CS540 Fall 2024 Homework 5

Linear Regression on Lake Mendota Ice

The Wisconsin State Climatology Office keeps a record on the number of days Lake Mendota was covered by ice at this site. When viewing this page, click the "Winter" column so the year is in **ascending** order.

Question 1: Data Curation

As with any real problems, the data is not as clean or as organized as one would like for machine learning. Curate a clean dataset from the Lake Mendota Ice data starting from year **1855-56** and ending in year **2022-23** (not 2023-24). You may do this manually. We care about the following aspects of the data:

- the starting year, x
  - For 1855-56, $x = 1855$
  - For 2008-09, $x = 2008$
- the number of ice days in that year, y
  - For 1855-56, $y = 118$
  - For 2008-09, $y = 97$

Some years, such as 2001-02, have multiple freeze thaw cycles. In these cases, you'll notice exactly one of the lines has a number for ice days. Use this number, which is the total number of ice days across the multiple freezes, as the y for that year. For example, in year 2001-02 your data will be $x = 2001$, $y = 21$.

Save your dataset as "hw5.csv". **You will include this file in your submission.** We provide you with an example toy.csv with the correct format but fake numbers. Your file should follow the standard format for ".csv" files. For example, the first 4 lines of your "hw5.csv" would be the following:

```
year,days
1855,118
1856,151
1857,121
```

Summary: Create a csv file as described above and name it to be "hw5.csv". You will submit this file!

Input Details for Remaining Questions

For the remaining questions, you need to write a python program hw5.py. Your code should take three arguments (we will explain them below):

1. the name of the csv file to read (i.e. hw5.csv)
2. the learning rate for gradient descent (i.e. 0.01)
3. the number of iterations to run gradient descent for (i.e. 100)

In particular, we will run your code as follows:

```
$ python3 hw5.py filename.csv learning_rate iterations
```

As a reminder, to get the arguments you can use the following code:

```
import sys

filename = sys.argv[1]
learning_rate = float(sys.argv[2])
iterations = int(sys.argv[3])
```

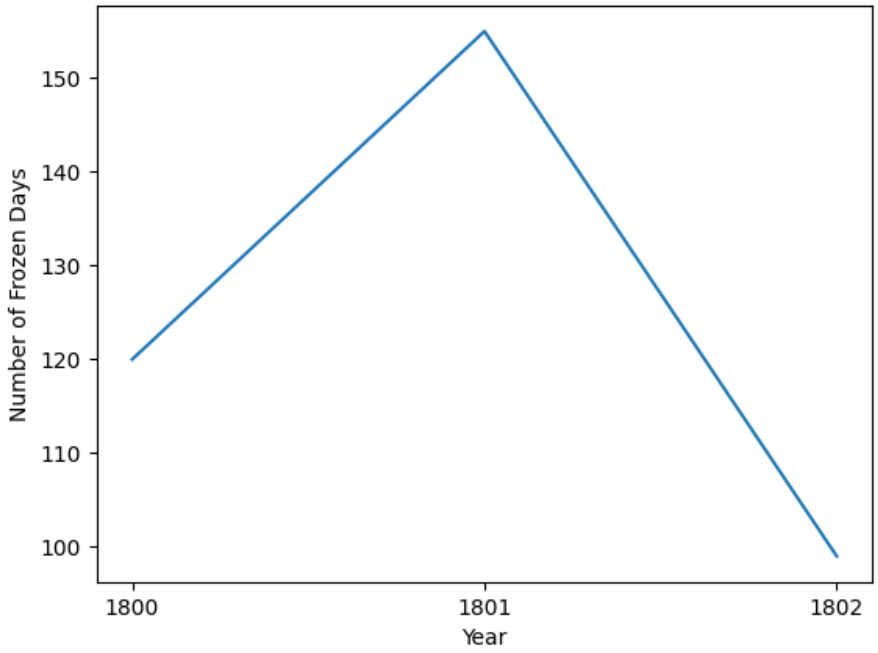
Your hw5.py will need to produce figures for Questions 2 and 5d, and print answers for Questions 3, 4, 5a-c, 6, 7, and 8. An example output for the printed statements is provided at the end of this writeup.

Question 2: Visualize Data

Plot the number of frozen days versus the year from the dataset in the inputted filename.csv. **Save the plot as data_plot.jpg exactly. Do not call plt.show().** You can save the plot by using the following code:

```
plt.savefig("data_plot.jpg")
```

For reference, here is the expected output data_plot.jpg for toy.csv:



Note: You do not need to exactly match the plot style, but you should have an x-axis label and y-axis label.

