1 Bayesian inference with dice

Imagine a classic scenario in statistical inference: you are to guess how many six-sided dice are in a bag, with the only clue at some point someone randomly rolled these n dice which produced a sum of 51. The following series of exercises breaks this problem into smaller programming tasks in Python that can help you discover the most likely number of dice. Whilst doing so you will learn about the limits of the programs you wrote, and how they can be overcome.

The idea behind inference

Say you suspected that there were m-dice in the bag. If you also knew how probable it was that m-dice produce a roll-sum of 51, then this gives you a pretty good estimate of the likelihood that there were in fact m-dice in the bag. This likelihood will be smaller for certain m and larger for others. Using this likelihood estimate, you should then be able to systematically check for which m this roll-sum of 51 is most likely.

Formally, this type of checking is described by Bayesian inference:

$$P(\text{sum} = 51|m\text{-dice}) \times P(m\text{-dice}) = P(m\text{-dice}|\text{sum} = 51) \times P(\text{sum} = 51)\ . \tag{1}$$

Never mind if this equation doesn't make too much sense now. What is important is we are after $P(m\text{-dice}|\text{sum} = 51)$, which reads "given a roll-sum of 51, what is the probability that it arose from m-dice". However, $P(m\text{-dice}|\text{sum} = 51)$ is not always easy to obtain, like in this case. Fortunately, computing $P(\text{sum} = 51|m\text{-dice})$, or "the probability of getting a sum of 51 given m-dice" is straightforward.

The goal

Hence, the key goal of these exercises is to determine the probability of obtaining a roll-sum of 51 given m-dice.

1.1 Enumerating all the roll-sums given m dice

We begin with a brute-force approach to find all the possible die-rolls that m-dice generate.

1.1.1 Counting outcomes.

Assuming each six-sided die's roll is independent of others, how many possible outcomes are there for 2-dice? How many outcomes for m dice?

1.1.2 Writing pseudocode.

Suppose you are given a list of all possible 2-die rolls: $r_2 = [[1,1],[1,2],[1,3],\ldots]$. Write a pseudocode that appends a third die roll to list. *Hint: is the third die's roll influenced by the rolls of the previous two dice?*

1.1.3 Implementing in Python.

Using only standard Python functions (i.e. no `import <module>`), write a list comprehension in Python that appends a 2-die roll list (r_2) to give all the possible die-rolls for 3 dice (r_3).

1.1.4 Scaling up.

Using a `for` loop in Python, repeat the process in the last part until you get all the die-rolls possible with 6-dice. How many unique die rolls are counted in this list and does this number correspond to the answer you found in 1.1.1?

1.1.5 Finding roll-sums.

Modify the output from 1.1.4 to find all the possible roll-sums with 6-dice. How many unique roll-sums are there?

1.1.6 Finding probabilities.

Given your output in 1.1.5, can you count the fraction of times each roll-sum occurs for the 6-dice situation?

Hints:

- *You will have to find how often each roll-sum occurs in your list of roll-sums. To do this, you'll need to know the unique* `set` *of roll-sums in this list.*
- *Once you know the unique roll-sums in this list, you'll have to count their occurrences in this list. You could either use the* `count` *function for lists. We would recommend* **AGAINST** *this because you will have to visit the entire list as many times as there are unique roll-sums. Instead, use a* `for` *loop that loops over this roll-sum list once, then incrementing the occurrence of each roll-sum to a running counter implemented as a Python dictionary or list.*
- *Don't forget to normalize the occurrences so they become a fraction of the total number of occurrences.*

1.1.7 The limits of scaling.

Applying your program in 1.1.5, you should only be able to obtain the roll-sums of about 9 dice on most current (circa 2016) computers. Will listing outcomes only up to 9-dice be sufficient to solve our 51 roll-sum problem? Why is it difficult for this method to go beyond 9 dice? *Hint: What is the minimum and maximum number of dice if the sum of their rolls is 51?*

1.2 A slightly better way of counting roll-sums given m dice.

It would appear that the computer might not have enough memory[1] to hold all the die-rolls with m-dice. However, our goal is to only keep track of their roll-sums rather than the actual sequence of rolls that produced each sum. For instance, a die-roll of $[1, 1, 2, 1, 1, 1]$ requires roughly six times more memory to store than their sum of 7.

