Lecture 8: Some worked examples

February 20, 2019

1 The numbers racket

The 'numbers game' was an underground lottery that flourished in American cities before the advent of legal state lotteries. In a simplified version, it worked like this: Every day, bettors submitted sequences of 3 digits to the bookmaker, paying one dollar for the bet. The next day, a random 3-digit sequence would be drawn: Often this was done by looking in the horse racing news for the total daily receipts of the local race track, and using the lowest-order three digits of the result. (For instance, if the track took in 236,402 dollars in bets, then the winning sequence would be 402.) Bettors who correctly guessed this result were paid $600 on the dollar.

Let's look at the problem from the standpoint of a bookmaker who has 800 daily customers. There are 1000 different 3-digit sequences. So we model the day's bets as 800 random shots fired at a target where the probability of striking the target is 0.001.Let X denote the number of winning bets made in a day. Then X has a binomial distribution with $n = 800, p = 0.001$. For such values, the binomial distribution is well-approximated by a Poisson distribution with $\lambda = np = 0.8$.

Let Y be the net daily receipts of the bookmaker . She takes in $800 and pays out $\$600\dot{X}$, so

$$Y = 800 - 600X.$$

What is her average daily take? By linearity of expected value,

$$E(Y) = E(800) - 600 \cdot E(X).$$

You might wonder what $E(800)$ means! This the expected value of the random variable that has the value 800 on every outcome, so $E(800) = 800$. Since the

expected value of a Poisson-distributed random variable is λ, we get

$$E(Y) = 800 - 600 \cdot 0.8 = 800 - 480 = 320.$$

In other words, our bookmaker takes in on average $320 per day.

How likely is it that she will have to pay out more than she takes in? If there is only one winning bet, she nets $200, but if there are two or more winning bets, then her net receipts are negative. This occurs with probability

$$\begin{aligned} P(X \geq 2) &= 1 - P(X \leq 1) \\ &= 1 - e^{-\lambda}(1 + \lambda) \\ &= 1 - e^{-0.8} \cdot 1.8 \\ &\approx 0.19. \end{aligned}$$

Since this occurs one out of five days, she keeps a reserve of cash on hand to cover the case of a several winning bets. Let's say that the cash reserve is $3000. What is the probability that the bettors 'break the bank'—that is, that she cannot cover the winning bets? This happens if the total payout exceeds the $800 received from customers plus the reserve cash; that is, if

$$600 \cdot X > 3800.$$

This is equivalent to $X > 6.33$, but since X only takes on integer values, this is the same as $X \geq 7$. We then calculate

$$P(X \geq 7) = 1 - e^{-\lambda} \sum_{k=0}^{6} \frac{\lambda^k}{k!} \approx 2.07 \times 10^{-5}.$$

Our bookmaker takes bets daily for thirty years. What is the probability that in that span of time, the bank will be broken? This is a problem we've seen before. The complementary event is that the bank is *not* broken for 30×365 successive days, which is

$$(1 - 2.07 \times 10^{-5})^{365 \times 30} \approx e^{-2.07 \times 10^{-5} \times 365} = 0.797.$$

So there is a probability of 0.203 that the bank will be broken at least once during this long career. (This problem was inspired by my hearing a podcast about the book *The World According to Fannie Davis: My Mother's Life in the Detroit Numbers*, by Bridget M. Davis.)

2 Coupon Collector's Problem

Every month you send away for a decorative plate with a scene from one of the 50 states of the US. The plate you receive is selected uniformly and at random from the set of 50 states. How many purchases will be required, on average, to collect all 50? (And where can you put all those duplicate plates, while you're waiting to complete the collection?)

An equivalent problem (with 50 replaced by 6): Repeatedly roll a die. How many rolls, on average, until you roll all six numbers? A general statement of the problem: There are n balls, labeled with the integers 1 through n, in a jar. You sample repeatedly with replacement until you have seens all n balls. Let X be the number of samples you draw.