Summary: Plot year vs. number of frozen days and save it to "data_plot.jpg" using plt.savefig.

Question 3: Data Normalization

Going forward, n will denote the number of data points. For each $i \in \{1, \ldots, n\}$, x_i and y_i will denote the feature and label, respectively, of the ith data point. For example, for the toy dataset we have $n = 3$, $x_1 = 1855$, and $y_1 = 118$.

Your task will be to fit the whole dataset using a linear regression model. Before doing so, you should perform min-max normalization on our input feature (that is, on the x's). For each i, denote the normalized version of x_i by

$$\widetilde{x}_i = \frac{x_i - m}{M - m},$$

where, $m = \min\{x_1, \ldots, x_n\}$ and $M = \max\{x_1, \ldots, x_n\}$.

Our goal is to find a weight w and bias b to minimize the Mean Squared Error loss (MSE) over the n data samples by solving

$$\underset{\begin{pmatrix} w \\ b \end{pmatrix} \in \mathbb{R}^2}{\arg\min} \frac{1}{2n} \sum_{i=1}^{n} \left(\hat{y}_i - y_i\right)^2,$$

where $\hat{y}_i = w\widetilde{x}_i + b$ is our model's output on the (normalized) feature $\widetilde{x}_i$. For ease of notation, we augment the feature by defining

$$\widetilde{\mathbf{x}}_i = \begin{pmatrix} \widetilde{x}_i \\ 1 \end{pmatrix}.$$

Now we can write $\hat{y}_i = \begin{pmatrix} w \\ b \end{pmatrix}^\top \widetilde{\mathbf{x}}_i$. Note the distinction between boldface $\widetilde{\mathbf{x}}_i$, which is a vector, and $\widetilde{x}_i$, which is a scalar. We organize the features in a $n \times 2$ array as follows:

$$\widetilde{X} = \begin{pmatrix} \widetilde{\mathbf{x}}_1^\top \\ \vdots \\ \widetilde{\mathbf{x}}_n^\top \end{pmatrix}.$$

Compute and store $\widetilde{X}$ as a numpy array, then print it out.

Summary: Print the normalized and augmented data matrix $\widetilde{X}$. Use the following format for your print statements:

```
print("Q3:")
print(X_normalized)
```

Remark: We perform normalization because on this problem it drastically improves the effectiveness and efficiency of the gradient descent in Question 5. Without normalization, we would need much more precise tuning for gradient descent to converge to the optimum, and it would converge much slower.

Question 4: Closed-Form Solution to Linear Regression

Let Y be the vector containing all y values:

$$Y = \begin{pmatrix} y_1 \\ \vdots \\ y_n \end{pmatrix}.$$

The minimizing weight and bias for MSE can equivalently be written as

$$\underset{\begin{pmatrix} w \\ b \end{pmatrix} \in \mathbb{R}^2}{\arg\min} \left\| \widetilde{X} \begin{pmatrix} w \\ b \end{pmatrix} - Y \right\|_2^2. \qquad \text{(Verify this yourself!)}$$

By setting the gradient to zero, we can explicitly solve for the optimal $\begin{pmatrix} w \\ b \end{pmatrix}$ to get

$$\begin{pmatrix} w \\ b \end{pmatrix} = \left(\widetilde{X}^\top \widetilde{X}\right)^{-1} \widetilde{X}^\top Y. \qquad \text{(Verify this yourself!)}$$

Your code should compute $\begin{pmatrix} w \\ b \end{pmatrix}$ as a numpy array, then print it out. Note that this involves taking the inverse of $\widetilde{X}^\top \widetilde{X}$, but for our datasets this will always be an invertible matrix.

Summary: Print the optimal weight and bias as a numpy array. Use the following format:

```
print("Q4:")
print(weights)
```

Question 5: Linear Regression with Gradient Descent

As we saw in Question 4, this particular optimization problem has an explicit closed-form solution. This will not always be the case, however, so we will also solve this problem using gradient descent. While on this problem gradient descent is certainly slower than computing the closed form solution, we can (and will, in future homeworks) use gradient descent and its variants on a much wider variety of optimization problems.

Recall from Lecture 8 that the gradient of the MSE loss C with respect to the weight w and bias w is

$$\nabla_{(w,b)} C = \begin{pmatrix} \frac{\partial C}{\partial w} \\ \frac{\partial C}{\partial b} \end{pmatrix} = \frac{1}{n} \left(\sum_{i=1}^{n} \hat{y}_i - y_i \right) \widetilde{\mathbf{x}}_i,$$

where $\hat{y}_i = \begin{pmatrix} w \\ b \end{pmatrix}^\top \widetilde{\mathbf{x}}_i$ (note that $\hat{y}_i$ here is instead denoted a_i in the lecture notes). Starting with the initialization

$$\begin{pmatrix} w_0 \\ b_0 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix},$$

run gradient descent with the learning rate and number of iterations provided as input to your program.

