Overview

Submit your writeup including any code and plots as a PDF via Gradescope.[1] We recommend reading through the entire homework beforehand and carefully using functions for testing procedures, plotting, and running experiments. Taking the time to reuse code will help in the long run!

Data science is a collaborative activity. While you may talk with others about the homework, please write up your solutions individually. If you discuss the homework with your peers, please include their names on your submission. Please make sure any handwritten answers are legible, as we may deduct points otherwise.

1. GLM for Dilution Assay

Being able to reformulate problems as generalized linear models (GLMs) enables you solve a wide variety of problems with existing packages. We recommend reviewing the examples of GLMs from Lectures 10 and 11. In particular, make sure you understand that formulating a GLM involves choosing an 1) output distribution and 2) link function that are appropriate for the application at hand.

In this problem, you'll retrace the footsteps of the statistician R. A. Fisher and develop one of the very first applications of GLMs. In a 1922 paper, Fisher formulated a GLM he used to estimate the unknown concentration ρ_0 of an infectious microbe in a solution. Without specialized technology to directly measure ρ_0 from the solution, Fisher devised the following procedure: we will progressively dilute the original solution, and after each dilution, we'll pour out some small volume v onto a sterile plate. If zero microbes land on the plate, it will remain sterile, but if any microbes land on a plate, they will grow visibly on it (we call this an "infected plate"). By observing whether or not the plate is infected at each dilution, and by formulating the relationship between this data and ρ_0 as a GLM, we can estimate ρ_0 from this data.

Specifically, let ρ_t denote the concentration at dilution t. Each time, we dilute the solution to be half its concentration, such that

$$\rho_t = \frac{\rho_0}{2^t} \tag{1}$$

for $t = 0, 1, \ldots$. When we pour out volume v of the solution onto the plate, and wait awhile to allow for microbe growth, we can observe whether a plate was infected (*i.e.*, has a non-zero number of microbes) or is sterile (*i.e.*, has zero microbes). Therefore, our data $Y_t \in \{0, 1\}$ is whether or not the plate is infected at each dilution.

We'll formulate a GLM that relates ρ_0 and t to the data Y_t. Estimating the parameters of this GLM will then allow us to estimate ρ_0, as will become clear in the last part.

[1] In Jupyter, you can download as PDF or print to save as PDF

(a) (2 points) At dilution t, the data $Y_t \in \{0, 1\}$ indicates whether or not the plate is infected. The chance that a plate gets infected is denoted by $\mu(t) := \mathbb{E}[Y_t]$. Write down an output distribution for Y_t that is appropriate for the values it takes on, using $\mu(t)$ as a parameter.

(b) (5 points) At dilution t, we pour out volume v onto a plate, so the expected number of microbes on the plate is $\rho_t v$. The actual number of microbes is distributed as a Poisson random variable with this mean $\rho_t v$:

$$\#\text{ microbes on plate at dilution } t \sim \text{Poisson}(\rho_t v). \tag{3}$$

Using this fact, write out an expression for $\mu(t) := \mathbb{E}[Y_t]$. Start with

$$\mu(t) = \mathbb{P}(\text{plate is infected at dilution } t) \tag{4}$$
$$= 1 - \mathbb{P}(\text{there are 0 microbes on plate at dilution } t). \tag{5}$$

(c) (5 points) Find a link function g such that

$$g(\mu(t)) = \beta_0 + \beta_1 t \tag{8}$$

for some constants β_0 and β_1.

(d) (3 points) Choosing an appropriate output distribution and link function as we've done in Parts (a) and (c) completes the GLM specification. Now, suppose you've estimated β_0 and β_1 (*e.g.*, using maximum-likelihood estimation). Write down an estimate of ρ_0.

2. Using Bootstrap to Evaluate Drug Bioequivalence

When drug companies introduce new drugs, the FDA requires them to show that the new drug is *bioequivalent* to the current drug used to treat the same condition. Bioequivalence means that the effect of the new drug is not substantially different from the effect of the current drug. The way the effect is measured is application-dependent—here, we'll look at drugs that infuse a certain hormone into the blood. A drug's effect is therefore the amount of hormone in the blood after administering the drug.

To formally define bioequivalence, let the random variables O, N, P denote the effect of the old drug, the effect of the new drug, and the effect of a placebo, respectively. The FDA requirement for bioequivalence is that

$$|\theta| \leq 0.2 \tag{11}$$

where

$$\theta = \frac{\mathbb{E}[N - O]}{\mathbb{E}[O - P]}. \tag{12}$$

In this problem, you'll estimate θ from a dataset and use the bootstrap to determine, with a certain confidence, whether or not we have bioequivalence.

