1. Math Stats

(a) Suppose we have n independent null p-values $P_1, \ldots, P_n$ that are uniformly distributed on $[0, 1]$. Recall that the level-α Bonferroni procedure rejects all P_i such that $P_i \leq \alpha/n$. What is the probability that the level-α Bonferroni procedure rejects any of the P_i? Is your answer smaller or larger than α?

solution: $\mathbb{P}(p_i \leq \alpha/n \text{ for some } i) = 1 - \mathbb{P}(p_i \geq \alpha/n \text{ for all } i) = 1 - (1 - \alpha/n)^n$, which we know is smaller than α by the union bound. Alternatively, we can show $1 - (1 - \alpha/n)^n \leq \alpha$ directly. Note this is equivalent to $1 - \alpha \leq (1 - \alpha/n)^n$, so it suffices to show the function $f(n) = (1-\alpha/n)^n$ is monotonically increasing. Since $\log f(n) = n \log(1-\alpha/n)$ has derivative $\frac{\partial}{\partial n}(\log f(n)) = \frac{\alpha}{n-\alpha} - \log\left(1 + \frac{\alpha}{n-\alpha}\right) \geq 0$, we conclude $f(n) \geq f(1)$. Note as $n \to \infty$ the probability converges to $1 - e^{-\alpha}$, which is close to α when α is small.

(b) Suppose we have a waiting time $T \sim \text{Exponential}(\lambda)$ and wish to test $H_0 : \lambda = 1$ versus $H_1 : \lambda = \bar{\lambda}$ for some $\bar{\lambda} > 1$. What test $\delta^*(T)$ maximizes $\text{TPR}(\delta)$ subject to $\text{FPR}(\delta) \leq \alpha$? *Hint.* Recall from section that the Neyman-Pearson lemma states $\delta^*(t) = 1\left\{\frac{f_1(t)}{f_0(t)} > \eta\right\}$, where f_i is the likelihood of T under H_i and $\eta \geq 0$ is some cutoff. Set the $\text{FPR}(\delta^*) = \alpha$ and write $\delta^*(T)$ explicitly in terms of T, α and $\bar{\lambda}$.

solution: By the Neyman-Pearson lemma, $\delta(T) = 1$ if $\frac{\bar{\lambda}e^{-\bar{\lambda}T}}{e^{-T}} > \eta$. Rearranging, $T < \frac{\log(\bar{\lambda}/\eta)}{\bar{\lambda}-1} =: \eta'$. The FPR $\alpha = \mathbb{P}(T < \eta') = \int_0^{\eta'} e^{-t}dt = 1 - e^{-\eta'}$. Rearranging, $\eta' = -\log(1-\alpha)$. Note this test does not depend on $\bar{\lambda}$.

(c) Suppose a particular drug test is 99% sensitive and 98% specific. The null hypothesis H_0 is that the subject is not using the drug. Assume a prevalence of $\pi_0 = 99.5\%$, i.e. only 0.5% of people use the drug. Consider a randomly selected individual undergoing testing. Rounding to the nearest three significant figures, find

(i) the probability of testing positive given H_0. **solution:** 0.02, or one-minus specificity.

(ii) the probability that they are not using the drug given they test positive. **solution:** Applying Bayes rule,

$$\mathbb{P}(H_0 \mid +) = \frac{\mathbb{P}(+ \mid H_0)\pi_0}{\mathbb{P}(+ \mid H_0)\pi_0 + \mathbb{P}(+ \mid H_1)(1 - \pi_0)}.$$

From part (a), $\mathbb{P}(+ \mid H_0) = .02$, and similarly $\mathbb{P}(+ \mid H_1) = 0.99$ is the given sensitivity, so

$$\mathbb{P}(H_0 \mid +) = \frac{0.02 \cdot 0.995}{0.02 \cdot 0.995 + 0.99 \cdot 0.005} = 0.801.$$

(iii) the probability of testing positive a second time given they test positive once. You may assume the two tests are statistically independent given drug user status. **solution:** Let $+_1$ denote a positive test result on the first test, and let $+_2$ denote a positive test result on the second test. By the law of total probability,

$$\mathbb{P}(+_2 \mid +_1) = \mathbb{P}(+_2 \mid +_1, H_0)\mathbb{P}(H_0 \mid +_1) + \mathbb{P}(+_2 \mid +_1, H_1)\mathbb{P}(H_1 \mid +_1).$$

Since the tests are independent given drug user status, $\mathbb{P}(+_2 \mid +_1, H_i) = \mathbb{P}(+_2 \mid H_i)$ for $i = 0, 1$. Hence,

$$\begin{aligned}\mathbb{P}(+_2 \mid +_1) &= \mathbb{P}(+_2 \mid H_0)\mathbb{P}(H_0 \mid +_1) + \mathbb{P}(+_2 \mid H_1)\mathbb{P}(H_1 \mid +_1) \\ &= .02 \cdot 0.801 + 0.99 \cdot (1 - .801) = 0.213.\end{aligned}$$

The positive test result will only be reproducible about 21.3% of the time.

This example comes from an excellent blog post [3], which explores the differences between Bayesian and frequentist approaches to testing in this setting.

(d) Consider the null hypothesis H_0 that random variable X has tail cdf $\bar{F}_0$, and the alternative hypothesis H_1 that X has tail cdf $\bar{F}_1$. Assume that $\bar{F}_1(x) \geq \bar{F}_0(x)$ for all x and that $\bar{F}_0$ is invertible. Show that, under the *alternative* H_1, the p-value $P = \bar{F}_0(X)$ is *sub-uniform*, i.e. $\mathbb{P}(P \leq p) \geq p$ for all $p \in [0, 1]$.

solution: We know that a p-value $P = \bar{F}_0(X)$ is uniformly distributed under its null H_0. By the monotonicity, $\mathbb{P}(P \leq p \mid H_1) \geq \mathbb{P}(\bar{F}_1(X) \leq p \mid H_1)$. The random variable $\bar{F}_1(X)$ is a p-value with respect to the 'new null' H_1, so $\mathbb{P}(\bar{F}_1(X) \leq p \mid H_1) = p$.

Alternatively, we can apply the same argument that the P is uniform under H_0,

$$\mathbb{P}(P \leq p \mid H_1) = \mathbb{P}(\bar{F}_0(X) \leq p \mid H_1) = \mathbb{P}\left(X \leq \bar{F}_0^{-1}(p) \mid H_1\right) = \bar{F}_1\left(\bar{F}_0^{-1}(p)\right),$$

where in the last step we used the assumption H_1 that X has tail cdf $\bar{F}_1$. Finally, by $\bar{F}_1(x) \geq \bar{F}_0(x)$ for all x, we have $\bar{F}_1\left(\bar{F}_0^{-1}(p)\right) \geq \bar{F}_0\left(\bar{F}_0^{-1}(p)\right) = p$. In this context, a p-value is more likely to be small under the alternative distribution than under the null distribution.

2. Online Experiments

In some applications of multiple testing, it is not possible to collect all p-values before making decisions about which hypotheses should be proclaimed discoveries. For example, in A/B testing in tech, p-values arrive in a continual stream, so decisions have to be made in an online fashion, without knowing the p-values of future hypotheses. In this question, we compare an online algorithm for FDR control called LORD with the classical Benjamini-Hochberg (BH) procedure. We will provide an implementation of the LORD algorithm, however, for completeness, we also state the steps of the LORD algorithm below. Don't worry if you don't have intuition for the α_t update; the important thing is that such an update ensures that FDR is controlled at any given time t.

Algorithm 1 The LORD Procedure

input: FDR level α, non-increasing sequence $\{\gamma_t\}_{t=1}^{\infty}$ such that $\sum_{t=1}^{\infty} \gamma_t = 1$, initial wealth $W_0 \leq \alpha$

Set $\alpha_1 = \gamma_1 W_0$

for $t = 1, 2, \ldots$ **do**

- p-value P_t arrives
- if $P_t \leq \alpha_t$, reject P_t
- $\alpha_{t+1} = \gamma_{t+1} W_0 + \gamma_{t+1-\tau_1}(\alpha - W_0)\mathbf{1}\{\tau_1 < t\} + \alpha \sum_{j=2}^{\infty} \gamma_{t+1-\tau_j}\mathbf{1}\{\tau_j < t\}$,
- where τ_j is time of j-th rejection $\tau_j = \min\{k : \sum_{l=1}^{k} \mathbf{1}\{P_l \leq \alpha_l\} = j\}$

end

While offline algorithms like Benjamini-Hochberg take as input a *set* of p-values, online algorithms take in an *ordered sequence* of p-values. This makes their performance sensitive to p-value ordering. In this exercise we analyze this phenomenon. Generate $N = 1000$ p-values in three different ways:

(i) For every $i \in \{1, \dots, N\}$, generate $\theta_i \sim \text{Bern}(1 - \pi_0)$. If $\theta_i = 0$, the p-value P_i is null, and should be generated from Unif$[0, 1]$. If $\theta_i = 1$, the p-value P_i is an alternative. Then, generate $Z_i \sim \mathcal{N}(3, 1)$, and let $P_i = \Phi(-Z_i)$, where Φ is the standard Gaussian $\mathcal{N}(0, 1)$ CDF.

(ii) For $i = 1, \dots, \pi_0 N$, set $\theta_i = 0$, meaning the hypothesis is truly null, and let $P_i \sim$ Unif$[0, 1]$. For $i = \pi_0 N + 1, \dots, N$, $\theta_i = 1$, and the hypothesis is truly alternative. Then, generate $Z_i \sim \mathcal{N}(3, 1)$, and let $P_i = \Phi(-Z_i)$, where Φ is the standard Gaussian $\mathcal{N}(0, 1)$ CDF.

(iii) For $i = 1, \dots, N - \pi_0 N$, set $\theta_i = 1$, meaning the hypothesis is alternative, generate $Z_i \sim \mathcal{N}(3, 1)$, and let $P_i = \Phi(-Z_i)$, where Φ is the standard Gaussian $\mathcal{N}(0, 1)$ CDF. For $i = N - \pi_0 N + 1, \dots, N$, $\theta_i = 0$, and the hypothesis is truly null; let $P_i \sim$ Unif$[0, 1]$.

