Lecture 11: Random Variables, Independence and Linearity of Expectation

Anup Rao

January 29, 2018

We discuss the use of hashing and pairwise independence.

Random variables $X_1, \ldots, X_n$ are said to be mutually independent if for every $x_1, \ldots, x_n$, we have

$$p(X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n) \\ = p(X_1 = x_1) \cdot p(X_2 = x_2) \ldots p(X_n = x_n).$$

It is possible that the random variables are not independent, even though every pair of random variables is independent. For example, suppose $X, Y \in \{0,1\}$ are uniformly random bits that are independent of each other. Set

$$Z = X + Y \mod 2.$$

Z is not independent of X, Y. However, we see that X, Z are mutually independent. Similarly, Y, Z are also mutually independent.

Another example: let $X \in \{0,1\}^n$ be a uniformly random string such that $X_1 + X_2 + \ldots + X_n = 0 \mod 2$. Then $X_1, \ldots, X_n$ are not mutually independent. However, every $n-1$ of those variables *are* mutually independent.

In general, we say that $X_1, \ldots, X_n$ are k-wise independent if every k variables are independent.

Hashing

Here is an application of pairwise independence to a data structures problem. Suppose we have subsets $S \subseteq T \subseteq [n]$. We would like to store S and allow membership queries into S. We want to preprocess and store S so that at runtime, if we get an element $i \in T$, we can quickly deduce whether $i \in S$ or not.

For example, imagine that T is the set of all students at UW, $[n]$ is the set of all people in the world, and S is the set of students enrolled in CSE312. We would like to be able to quickly check whether a student at UW is enrolled in CSE312 or not.

An obvious way to solve this problem is to just store S as a sorted list. Then if we want to check whether or not $i \in S$, we can just scan the list. This takes time proportional to $|S|$. Can we speed this up?

Another way to solve this problem is to store the indicator vector of S as a vector in $\{0,1\}^n$. Then membership queries can be checked in constant time. But this requires n bits of space! Do we really need n bits of space?

The idea of *hashing* is to map each element of the space to a much smaller set of hashes, in such a way that collisions are unlikely. Then membership can be checked quickly, and using small space at the same time. Let $h : [n] \to [m]$ be a uniformly random function, with $m \ll n$. Since h is uniformly random, $h(1), h(2), \ldots, h(n)$ are mutually independent and uniformly random in $[m]$.

Now, here is the idea.

1. Pick a random function $h : [n] \to [m]$ as described above.
2. Initialize a bit vector $Z \in \{0,1\}^m$ by setting all coordinates to 0.
3. For every element $j \in S$, set $Z_{h(j)} = 1$.

To check if $i \in T$ belongs to S or not, inspect $Z_{h(i)}$. If $Z_{h(i)} = 0$, we conclude that $i \notin S$. If $Z_{h(i)} = 1$, we conclude that $i \in S$.

The algorithm makes an error only if there are distinct i, j such that $i \in S$, $j \in T$ and $h(i) = h(j)$. In other words, an error can happen only if h is not an injective function on T.

Claim 1. *$p(h$ is not injective on $T) \leq (1/m) \cdot \binom{|T|}{2}$*

Proof. Let $E_{i,j}$ denote the event that $h(i) = h(j)$. Then $p(E_{i,j}) = 1/m$. If E denotes the event that h is not injective on T, then we see that $E = \bigcup_{i \neq j \in T} E_{i,j}$. So

$$p(E) \leq \sum_{i \neq j \in T} p(E_{i,j}) = \sum_{i \neq j \in T} (1/m) = (1/m) \cdot \binom{|T|}{2}.$$

□

So, as long as we set $m \gg \binom{|T|}{2}$, the probability that h is not injective on T is extremely small. This means, we only need to store a bit vector whose length is about $O(|T|^2)$.

All of this is great, but it is not actually practical to sample a uniformly random function $h : [n] \to [m]$. Imagine trying to sample a uniformly random hash for every single possible human name. Luckily, it turns out that it is enough to sample a pairwise independent hash. We say that the hash function is pairwise independent if for every $i \neq j$, $h(i)$ and $h(j)$ are independent. If you look at the proof of the claim above, you see that we only need the hash function to be pairwise independent for the analysis to go through.

Moreover, there are several nice constructions that give pairwise independent functions. For example, let $m = p$ be a prime number. Let $a, b \in [p]$ be independent and uniformly random. Define

$$h(x) = ax + b \mod p.$$

Then h is a pairwise independent function mapping $[n]$ to $[p]$. h can also be computed very quickly. So, setting $p \gg |T|^2$ be a large

enough prime number, we get a hash function that allows us to store S and quickly answer membership queries.