(a) (2 points) The CSV file `bioequivalence.csv` provided for this homework contains the following data on the level of a hormone in 8 subjects' blood, after medications were administered.

subject	placebo	old	new
1	9243	17649	16449
2	9671	12013	14614
3	11792	19979	17274
4	13357	21816	23798
5	9055	13850	12560
6	6290	9806	10157
7	12412	17208	16570
8	18806	29044	26325

Download the data, and use it to compute the plug-in estimate $\hat{\theta}$ of θ.

(b) (10 points) Part (a) gave an estimate of θ, but by itself it doesn't capture the certainty we have in the estimate, so we can't use it to conclude that we have bioequivalence with a given confidence level. Instead, we'll compute a bootstrap confidence interval to do this.

(i) Implement a function `bootstrap_bioequivalence(N, O, P, B)` which takes in the following inputs:

- $N = (N_1, \ldots, N_n)$, an array of the effects of the new drug on n subjects
- $O = (O_1, \ldots, O_n)$, an array of the effects of the old drug on n subjects
- $P = (P_1, \ldots, P_n)$, an array of the effects of the placebo on n subjects
- B, an integer which is the number of bootstrap replicates

and outputs a length-B array of bootstrap replicates of $\hat{\theta}$.

(ii) Using `bootstrap_bioequivalence(N, O, P, B)`, compute $B = 10000$ bootstrap replicates of $\hat{\theta}$. Plot a histogram of these replicates, and label the x- and y- axes.

(iii) Using the replicates from (ii), compute a 95-percentile confidence interval for θ (make sure to include the code you use to compute this). Hint: Use the function `np.percentile`.

(c) (3 points) Based on Part (b), can we conclude the new drug and old drug are bioequivalent, at the 95% confidence level? That is, does the 95% confidence interval fall within the FDA requirement for bioequivalence?

3. Image Denoising with Gibbs Sampling

In this problem, we derive a Gibbs sampling algorithm to restore a corrupted image [1]. A grayscale image can be represented by a 2-dimensional array X of shape $n \times m$, where the intensity of the (i, j)-th pixel is X_{ij}. In this problem, we are given an image X whose pixels have been corrupted by noise, and the goal is to recover the original image Z.

(a) (2 points) Load the grayscale image `X.pkl` as a numpy array X. Visualize the image.

From plotting the image X, it is clear that it has been corrupted with noise. Let Z denote the original image, which we also represent as an $n \times m$ array. Let $\mathcal{I} = \{(i, j) : 1 \leq i \leq n \text{ and } 1 \leq j \leq m\}$ denote the collection of all pixels in the image, represented by the corresponding index of the array. Given a pixel (i, j), define the set of *neighboring pixels* to be

$$N_{(i,j)} = \{(i', j') \in \mathcal{I} : (i = i' \text{ and } |j - j'| = 1) \textbf{ or } (|i - i'| = 1 \text{ and } j = j')\}.$$

To capture the fact that, in natural images, neighboring pixels are likely be similar, we consider the following prior over the original image:

$$p(Z) \propto \exp\left(-\frac{1}{2}\sum_{(i,j)\in\mathcal{I}}\left[aZ_{ij}^2 - b\sum_{(i',j')\in N_{(i,j)}} Z_{ij}Z_{i'j'}\right]\right).$$

Assuming the image has been corrupted with Gaussian noise $X_{(i,j)} \mid Z_{(i,j)} \sim \mathcal{N}(Z_{(i,j)}, \tau^{-1})$ (independently across pixels $(i, j) \in \mathcal{I}$), the complete posterior can be written as

$$p(X \mid Z) \propto \exp\left(-\frac{1}{2}\sum_{(i,j)}\left[(a+\tau)Z_{ij}^2 - 2\tau Z_{ij}X_{ij} - b\sum_{(i',j')\in N_{(i,j)}} Z_{ij}Z_{i'j'}\right]\right) \tag{15}$$

Let $S_{ij} = \sum_{(i',j')\in N_{(i,j)}} Z_{i'j'}$. By completing the square in the posterior (15), we have

$$Z_{ij} \mid (Z_{i'j'})_{(i',j')\neq(i,j)}, X \sim \mathcal{N}\left(\frac{\tau X_{ij} + bS_{ij}}{a+\tau}, \frac{1}{a+\tau}\right) \tag{16}$$

(b) (2 points) Fill in the missing line of pseudocode for a Gibbs sampler of the posterior, $p(Z|X)$. **Be specific with each conditioned variable and sub/superscript!**