(a) Run the LORD algorithm with $\alpha = 0.05$ on three p-value sequences, given as in (i), (ii) and (iii), respectively. Compute the false discovery proportion (FDP) and sensitivity. Repeat this experiment 100 times to estimate FDR as the average FDP over 100 trials, as well as the average sensitivity. Do this for all $\pi_0 \in \Pi_0 := \{0.1, 0.3, 0.5, 0.7, 0.9\}$. Make the following plots:

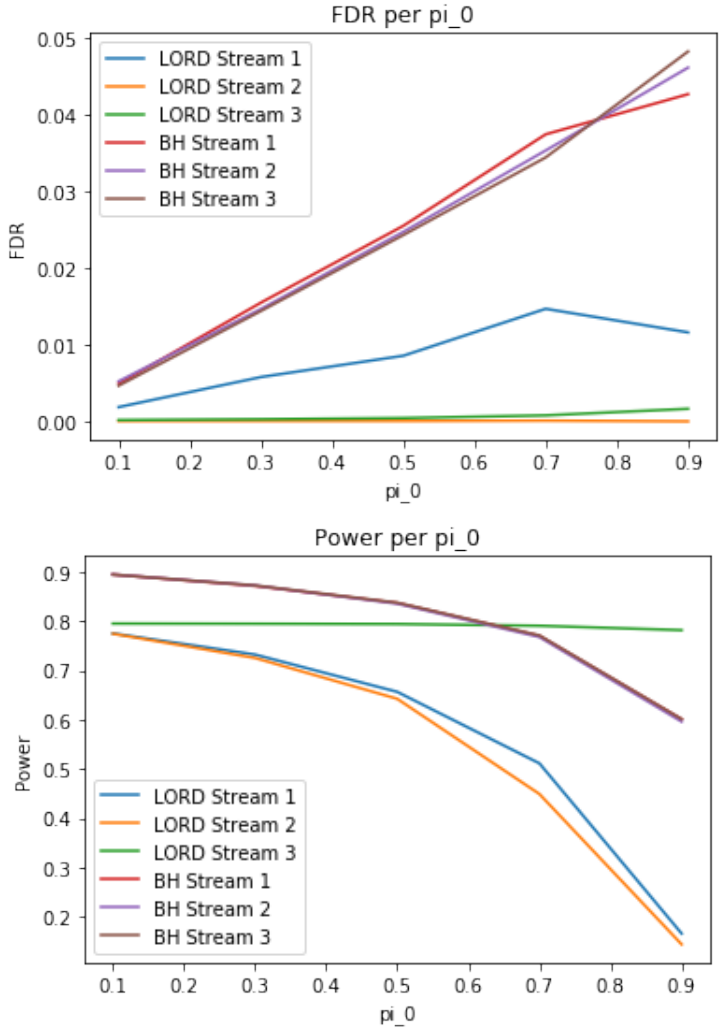
- FDR estimated over 100 trials on the y-axis against $\pi_0 \in \Pi_0$ on the x-axis, for the three different scenarios (i), (ii) and (iii).

- Expected sensitivity estimated over 100 trials on the y-axis against $\pi_0 \in \Pi_0$ on the x-axis, for the three different scenarios (i), (ii) and (iii).

For which of the three scenarios (i), (ii), (iii) does LORD achieve highest average sensitivity? Can you give an intuitive explanation for this?

(b) Now also run the Benjamini-Hochberg procedure with $\alpha = 0.05$ for settings (i), (ii), (iii) on the whole batch; generate all of N p-values, and then apply BH. Make the same plots as in part (a). How does the sensitivity of BH compare to the sensitivity of LORD? How does the sensitivity of BH compare in settings (ii) and (iii)?

solutions to (a) and (b): LORD achieves highest average sensitivity when the alternatives come first. This follows from the "wealth" interpretation of online FDR algorithms. BH shows almost no difference among the three scenarios because the number of nulls and their values are roughly the same in all three streams, and their ordering does not matter. In this setting the FDR of BH is close to the known upper bound $\pi_0 \alpha$. LORD appears to be more conservative with the FDR staying substantially below $\alpha = 0.05$.



3. Bias in Police Stops

The following example is taken from [1, Ch. 6]:

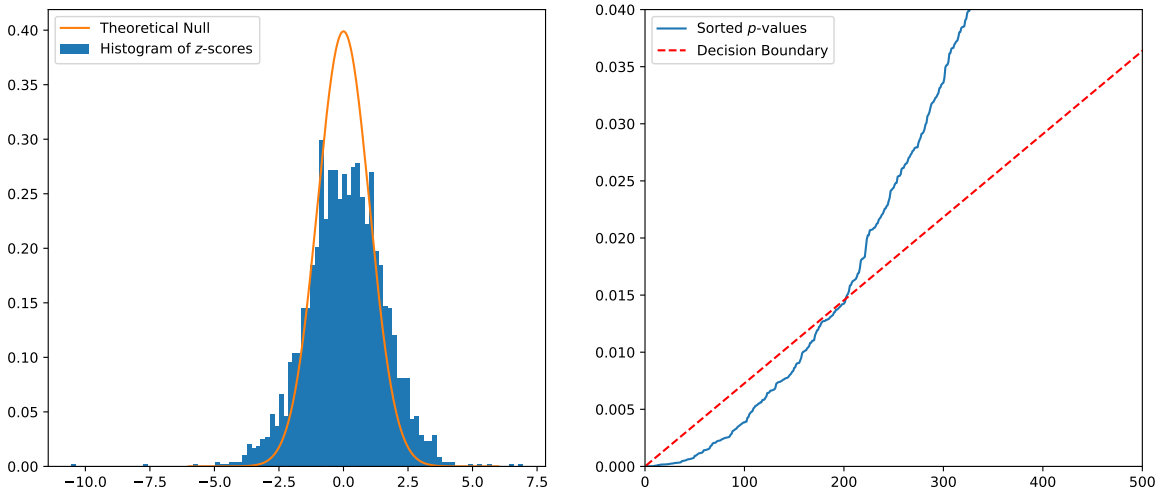
> A study of possible racial bias in police pedestrian stops was conducted in New York City in 2006. Each of $N = 2749$ officers was assigned a score z_i on the basis of their stop data, with large positive values of z_i being possible evidence of bias. In computing z_i, an ingenious two-stage logistic regression analysis was used to compensate for differences in the time, place, and context of the individual stops.

We provide the data in a file `policez.csv`.

(a) In one plot, make a normalized histogram of the z-scores and a line plot of the pdf of the theoretical null $\mathcal{N}(0, 1)$. Describe how the fit looks.

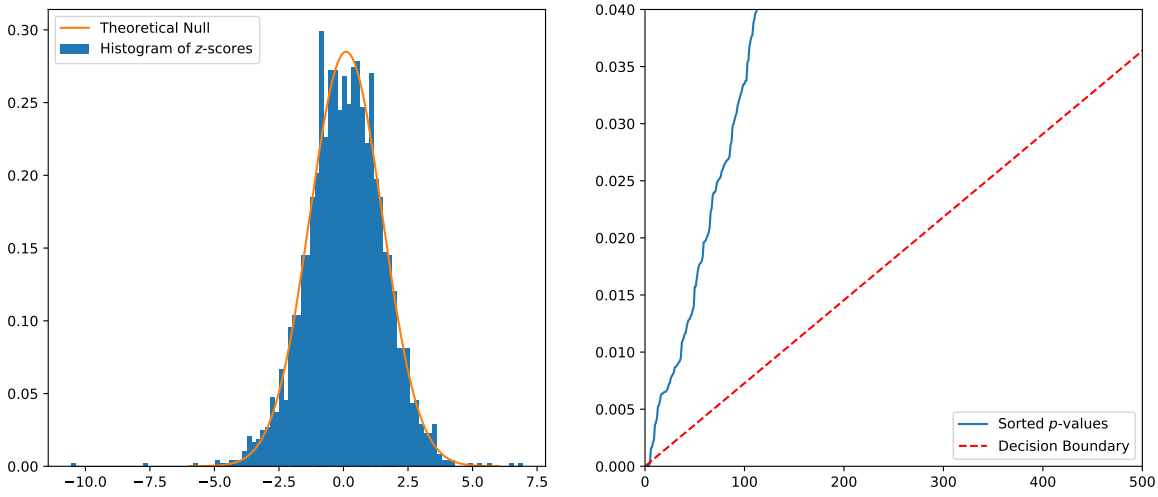
(b) Compute p-values $P_i = \Phi(-z_i)$ and then apply the BH procedure with $\alpha = 0.2$. Plot the sorted p-values as well as the decision boundary. How many discoveries did you make?

solutions to (a) and (b): The z-values have heavier tails than the theoretical null. This could either mean a significant fraction of the subjects are non-null, or that the theoretical null $\mathcal{N}(0,1)$ otherwise fails to describe these statistics [1, see section 6.4]. Applying BH with these p-values, we made 201 discoveries.



(c) A better fit to the z-scores is given by $\mathcal{N}(0.10, 1.40^2)$, called the empirical null. Repeat steps **(a)** and **(b)** treating the empirical null as the null distribution.

solution: The empirical null has a much better fit. Set the p-values to $P_i = \Phi(-\frac{z_i - .1}{1.4})$ Applying BH with these p-values, we made 5 discoveries.



(d) What assumption(s) are we implicitly making in part **(c)** by replacing the theoretical null $\mathcal{N}(0, 1)$ with one which fits the data well $\mathcal{N}(0.10, 1.40^2)$? What are the limitations of using the theoretical null? Which approach would you take when reporting discoveries of bias in this example? What other limitations do you see to this approach to modeling bias?

solution: See [1, section 6.4] for a discussion of issues with the theoretical null. Correlation between the z_i might explain over-dispersion.

By fitting a single normal distribution to the nulls, we are essentially assuming π_0 is large (in fact, $\pi_0 \geq .9$ was assumed in fitting the empirical null). The authors of the original study [4] noted this limitation of the empirical null:

> Implicit in the proposed framework, which draws on a multiple-comparison idea relevant to hypothesis testing, is an assumption that numerous officers have the same level of bias, which is either near zero or identically equal to zero. Although the method compares officers to their peers, it is not necessarily the case that their peers are unbiased. If, for example, all of the officers in a precinct act in a racially biased manner then when each is compared with the others, none of the officers in this precinct will be flagged as problematic. Only in the case that most officers are unbiased and only a few are problematic, the setting several police executives have suggested, will the method actually measure race bias among officers.

Studies of racial bias in policing should not consider individual officers as the only unit of analysis, but should also look at deployment and enforcement disparities within and across entire departments. See also [2] for a Bayesian regression analysis of stop-and-frisk records.

References

[1] Bradley Efron. *Large-scale inference: empirical Bayes methods for estimation, testing, and prediction.* Cambridge University Press, 2012.

[2] Andrew Gelman, Jeffrey Fagan, and Alex Kiss. *An analysis of the New York City police department's "stop-and-frisk" policy in the context of claims of racial bias.* Journal of the American Statistical Association, Vol. 102, 2007.

[3] Adityanand Guntuboyina. *Frequentist vs Bayes testing* (blog) September 16, 2019, https://statpacking.wordpress.com/2019/09/16/frequentist-vs-bayes-testing/.

[4] Greg Ridgeway and John M. MacDonald *Doubly robust internal benchmarking and false discovery rates for detecting racial bias in police stops.* Journal of the American Statistical Association, Vol. 104, 2009.

1. Ridge as MAP

In this problem, we work through the maximum *a posteriori* (MAP) interpretation of ridge regression. Suppose $x_1, \ldots, x_n \in \mathbb{R}^d$ are fixed feature vectors. Assume the linear model, where we observe

$$y_i = \beta^\top x_i + \varepsilon_i,\ i = 1, \ldots, n,$$

where $\varepsilon_i \sim N(0, \sigma^2)$ are independent of each other, and $\beta \in \mathbb{R}^d$ and $\sigma^2 > 0$ are unknown.

Let $y = (y_1, \ldots, y_n), \varepsilon = (\varepsilon_1, \ldots, \varepsilon_n)$, and let X denote the matrix whose i-th row is equal to x_i. Using this notation, we may more succinctly write the linear model as

$$y = X\beta + \varepsilon, \quad \varepsilon \sim N(0, \sigma^2 I_n).$$

We model the regression weights as a random variable with the following prior distribution:

$$\beta \sim N(0, \sigma_\beta^2 I_d).$$

where $\sigma_\beta^2 > 0$ is hyperparameter we choose.

