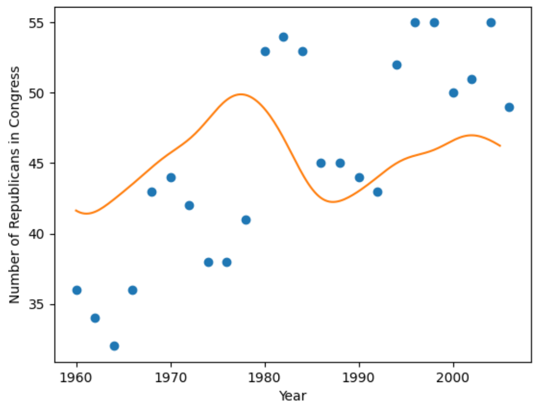
Solution 4.1:
(a)



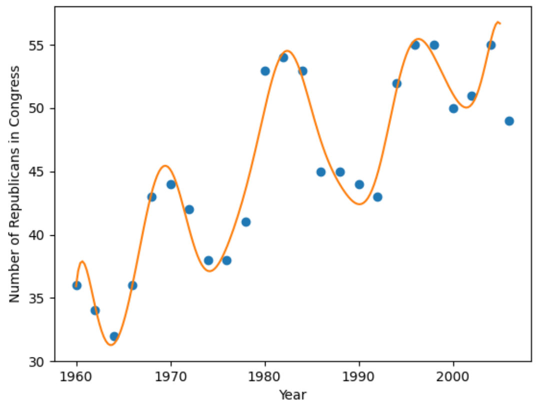
(b)



(c)

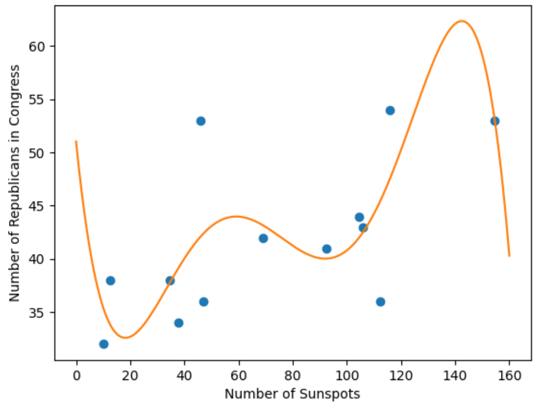


(d)

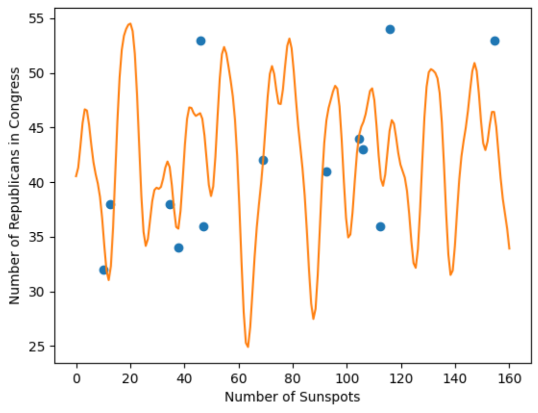


Basis	Residual sum of squares error
(a)	394.9803839890881
(b)	54.27309661671943
(c)	1082.8088559867188
(d)	39.00107715663492

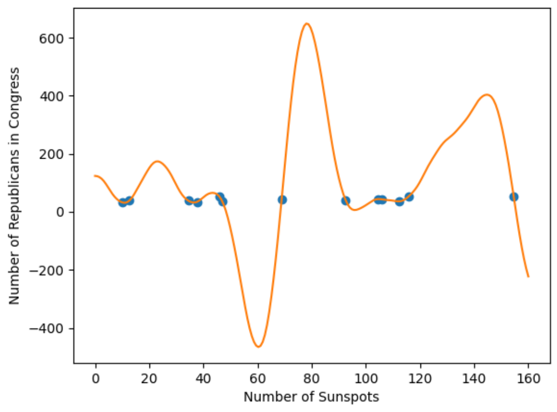
Solution 4.2:
(a)



(c)



(d)



Basis	Residual sum of squares error
(a)	351.2279357741859
(c)	375.10675778167405
(d)	$2.186325095811903 \times 10^{-22}$

Basis (d) provided the "best" fit for the training data in terms of least squares loss, but this is not necessarily easily generalizable given how it seems to overfit these data. Therefore, (a) seems more generalizable and can be argued to provide the "best" fit.

Even though basis (d) fit the data very well, simple logic tells is that the number of sunspots does not control the number of Republicans in the senate. This is an example of the old adage that "correlation does not equal causation"; in other words, regression can tell us that there is an association between two variables, but we cannot necessarily infer causality. Thus, even though we can make a model that fits the training data, it does not mean that sunspots control number of republicans.

Name

Ben Ray

Collaborators and Resources

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

Skyler's Code Review notes and Stack Overflow for help with Python syntax
CS 181 Office hours

Calibration

Approximately how long did this homework take you to complete (in hours)?
20