Problem Set 8

Topic: Generalized Linear Models (GLMs)
Due date: **Tuesday March 25th**
Instructor: Chris Rodgers, christopher.rodgers@emory.edu

You may complete these problems in Python, Matlab, R, or whatever language you prefer. **You must include sufficient comments in your code so that I can understand your code**. Remember from the syllabus: "*Collaboration on assignments is acceptable (and, in fact, is encouraged!), but please list the names of everyone with whom you worked, and each student must turn in their own, separately generated, solutions. For coding exercises, please turn in all code.*"

This problem set looks long, but it is because I have provided lots of hints. I am available by email to answer questions. However, I cannot answer questions that are sent after 5pm on Monday March 24th, so please start working early.

For all of these problems, we will use the nomenclature $\mathbf{Y \approx f(X\theta)}$, where $\mathbf{X}$ is the input data, $\mathbf{Y}$ the output data, $\mathbf{f}$ the non-linearity, and $\boldsymbol{\theta}$ the parameters.

Problem 1: Use regression to fit data to a polynomial

In this problem, you will use the regression approach that we discussed in class to fit a polynomial to simulated data. This problem is an example of how you can make GLMs more versatile by including specific non-linear features in the input matrix $\mathbf{X}$, which is called "feature engineering".

In this problem, $\mathbf{f}$ is the identity function, meaning there is no non-linearity in the GLM.

Problem 1.1: Write code to generate data from a polynomial with added noise.

- Generate a one-dimensional array $\mathbf{w}$ (length $\mathbf{N}$=51) that spans the range -1 to 1.
- Generate $\mathbf{Y\_noiseless}$ (length 51) such that $\mathbf{Y\_noiseless = 2w^3 + w^2 - w}$. Note: these coefficients `[2, 1, -1]` are $\boldsymbol{\theta}$.
- Generate a one-dimensional array of $\mathbf{noise}$ (length 51) where each entry is drawn from a normal distribution with mean 0 and standard deviation 0.1.
- Generate the variable $\mathbf{Y}$ which is equal to $\mathbf{Y\_noiseless + noise}$.
- Make a plot of $\mathbf{Y\_noiseless}$ vs $\mathbf{w}$, and also $\mathbf{Y}$ vs $\mathbf{w}$.

For the rest of Problem 1, you will use only the variables $\mathbf{w}$ and $\mathbf{Y}$, and you will attempt to recover the coefficients $\boldsymbol{\theta}$ that you used to generate the data (as if $\boldsymbol{\theta}$ were unknown).

Problem 1.2: Generate $\mathbf{X}$, the "design matrix" that you would use to estimate $\boldsymbol{\theta}$ with a GLM. We did a similar problem in class when we set up a design matrix that included a certain amount of stimulus history. Here, you will not be including stimulus history. Instead, you want to set up the variable $\mathbf{X}$ to include information about polynomial powers of $\mathbf{w}$. $\mathbf{X}$ should be a matrix of numbers that you compute with code.

Hint: The shape of $\mathbf{X}$ will be $\mathbf{N}$ rows and 3 columns. Each column corresponds to a feature of $\mathbf{w}$. Each feature is a different part of the polynomial that you used to generate $\mathbf{Y}$.

Problem 1.3: Use a GLM fitting function to calculate θ using only Y and X. In Python, you can do this with:

```
import sklearn.linear_model
reg = sklearn.linear_model.LinearRegression()
reg.fit(X, y)
print(reg.coef_)
```

In Matlab you can use `glmfit`. There are many tutorials you can find about this. You do not need an "intercept term" for this, because you did not add a y-intercept when you generated Y_noiseless.

Show your code and the estimate you got for θ. Does the estimate get better if you decrease the amount of noise you added?

Problem 1.4: Calculate θ using the following equation:

$$\theta = (X^T X)^{-1} X^T Y$$

This is the closed-form solution for the GLM, which is only valid if f is the identity function.

Here is pseudo-code to do this:

```
invert_matrix(X.transpose * X) * X.transpose * Y
```

How does this compare to the result you got in problem 1.3?

Hint: to multiply matrices, use the * (asterisk) in Matlab. In Python, use the @ (at sign).

