Juliette Woodrow
CS 109

Section 6

1 Beta Sum Warmup

What is the distribution of the sum of 100 IID Betas? Let X be the sum:

$$X = \sum_{i=1}^{100} X_i \quad \text{where each } X_i \sim \text{Beta}(a = 3, b = 4)$$

Note the expectation and variance of a Beta:

$$E[X_i] = \frac{a}{a+b} \qquad \text{Var}(X_i) = \frac{ab}{(a+b)^2(a+b+1)} \qquad \text{Where } X_i \sim \text{Beta}(a, b)$$

By the Central Limit Theorem, the sum of equally weighted IID random variables will be Normally distributed. We calculate the expectation and variance of X_i using the Beta formulas:

$$\begin{aligned} E(X_i) &= \frac{a}{a+b} && \text{Expectation of a Beta} \\ &= \frac{3}{7} \approx 0.43 \\ \text{Var}(X_i) &= \frac{ab}{(a+b)^2(a+b+1)} && \text{Variance of a Beta} \\ &= \frac{3 \cdot 4}{(3+4)^2(3+4+1)} \\ &= \frac{12}{49 \cdot 8} \approx 0.03 \end{aligned}$$

$$\begin{aligned} X &\sim N(\mu = n \cdot E[X_i], \sigma^2 = n \cdot \text{Var}(X_i)) \\ X &\sim N(\mu = 100 * 0.43, \sigma^2 = 100 * 0.03) \\ X &\sim N(\mu = 43, \sigma^2 = 3) \end{aligned}$$

2 Timing Attack

In this problem we will see how to crack a password in linear time by measuring how long the password check takes to execute (see code below).

```
# An insecure string comparison
def does_password_match(guess, password):
    n_guess = len(guess)
    n_password = len(password)
    if n_guess != n_password:
        return False                    # 4 lines executed to get here
    for i in range(n_guess):
        if guess[i] != password[i]:
            return False                # 6 + 2i lines executed to get here
    return True                         # 5 + 2n lines executed to get here
```

Assume that our server takes T ms to execute any line in the code where $T \sim N(\mu = 5, \sigma^2 = 0.5)$ milliseconds. The amount of time taken to execute a line is always independent of other lines. On our site, all passwords only use lower case letters and are between 5 and 10 letters long, inclusive. A hacker is trying to crack the root password which is "gobayes" by carefully measuring how long the code takes to tell her that her guesses are incorrect.

a. What is the distribution for the time it takes to execute k lines of code?

Let T_k be the amount of time to execute k lines. $T_k = \sum_{i=1}^{k} X_i$ where X_i is the amount of time to execute line i. $X_i \sim N(\mu = 5, \sigma^2 = 0.5)$. Since T_k is the sum of k independent normals:

$$T_k \sim N(\mu = 5k, \sigma^2 = 0.5k)$$

b. First, the hacker needs to find the length of the password. What is the probability that the time taken to check a guess of correct length (when the server executes 6 lines) is longer than the time taken to check a guess of an incorrect length (when the server only executes 4 lines)? Assume the first letter of the guess does not match the password's first letter. Hint: $P(A > B) = P(A - B > 0)$.

Time to run 6 lines of code: $T_6 \sim N(\mu = 30, \sigma^2 = 3)$
Time to run 4 lines of code: $T_4 \sim N(\mu = 20, \sigma^2 = 2)$ Then we apply a linear transform to T_4 so we can subtract it from T_6 by "adding" two Normal RVs.

$$\begin{aligned} -T_4 &\sim N(\mu = -20, \sigma^2 = 2) \\ T_6 - T_4 &\sim N(\mu = 10, \sigma^2 = 5) \\ P(T_6 > T_4) &= P(T_6 - T_4 > 0) \\ &= 1 - F_{T_6 - T_4}(0) \\ &= 1 - \Phi\left(\frac{0 - 10}{\sqrt{5}}\right) \approx 1.0 \end{aligned}$$

c. (Optional) Now that our hacker knows the length of the password, to get the actual string, she will try to determine one letter at a time, starting with the first letter. To start, the hacker tries the string "aaaaaaa" and sees that it takes 27ms. Based on this timing, how much more probable is it that first character did not match (server executes 6 lines) than the first character did match (server executes 8 lines)? Assume that all letters in the alphabet are equally likely to be the first letter.