We assume that each time we sample, the probability of obtaining a particular item is $1/n$. Let X be the random variable denoting the number of times you have to sample until all n items are collected. The problem is solved by a clever decomposition of X into the sum of simpler random variables:

$$X = X_1 + \cdots + X_n,$$

where X_i is the number of draws required to collect the i^{th} distinct item after $i-1$ items have been collected. This is best explained with an example: Let $n = 8$, and suppose the sequence of values collected is

```
3 3 1 3 3 8 2 7 7 4 6 3 4 2 5
```

(This was actually generated when I ran the random simulation.) Then $X_1 = 1$, because the first item we collect (3) is new. We need to make two more draws to collect the next new item (1), so $X_2 = 2$. Then three more draws to collect the *next* new item (8), so $X_3 = 3$. You should check that $X_4 = 1$, $X_5 = 1$, $X_6 = 2$,$X_7 = 1$, and $X_8 = 4$. These add up to 15, which is the value of X for this outcome.

For each i, X_i is a geometric random variable with parameter $p = \frac{n-i+1}{n}$, so $E(X_i) = \frac{1}{p} = n \cdot \frac{1}{n-i+1}$, and thus

$$E(X) = E(X_1) + \cdots + E(X_n) = n \cdot \left(\frac{1}{n} + \frac{1}{n-1} + \cdots + \frac{1}{2} + 1 \right).$$

For $n = 50$, the sum

$$\sum_{i=1}^{50} \frac{1}{i}$$

has value about 4.5, so we need to collect about $50 \times 4.5 = 225$ plates on average to get all 50.

There is a simple asymptotic approximation for this sum. A basic calculus argument approximating the area under the graph of $y = 1/x$ by rectangles shows that

$$\ln n < \sum_{i=1}^{n} \frac{1}{i} < 1 + \ln n,$$

so asympotically, the expected value is $n \ln n$. For $n = 50$, this approximation gives about 196 samples for the expected value.

Observe how the sum rule made it relatively easy to calculate $E(X)$ without having to deal with the more difficult problem of finding the PMF of X and directly computing its expected value.

3 Average case analysis of quicksort

Let's recall how the quicksort algorithm works. We have an array of n items: We can suppose that these items are the integers $1, \ldots, n$ in some order. We begin by *partitioning* the array. This is done by treating the leftmost array entry as the 'pivot' value and comparing each of the remaining array elements once to the pivot and doing some rearranging so that all the entries less than the pivot value precede all the entries that are greater than the pivot value. For example

```
6 8 1 4 7 2 9 10 5 3
```

is transformed to

```
6 3 1 4 5 2 9 10 7 8
```

We then swap the pivot value 6 with the rightmost value less than the pivot (2) to get the partitioned array

```
2 3 1 4 5 | 6 | 9 10 7 8
```

The algorithm proceeds by applying this partitioning procedure recursively to the two subarrays to the left and to the right of 6.

You don't need to remember the details of how the partitioning step goes about rearranging the elements. The only things relevant for our purposes are *(a)* Each element gets compared to the pivot value exactly once, so the partitioning step applied to an array of n items does $n - 1$ comparisons; and *(b)* the number of

times elements are moved is smaller than the number of comparisons, and thus *the running time of the algorithm is dominated by the number of comparisons.*

The first partition always requires $n-1$ comparisons. The total number of comparisons make to complete the sorting varies considerably. If the array is partitioned into roughly two equal halves at each step, then the total number of generations (the depth of the recursion tree) is about $\log_2 n$. We do fewer than $n-1$ comparisons in each generation, so *in the best case* the running time is bounded above by a constant times $n \log n$. But equal partitions at each step is too much to hope for, and things can get pretty bad. For example, if the array is already sorted at the outset, then then after the first partition, the left subarray is empty, and the right subarray contains $n-1$ elements Thus we will do $n-2$ comparisons in the second generation, and likewise $n-3$ in the third, *etc.*, so the total number of comparisons is

$$1 + 2 + \cdots + (n-1) = \frac{n^2 - n}{2},$$

which is no better than slow sorting methods like selection sort.

So is quicksort quick? We will show that the *average* running time is proportional to $n \log n$. To do this, we use a probability model. The underlying probability space is the set of all $n!$ permutations of $\{1, \ldots, n\}$, with a uniform probability distribution. (One can ask whether the uniformity assumption is reasonable; more about this below.) The random variable X that we investigate is the number of comparisons quicksort makes in sorting a given permutation, and we ask for the value of $E(X)$—this represents the average running time.

