**Application: A Probabilistic Search Size Estimation Algorithm**

Suppose we have a database filled with $N$ entries, and there's something specific that we'd like to search for. We're going to develop and analyze a probabilistic algorithm not for searching the database, but rather for *counting how many* solutions there are. (A *solution* in this context is any database entry that satisfies the desired search criteria.) For example, perhaps we are managing a large domain of web pages, and we'd like to know how many of them are sports related. Or perhaps we're designing image editing software, and we'd like to know how many pixels there are of a specific color.

Whatever the case, a simple probabilistic algorithm for doing this might go as follows: Rather than sample all $N$ entries in the database, we might instead pick some smaller number of elements from the array at random. It seems reasonable to assume that the percentage of our smaller sample which are solutions is a decent guesstimate of the percentage of solutions in the entire database.

For example, suppose our database has 100 entries, and I pick 20 entries uniformly at random. (That is to say, all entries are equally likely.) Suppose that out of those 20 entries, 5 of them are the desired search result. So it turned out that one fourth of the random sample were solutions, so it might be reasonable to assume that one fourth of *the entire database* are solutions. One fourth of 100 is 25, so my guess is that there are around 25 solutions to this search problem.

Let's refine this idea and analyze it using what we know about probability theory. As we said, $N$ will denote the total number of entries in the database. Let's also let $k$ be a smaller number denoting how many entries we'll sample at random, and let $M$ be the total number of solutions there are in the database. For each $i$ from 1 to $k$, we'll define the Bernoulli random variable:

$$X_i = \begin{cases} 1 & \text{if the } i^{th} \text{ entry chosen is a solution} \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

For the sake of simplifying our analysis, let's assume that we're sampling with replacement. Under this assumption, the probability of finding a solution will never change - It will always just be $\frac{M}{N}$. Then

$$X_i \sim Bernoulli(p = \frac{M}{N}) \text{ for each } i = 1, 2, ..., k \tag{2}$$

So our random variables are identically distributed. Furthermore, sampling with replacement also means that any one choice has no effect on any other choices. That is to say, $X_i$ and $X_j$ are independent for any $i \neq j$. So we have a collection of independent and identically distributed (iid) Bernoulli random variables. From these, we can define the output of our search result. Note that the total number of solutions to our search problem is just the sum of these $X_i$'s! So the output to our probabilistic program is itself a random variable - a function of these!

$$S = N(\frac{\sum_{i=1}^{k} X_i}{k}) \tag{3}$$

To see this, note that the sum divided by $k$ in parentheses is the percentage (as a proportion) of our sample which are solutions. Multiplying this by N then gives us the projected percentage of total solutions.

Given what we now know, we can thoroughly analyze this random variable without even knowing it's distribution! We first find the expected value and the standard deviation:

To analyze the quality of this algorithm, we need to know how it's precision and accuracy scales with our sample size, $k$. We'll answer the following question:

How big must our sample size $k$ be in order to ensure that, with probability at least $\frac{3}{4}$, we will be within $\sqrt{M}$ of the correct value?

So if there are, say, 36 solutions, I'm asking how big my random sample needs to be such that I can be *at least* 75% sure that I'm *at most* 6 off from the correct answer. The 75% probability is the accuracy of the algorithm, while the error of 6 is the precision. To have a decent algorithm, we need to be able to good values for both of these at once. Chebyshev's Inequality gives us the tools we need to find our desired $k$:

This result might look like bad news - It involves $M$, the thing we're trying to estimate in the first place! However:

So by introducing probability theory we've come up with an algorithm which effectively cuts the size of the database we're searching through by a factor of 4. This isn't all that great. It's still just $O(N)$ - what it would be if we had just brute-force combed through the entire database! However, it can actually be proven that *this is algorithm is the best that any classical algorithm can ever do*. More amazing still, it can be shown that, through a process called *phase estimation*, a *quantum computer* can in theory perform this task probabilistically, with complexity $O(log(N))$.

So while our algorithm might not be the greatest thing ever conceived of by mankind, I still think that this is a great application of what we've done so far. Using the theory of expectation, there's a lot that we can say and do with probabilities - without actually calculating a single one!