If interested, you can review the derivation of this equation on slide 41 of Dr Anqi Wu's slides. Intuitively, the mathematical derivation for solving all linear regression problems relies on calculating the error as a function of theta, and then calculating the global minimum of this error by finding where its derivative is zero.

Problem 2: Implement regularization and cross-validation

In this problem, you will implement regularization to control over-fitting in a GLM, and you will implement cross-validation to assess the effect of over-fitting.

Begin by downloading the datafile ("RGCdata.mat") at this link: https://osf.io/mzujs/download
To load the data in Matlab, use `load`.
In Python, use the following code snippet:

```
loaded_data = scipy.io.loadmat('RGCdata.mat')
Stim = loaded_data['Stim'].flatten()
SpCounts = loaded_data['SpCounts']
```

`Stim` is a 1d-array of the stimulus intensity, which can take only one of two values. `SpCounts` is a 2d-array of the spike counts from 4 neurons. Use only the spike counts from the first neuron and discard the other three neurons.

You can see lots of code and explanation for this dataset on Neuromatch Academy.

Problem 2.1: Plot the first 100 values of `Stim` and the first 100 values of `SpCounts`. Remember, you should only be using the first neuron, not all four. Also remember that you should be providing code for all problems to show your work, not just a plot.

Problem 2.2: Set up the design matrix $\mathbf{X}$, using the same approach that we used in class. Include 25 samples of history, so the number of columns in $\mathbf{X}$ should be 25.

There are lots of hints about how to do this on NMA. There are several slightly different ways to compute X and Y, depending on how you handle the data at the beginning of the recording.

You will need X and Y for Problem 2.3. Therefore, if you are unable to complete Problem 2.2, use the values of X and Y that I have provided here.
Please note that if you are using Python, you may want to "flatten" the 1-dimensional Y array stored in this file.
https://numpy.org/doc/stable/reference/generated/numpy.ndarray.flatten.html

Problem 2.3: Calculate θ using the following equation:

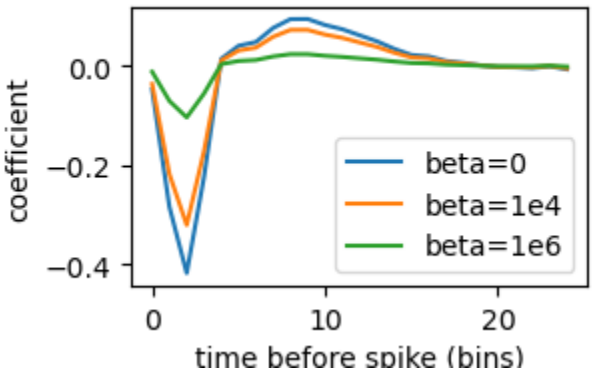
$$\theta = (X^T X + \beta I)^{-1} X^T Y$$

This equation differs from the one in Problem 1.4 because it includes regularization controlled by the regularization parameter β, a scalar.

Note: I is the identity matrix with shape (25, 25) and with ones along the diagonal and zeros everywhere else. You can make this matrix with `np.eye` in Python or `eye` in Matlab. This regularized approach is known as "ridge regression" because the identity matrix looks like a ridge.

Alternatively: If you prefer, instead of implementing the equation above, you can calculate θ using the function `ridge` in Matlab or the object `sklearn.linear_model.Ridge` in Python. Note that β is called α in Python and k in Matlab.

Whichever method you choose, compute θ for $\beta = 0$, $\beta = 10000$, and $\beta = 1000000$. Plot theta for each value of β on the same plot. Your plot should look something like this. (Note: On 2024-03-07, this figure was updated.)



Problem 2.4: Following up on problem 2.3, describe in words how changing β affects θ. Describe in words the neurobiological explanation for why θ approaches zero for long time lags. (Hint: think about what would happen if you considered an extremely long history instead of just 25 samples.)

Problem 2.5: Explain in words (or write pseudo-code) to implement cross-validation for the calculation you did in Problem 2.3. By "pseudo-code", I mean something that looks like code but does not actually have to run or be syntactically correct. Do not use any pre-designed cross-validation functions in your answer, such as these. It is important that you understand the logic behind how cross-validation works.

Hint: start by splitting the data into two parts, a training set and a test set. Then calculate performance on the training set and the test set. Be specific in your answer. For instance, how exactly will you split the data into two parts? There are multiple correct answers...