Let M be the event that the first letter matched. The problem's wording indicates that we are looking for a ratio between the probability of M^C to the probability of M, conditioned on observing a code execution time of 27ms. Let $T_?$ be the time it took to check the password, with the number of lines executed being unknown. To calculate this ratio, we can apply Bayes' Theorem:

$$\begin{aligned}\frac{P(M^C|T_? = 27)}{P(M|T_? = 27)} &= \frac{f(T_? = 27|M^C)P(M^C)}{f(T_? = 27|M)P(M)} \\ &= \frac{f(T_? = 27|M^C)\frac{25}{26}}{f(T_? = 27|M)\frac{1}{26}} \\ &= 25 \cdot \frac{f(T_? = 27|M^C)}{f(T_? = 27|M)}\end{aligned}$$

From here, we can reason about how $T_?$ is distributed if we know M or M^C. If M happened, then $T_? = T_8$; otherwise, $T_? = T_6$. So we can re-write the above as:

$$\begin{aligned}&= 25 \cdot \frac{f(T_6 = 27)}{f(T_8 = 27)} \\ &= 25 \cdot \frac{\frac{1}{\sqrt{6\pi}} e^{-\frac{(27-30)^2}{6}}}{\frac{1}{\sqrt{8\pi}} e^{-\frac{(27-40)^2}{8}}} \\ &= 25 \cdot \frac{\sqrt{8}}{\sqrt{6}} \cdot \frac{e^{-\frac{9}{6}}}{e^{-\frac{169}{8}}} \\ &\approx 9.6 \text{ million}\end{aligned}$$

d. (Optional) If it takes the hacker 6 guesses to find the length of the password, and 26 guesses per letter to crack the password string, how many attempts does she need to crack our password, "gobayes"? Yikes!

Since the password is length 7: $6 + 7 \cdot 26 = 188$.

3 Binary Tree

Consider the following function for constructing binary trees:

```
def random_binary_tree(p):
    """
    Returns a dictionary representing a random binary tree structure.
    The dictionary can have two keys, "left" and "right".
    """
    if random_bernoulli(p):   # returns true with probability p
        new_node = {}  # initialize one new node
        new_node["left"] = random_binary_tree(p)
        new_node["right"] = random_binary_tree(p)
        return new_node
    else:
        return None
```

The `if` branch is taken with probability p (and the `else` branch with probability $1 - p$). A tree with no nodes is represented by `None`; so a tree node with no left child has `None` for the `left` field (and the same for the right child).

(a) Let X be the number of nodes in a tree returned by `random_binary_tree(p)`. You can assume $0 < p < 0.5$. What is $E[X]$, in terms of p?

The number of nodes in the tree depends on whether or not the if statement is true or false. It is true with probability p and false with probability 1 - p. This suggests that in order to find $E[X]$, we need to define a background random variable, Y, corresponding to the result of `random_bernoulli(p)`, where $P(Y = 1) = p$ and $P(Y = 0) = 1 - p$. Then, we can apply the Law of Total Expectation:

$$E[X] = P(\text{if}) \cdot E[X \mid \text{if}] + P(\text{else})E[X \mid \text{else}]$$

$$E[X] = p \cdot E[X \mid \text{if}] + (1 - p)E[X \mid \text{else}]$$

$$E[X] = p \cdot E[X \mid Y = 1] + (1 - p)E[X \mid Y = 0]$$

$E[X \mid Y = 0] = 0$ because if the `else` statement is executed, we return a tree with no nodes.

Let X_1 and X_2 be number of nodes returned by the left and right calls to `random_binary_tree`. We can write $E[X \mid Y = 1]$ as $E[1 + X_1 + X_2]$ (parent + left child + right child) because that represents the total number of nodes that are added to the tree if the `if` statement is true. Then, because the recursive call is identical to the original function call, we know that $E[X_1] = E[X_2] = E[X]$, so

$$E[X \mid Y = 1] = E[1 + X_1 + X_2] = 1 + E[X] + E[X] = 1 + 2E[X]$$

Putting this all together:

$$
\begin{aligned}
E[X] &= p \cdot E[X \mid Y = 1] + (1-p)E[X \mid Y = 0] \\
&= p \cdot E[1 + X_1 + X_2] + (1-p) \cdot 0 \\
E[X] &= p \cdot (1 + 2E[X]]) \\
E[X] &= p + 2pE[X] \\
E[X] - 2pE[X] &= p && \text{(getting } E[X] \text{ alone on the LHS)} \\
(1-2p)E[X] &= p \\
E[X] &= \frac{p}{1-2p}
\end{aligned}
$$

(b) (Optional) Why did we need to assume that p is less than 0.5?

We needed $p < 0.5$ so that the expected tree size is finite.

From earlier,

$$E[X] = \frac{p}{1-2p}.$$

If $p \geq 0.5$, then $1 - 2p \leq 0$, so the denominator is zero or negative and $E[X]$ either blows up or is undefined.

Intuitively, each node creates $2p$ children on average. If $2p \geq 1$, the tree keeps growing instead of dying out, so the expected number of nodes becomes infinite. Therefore, $p < 0.5$.