(a) Write the posterior distribution for β after observing the data, $p(\beta|X, y)$. *Hint:* use Bayes' rule and the probability density functions of multivariate Gaussians. Also use the fact that for a vector z, $z^T z = \|z\|_2^2$, where $\|z\|_2$ is the Euclidean norm of z.

solution: From the linear model, $y \mid X, \beta \sim N(X\beta, \sigma^2 I_n)$. Using the proportionality form of Bayes rule,

$$\begin{aligned} p(\beta \mid X, y) &\propto p(y \mid X, \beta) p(\beta) \\ &\propto \exp\left(-\frac{1}{2} \left[\underbrace{(y - X\beta)'(\sigma^2 I_n)^{-1}(y - X\beta) + \beta'(\sigma_\beta^2 I_d)^{-1}\beta}_{\triangleq q(\beta)} \right] \right) \end{aligned}$$

Both terms in the exponent $q(\beta)$ are quadratic, so the posterior distribution $\beta \mid X, y \sim N(\mu^*, \Sigma^*)$ for some d-dimensional mean μ^* and $d \times d$ covariance matrix Σ^*. The posterior density $p(\beta \mid X, y)$ is proportional to the density of $N(\mu^*, \Sigma^*)$, so

$$\exp\left(-\frac{1}{2} q(\beta)\right) \propto \exp\left(-\frac{1}{2} \left[\underbrace{(\beta - \mu^*)'(\Sigma^*)^{-1}(\beta - \mu^*)}_{\triangleq \bar{q}(\beta)} \right] \right),$$

which, taking logarithms, implies

$$q(\beta) = \bar{q}(\beta) + C$$

for some constant C. Expanding the quadratics,

$$\begin{aligned} q(\beta) &= \beta' \left(\frac{1}{\sigma^2} X'X + \frac{1}{\sigma_\beta^2} I_d \right) \beta - 2 \left(\frac{1}{\sigma^2} y'X \right) \beta + C' \\ \bar{q}(\beta) &= \beta' \left(\Sigma^*\right)^{-1} \beta - 2 \left((\mu^*)'(\Sigma^*)^{-1} \right) \beta + C'' \end{aligned}$$

implying $\Sigma^* = \left(\frac{1}{\sigma^2} X'X + \frac{1}{\sigma_\beta^2} I_d \right)^{-1}$ and $\mu^* = \Sigma^* \left(\frac{1}{\sigma^2} X'y \right) = \left(X'X + \frac{\sigma^2}{\sigma_\beta^2} I_d \right)^{-1} X'y$.

(b) Show that the MAP estimator of β,

$$\hat{\beta}_{MAP} := \arg\max_{\beta} p(\beta|X, y)$$

solves the regularized least-squares problem,

$$\arg\min_{\beta} \|X\beta - y\|_2^2 + \lambda\|\beta\|_2^2$$

with $\lambda = \frac{\sigma^2}{\sigma_\beta^2}$. *Hint:* use part (a).

solution: By manipulating the definition of $\hat{\beta}_{MAP}$,

$$\begin{aligned}
&\arg\max_{\beta} \frac{1}{(2\pi)^{(n+d)/2}\sigma^n\sigma_\beta^d} e^{-\frac{1}{2\sigma^2}\|X\beta-y\|_2^2-\frac{1}{2\sigma_\beta^2}\|\beta\|_2^2} \\
&= \arg\max_{\beta} \log\left(\frac{1}{(2\pi)^{(n+d)/2}\sigma^n\sigma_\beta^d} e^{-\frac{1}{2\sigma^2}\|X\beta-y\|_2^2-\frac{1}{2\sigma_\beta^2}\|\beta\|_2^2}\right) \\
&= \arg\max_{\beta} \log\left(\frac{1}{(2\pi)^{(n+d)/2}\sigma^n\sigma_\beta^d}\right) - \frac{1}{2\sigma^2}\|X\beta - y\|_2^2 - \frac{1}{2\sigma_\beta^2}\|\beta\|_2^2 \\
&= \arg\min_{\beta} \frac{1}{2\sigma^2}\|X\beta - y\|_2^2 + \frac{1}{2\sigma_\beta^2}\|\beta\|_2^2 \\
&= \arg\min_{\beta} \|X\beta - y\|_2^2 + \frac{\sigma^2}{\sigma_\beta^2}\|\beta\|_2^2
\end{aligned}$$

Another way to solve this is to note that the mode of the posterior distribution from part (a) is μ^* (since the mode of a Gaussian equals its mean), so

$$\hat{\beta}_{MAP} = \mu^* = \left(X'X + \frac{\sigma^2}{\sigma_\beta^2} I_d\right)^{-1} X'y,$$

which is precisely the closed form solution to ridge regression with $\lambda = \frac{\sigma^2}{\sigma_\beta^2}$.

(c) In the regularized least-squares problem, λ is the regularization term: large values of λ penalize weight vectors with large norms. Since $\hat{\beta}_{MAP}$ is the solution to the regularized least-squares problem with $\lambda = \frac{\sigma^2}{\sigma_\beta^2}$, explain how our modeling decision (*i.e.*, choice of σ_β^2) influences $\hat{\beta}_{MAP}$.

solution: Smaller σ_β^2 indicates that our prior is "more confident" and corresponds to greater regularization, causing the norm of $\hat{\beta}_{MAP}$ to be smaller (closer to the mean of the prior, which is the zero vector).

2. Rejection Sampling

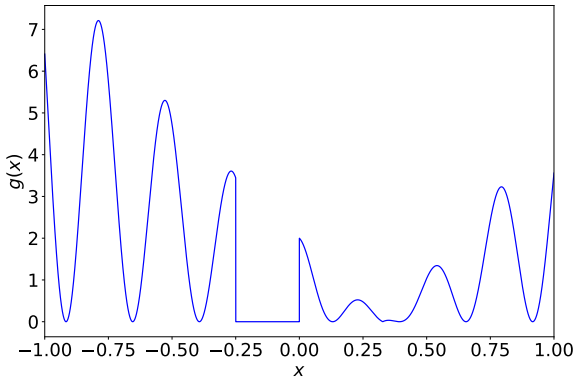
Consider the function

$$g(x) = \cos^2(12x) \times |x^3 + 6x - 2| \times \mathbb{1}_{x\in(-1,-.25)\cup(0,1)}.$$

In this problem, we use rejection sampling to generate random variables with pdf $f(x) = cg(x)$.

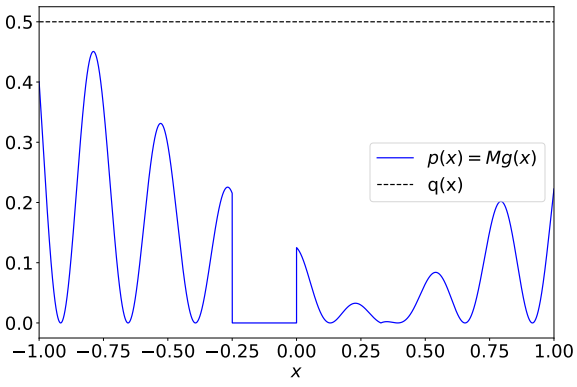
(a) Plot g over its domain. What is a uniform proposal distribution q that covers the support of f? What is a constant M such that the scaled target distribution $p(x) = Mg(x)$ satisfies $p(x) \leq q(x)$ for all x?

solution: One proposal is $q(x) = \frac{1}{2}\mathbb{1}_{x\in(-1,1)}$. Since $g(x) \leq 8$ on the range $-1 \leq x \leq 1$ we will take $M = \frac{1}{16}$ so that $p(x) \leq q(x)$. Since $\max_x g(x) \approx 7.21$ we can really take any $M \leq \frac{1}{14.42}$ for this proposal distribution.



(b) Suppose you run rejection sampling with target p and proposal q from part (a) until you generate n samples and your sampler runs a total of $N \geq n$ times, including n acceptances and $N - n$ rejections. Explain how you can use n, N and M to estimate c.

solution: Below we plot the target p and proposal q.



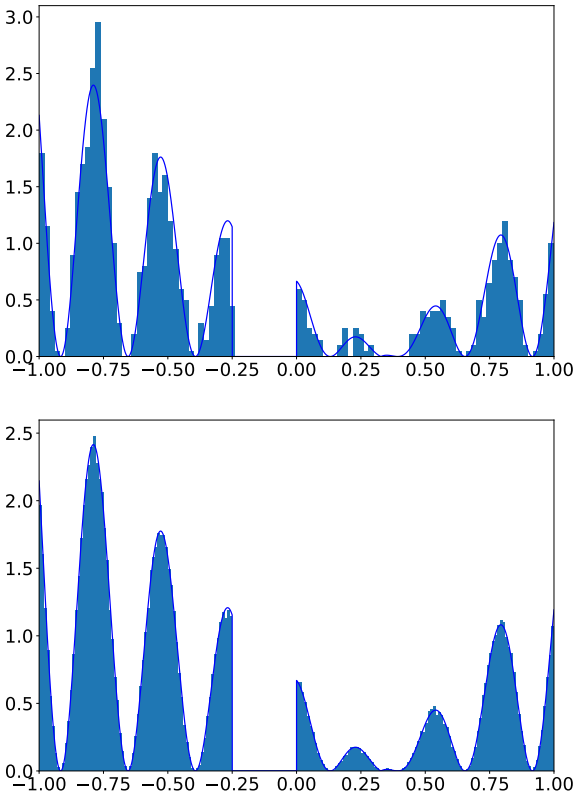
Rejection sampling is set up so that we accept a sample if it lies below the blue target curve. In particular

$$\frac{n}{N} \approx \frac{\int_{-1}^{1} p(x)dx}{\int_{-1}^{1} q(x)dx} = M \int_{-1}^{1} g(x)dx = \frac{M}{c} \int f(x)dx = \frac{M}{c}.$$

Thus, we shall use $\hat{c} = \frac{NM}{n}$ to estimate c.

(c) Use rejection sampling to generate a sample of size 10^3 from f and overlay a line plot of f atop a normalized histogram of your samples. Repeat this step with 10^6 samples. *Hint:* to plot f, first use your values of n, N and M to estimate c using your answer from part (b).

solution:



3. Gibbs Sampling

Graphical models are often useful for modeling phenomena involving multiple variables. In this problem, you'll formulate a graphical model, then demonstrate how to sample from the posterior using Gibbs sampling.

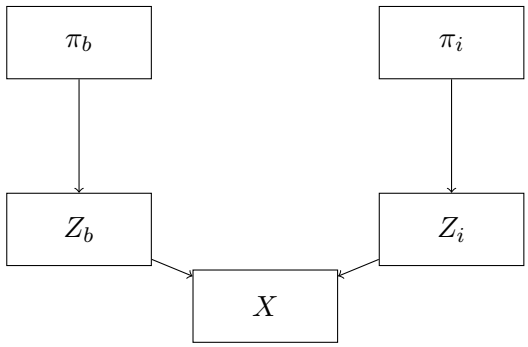
(a) Consider the following scenario: suppose the probability that a burglar breaks into your car is π_b, and the probability that an innocent passerby accidentally touches your car is π_i. Let Z_b be a binary random variable that is 1 if there is a burglar, and 0 otherwise. Likewise, let Z_i be a binary random variable that is 1 if there is an innocent passerby, and 0 otherwise. Suppose Z_b and Z_i are independent of each other.