Algorithm 1 Gradient Descent

Require: Learning rate α, number of iterations T, initialization (w_0, b_0).
  for $t = 0, \ldots, T-1$ **do**
    $\begin{pmatrix} w_{t+1} \\ b_{t+1} \end{pmatrix} \leftarrow \begin{pmatrix} w_t \\ b_t \end{pmatrix} - \alpha \nabla_{(w,b)} C$
  end for

(a) During gradient descent, print out the current weight w_t and bias b_t (as a numpy array) once every 10 iterations (that is, when $t \mod 10 = 0$). In particular, the first line to be printed should be [0 0], and when the learning rate and number of iterations are properly tuned, the sequence should converge close to the solution from Question 4.

(b + c) Find and print out a learning rate and number of iterations such that gradient descent with initialization [0 0] on the Mendota ice dataset converges close to the solution from Question 4. Specifically, when you run

```
$ python3 hw5.py hw5.csv your_learning_rate your_iterations
```

the last weight and bias printed out during part (a) should each be within 0.01 of the weight and bias obtained from the closed-form solution in Question 4. You should find a learning rate large enough that number of iterations needed is no more than 500. You need to exercise caution, however, because **if the learning rate is too large then the weight and bias will diverge**. As such, you should start with a small learning rate (like 0.01) and experiment with increasingly larger values to speed up convergence. You can also implement part (d) first, and use the plot to help guide your search. If you are still having trouble getting your weight and bias to converge to the correct values, double check that you are correctly computing the gradient and the update step.

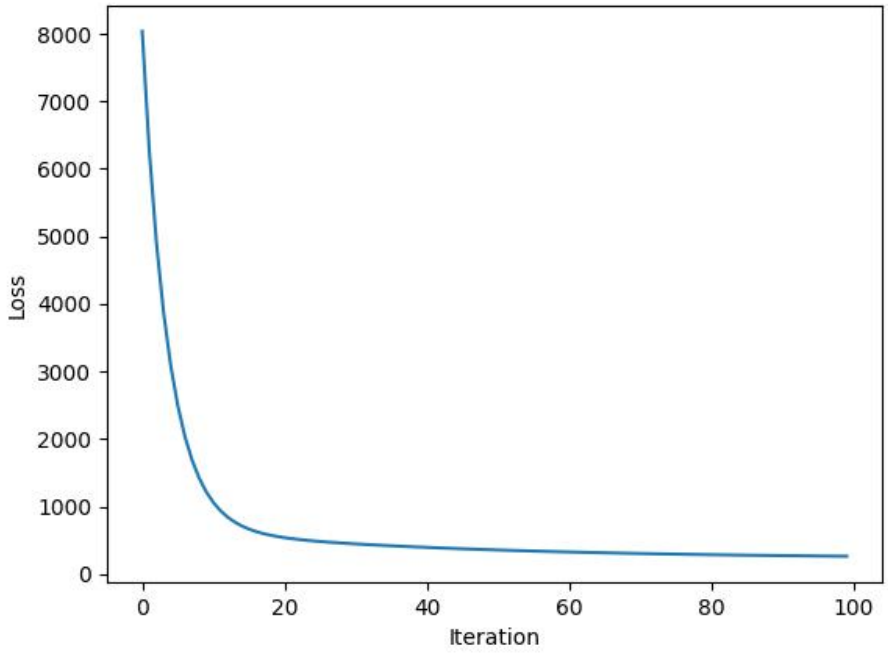
(d) Plot the MSE loss

$$C(w_t, b_t) = \frac{1}{2n} \sum_{i=1}^{n} \left((w_t \widetilde{x}_i + b_t) - y_i \right)^2 = \frac{1}{2n} \left\| \widetilde{X} \begin{pmatrix} w_t \\ b_t \end{pmatrix} - Y \right\|_2^2$$

produced by the weight w_t and bias b_t versus the iteration t. To do so, you should compute and store the loss during each iteration of gradient descent. Like before, **save the plot as loss_plot.jpg exactly, and do not call plt.show()**. You can save the plot by running the following code:

```
plt.savefig("loss_plot.jpg")
```

For reference, here is the expected output loss_plot.jpg on toy.csv with learning rate 0.1 and 100 iterations:



Note: You do not need to exactly match the plot style, but you should have an x-axis label and a y-axis label.

