Problem Set 6

Due Wed. November 6, 10 am

Comments

- This covers material in Units 8 and 9.
- It's due at 10 am (Pacific) on November 6, 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 we said that in the exponent of the 8 byte floating point representation, $e \in \{0, ..., 2047\}$, since e is represented with 11 bytes ($2^{11} = 2048$). So that would suggest that the largest and smallest magnitude numbers (ignoring the influence of the 52 bit in the mantissa) are 2^{1024} and 2^{-1023}.

    a. However, 2^{1024} overflows. Create a numpy `float64` number larger than 10^{308} that overflows and is represented as `inf`. What is its bit-wise representation? Why does it now make sense that we can't work with 2^{1024} as a regular number?

    Note that there is still a bit of a mystery here in that it doesn't appear that the sequence 0 followed by 63 1s (or simply 64 1s) is used in any way, even though those would be a natural candidate to represent `inf` (and `-inf`). Extra credit for investigating and figuring out why that is (it might be difficult to determine and I don't know the answer myself).

    b. What is the bitwise repr of 2^{-1022}? Given that, what would the bit-wise repr of 2^{-1023} be (work this out conceptually, not by trying to use 2^{-1023} in Python)? What number does that bitwise representation actually represent?

    I asked that you not try to use 2^{-1023} in Python. Doing that and exploring what is going on is part (c).

    c. Extra credit: 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}. 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 saw above. Start by looking at the bit-wise representation of 1×2^{-1023} and see it is not the same as what you worked out in part (b).

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.

2. Consider the following estimates of the variance of a set of numbers. The results depend on whether the magnitude of the numbers is large or small. You can assume that for a vector `w`, var(w) is calculated as $\sum_{i=1}^{n}(w_i - \bar{w})^2/(n-1)$.

```
import numpy as np
rng = np.random.default_rng(seed = 1)
def dg(x, form = '.20f'):
  print(format(x, form))

z = rng.normal(size = 100)
x = z + 1e12
## Calculate the empirical variances
dg(np.var(z))
dg(np.var(x))
```

```
0.72514887009499828796
0.72514631554484365594
```

Explain why these two estimates agree to only a small number of decimal places and which of the two is the more accurate answer, when mathematically the variance of `z` and the variance of `x` are exactly the same (since `x` is just the addition of a constant to `z`). How many digits of accuracy do you expect in the less accurate of the two?

3. Consider the following, in which we run into problems when trying to calculate on a computer. Suppose I want to calculate a predictive density for new data (e.g., in a model comparison in a Bayesian context):

$$f(y^*|y,x) = \int f(y^*|y,x,\theta)\pi(\theta|y,x)d\theta = E_{\theta|y,x}f(y^*|y,x,\theta).$$

Here $\pi(\theta|y,x)$ is the posterior distribution (the distribution of the parameter, θ, given the data, y, and predictors, x). All of θ, y, and x will generally be vectors.

If we have a set of samples for θ from the posterior distribution, $\theta_j \sim \pi(\theta|y,x)$, $j = 1, \ldots, m$, we can estimate that quantity for a vector of conditionally IID observations using a Monte Carlo estimate of the expectation:

$$f(y^*|y,x) \approx \frac{1}{m}\sum_{j=1}^{m}\prod_{i=1}^{n} f(y_i^*|y,x,\theta_j).$$

a. Explain why I should calculate the product in the equation above on the log scale. What is likely to happen if I just try to calculate it directly?

b. Here's a re-expression, using the log scale for the inner quantity,

$$\frac{1}{m}\sum_{j=1}^{m}\exp\sum_{i=1}^{n}\log f(y_i^*|y,x,\theta_j),$$

which can be re-expressed as

$$\frac{1}{m}\sum_{j=1}^{m}\exp(v_j)$$

where

$$v_j=\sum_{i=1}^{n}\log f(y_i^*|y,x,\theta_j).$$

What is likely to happen when I try to exponentiate v_j?

c. Consider the log predictive density,

$$\log f(y^*|y,x)\approx\log\left(\frac{1}{m}\sum_{j=1}^{m}\exp(v_j)\right).$$

Figure out how you could calculate this log predictive density without running into the issues discussed in parts (a) and (b).

Hint: recall that with the logistic regression example in class, we scaled the problematic expression to remove the numerical problem. Here you can do something similar with the $\exp(v_j)$ terms, though at the end of the day you'll only be able to calculate the log of the predictive density and not the predictive density itself.

4. Experimenting with importance sampling.

a. Use importance sampling to estimate the mean (i.e., $\phi = E_f X$) of a truncated t distribution with 3 degrees of freedom, truncated such that $X < -4$. Have your sampling density be a normal distribution centered at -4 and then truncated so you only sample values less than -4 (this is called a half-normal distribution). You should be able to do this without discarding any samples (how?). Use $m = 10000$ samples. Create histograms of the weights $f(x)/g(x)$ and the summand $h(x)f(x)/g(x)$ to get a sense for whether $\mathrm{Var}(\hat{\phi})$ is large. Note if there are any extreme weights that would have a very strong influence on $\hat{\phi}$. Estimate $\mathrm{Var}(\hat{\phi})$. Hint: remember that your $f(x)$ needs to be appropriately normalized or you need to adjust the weights per the class notes. For comparison, based on using numerical integration, which is feasible in this simple one-dimensional case but increasingly infeasible in higher dimensions, the mean is -6.216.

b. Now use importance sampling to estimate the mean of the same truncated t distribution with 3 degrees of freedom, truncated such that $X < -4$, but have your sampling density be a t distribution, with 1 degree of freedom (not 3), centered at -4 and truncated so you only sample values less than -4. Again you shouldn't have to discard any samples. Respond to the same questions as above in part (a). In addition, compute a 95% (simulation) uncertainty interval for your estimate, using the Monte Carlo simulation error, $\sqrt{\widehat{\mathrm{Var}}(\hat{\phi})}$.