Let X be a binary random variable that is 1 if your car alarm goes off. The probability your car alarm goes off depends on Z_b and Z_i, and is known to be:

Z_b	Z_i	$\mathbb{P}(X = 1 \mid Z_b, Z_i)$
0	0	0
0	1	0.05
1	0	0.85
1	1	0.90

Draw the graphical model depicting the direct relationships between π_b, π_i, Z_b, Z_i, and X.

solution:



(b) Suppose you know the parameters π_b and π_i, as well as $\mathbb{P}(X = 1 \mid Z_b, Z_i)$ as specified in Part (a). X is the observed variable, and Z_i and Z_b are the latent (unobserved) variables. We want to sample from $\mathbb{P}(Z_i, Z_b \mid X, \pi_b, \pi_i)$, the posterior over the latent variables conditioned on everything else. We'll use Gibbs sampling to do this:

(i) Suppose we are running Gibbs sampling, and on each iteration we sample Z_b first and then sample Z_i. We observed $X = 0$, and the values of Z_b and Z_i from iteration t are $Z_b^{(t)} = 0$ and $Z_i^{(t)} = 1$.

Derive the distribution used for the Gibbs sampling update of $Z_b^{(t+1)}$. Your solution should be in terms of π_b, π_i, and constants.

solution: For the Gibbs sampling update of $Z_b^{(t+1)}$, we have

$$\begin{aligned}
\mathbb{P}(Z_b \mid Z_i, X, \pi_b, \pi_i) &= \frac{\mathbb{P}(Z_b, Z_i, X \mid \pi_b, \pi_i)}{\mathbb{P}(Z_i, X \mid \pi_b, \pi_i)} \\
&= \frac{\mathbb{P}(X \mid Z_b, Z_i, \pi_b, \pi_i)\mathbb{P}(Z_b, Z_i \mid \pi_b, \pi_i)}{\mathbb{P}(Z_i, X \mid \pi_b, \pi_i)} \\
&= \frac{\mathbb{P}(X \mid Z_b, Z_i)\mathbb{P}(Z_b \mid \pi_b)\mathbb{P}(Z_i \mid \pi_i)}{\mathbb{P}(Z_i, X \mid \pi_b, \pi_i)} \\
&= \frac{\mathbb{P}(X \mid Z_b, Z_i)\mathbb{P}(Z_b \mid \pi_b)\mathbb{P}(Z_i \mid \pi_i)}{\sum_{z\in\{0,1\}} \mathbb{P}(Z_b = z, Z_i, X \mid \pi_b, \pi_i)} \\
&= \frac{\mathbb{P}(X \mid Z_b, Z_i)\mathbb{P}(Z_b \mid \pi_b)\mathbb{P}(Z_i \mid \pi_i)}{\sum_{z\in\{0,1\}} \mathbb{P}(X \mid Z_b = z, Z_i)\mathbb{P}(Z_b = z \mid \pi_b)\mathbb{P}(Z_i \mid \pi_i)}.
\end{aligned}$$

Now we plug in the specific values given in the problem, including $X = 0$ and $Z_i = Z_i^{(t)} = 1$. Since Z_b is a binary random variable, to find its distribution we can just compute $\mathbb{P}(Z_b = 1 \mid Z_i = 1, X = 0, \pi_b, \pi_i)$. Plugging in the values of $\mathbb{P}(X \mid Z_b = 1, Z_i = 1)$ given in Part (a), and $\mathbb{P}(Z_b = 1) = \pi_b$ and $\mathbb{P}(Z_i = 1) = \pi_i$, we have

$$\begin{aligned}
&\mathbb{P}(Z_b = 1 \mid Z_i = 1, X = 0, \pi_b, \pi_i) \\
&= \frac{\mathbb{P}(X = 0 \mid Z_b = 1, Z_i = 1)\mathbb{P}(Z_b = 1 \mid \pi_b)\mathbb{P}(Z_i = 1 \mid \pi_i)}{\sum_{z\in\{0,1\}} \mathbb{P}(X = 0 \mid Z_b = z, Z_i = 1)\mathbb{P}(Z_b = z \mid \pi_b)\mathbb{P}(Z_i = 1 \mid \pi_i)} \\
&= \frac{0.1 \cdot \pi_b\pi_i}{0.95 \cdot (1 - \pi_b) \cdot \pi_i + 0.1 \cdot \pi_b\pi_i}.
\end{aligned}$$

That is, $Z_b^{(t+1)}$ is a Bernoulli random variable with probability of one equal to

$$\frac{0.1 \cdot \pi_b\pi_i}{0.95 \cdot (1 - \pi_b) \cdot \pi_i + 0.1 \cdot \pi_b\pi_i}.$$

(ii) Now, suppose we draw $Z_b^{(t+1)} = 1$ from the distribution derived in Part (b.i). Derive the distribution used for the Gibbs sampling update of $Z_i^{(t+1)}$. Your solution should be in terms of π_b, π_i, and constants.

solution: By the same reasoning as Part (b.i), for the Gibbs sampling update of $Z_i^{(t+1)}$ we can focus on finding $\mathbb{P}(Z_i = 1 \mid Z_b = 1, X = 0, \pi_b, \pi_i)$ (since we drew $Z_b^{(t+1)} = 1$):

$$\begin{aligned}
&\mathbb{P}(Z_i = 1 \mid Z_b = 1, X = 0, \pi_b, \pi_i) \\
&= \frac{\mathbb{P}(X = 0 \mid Z_b = 1, Z_i = 1)\mathbb{P}(Z_b = 1 \mid \pi_b)\mathbb{P}(Z_i = 1 \mid \pi_i)}{\sum_{z\in\{0,1\}} \mathbb{P}(X = 0 \mid Z_b = 1, Z_i = z)\mathbb{P}(Z_i = z \mid \pi_i)\mathbb{P}(Z_b = 1 \mid \pi_b)} \\
&= \frac{0.1 \cdot \pi_b\pi_i}{0.15 \cdot (1 - \pi_i)\pi_b + 0.1 \cdot \pi_b\pi_i}.
\end{aligned}$$

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.

Solution:

$$Y_t \sim \text{Bernoulli}(\mu(t)). \tag{2}$$

(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

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

Solution: Using the probability under a Poisson that the count equals zero, we have

$$\mu(t) = 1 - \mathbb{P}(\text{there are 0 microbes on plate}) \tag{6}$$
$$= 1 - \exp(-\rho_t v). \tag{7}$$

(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.

Solution:

$$\log(-\log(1 - \mu(t))) = \log \rho_t + \log v \tag{9}$$
$$= \log \rho_0 + \log v - \log 2t \tag{10}$$

so $\beta_0 = \log v + \log \rho_0$ and $\beta_1 = -\log 2$.

(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.

Solution: To get an estimate of ρ_0, take $\rho_0 = \exp(\beta_0 - \log v)$ (since v is known).

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.

Solution:

$$\hat{\theta} = \frac{-452.3}{6342} \tag{13}$$

$$= -0.0713. \tag{14}$$

(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

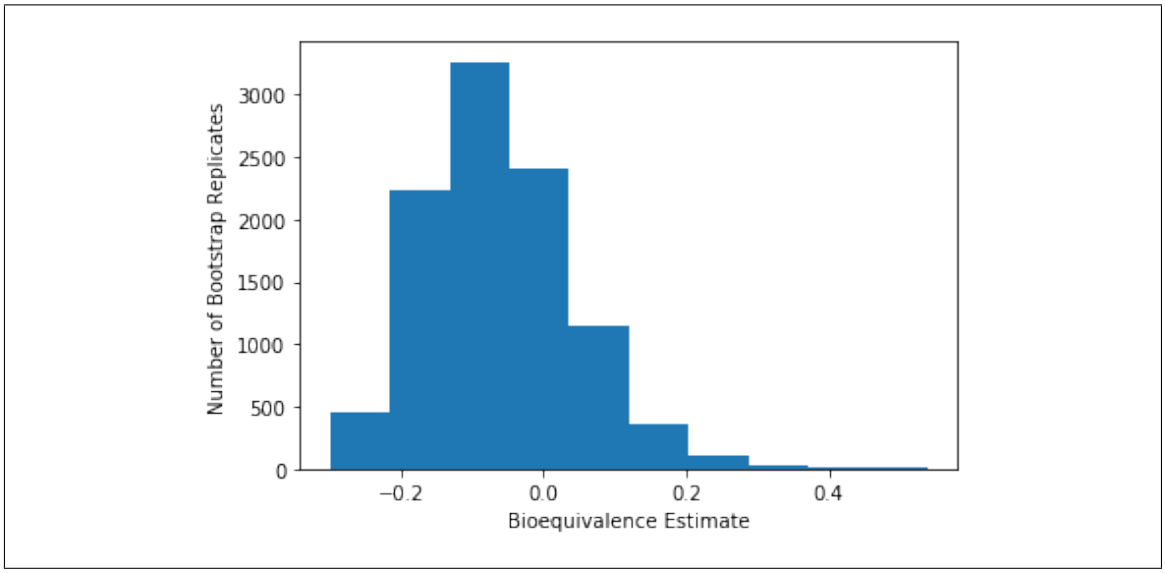
and outputs a length-B array of bootstrap replicates of $\hat{\theta}$.

Solution:

```
def bootstrap_bioequivalence(N, O, P, B):
    bootstrap_replicates = []
    for b in range(B):
        idx = np.random.choice(n, size=N.size, replace=True)
        bootstrap_replicates.append(np.mean(N[idx] - O[idx])
                                    / np.mean(O[idx] - P[idx]))
return np.array(bootstrap_replicates)
```

(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.

Solution:



(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`.

Solution: The 95-percentile confidence interval is $(-0.23, 0.16)$.

(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?

Solution: No, since $(-0.23, 0.16)$ does not fall in $[-0.2, 0.2]$.

3. Image Denoising with Gibbs Sampling

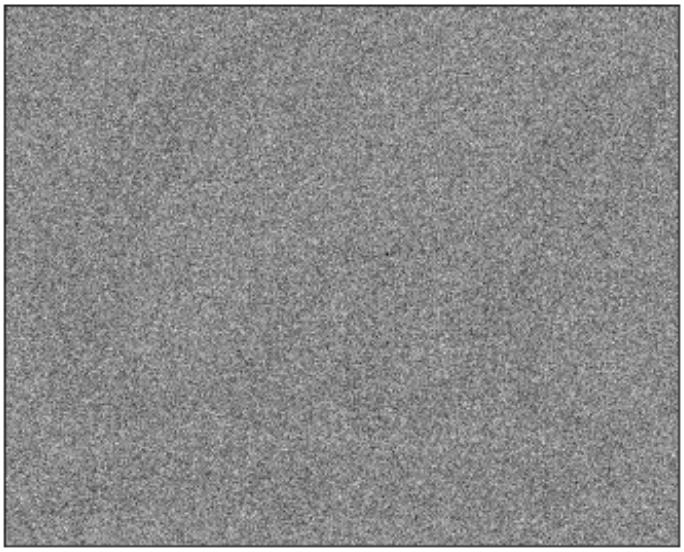
In this problem, we derive a Gibbs sampling algorithm to restore a corrupted image [1]. A grayscale image can be represented by a 2-dimensional array X of shape $n \times m$, where the intensity of the (i, j)-th pixel is X_{ij}. In this problem, we are given an image X whose pixels have been corrupted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

Solution: We used a reverse grayscale colormap:

```
import matplotlib.pyplot as plt
plt.imshow(X, cmap=plt.cm.get_cmap('gray').reversed())
```

We see that the resulting image is quite noisy.



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.*
  - ...
  - Sample $Z_{n,m}^{(t)} \sim p(Z_{n,m} \mid Z_{1,1} = Z_{1,1}^{(t)}, Z_{1,2} = Z_{1,2}^{(t)}, \ldots, Z_{n,m-1} = Z_{n,m-1}^{(t)}, X)$

Solution: $p\left(Z_{1,3} \mid Z_{1,1} = Z_{1,1}^{(t)}, Z_{1,2} = Z_{1,2}^{(t)}, Z_{1,4} = Z_{1,4}^{(t-1)}, \ldots, Z_{n,m} = Z_{n,m}^{(t-1)}, X\right)$.