1.2.1 Keeping only sums.

Can you modify your Python program in 1.1.5 to only keep sums of die-rolls instead of the die-roll sequences? For how many more dice can your program compute roll-sums, and what is limiting you now?

1.3 Monte Carlo method.

Recall the goal is to obtain a 'most-likely' guess for the number of dice in the bag given their roll-sum of 51. What we have done so far is to enumerate all possible roll-sums for m-dice, which is fairly time consuming since the number of roll-sums grows rapidly with m (see 1.1.1).

On the other hand, we can produce a pretty good estimate if we can enumerate most but not all of these roll-sums. One way of getting most of these roll-sums is to simulate very many random m-die rolls. The hope is to obtain sufficient statistics with *enough* random, simulated die-rolls.

1.3.1 The random Python module.

Using the `randint` function in Python's `random` module, write a single-line of Python code that generates a random m-dice roll. Convert this result into a roll-sum. *Hint: you might be tempted to generate random roll-sums directly instead of generating the sequence of die-rolls. Don't. The distribution of roll-sums is not as straightforward as you might think.*

[1]To be precise, there is insufficient Random Access Memory (RAM) to hold large sequences of die throws. What can't be written into the RAM is then written temporarily to your computer's hard disk, which takes much longer to read and write than RAM does.

1.3.2 The random Python module.

Define a Python function `monte_carlo`(m, N) that calls your code in 1.3.1 N times through a `for` loop. For now, set $N = 10,000$. Then repeat what you did in 1.1.6 to convert these roll-sums into probabilities. How do these probabilities compare with those from the complete enumeration you did for the 6-dice case in 1.1.6?

1.3.3 Using your Monte Carlo simulation.

Using your `monte_carlo`(m, N) function in 1.3.2, find the probability that you will obtain a roll-sum of 51 if there were 10 dice in the bag. Name this new Python function `likelihood_monte_carlo`$(m, N, s = 51)$.

Try different values of $N \geq 1000$. How much does N matter?

1.4 Inference basics. Hang on to your socks!

You might have noticed that the `likelihood_monte_carlo`$(m, N, s = 51)$ function in 1.3.3 actually estimates $P(\text{sum} = 51|m\text{-dice})$ in equation (1)! Re-writing that equation so that what we want is on the left hand side:

$$P(m\text{-dice}|\text{sum} = 51) = \frac{P(\text{sum} = 51|m\text{-dice}) \times P(m\text{-dice})}{P(\text{sum} = 51)} . \tag{2}$$

The term on the left-hand side is called the *posterior probability.* Now, what's left is to figure out $P(\text{sum} = 51)$ and $P(m\text{-dice})$.

1.4.1 The art of picking a prior distribution.

The probability $P(m\text{-dice})$ is referred to in Bayesian inference as a *prior distribution.* It is what we believe, *a priori*, the probability that there could have been m dice in the bag, and should not be influenced by the knowledge that a roll-sum of these dice was 51.

Guessing the appropriate prior, I'm afraid, is an art. The general rule is to assume the 'most ignorant' prior distribution so that we least bias our inferences. Assuming we were told ahead of time that there cannot be more than 100 dice in the bag, what then is this 'most ignorant' and least biased $P(m\text{-dice})$? *Hint: you are 'ignorant' when you don't have any basis to favor m dice over say n dice, as long as* $1 \leq n, m \leq 100$. *This is sometimes called a uniform prior.*