Determining the exact distribution of X seems like a horribly complicated problem, but we can find $E(X)$ through another clever trick using the linearity of expectation. If $1 \leq i < j \leq n$, we let $X_{i,j}$ be the random variable whose value is the number of times i is compared to j during sorting. Thus

$$X = \sum_{1 \leq i < j \leq n} X_{i,j},$$

so

$$E(X) = \sum_{1 \leq i < j \leq n} E(X_{i,j}).$$

Observe that $X_{i,j}$ can only be 0 or 1: i and j will be compared if one of them is chosen as the pivot value for a subarray that contains the other element. For example if i is the first element of a subarray about to be partitioned, then i and j

will be compared only if j is an element of that subarray. After the partitioning, i will be in position i, and will not be compared again to any other element. Thus $X_{i,j}$ is a Bernoulli random variable, and $E(X_{i,j})$ is simply the probability that i and j are compared during sorting.

This probability is easy to compute. To illustrate the method, suppose we want to find the probability that 2 and 8 are compared. We ask which among the values 2,3,4,5,6,7,8 will first be chosen as a pivot value for a partition step. If it is 3,4,5,6 or 7, then the resulting partition step will put 2 and 8 in different parts of the partition, in which case 2 and 8 will never be compared during the sorting. If 2 is chosen before the others as a pivot, then 2,3,4,5,6,7,8 are all in the subarray to be partitioned, so 2 and 8 will be compared; and likewise is 8 is first chosen as a pivot value. Thus 2 and 8 will be compared if and only if the first member of $\{2, 3, 4, 5, 6, 7, 8\}$ to be chosen as a pivot is 2 or 8. Because all possible permutations are equally likely at the outset, the probability that any one of these 7 elements is chosen before the others is $\frac{1}{7}$. Thus $E(X_{2,8}) = \frac{2}{7}$.

In general, the same reasoning shows $E(X_{i,j}) = \frac{2}{j-i+1}$. We then have

$$\begin{aligned} E(X) &= \sum_{1 \le i < j \le n} \frac{2}{j-i+1} \\ &= 2\big((\frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n}) + (\frac{1}{2} + \cdots + \frac{1}{n-1}) + \cdots + \frac{1}{2}\big) \\ &< 2n \cdot \ln n \text{ (by the approximation we used above for the sum of reciprocals of positive integers} \end{aligned}$$

So quicksort really is quick. We can similarly get a lower bound of the form $cn \cdot \ln n$ for some constant c.

We've presented quicksort as a deterministic algorithm (one that makes no random choices) and computed the average case behavior assuming a certain probability distribution on the inputs. Alternatively you can use a probabilistic version of quicksort where the pivot values are chosen at random, and ask for the expected value of the running time on a fixed array. Very similar reasoning gives the same average-case result: For any starting array, the average running time of the algorithm is proportional to $n \log n$.

4 A bit more about expected value–product of random variables

If X, Y are *independent* random variables defined on same sample space, then

$$E(XY) = E(X)E(Y).$$

The proof is simple, but independence is a crucial requirement.

Example. Roll two dice, and let X_i, $i = 1, 2$, be the value displayed on the i^{th} die. X_1, X_2 are independent random variables, so $E(X_1X_2) = E(X_1) \cdot E(X_2) = 3.5^2 = 12.25$. Note that we do not have to go through the chore of finding the PMF of X_1X_2.

Example. Let X_1, X_2 be as above, and let $Y_1 = \max(X_1, X_2)$, $Y_2 = \min(Y_1, Y_2)$. Then $Y_1Y_2 = X_1X_2$, because it's still just the product of the two dice. So $E(Y_1Y_2) = 12.25$. You can verify directly by a computation with the PMF that $E(Y_1) = 4.47$. Since

$$E(Y_1)E(Y_2) = E(Y_1 + Y_2) = E(X_1 + X_2) = 7,$$

thus $E(Y_2) = 2.53$, and $E(Y_1)E(Y_2) = 11.31 \neq E(Y_1Y_2)$. Thus this product rule can fail if Y_1, Y_2 are not independent.

Example. An even simpler example of how the product rule can fail: $E(X_1^2)$ is just the mean of i^2 for $i = 1, \ldots, 6$, namely 15.167, which is not equal to $E(X_1)^2 = 12.25$. In general, if f is a function, it is NOT true that $E(f(X)) = f(E(X))$. However we do have

$$E(f(X)) = \sum_a f(a) \cdot P_X(a) = \sum_{s \in S} P(s) \cdot f(X(s)).$$