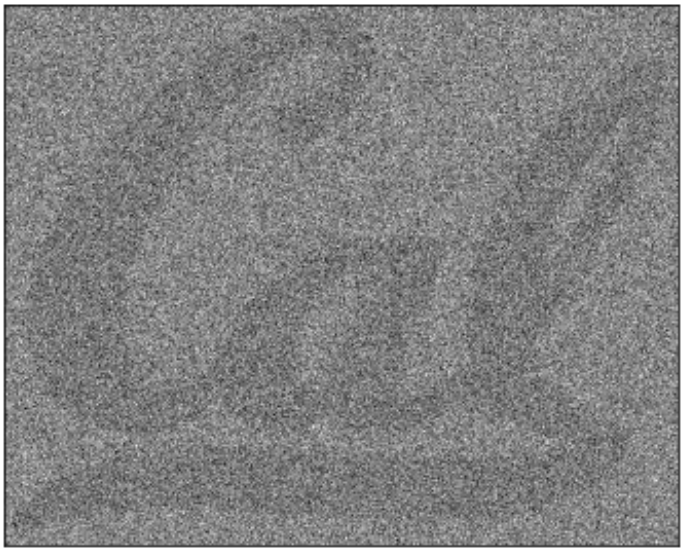
(c) (3 points) Write the pseudo-code from Part (b) more explicitly both by using a double for-loop over $(i, j) \in \mathcal{I}$ and by being explicit about the conditional distributions of the form $p(Z_{1,1} \mid Z_{1,2} = Z_{1,2}^{(t-1)}, Z_{1,3} = Z_{1,3}^{(t-1)}, \ldots, Z_{n,m} = Z_{n,m}^{(t-1)},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

Solution:

- Initialize $Z^{(0)} = X$.
- For $t = 1, \ldots, T$:
  - Let $Z = Z^{(t-1)}$.
  - For $i = 1, \ldots, n$:
    - For $j = 1, \ldots, m$:
      - Let $S_{ij} = \sum_{(i',j') \in N_{(i,j)}} Z_{i'j'}$
      - Sample $Z_{i,j} \sim \mathcal{N}\left(\frac{\tau X_{ij} + bS_{ij}}{a+\tau}, \frac{1}{a+\tau}\right)$
  - Let $Z^{(t)} = Z$.

(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 $Z^{(100)}$.

Solution: The sample $Z^{(1)}$ unsurprisingly looks pretty similar to X but you can definitely see the logo better. Based on how long a single iteration took, reaching $Z^{(100)}$ would take about twelve minutes.



```
Z = X
for i in range(n):
    for j in range(m):
        N_ij = [(i-1,j), (i, j-1), (i+1, j), (i, j+1)]
        S_ij = sum(Z[u, v] for (u,v) in N_ij
                   if 0 <= u < n and 0 <= v < m)
        Z[i,j] = ((tau*X[i,j] + b*S_ij) / (a+tau)
                  + np.random.randn()*np.sqrt(1/(a+tau)))
```

(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.

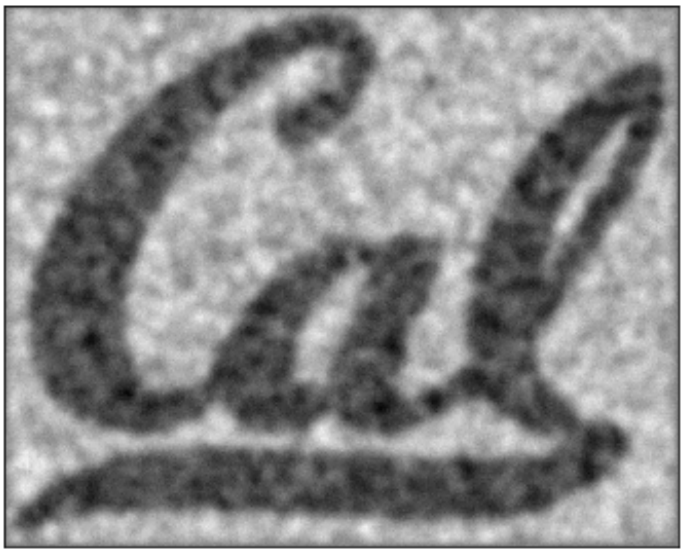
Solution: Note in particular the posterior (15) can be written as

$$p(X \mid Z) \propto \exp\left(-\frac{1}{2} \sum_{(i,j) \in \mathcal{I}_{\text{even}}} \left[(a+\tau) Z_{ij}^2 - 2\tau Z_{ij} X_{ij} - b \sum_{(i',j') \in N_{(i,j)}} Z_{ij} Z_{i'j'} \right] \right)$$
$$\times \exp\left(-\frac{1}{2} \sum_{(i,j) \in \mathcal{I}_{\text{odd}}} \left[(a+\tau) Z_{ij}^2 - 2\tau Z_{ij} X_{ij} - b \sum_{(i',j') \in N_{(i,j)}} Z_{ij} Z_{i'j'} \right] \right)$$

And, crucially, all the neighbors $N_{(i,j)}$ are odd when $i+j$ is even and vice versa.

(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.

Solution: It took about 14 seconds to sample $Z^{(100)}$.



```
A, B = np.meshgrid(np.arange(m), np.arange(n))

even_ = (A + B) % 2 == 0
odd_  = (A + B) % 2 == 1

Z = np.pad(X, 1)
even = np.pad(even_, 1)
odd  = np.pad(odd_, 1)

for it in range(100):
    S = Z[2:, 1:-1] + Z[:-2, 1:-1] + Z[1:-1, 2:] + Z[1:-1, :-2]
    delta = ((tau*X + b*S) / (a+tau)
             + np.random.randn(n, m)*np.sqrt(1/(a+tau)))
    Z[even] = np.pad(delta*even_, 1)[even]

    S = Z[2:, 1:-1] + Z[:-2, 1:-1] + Z[1:-1, 2:] + Z[1:-1, :-2]
    delta = ((tau*X + b*S) / (a+tau)
             + np.random.randn(n, m)*np.sqrt(1/(a+tau)))
    Z[odd] = np.pad(delta*odd_, 1)[odd]
```

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.

Overview

Submit your writeup including all 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 test, maintain, and 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 Observational Data on Infant Health

The Infant Health and Development Program (IHDP) was an experiment treating low-birth-weight, premature infants with intensive high-quality childcare from a trained provider. The goal is to estimate the causal effect of this treatment on the child's cognitive test scores. The data *does not* represent a randomized trial with randomly allocated treatment, so there may be confounders between treatment and outcome. In this problem, we devise a propensity score model to control for observed confounders. Review Lecture 15 to make sure you understand the details of propensity score modeling in causal inference.

(a) (2 points) The CSV file `ihdp.csv` has 27 columns:

- Column 1 is the treatment $z_i \in \{0, 1\}$, which indicates whether or not the treatment was given to the infant.
- Column 2 is the outcome $y_i \in \mathbb{R}$, the child's cognitive test score.
- Columns 3-27 contain 25 features of the mother and child (*e.g.* the child's birth weight, whether or not the mother smoked during pregnancy, her age and race). Since this dataset was not collected by a randomized trial, these features could all confound z_i and y_i, and are denoted by $x_i \in \mathbb{R}^{25}$.

In this part, you'll estimate $\hat{e}(x)$ by fitting a logistic regression model that predicts z_i from x_i. For any x_i, $\hat{e}(x_i)$ is then the predicted probability that $z_i = 1$ made by the logistic regression model on x_i. Specifically:

1. Read the data in `ihdp.csv` (*e.g.* using the `csv` package in Python) into three arrays: $Z \in \{0, 1\}^n$ containing the treatments, $Y \in \mathbb{R}^n$ containing the outcomes, and $X \in \mathbb{R}^{n \times 25}$ containing the features.

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

2. To fit a logistic regression model, use the `scikit-learn` package in Python, which is imported as `sklearn`. Start with the following two lines:

   ```
   from sklearn.linear_model import LogisticRegression as LR
   lr = LR(penalty='none', max_iter=200, random_state=0)
   ```

3. Use the `lr.fit()` method to fit the logistic regression model $\hat{e}(x)$

See the documentation here

Solution: `lr.fit(X, Z)`

(b) (2 points) Write a function `estimate_treatment_effect` to estimate treatment accounting for the propensity. It should take as arguments a fitted regression model (the `LogisticRegression` object `lr` from the previous part), X, Y, and Z, and output a single value, which is the estimate of the average treatment effect.

Hint: Use the inverse propensity weighted estimator:

$$\hat{\tau} = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{z_i y_i}{\hat{e}(x_i)} - \frac{(1-z_i)y_i}{1-\hat{e}(x_i)}\right). \tag{1}$$

See the `LogisticRegression` object's predict_proba method.

Solution:

```
def estimate_treatment_effect(lr, X, Y, Z):
    ex = lr.predict_proba(X)[:, 1]
    return np.mean(Z * Y / ex) - np.mean((1 - Z) * Y / (1 - ex))
```

(c) (3 points) Use the function `estimate_treatment_effect` from the previous part to estimate the treatment effect on the IHDP dataset. Report this estimate. According to the estimate, did the treatment have a beneficial causal effect on the outcome (*i.e.* cause cognitive test scores to increase)?

Solution: 3.7. Since this estimate of the treatment effect is positive, it indicates that the treatment had a beneficial effect on cognitive test scores.

(d) (3 points) The naïve estimator is the difference between the sample means:

$$\tilde{\tau} = \frac{1}{n_1}\sum_{i=1}^{n} y_i z_i - \frac{1}{n_0}\sum_{i=1}^{n} y_i(1-z_i), \tag{2}$$

where $n_1 = \sum_{i=1}^{n} z_i$ and $n_0 = n - n_1$. Report this estimate on the IHDP dataset. Why is it different from the estimate you computed in the previous part? Are there any circumstances under which these two estimators should produce the same estimates?

Solution: 4.0. Unlike the estimator derived in the previous problem, this naive estimator does not take into account the existence of confounders X between Z and Y. Here, confounders have inflated the apparent causal effect of the treatment on the outcome, as the naïve estimate is larger than the one in the part (c).

Even if Z and Y are not confounded at all, however, these two estimators will still in general produce different values. One condition under which $\tilde{\tau} = \hat{\tau}$ is that $\hat{e}(x) = n_1/n$ for all x.

2 Concentration of Counts

In this problem we apply concentration inequalities to sums of independent but not identically distributed random variables. If the university has n students, let X_j denote the indicator that student j is on CalCentral. We will (unrealistically) assume the X_j are independent for $j = 1, \ldots, n$, and we would like to get high-probability bounds on the total number of students on CalCentral $S = X_1 + X_2 + \cdots + X_n$, since if too many students are on the site at once it will crash. Assume we know from historical data that $\mathbb{P}(X_j = 1) = p_j$.

(a) (2 points) Write $\mu = \mathbb{E}[S]$ and $\sigma^2 = Var(S)$ in terms of $p_1, p_2, \ldots, p_n$.

Solution:

$$\mathbb{E}[S] = \mathbb{E}[X_1 + X_2 + \ldots + X_n] = \sum_{i=1}^{n} p_i$$

and by independence

$$\sigma^2 = Var(X_1 + X_2 + \ldots + X_n) = \sum_{i=1}^{n} p_i(1 - p_i)$$

(b) (2 points) Find Markov's bound on $\mathbb{P}(S \geq K\mu)$ for $K > 1$.