Summary: Print the weight and bias every 10 iterations of gradient descent, plot loss over time, and print out your own tuned learning rate and number of iterations. Use the following format:

```
print("Q5a:")
# run gradient descent, every 10 iterations do print(weights)
print("Q5b: *your learning rate*")
print("Q5c: *your number of iterations*")
```

Question 6: Prediction

Going forward, the weight w and bias b will refer to the ones obtained by the closed-form solution in Question 4. For this question, print out the model's prediction for the number of ice days for winter 2023-24 (last winter). That is, for $x = 2023$, compute

$$\hat{y} = w\widetilde{x} + b = w\frac{x - m}{M - m} + b,$$

where m and M are the same as we defined in Question 3 above. You do **not** need to recompute m and M, as are **not** adding the year 2023-24 to the dataset, but rather testing how well our model works on unseen data.

The Wisconsin State Climatology Office has released the official 2023-24 numbers, so you can check to see how close your prediction was. Don't be alarmed if your prediction is somewhat higher than the real value, as last winter was quite warm!

Summary: Print the model's prediction for the number of ice days for 2023-24 using the following format (include a space after ":"):

```
print("Q6: " + str(y_hat))
```

Question 7: Model Interpretation

(a) What is the sign of the weight w? If the sign is positive, print ">". If negative, print "<". If zero, print "=". Do not hardcode the symbol – we will test your code on different datasets that could result in different answers.

(b) Briefly explain the meaning of each of the three possible signs of w as it relates to Mendota ice days. Specifically, what would $w > 0$ indicate, what would $w < 0$ indicate, and what would $w = 0$ indicate? Separate your sentences by **punctuation and spaces only**. Do not use new lines within your answer.

Summary: Print the sign of w and interpret all three possible signs. Use the following format (include a space after ":"):

```
print("Q7a: " + symbol)
print("Q7b: ...")
```

Question 8: Model Limitations

(a) With the weight w and bias b, predict the year x^* by which Lake Mendota will no longer freeze. That is, find and print $x^* \in \mathbb{R}$ satisfying

$$w\widetilde{x}^* + b = 0 \iff w\frac{x^* - m}{M - m} + b = 0,$$

(b) Briefly explain whether x^* is a compelling prediction, and discuss some potential limitations that may cause this prediction to be inaccurate. Your answer should be specifically based on hw5.csv (rather than toy.csv or some other dataset). Separate your sentences by **punctuation and spaces only**. Do not use new lines within your answer.

Summary: Print the model's prediction for the year Lake Mendota will no longer freeze, and write a few sentences analyzing the prediction. Use the following format (include a space after ":"):

```
print("Q8a: " + str(x_star))
print("Q8b: ...")
```

Submission Details

- There is no starter code for this assignment.
- The files you need to upload are **hw5.py** and **hw5.csv**.
- All code, except importing modules, should be contained in functions or under if __name__=="__main__":

- Only python built-in library, numpy, matplotlib and pandas are allowed. Do not import other modules.
- Remember to remove all debugging output before submission.
- Do not include plt.show() in your submission.
- Please submit your hw5.csv and hw5.py to Gradescope. No late submissions will be accepted.

Example

toy.csv:

```
year,days
1800,120
1801,155
1802,99
```

Run parameters:

```
$ python3 hw5.py toy.csv 0.1 100
```

Output:
(Note that data_plot.jpg and loss_plot.jpg should also be produced but are not shown here, as the example plots are already shown above.)

```
Q3:
[[0.  1. ]
 [0.5 1. ]
 [1.  1. ]]
Q4:
[-21.         135.16666667]
Q5a:
[0. 0.]
[30.70107492 74.43550295]
[32.77049906 96.45819629]
[ 28.31457658 104.86927867]
[ 22.82179505 109.499432  ]
[ 17.6015202  112.86972697]
[ 12.91861334 115.65321437]
[  8.7825806  118.05246758]
[  5.14554887 120.14743019]
[  1.95132189 121.98360183]
Q5b: your answer
Q5c: your answer
Q6: -2206.3333333333335
Q7a: <
Q7b: your answer
Q8a: 1812.873015873016
Q8b: your answer
```

We give you toy_output.txt containing the above output. You can use vimdiff on CSL Machines to compare your result with this example.

```
$ python3 hw5.py toy.csv 0.1 100 | tee your_output
$ vimdiff your_output toy_output.txt
```

Note that your result might be different from the example output because the code was running on different environments. We will grade your work on Gradescope which will run submissions on the same environment. In addition, a tolerance has been built into the grader for comparing numerical values.