1.4.2 Joint probabilities.

To move forward, we need to take a short excursion and learn about *joint probabilities*. One way of writing the Bayes's rule in equation (1) is

$$\begin{aligned} P(\text{sum} = 51|m\text{-dice}) \times P(m\text{-dice}) &= P(m\text{-dice}|\text{sum} = 51) \times P(\text{sum} = 51) \\ &= P(m\text{-dice}, \text{sum} = 51) \ . \end{aligned} \tag{3}$$

This last equation introduces a new term, $P(m\text{-dice}, \text{sum} = 51)$, a joint probability. In words, this is the probability that there is m-dice in the bag and their roll-sum is 51. "How would one get this joint probability" you ask? Here's how:

1. Prepare a very, very large number of bags of dice such that the number of dice in each bag followed your prior distribution $P(m\text{-dice})$. Also prepare a very large sheet of paper that keeps track of the occurrences of roll-sums and number of dice.

2. Now, roll the dice in each bag to obtain their roll-sum.

3. Record the number of dice in this bag, m, and their roll-sum into the very large sheet of paper mentioned in the first step.

4. Repeat this for all your bags of dice.

5. Once you are done, convert the occurrences on your sheet of paper into probabilities. When you are done, you should get the total probability of any m-dice plus roll-sum combination to equal one. This is because if you have prepared enough bags of dice and rolled them, you will encounter every possible combination of m and roll-sum.

You don't have to do anything for this question. Just be in awe about how challenging it is to perform this experiment to determine the joint probability! In the next part we will find a way of working around it using equation (3).

1.4.3 Calculating $P(\text{sum} = 51)$ using joint probabilities.

What is the probability of $P(\text{sum} = 51)$? This seems like a tough thing to know or calculate. However, if you had the joint probability in question 1.4.2, you could formally obtain

$$P(\text{sum} = 51) = \sum_m P(m\text{-dice}, \text{sum} = 51) \ . \tag{4}$$

Think about this equation for a bit. In simple terms convince someone else, or your alter ego, of it validity.

1.4.4 Calculating $P(\mathbf{sum} = 51)$, part two.

From question 1.4.2, we recalled the difficulty of obtaining the joint distribution. However, equation (3) tells us that we might be able to compute this using the `monte_carlo`(m, N) Python function in 1.3.3, which actually estimates $P(\text{sum} = 51|m\text{-dice})$. Here's what we mean:

$$P(\text{sum} = 51) = \sum_m P(m\text{-dice}, \text{sum} = 51)$$

substituting with equation (3)

$$\begin{aligned} &= \sum_m P(\text{sum} = 51|m\text{-dice}) \times P(m\text{-dice}) \\ &\approx \sum_m \texttt{likelihood\_monte\_carlo}(m, N, s = 51) \times P(m\text{-dice}) \ . \end{aligned} \tag{5}$$

With this new-found knowledge, go forth and compute $P(\text{sum} = 51)$. *Hint: use a Python* `for` *loop to sum over m, all possible dice numbers in your prior distribution $P(m$-dice$)$. For more see what you wrote in1.4.1. Note that $P(sum = 51)$ might not be accurate if you've used simulated each bag of dice too few times (i.e. N is too small). So gradually increase N until $P(sum = 51)$ stops fluctuating.*

1.4.5 All together now.

Recall that we are trying to solve the inference problem in equation (2). Put simply, we would like to know the probability that m-dice can generate a roll-sum of 51, or equivalently $P(m\text{-dice}|\text{sum} = 51)$. You might also recall that this is called the *posterior probability*, which is a fancy way of saying 'taking the evidence into consideration'.

If you have gotten this far, you have secretly already numerically computed all the terms needed for this posterior. In fact, you might have realized that your calculation in equation (5) is suspiciously close to that of equation (2).

You are on your own now, young Padawan. Compute the posterior $P(m\text{-dice}|\text{sum} = 51)$ and find for which number of dice m is the roll-sum 51 most likely.

1.4.6 A faster way to get the solution?

You might have realized from your solution to 1.4.5 that the most likely number of dice is pretty close to 51/3.5. Is this a coincidence? Can you explain this simply?

1.5 A closed-form solution.

More soon...