Solution:

$$\mathbb{P}(S \geq K\mu) < \frac{\mu}{K\mu} = \frac{1}{K}$$

(c) (3 points) Find Chebyshev's bound on $\mathbb{P}(S \geq K\mu)$ in terms of μ, K and σ.

Solution:

$$\mathbb{P}(S \geq K\mu) \leq \mathbb{P}((S-\mu)^2 \geq (K-1)^2\mu^2) < \frac{\sigma^2}{(K-1)^2\mu^2}$$

(d) (2 points) If all the p_j's are equal to p, what is the value of the bound in (c)? How does the dependence on n compare to the bound you got in part (b)?

Solution:

$$\mathbb{P}(S \geq nKp) < \frac{(1-p)}{np(K-1)^2}$$

Tighter because it decreases with n.

(e) (1 point) Show that the moment generating function of X_j is given by $M_{X_j}(t) = 1 + p_j(e^t - 1)$ for all t. (Recall the definition: $M_{X_j}(t) = \mathbb{E}[e^{tX_j}]$)

Solution:

$$M_{X_j}(t) = \mathbb{E}\left[e^{tX_j}\right] = (1-p_j) + p_j e^t = 1 + p_j(e^t - 1)$$

(f) (2 points) Show that $M_S(t) \leq \exp\big(\mu(e^t - 1)\big)$ for all t. *Hint:* use the fact that S is the sum of independent random variables, and that $e^x \geq 1 + x$ for all $x \geq 0$.

Solution: Using the fact that the mgf of a sum of independent random variables is the product of their mgfs, we get that:

$$M_S(t) = \prod_{j=1}^{n}(1 + p_j(e^t - 1))$$

Using the supplied upper bound we get that:

$$M_S(t) = \prod_{j=1}^{n}(1 + p_j(e^t - 1)) \leq \prod_{j=1}^{n} e^{p_j(e^t-1)} = e^{\mu(e^t-1)}$$

(g) (3 points) Use Chernoff's method and the bound in (f) to show that

$$\mathbb{P}\big(S \geq K\mu\big) \;\leq\; \exp\Big(-\mu(K\log K + 1 - K)\Big)$$

If $p_j = p$ for all j, how does this bound depend on n? *Hint:* You may use the fact that $K \log K + 1 - K \geq 0$ for all $K > 0$.

Solution: Recall that for the Chernoff Bound, we simply use Markov's inequality on the MGF. Therefore:

$$\mathbb{P}(S > K\mu) = \mathbb{P}(e^{tS} > e^{tK\mu}) \leq e^{-tK\mu} M_S(t) \leq \exp\left(\mu(e^t - 1) - tK\mu\right)$$

The right hand side is minimized by taking $t = \log(K)$, which gives

$$\mathbb{P}(S > K\mu) \leq \exp\left(-\mu(K \log K + 1 - K)\right)$$

When $\mu = np$ then the bound above will decrease exponentially as n gets large.

Overview

Submit your writeup including all code and plots as a PDF via Gradescope. We recommend reading through the entire homework beforehand and carefully using functions for testing procedures, plotting, and running experiments. Taking the time to test, maintain, and 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 hand-written answers are legible, as we may deduct points otherwise.

1 Simulation Study of Bandit Algorithms

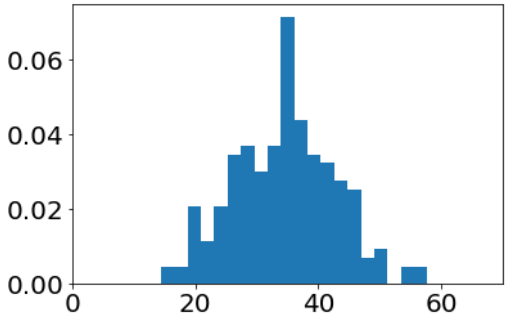
In this problem, we evaluate the performance of three algorithms for the multi-armed bandit problem. The general protocol for the multi-armed bandit problem with K arms and n rounds is as follows: in each round $t = 1, \ldots, n$ the algorithm chooses an arm $A_t \in \{1, \ldots, K\}$ and then observes reward r_t for the chosen arm. The bandit algorithm specifies how to choose the arm A_t based on what rewards have been observed so far. In this problem, we consider a multi-armed bandit $K = 2$ arms, $n = 50$ rounds, and where the reward at time t is $r_t \sim \mathcal{N}(A_t - 1, 1)$, i.e. $\mathcal{N}(0, 1)$ for arm 1 and $\mathcal{N}(1, 1)$ for arm 2.

(a) (2 points) Consider the multi-armed bandit where the arm $A_t \in \{1, 2\}$ is chosen according to the ε-greedy algorithm (below) with $c = 4$ and $\varepsilon = 1/2$. Let $G_n = \sum_{t=1}^{n} r_t$ denote the total reward after $n = 50$ iterations. Simulate the random variable G_n a total of $B = 200$ times and save the values $G_n^{(b)}$, $b = 1, \ldots, B$ in a list. Report the expected value $\frac{1}{B} \sum_{b=1}^{B} G_n^{(b)}$ of the total reward and plot a normalized histogram of the rewards.

Algorithm 1 ε-Greedy Algorithm

input: Number of initial pulls c per arm; additional exploratory fraction $\varepsilon \in (0, 1)$
for $t = 1, \ldots, cK$: **do**
  Choose arm $A_t = (t \bmod K) + 1$
end
for $t = cK + 1, cK + 2, \ldots, n$: **do**
  Toss coin with success probability ε:
  if *success* **then**
    Choose arm A_t uniformly at random
  else
    Choose arm A_t with the highest average reward so far.
  end
end

Solution: The average reward was anywhere in the range $[32, 37]$ over repeated runs of the entire simulations.

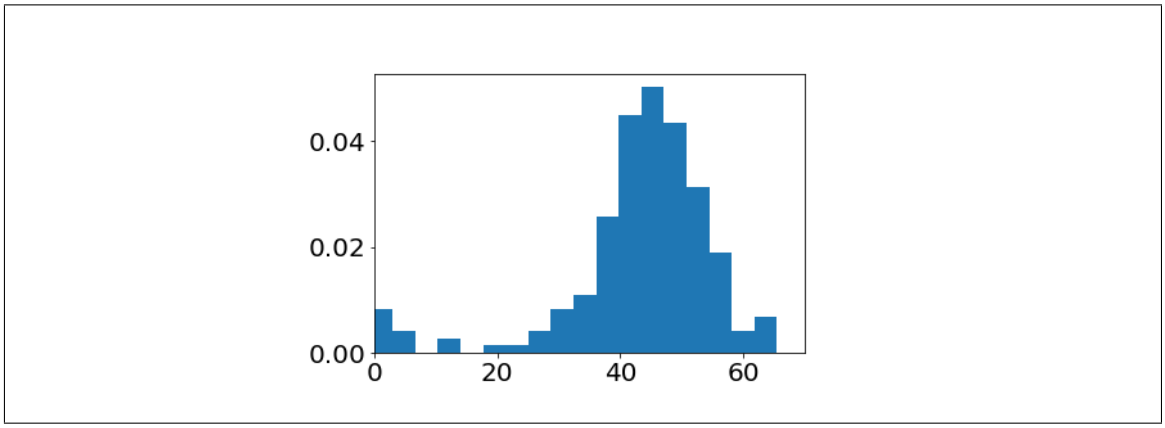


(b) (4 points) Consider the multi-armed bandit where the arm $A_t \in \{1, 2\}$ is chosen according to the explore-then-commit algorithm (below) with $c = 4$. Repeat the simulation in Part (a) using the explore-then-commit algorithm, again reporting the expected total reward and the histogram of $G_n^{(b)}$ after $n = 50$ rounds. How does the reward G_n compare to your results from part (a)? Compare both the average and the overall distribution.

Algorithm 2 Explore-then-Commit Algorithm

input: Number of initial pulls c per arm
for $t = 1, \ldots, cK$: **do**
| Choose arm $A_t = (t \bmod K) + 1$
end
Let $\hat{A} \in \{1, \ldots, K\}$ denote the arm with the highest average reward so far.
for $t = cK + 1, cK + 2, \ldots, n$: **do**
| Choose arm $A_t = \hat{A}$
end

Solution: The average reward was in the range $[39, 46]$ over many repeated runs of the entire simulation. This means the overall reward is higher with explore-then-commit than ε-greedy, but from the histogram we see that some fraction of the time, the algorithm commits to the wrong arm and thus incurs a small reward.

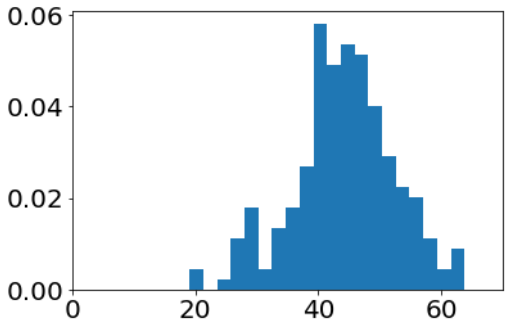


(c) (4 points) Consider the multi-armed bandit where the arm $A_t \in \{1, 2\}$ is chosen according to the UCB algorithm (below) with $c = 4$. Repeat the simulation in Part (a) using the UCB algorithm, again reporting the expected total reward and the histogram of $G_n^{(b)}$ after $n = 50$ rounds. How does the reward G_n compare to your results from part (b)? Compare both the average and the overall distribution. *Note:* if $T_A(t)$ denote the number of times arm A has been chosen (before time t) and $\hat{\mu}_{A,t}$ is the average reward from choosing arm A (up to time t), then use the upper confidence bound $\hat{\mu}_{A,T_A(t-1)} + \sqrt{\frac{2\log(20)}{T_A(t-1)}}$. Note also that this algorithm is slightly different than the one used in lab as we are using an initial exploration phase.

Algorithm 3 UCB Algorithm

input: Number of initial pulls c per arm
for $t = 1, \ldots, cK$: **do**
| Choose arm $A_t = (t \bmod K) + 1$
end
for $t = cK + 1, cK + 2 \ldots$: **do**
| Choose arm A_t with the highest upper confidence bound so far.
end

Solution: The average reward was in the range $[41, 46]$ over many repeated runs of the entire simulation. The distribution looks very similar to the main mode in part (b), since it usually takes UCB a small number of iterations to basically settle on the best arm. It does not suffer from the same issue of getting stuck in the worse arm.



