Problem Set 6

Due Friday Nov. 3, 10 am

Comments

- This covers material in Units 8 and 9.
- It's due at 10 am (Pacific) on November 3, both submitted as a PDF to Gradescope as well as committed to your GitHub repository.
- Please see PS1 for formatting and attribution requirements.
- Note that is is fine to hand-write solutions to the the non-coding questions, but make sure your writing is neat and insert any hand-written parts in order into your final submission.

Problems

1. In class and in the Unit 8 notes, I mentioned that integers as large as 2^{53} can be stored exactly in the double precision floating point representation. (Note that for this problem, you don't need to write out `e` in base 2; you can use base 10).

    a. Demonstrate how the integers 1, 2, 3, …,$2^{53}-2$, $2^{53}-1$ can be stored exactly in the $(-1)^S \times 1.d \times 2^{e-1023}$ format where d is represented as 52 bits. I'm not expecting anything particularly formal - just write out for a few numbers and show the pattern.

    b. Then show that 2^{53} and $2^{53}+2$ can be represented exactly but $2^{53}+1$ cannot, so the spacing of numbers of this magnitude is 2. Finally show that for numbers starting with 2^{54} that the spacing between integers that can be represented exactly is 4. Then confirm that what you've shown is consistent with the result of executing $2.0^{53}-1$, 2.0^{53}, and $2.0^{53}+1$ in Python (you can use base Python floats or numpy).

    c. Finally, calculate the relative error in representing numbers of magnitude 2^{53} in base 10. (This should, of course, look very familiar, and should be the same as the relative error for numbers of magnitude 2^{54} or any other magnitude…)

2. If we want to estimate a derivative of a function on a computer (often because it is hard to calculate the derivative analytically), a standard way to approximate this is to compute:

$$f'(x) \approx \frac{f(x+\epsilon)-f(x)}{\epsilon}$$

    for some small ϵ. Since the limit of the right-hand side of the expression as $\epsilon \to 0$ is exactly $f'(x)$ by the definition of the derivative, we presumably want to use ϵ very small from the perspective of using a difference to approximate the derivative.

a. Considering the numerator, if we try to do this on a computer, in what ways (there are more than one) do the limitations of arithmetic on a computer affect our choice of ϵ? (Note that I am ignoring the denominator because that just scales the magnitude of the result and itself has 16 digits of accuracy.)
b. Write a Python function that calculates the approximation and explore how the error in the estimated derivative for some x varies as a function of ϵ for a (non-linear) function that you choose such that you can calculate the derivative analytically (so that you know the truth).

3. Consider multiclass logistic regression, where you have quantities like this:

$$p_j = \text{Prob}(y = j) = \frac{\exp(x\beta_j)}{\sum_{k=1}^{K} \exp(x\beta_k)} = \frac{\exp(z_j)}{\sum_{k=1}^{K} \exp(z_k)}$$

for $z_k = x\beta_k$. Here p_j is the probability that the observation, y, is in class j.

a. What will happen if the z values are very large in magnitude (either positive or negative)?
b. How can we reexpress the equation so as to be able to do the calculation even when either of those situations occurs?

4. Let's consider importance sampling and explore the need to have the sampling density have heavier tails than the density of interest. Assume that we want to estimate $\phi = EX$ and $\phi = E(X^2)$ with respect to a density, f. We'll make use of the Pareto distribution, which has the pdf $p(x) = \frac{\beta\alpha^\beta}{x^{\beta+1}}$ for $\alpha < x < \infty$, $\alpha > 0$, $\beta > 0$. The mean is $\frac{\beta\alpha}{\beta-1}$ for $\beta > 1$ and non-existent for $\beta \leq 1$ and the variance is $\frac{\beta\alpha^2}{(\beta-1)^2(\beta-2)}$ for $\beta > 2$ and non-existent otherwise.

a. Does the tail of the Pareto decay more quickly or more slowly than that of an exponential distribution?
b. Suppose f is an exponential density with parameter value equal to 1, shifted by two to the right so that $f(x) = 0$ for $x < 2$. Pretend that you can't sample from f and use importance sampling where our sampling density, g, is a Pareto distribution with $\alpha = 2$ and $\beta = 3$. Use $m = 10000$ to estimate EX and $E(X^2)$ and compare to the known expectations for the shifted exponential. Recall that $\text{Var}(\hat{\phi}) \propto \text{Var}(h(X)f(X)/g(X))$. Create histograms of $h(x)f(x)/g(x)$ and of the weights $f(x)/g(x)$ to get an idea for whether $\text{Var}(\hat{\phi})$ is large. Note if there are any extreme weights that would have a very strong influence on $\hat{\phi}$.
c. Now suppose f is the Pareto distribution described above and pretend you can't sample from f and use importance sampling where our sampling density, g, is the exponential described above. Respond to the same questions as for part (b), comparing to the known values for the Pareto.

5. Extra credit: This problem explores the smallest positive number that base Python or numpy can represent and how numbers just larger than the smallest positive number that can be represented.

a. By trial and error, find the base 10 representation of the smallest positive number that can be represented in Python. Hint: it's rather smaller than 1×10^{-308}.

b. Explain how it can be that we can store a number smaller than 1×2^{-1022}, which is the value of the smallest positive number that we discussed in class. Start by looking at the bit-wise representation of 1×2^{-1022}. What happens if you then figure out the natural representation of 1×2^{-1023}? You should see that what you get is actually a very "well-known" number

that is not equal to 1×2^{-1023}. Given the actual bit-wise representation of 1×2^{-1023}, show the progression of numbers smaller than that that can be represented exactly and show the smallest number that can be represented in Python written in both base 2 and base 10.

Hint: you'll be working with numbers that are not normalized (i.e., denormalized); numbers that do not have 1 as the fixed number before the radix point in the floating point representation we discussed in Unit 8.