Lecture 28: Probability (fin)

Birthday paradox

What is the probability that 2 students in a class of n share the same birthday?

Probability that a *given* pair of people don't share the same birthday is given by $\frac{364}{365}$.

But there are $\binom{n}{2}$ pairs. Therefore, probability that none of these pairs share the same birthday is the same as saying that we toss a biased coin which shows heads with probability $\frac{364}{365}$, toss it $\binom{n}{2}$ times, and every single toss comes up heads. This happens with probability:

$$\left(\frac{364}{365}\right)^{\binom{n}{2}}$$

which is very, very small for n even moderately large. For $n = 23$, the above ratio is already below 50%. In other words, there is at least a 1/2 chance that 2 students in a class of 23 have the same birthday.

Note that we assumed in the above calculation that the $\binom{n}{2}$ pairs were independent of each other. This is not quite true (since the pairs have overlap). A more precise answer can be achieved by enumerating the sample space. The number of possible outcomes (sample space) is given by

$$365^n$$

since there are n people with 365 choices for birthdays each. For all of them to have *different* birthdays, the number of choices is limited to

$$365 \times 364 \times 363 \times \ldots \times (365 - (n-1)) = \frac{365!}{(365-n)!}$$

by the arrangement principle. Therefore the probability that some pair share the same birthday equals

$$1 - \frac{1}{365^n}\frac{365!}{(365-n)!}.$$

Growth of functions

One last point about counting, which you have probably already seen in earlier classes. In problems involving counting, it is often the case that we are not interested in the final answer itself, but rather how the answer *scales* as the size of the problem grows. There is a systematic way to deal with such situations, illustrated via an example.

> The ISU network has n computer terminals. Suppose a certain networking algorithm (say, Algorithm A) requires listing all possible combinations of 2 computers, while a second algorithm (Algorithm B) requires listing combinations of 3 computers. Which algorithm is more efficient (i.e., which one is likely to terminate faster?)

One can see that the number of possible combinations of 2 computers is nothing but:

$$\binom{n}{2} = \frac{n!}{(n-2)!2!} = \frac{n(n-1)}{2} = \frac{n^2 - n}{2}$$

while the number of combinations of 3 computers i:

$$\binom{n}{3} = \frac{n!}{(n-3)!3!} = \frac{n(n-1)(n-2)}{6} = \frac{n^3 - 3n^2 + 2n}{6}.$$

Now the question is: Which is "bigger": $\binom{n}{2}$ or $\binom{n}{3}$

Well, it turns out that it depends on n. For small n, $n(n-1)/2$ is bigger, while for *all* n bigger than 5, $n(n-1)(n-2)/6$ is bigger than $n(n-1)/2$ (as an **exercise**, try proving this via induction.) So, morally, $\frac{n(n-1)(n-2)}{6}$ is the "bigger" quantity.

The reason why this happens is that the dominating factor in the two expressions is the exponent of the leading term. In the first case, the exponent is two (due to the n^2 term) while in the second case, the exponent is three (due to the n^3 term). The constants don't matter – no matter what they are, after n grows past a certain threshold, the function with the higher exponent will start dominating the function with the smaller one.

This is also why the function $\frac{n^2}{100}$ is "bigger" than $100n$, although it appears to be smaller for all n up to 10,000.

This leads to what is called "big-oh" notation. Formally, the definition is given as follows:

> Let $f(n)$ be a function mapping the natural numbers to positive reals. Let $g(n)$ be a function such that for *some* N, $f(n) \leq Cg(n)$ for all $n \geq N$. Then, $f \in O(g)$ or "f is big-oh of g".

So in the above example, we can write $\binom{n}{2} \in O(n^2)$ and $\binom{n}{3} \in O(n^3)$, and drop all other lower-order terms and constants, leading us to conclude that the latter is "bigger".