Here is code for all three parts:

```
def eps_greedy(Z, c=4, eps=.05):
    n, K = Z.shape

    arm = np.zeros(n, dtype=int)
    rew = np.zeros(n)

    T = c*K
    # initial exploration phase
    arm[:T] = np.arange(T) % K
    rew[:T] = Z[np.arange(T), arm[:T]]

    # greedy phase
    for t in range(T, n):
        if np.random.rand() < eps:
            arm[t] = np.random.choice(K)
        else:
            arm[t] = np.argmax([
                        np.mean(rew[arm==k][:t]) for k in range(K)])
        rew[t] = Z[t, arm[t]]

    return rew

def explore_commit(Z, c=4):
    n, K = Z.shape

    arm = np.zeros(n, dtype=int)
    rew = np.zeros(n)

    # initial exploration phase
```

```
    T = c*K
    arm[:T] = np.arange(T) % K
    rew[:T] = Z[np.arange(T), arm[:T]]

    # commit phase
    best = np.argmax([np.mean((rew[:T])[arm[:T]==k]) for k in range(K)])
    arm[T:] = best
    rew[T:] = Z[T:, best]
    return rew

def ucb(Z, c=4, eps=.05):
    n, K = Z.shape

    arm = np.zeros(n, dtype=int)
    rew = np.zeros(n)

    # initial exploration phase
    T = c*K
    arm[:T] = np.arange(T) % K
    rew[:T] = Z[np.arange(T), arm[:T]]

    # ucb phase
    for t in range(T, n):
        arm[t] = np.argmax([
                  np.mean((rew[:t])[arm[:t]==k])
                  +np.sqrt(2*np.log(20)/sum(arm[:t]==k)) for k in range(K)])
        rew[t] = Z[t, arm[t]]

    return rew

### Run Simulation
from collections import defaultdict

n, K, B = 50, 2, 200
res = defaultdict(list)

for b in range(B):
        Z = np.random.randn(n, K)
        Z[:, 0] += 1

        res['eps'].append(sum(eps_greedy(Z, eps=.5)))
```

```
        res['etc'].append(sum(explore_commit(Z)))
        res['ucb'].append(sum(ucb(Z)))

{alg:np.mean(v) for alg, v in res.items()}
```

2 Regret of Explore-then-Commit

In this problem, we analyze the regret of the Explore-then-Commit algorithm for the multi-armed-bandit (MAB) problem.

We consider a stochastic MAB problem with a set of $K = 2$ arms $\mathcal{A} = \{1, 2\}$. Recall from Lab 8 that each arm $A \in \mathcal{A}$ is associated with a reward distribution $X_A \sim \mathbb{P}_A$, with corresponding mean $\mu_A = \mathbb{E}[X_A]$. **We will assume throughout this problem that the first arm has higher average reward, i.e.** $\mu_1 > \mu_2$**.** At each round $t = 1, \ldots, n$ our algorithm chooses an arm $A_t \in \mathcal{A}$ and receives a corresponding reward $X_{A_t}^{(t)} \sim \mathbb{P}_{A_t}$, independent of all previous rewards.

If we knew arm 1 has higher average reward we would choose $A_t = 1$ each round in order to maximize the expected total reward. In practice, however, we do not know which arm is better since the means $\{\mu_1, \mu_2\}$ are unknown. The expected reward of our algorithm will always be less than $n\mu_1$, and we quantify the price we pay for not knowing the better arm via the *regret*

$$R_n := n\mu_1 - \mathbb{E}\left[\sum_{t=1}^{n} X_{A_t}^{(t)}\right].$$

In this problem we analyze the regret of the explore-then-commit (ETC) algorithm from problem 1(b). Recall from Algorithm 2 that ETC proceeds in two phases. In the exploration phase, each arm $A \in \mathcal{A}$ is pulled c times in order to produce an estimate $\hat{\mu}_A = \frac{1}{c}\sum_{t \le cK : A_t = A} X_A^{(t)}$ of the mean reward for that arm. In the commit phase, i.e. for every $t > cK$, we choose $A_t = \hat{A}$, where $\hat{A} := \arg\max_{a \in \mathcal{A}} \hat{\mu}_A$ is the apparent best arm at the end of the exploration phase.

In the first part of our analysis, we evaluate the probability that we incorrectly identify arm 2 as the best arm, i.e. $\mathbb{P}(\hat{A} = 2)$.

(a) (3 points) Assume each reward is in the unit interval $[0, 1]$, i.e. $0 \le X_A \le 1$ for all $A \in \mathcal{A}$. Show that

$$\mathbb{P}\left(\hat{A} = 2\right) \le \exp\left(-\frac{c\Delta^2}{2}\right),$$

where $\Delta = \mu_1 - \mu_2$. *Hint:* apply Hoeffding's inequality from Lecture 18.

Solution: We have

$$\begin{aligned}\mathbb{P}(\hat{A}=2) &= \mathbb{P}\left(\hat{\mu}_2 \geq \hat{\mu}_1\right) \\ &= \mathbb{P}\left(\hat{\mu}_2 - \hat{\mu}_1 \geq 0\right) \\ &= \mathbb{P}\Big(\left(\hat{\mu}_2 - \hat{\mu}_1\right) - \mathbb{E}[\hat{\mu}_2 - \hat{\mu}_1] \geq \Delta\Big),\end{aligned}$$

where the last step used $\Delta = \mu_1 - \mu_2$.

Now $\hat{\mu}_2 - \hat{\mu}_1$ is the average of c IID random variables, each of which is bounded on $[-1, 1]$. Hoeffding's inequality provides

$$\mathbb{P}\Big(\left(\hat{\mu}_2 - \hat{\mu}_1\right) - \mathbb{E}[\hat{\mu}_2 - \hat{\mu}_1] \geq \Delta\Big) \leq \exp\left(-\frac{2c\Delta^2}{(1-(-1))^2}\right).$$

In parts (b) and (c), we write the regret in terms of the probability $\mathbb{P}(\hat{A}=2)$.

(b) (3 points) Let m denote the number of times arm 2 has been pulled, up to and including time n. Show

$$R_n = \Delta \mathbb{E}[m].$$

Hint: Start from the following:

$$\begin{aligned}R_n &:= n\mu_1 - \mathbb{E}\left[\sum_{t=1}^{n} X_{A_t}^{(t)}\right] = \mathbb{E}\left[\sum_{t=1}^{n}\left(\mu_1 - X_{A_t}^{(t)}\right)\right] \\ &= \mathbb{E}\left[\sum_{t=1}^{n} \mathbb{I}\{A_t = 1\}\left(\mu_1 - X_1^{(t)}\right)\right] + \mathbb{E}\left[\sum_{t=1}^{n} \mathbb{I}\{A_t = 2\}\left(\mu_1 - X_2^{(t)}\right)\right].\end{aligned}$$

Note also that for all t, A_t is independent of $X_A^{(t)}$ for all $A \in \{1, 2\}$.

Solution: Working from the hint,

$$\begin{aligned}R_n &= \sum_{A\in\mathcal{A}}\sum_{t=1}^{n} \mathbb{E}\left[\mathbb{I}\{A_t = A\}\left(\mu_{A^*} - X_A^{(t)}\right)\right] && \text{linearity of expectation} \\ &= \sum_{A\in\mathcal{A}}\sum_{t=1}^{n} \mathbb{E}\left[\mathbb{I}\{A_t = A\}\right]\mathbb{E}\left[\mu_{A^*} - X_A^{(t)}\right] && \text{by independence} \\ &= \sum_{A\in\mathcal{A}}\sum_{t=1}^{n} \mathbb{E}\left[\mathbb{I}\{A_t = A\}\right]\Delta_A && \text{by definition of } \Delta_A \\ &= \sum_{A\in\mathcal{A}} \Delta_A \sum_{t=1}^{n} \mathbb{E}\left[\mathbb{I}\{A_t = A\}\right] = \sum_{A\in\mathcal{A}} \Delta_A \mathbb{E}[T_A(n)] && \text{linearity of expectation}\end{aligned}$$

(c) (2 points) Show that if $n > 2c$, then

$$\mathbb{E}[m] = c + (n - 2c)\mathbb{P}\left(\hat{A} = 2\right)$$

Hint: If $n > cK$, every arm is pulled deterministically c times during the first cK rounds. Afterward, an arm is only pulled if it is the one we have committed to.

Solution: The hint gives $m = c + (n - Kc)\mathbb{I}[\hat{A} = A]$. Taking expectations proves the claim.

In parts (d) and (e), we finalize our bound on the regret R_n.

(d) (2 points) Show that

$$R_n \leq \Delta\left(c + (n - 2c)\exp\left(-\frac{c\Delta^2}{2}\right)\right).$$

Hint: combine parts (a)-(c).

Solution:

$$R_n \overset{(b)}{=} \Delta\mathbb{E}[m] \overset{(c)}{=} \Delta\left(c + (n - 2c)\mathbb{P}(\hat{A} = 2)\right) \overset{(a)}{\leq} \Delta\left(c + (n - 2c)\exp\left(-\frac{c\Delta^2}{2}\right)\right)$$

(e) (3 points) Suppose you knew the sub-optimality gap Δ. Solve for (and report) a value of c which guarantees that:

$$\exp\left(-\frac{c\Delta^2}{2}\right) \leq \frac{1}{n}.$$

For this number of exploratory pulls c, what is the upper bound on the regret from part (d)? Your answer should be in terms of n and Δ. Does this bound grow linearly in n, or does it do better (i.e. is it sublinear)?

Solution: Solving for c gives

$$c = \frac{2\log n}{\Delta^2}.$$

Plugging this into our regret bound from part (e) gives

$$R_n \leq \frac{2\log n}{\Delta} + \Delta.$$

This is sublinear in n.

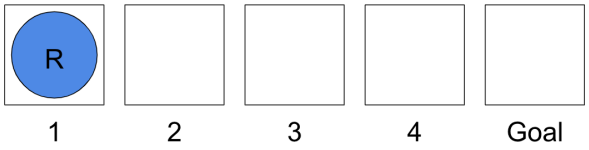
Overview

Submit your writeup including all code and plots as a PDF via Gradescope. We recommend reading through the entire homework beforehand and carefully using functions for testing procedures, plotting, and running experiments. Taking the time to test, maintain, and 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 Markov Decision Process for Robot Soccer

A soccer robot R is on a fast break toward the goal, starting in position 1. From positions 1 through 3, it can either shoot (S) or dribble the ball forward (D). From 4 it can only shoot. If it shoots, it either scores a goal (state G) or misses (state M). If it dribbles, it either advances a square or loses the ball, ending up in state M.



In this Markov Decision Process (MDP), the states are 1, 2, 3, 4, G, and M, where G and M are terminal states. The transition model depends on the parameter y, which is the probability of dribbling successfully (*i.e.*, advancing a square). Assume a discount of $\gamma = 1$. For $k \in \{1, 2, 3, 4\}$, we have

$$\begin{aligned} \mathbb{P}(G \mid k, S) &= \frac{k}{6} \\ \mathbb{P}(M \mid k, S) &= 1 - \frac{k}{6} \\ \mathbb{P}(k+1 \mid k, D) &= y \\ \mathbb{P}(M \mid k, D) &= 1 - y, \\ R(k, S, G) &= 1 \end{aligned}$$

and rewards are 0 for all other transitions.

(a) (3 points) Denote by V^π the value function for the specific policy π. What is $V^\pi(1)$ for the policy π that always shoots?

Solution:

$$\begin{aligned} V^{\pi}(1) &= \mathbb{P}(G|1,S)R(1,S,G) + \mathbb{P}(M|1,S)R(1,S,M) \\ &= (1/6)(1) + (5/6)(0) \\ &= 1/6 \end{aligned}$$

(b) (4 points) Denote by $Q^*(s,a)$ the value of a q-state (s,a), which is the expected utility when starting with action a at state s, and thereafter acting optimally. What is $Q^*(3,D)$ in terms of y?

Solution:

$$\begin{aligned} Q^*(3,D) &= \mathbb{P}(4|3,D)(R(3,D,4) + V^*(4)) + \mathbb{P}(M|3,D)R(3,D,M) \\ &= \mathbb{P}(4|3,D)V^*(4) \\ &= \mathbb{P}(4|3,D)Q^*(4,S) \\ &= \mathbb{P}(4|3,D)(\mathbb{P}(G|4,S)R(4,S,G) + \mathbb{P}(M|4,S)R(4,S,M)) \\ &= \mathbb{P}(4|3,D)\mathbb{P}(G|4,S)R(4,S,G) \\ &= y(4/6)(1) = \frac{2}{3}y \end{aligned}$$

where $V^*(s)$ denotes the value of a state s, which is the expected utility when starting in state s and acting optimally.

