Stats 598z: Homework 2

HW2 due before class on Feb 8

Important:

`R` code, tables and figures should be part of a single .pdf or .html files from `R Markdown` and `knitr`. See the class reading lists for a short tutorial.

If you collaborated with anyone else, mention their names and the nature of the collaboration

1 Problem 1: Rejection sampling from a truncated Gaussian [55pts]

The `rnorm()` function samples from a Gaussian distribution.

1. Draw 10000 samples from a Gaussian with mean 1 and standard deviation 2. Plot the histogram using `hist()` [5pts]

We now want to sample from a *truncated Gaussian*: a distribution that has zero probability outside some interval `(lower, upper)`, and is like the Gaussian within that interval. A simple way to do this is by *rejection sampling*: keep sampling from a Gaussian distribution until you get a sample in `(lower, upper)`, and return that sample. You can do this with a `while` loop. To get more than one sample, enclose the `while` loop in a `for` loop.

2. Write a function `trunc_norm()` to do this. This should accept five inputs: the mean and standard deviation of the Gaussian, the lower and upper truncation limits and the number of samples from the truncated Gaussian (call the last `num_samp`). Use appropriate defaults for the arguments (`lower` and `upper` should default to `-Inf` and `Inf`). [15pts]

3. Plot the histogram of 10000 samples from a Gaussian with mean 1, standard deviation 2, truncated to $(0, 5)$. [5pts]

4. All the looping in the previous function can be quite inefficient, and we will try to introduce some vectorization. Instead of generating one sample at a time from `rnorm()`, draw `num_samp` samples. If all these samples lie in `(lower, upper)`, return them and we're done. If not, keep the samples that *do* lie in the interval, and discard the rest. Keep repeating this process until you have accumulated `num_sampl` samples. Note, that this only needs a while loop which iterates a much smaller number of times. Call the overall function `trunc_norm_vec()` [15pts]

5. Plot the histogram of 10000 samples from a Gaussian with mean 1, standard deviation 2, truncated to $(0, 5)$. It should be nearly identical to the previous one. [5pts]

6. Use `system.time()` to compare the efficiency of both functions. Use more samples if you must. [10pts]

2 Problem 2: Calculating entropy [55pts]

We saw how to calculate the entropy of a probability distribution in class. We will now do this for 10000 probability distributions (each of which are 6-dimensional, so you can think of them as 10000 different loaded dice). For efficiency, we want our code to be vectorized. Recall for any distribution the entropy is $-\sum_{i=1}^{6} p \log(p)$.

1. We will first generate the 10000 probabilities and store them in a 10000×6 matrix. We do this in three steps:

   (a) Sample 60000 variables from a Gaussian with mean 1 and standard deviation 1 and store them in a 10000×6 matrix. [5pts]

   (b) Set all elements that are negative to 0. [5pts]

   (c) Rescale the rows of the matrix so that they add up to 1 (as in homework 1). [5pts]

   The result is 10000 probability distributions (that are nonnegative and add up to 1).

2. Directly applying the formula for entropy will give you `NaN`'s, since some components are 0. One approach to fix this is to loop through each row, and use the method described in class. This is inefficient. Instead write down vectorized code that does this without any `for` loops. (Hint: you can try subsetting but you might run into trouble if you're not careful: it's probably simplest to use `ifelse()`). Your code should produce no warnings. [20pts]

3. Plot a histogram of the resulting 10000 calculated entropies. [4pts]

4. What is the largest entropy you observe? What is the corresponding probability vector? [4pts]

5. Make a guess about the probability distribution that has the theoretical maximum entropy. [2pts]