- Initialize $Z^{(0)} = X$.
- For $t = 1, \ldots, T$:
  - Sample $Z_{1,1}^{(t)} \sim p(Z_{1,1} \mid Z_{1,2} = Z_{1,2}^{(t-1)}, Z_{1,3} = Z_{1,3}^{(t-1)}, \ldots, Z_{n,m} = Z_{n,m}^{(t-1)}, X)$.
  - Sample $Z_{1,2}^{(t)} \sim p(Z_{1,2} \mid Z_{1,1} = Z_{1,1}^{(t)}, Z_{1,3} = Z_{1,3}^{(t-1)}, \ldots, Z_{n,m} = Z_{n,m}^{(t-1)}, X)$.
  - Sample $Z_{1,3}^{(t)} \sim$ *# TODO: fill this in.*
  - $\ldots$
  - Sample $Z_{n,m}^{(t)} \sim p(Z_{n,m} \mid Z_{1,1} = Z_{1,1}^{(t)}, Z_{1,2} = Z_{1,2}^{(t)}, \ldots, Z_{n,m-1} = Z_{n,m-1}^{(t)}, X)$

(c) (3 points) Write the pseudo-code from Part (b) more explicitly both by using a double for-loop over $(i, j) \in \mathcal{I}$ and by being explicit about the conditional distributions of the form $p(Z_{1,1} \mid Z_{1,2} = Z_{1,2}^{(t-1)}, Z_{1,3} = Z_{1,3}^{(t-1)}, \ldots, Z_{n,m} = Z_{n,m}^{(t-1)}, X)$. In your pseudo-code, use `np.random.randn()` to generate a $\mathcal{N}(0, 1)$ random variable at each step.

(d) (5 points) Implement the Gibbs sampler from Part (c) with $a = 250, b = 62.5$, and $\tau = 0.01$. Run your code for $T = 1$ iteration, i.e. update each coordinate exactly once. Visualize the resulting image $Z^{(1)}$. Time your code and estimate how long it would take to compute $Z^{(100)}$.

(e) (2 points) The bottleneck in running the Gibbs sampler from Part (d) is sampling a single pixel Z_{ij} with the values of all others held fixed. Fortunately, it is possible to speed up the sampling process with an improvement known as *blocked Gibbs sampling*. Specifically, define two subsets of the pixels $\mathcal{I}_{\text{even}} = \{(i, j) : i + j \text{ is even}\}$ and $\mathcal{I}_{\text{odd}} = \{(i, j) : i + j \text{ is odd}\}$. The blocked Gibbs sampler proceeds as follows:

- Initialize $Z^{(0)} = X$.
- For $t = 1, \ldots, T$:
  - Let $Z = Z^{(t-1)}$.
  - Let Δ be an $n \times m$ matrix with $\mathcal{N}(0, \frac{1}{a+\tau})$ entries.
  - For $(i, j) \in \mathcal{I}_{\text{even}}$:
    - Let $S_{ij} = \sum_{(i',j') \in N_{(i,j)}} Z_{i'j'}$
  - Update $Z_{\mathcal{I}_{\text{even}}} = \frac{\tau}{a+\tau} X_{\mathcal{I}_{\text{even}}} + \frac{b}{a+\tau} S_{\mathcal{I}_{\text{even}}} + \Delta_{\mathcal{I}_{\text{even}}}$.
  - For $(i, j) \in \mathcal{I}_{\text{odd}}$:
    - Let $S_{ij} = \sum_{(i',j') \in N_{(i,j)}} Z_{i'j'}$
  - Update $Z_{\mathcal{I}_{\text{odd}}} = \frac{\tau}{a+\tau} X_{\mathcal{I}_{\text{odd}}} + \frac{b}{a+\tau} S_{\mathcal{I}_{\text{odd}}} + \Delta_{\mathcal{I}_{\text{odd}}}$.
  - Let $Z^{(t)} = Z$.

The advantage of this approach is that the inner for-loops can be *vectorized*. Explain why updating half the variables $Z_{\mathcal{I}_{\text{even}}}$ (and then $Z_{\mathcal{I}_{\text{odd}}}$) at once is justified.

(f) (1 point) Implement the Gibbs sampler from Part (e) using $a = 250, b = 62.5$ and $\tau = 0.01$. Run your code for $T = 100$ iterations, and visualize the resulting image $Z^{(100)}$. Time your code and report how long it took. *Hint:* Compute the entire $n \times m$ matrix S at once using matrix operations on Z. You may find it helpful to pad the matrix Z with a border of zeros using `Z_bar = np.pad(Z, 1)`. Then use slicing on the $(n + 2) \times (m + 2)$ matrix `Z_bar` to compute S.

References

[1] Stuart Geman and Donald Geman (1984). *Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images*. IEEE Transactions on Pattern Analysis and Machine Intelligence, (6), 721-741.