(c) (5 points) Denote by $V_t^*(s)$ the value of a state s at iteration t, which is the expected utility when starting in state s and acting optimally. Using $y = \frac{3}{4}$, complete the first two iterations ($t = 1, 2$) of value iteration. Iteration 0 corresponds to having value 0 in every state: $V_0^*(1) = V_0^*(2) = V_0^*(3) = V_0^*(4) = 0$.
Hint: Recall that $V_{t+1}^*(s) = \max_{a \in A} \sum_{s'} \mathbb{P}(s'|s,a)(R(s,a,s') + V_t^*(s'))$.

Solution: As an example, consider computing $Q_2^*(1,D)$:

$$Q_i^*(s,a) = \sum_{s'} T(s,a,s')(R(s,a,s') + \gamma V_{i-1}^*(s'))$$

$$
\begin{aligned}
&Q_2^*(1, D) \\
&= \sum_{s' \in \{2, M\}} \mathbb{P}(s'|1, D)(R(1, D, s') + \gamma V_1^*(s')) \\
&= \mathbb{P}(2|1, D)(R(1, D, 2) + \gamma V_1^*(2)) + \mathbb{P}(M|1, D)(R(1, D, M) + \gamma V_1^*(M)) \\
&= y(0 + (1)(1/3)) + (1 - y)(0 + (0)(0)) \\
&= \frac{1}{4}
\end{aligned}
$$

i	$Q_i(1, S)$	$Q_i(2, S)$	$Q_i(3, S)$	$Q_i(4, S)$
0	0	0	0	0
1	$\frac{1}{6}$	$\frac{1}{3}$	$\frac{1}{2}$	$\frac{2}{3}$
2	$\frac{1}{6}$	$\frac{1}{3}$	$\frac{1}{2}$	$\frac{2}{3}$

i	$Q_i(1, D)$	$Q_i(2, D)$	$Q_i(3, D)$
0	0	0	0
1	0	0	0
2	$\frac{1}{4}$	$\frac{3}{8}$	$\frac{1}{2}$

i	$V_i^*(1)$	$V_i^*(2)$	$V_i^*(3)$	$V_i^*(4)$
0	0	0	0	0
1	$\frac{1}{6}$	$\frac{1}{3}$	$\frac{1}{2}$	$\frac{2}{3}$
2	$\frac{1}{4}$	$\frac{3}{8}$	$\frac{1}{2}$	$\frac{2}{3}$

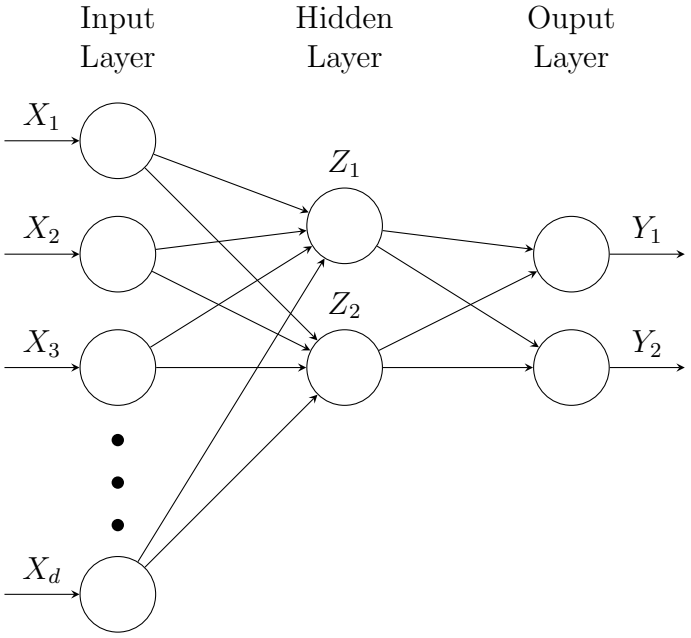
(d) (3 points) For what range of values of y is $Q^*(3, S) \geq Q^*(3, D)$?

Solution:

$$
\begin{aligned}
Q^*(3, S) &\geq Q^*(3, D) \\
\mathbb{P}(G|3, S) &\geq \mathbb{P}(4|3, D)\mathbb{P}(G|4, S) \\
\frac{1}{2} &\geq \frac{2}{3}y \\
\frac{3}{4} &\geq y \geq 0
\end{aligned}
$$

2 Random, Engineered, or Learned Features

In this problem, we compare three methods for choosing features when the feature representation is smaller than the input space. The input $X = (X_1, \ldots, X_d)$ is a d-dimensional vector, which we summarize using $Z = (Z_1, Z_2)$, a 2-dimensional vector. We then model the 2-dimensional output Y as a linear function $Y = f(Z)$ of Z. The relationships in this model are depicted in the diagram below:



We observe n pairs $(X^{(i)}, Y^{(i)})_{i=1}^n$ and test different methods for summarizing the input $X^{(i)}$ using $Z^{(i)} = g(X^{(i)})$. In each case, given the derived features $(Z^{(i)})_{i=1}^n$ we model the output $(Y^{(i)})_{i=1}^n$ as a linear function $Y^{(i)} = f(Z^{(i)})$ of the features. The data is contained in two pickle files: an $n \times d$ array `X.pkl` and a $n \times 2$ array `y.pkl`. We denote the $n \times d$ array of input vectors in bold $\mathbf{X}$, where the i^{th} row of $\mathbf{X}$ is $X^{(i)}$, and similarly we use bold notation $\mathbf{Y}$ and $\mathbf{Z}$ for the $n \times 2$ arrays of outputs and features, respectively.

(a) (4 points) **Random features**: First sample $W_1, W_2 \sim \mathcal{N}(0, I_d)$, and then normalize these to be unit vectors by defining $v_k = \frac{W_k}{\|W_k\|_2}$ for $k \in \{1, 2\}$. Construct the features with the dot product

$$Z_k^{(i)} = v_k^\mathsf{T} X^{(i)}.$$

Then, estimate the coefficients β in a linear model $Y = \beta^\mathsf{T} Z + \varepsilon$ using ordinary least squares. Since Y is a 2-dimensional response vector and Z is a 2-dimensional vector of covariates, the coefficients β in this linear model are a 2×2 matrix, and the ordinary least squares estimate of β can be written as

$$\hat{\beta} = (\mathbf{Z}^\mathsf{T}\mathbf{Z})^{-1}\mathbf{Z}^\mathsf{T}\mathbf{Y}$$

Compute the average training error for each response $r \in \{1, 2\}$

$$RSS(r) = \sum_{i=1}^{n} (Y_r^{(i)} - (\hat{\beta}^\mathsf{T} Z^{(i)})_r)^2.$$

Finally, *repeat this process* $M = 200$ times to obtain different values $RSS_1(r), \ldots, RSS_M(r)$ and report the average value $\frac{1}{M} \sum_{m=1}^{M} RSS_m(r)$ for $r \in \{1, 2\}$. *Hint:* Once you generate the random features Z, you can fit the linear model using

```
import statsmodels.api as sm

yhat = sm.OLS(y, Z).fit().predict()
RSS = sum((yhat-y)**2)
```

Note that you will need to repeat this in a for loop $M = 200$ times, using the same values for $\mathbf{y}$ but different values of $\mathbf{Z}$ due to the randomness in the vectors v_k.

Solution: The average RSS for the two responses was $[4144, 5616]$ (answers for the second response will vary by a few hundred).

```
n, d = X.shape

M, tot = 200, 0
for m in range(M):
    v = np.random.randn(d, 2)
    v /= np.linalg.norm(v, axis=0)
    Z = X@v

    yhat = sm.OLS(y, Z).fit().predict()
    tot += sum((yhat-y)**2)
print(tot/M)
```

(b) (5 points) **Engineered features**: Similar to part (a), we will construct the derived features Z_1, Z_2 by projecting the input features along two directions:

$$Z_k^{(i)} = v_k^\mathsf{T} X^{(i)},$$

but rather than choosing the directions v_k at random, we will choose them using to capture as much variation in the input $\mathbf{X}$ as possible. This can be achieved through *principal component analysis*, and the optimal directions v_1 and v_2 may be computed as follows:

```
import numpy as np
```

```
_, U = np.linalg.eigh(X.T @ X)
v1 = U[-1]
v2 = U[-2]
```

Compute the features $Z_k^{(i)} = v_k^\mathsf{T} X^{(i)}$ using v_k as above, and fit a linear model as in part (a). Report the $RSS(r)$ for this approach for both responses $r \in \{1, 2\}$. Compare each answer to part (a). Explain why the engineered features can be worse than random.

Solution: The RSS for the two responses was $[4173, 3696]$. Note that the RSS for the first output is slightly worse than in part (a), which uses the same number of linear features chosen at random. This indicates that the features we choose via PCA must be poorly correlated with the response $Y_1^{(i)}$. Indeed, PCA selects directions v_k *only* with reference to the covariate matrix $\mathbf{X}$. Suppose, for instance

$$Y_1^{(i)} = v_d^\mathsf{T} X^{(i)} + \varepsilon^{(i)},$$

where v_d is the *smallest* principal component. Then $Y_1^{(i)}$ will be nearly uncorrelated with $Z_k^{(i)}$ since v_k (for $k = 1, 2$) and v_d are orthogonal.

```
_, U = np.linalg.eigh(X.T @ X)

v = U[-2:].T

Z = X@v

yhat = sm.OLS(y, Z).fit().predict()
print(sum((yhat-y)**2))
```

(c) (6 points) **Learned features**: Fit a two-layer neural network with linear layers using `PyTorch`. As in the diagram above, the features Z define the hidden layer and so the values of Z will be estimated in the learning algorithm. For the loss function, use `torch.nn.MSELoss()` and train the model using 10^4 iterations of stochastic gradient descent with a learning rate of 0.01. Report the final errors

$$RSS(r) = \sum_{i=1}^{n} (Y_r^{(i)} - \hat{Y}_r^{(i)})^2,$$

where $\hat{Y}_r^{(i)}$ are the model predictions based on $X^{(i)}$. *Hint:* you may find the example neural network code from lecture 24 to be a helpful starting place.

Why do the features from part (b)—which are designed to explain as much of the variance in the input X as possible—not perform as well as the learned features from

part (c)? What sorts of functions can your two-layer linear neural net from part (c) express?

Solution: The RSS for the two responses was $[3926, 3671]$. This neural net composes two linear functions, so overall it can only express linear functions. Note, however, that the neural net chooses the features Z_k in order to minimize the total RSS, so it's no surprise that it outperforms part (b).

```
Xt = torch.tensor(X, dtype=torch.float32)
yt = torch.tensor(y, dtype=torch.float32)
layer1 = torch.nn.Linear(d, 2)
layer2 = torch.nn.Linear(2, 2)
model = lambda x : layer2(layer1(x))
loss_fn = torch.nn.MSELoss()
optimizer = torch.optim.SGD(
                itertools.chain(
                        layer1.parameters(),
                        layer2.parameters()), lr=.01)

for epoch in range(10000):
    optimizer.zero_grad()
    outputs = model(Xt)
    loss = loss_fn(outputs, yt)
    loss.backward()
    optimizer.step()
print(sum((y - outputs.detach().numpy())**2))
```