COS597D: Information Theory in Computer Science October 19, 2011

Lecture 10

Lecturer: Mark Braverman *Scribe: Andrej Risteski**

1 Kolmogorov Complexity

In the previous lectures, we became acquainted with the concept of Shannon entropy, which is designed to capture distributions X over sets, i.e. relative frequencies of elements in a given set.
Intuitively, however, we would like a notion that captures the fact that:

- 111...111 is not very random.
- 3.14159... is not very random. (Since we can write a short program outputting π to arbitrary accuracy and so predict the next bit easily.)
- 78149512... is "fairly" random.

In other words, we want to describe how "compressible" and/or "hard to describe" a string is.

Definition 1. *Let U be a universal computer. A valid execution of U on p is one that terminates and reads the entire string p.*

Note that the programs on which U executes validly form a prefix-free set. Indeed, if p is a prefix of p', U is valid on p, it will terminate once it's done reading p – so it cannot read all of p' – so the execution on p' cannot be valid.

Definition 2. *Kolmogorov complexity $K_U(x)$ of string x is defined as $\min_{p:U(p)=x} l(p)$, where $l(p)$ denotes the length of string p.*

In other words, the Kolmogorov complexity of x is the shortest program outputting x. Let's note that this definition is fairly robust.

Claim 3. *If U is a universal computer, A another computer, there is some constant C_A, s.t. $K_U(X) \leq K_A(x) + C_A$ for all strings x.*

Proof This is almost immediate. One can write a compiler for translating A to U. This is come constant-sized program – so the claim holds. ■

Having the claim above in mind, we will henceforth drop the subscript U in the notation for Kolmogorov complexity. Now, let's note a few properties about Kolmogorov complexity.

Claim 4. $K(X|l(x)) \leq l(x) + C$

Proof Just use the trivial encoding as a program which outputs the sequence of bits in x. ■

Claim 5. $K(X) \leq l(x) + 2\log l(x) + C$

*Portions of the notes are based on "Elements of Information Theory", T.M.Cover and J.A.Thomas

Proof We need to somehow encode the length of the string, and then we can use the trivial encoding as above. We can do this by encoding the length of the string as binary number, and then doubling each of the digits in the binary representation, followed by the string "01". For example, if the length is 6, i.e. 110 in binary, we encode it as 11110001. This gives us the bound we need. ■

As a remark, we can keep doing the trick in the proof above recursively. For instance, by doing it one more time, we can get Kolmogorov complexity $l(x) + \log l(x) + 2\log\log l(x) + C$. By continuing the recursion until the terms are positive, we get an encoding of size $l(x) + \log l(x) + \log\log l(x) + \cdots + c$.

Claim 6. *The number of strings with $K(x) = k$ satisfies $|\{x : K(x) < k\}| < 2^k$.*

Proof Trivial. There are at most 2^k programs of length k – so they encode at most 2^k different strings. ■

Theorem 7. *Kolmogorov complexity is not computable.*

Proof

Suppose there is a program $Q(x)$ that outputs $K(x)$.
Then, consider the following program:

P(n):
For all $x \in \{0,1\}^n$
If x is the first string with $K(x) \geq n$, using the description of Q to check this.
Output x and halt.

Now, if the sequence of strings outputted for each input length n for P is $x_1, x_2, \ldots, x_n$, then $K(x_n) \geq n$. However, $K(x_n) \leq c + 2\log n$ – by just using the program above, which has a constant-size description (given that the description of Q is just a constant as well), and using $2\log n$ bits for encoding n. So we have a contradiction. ■

2 Kolmogorov Complexity and Entropy

We note here a relationship between Kolmogorov complexity and entropy.

Lemma 8. *For any computer U,* $\sum_{p:U(p) \text{ halts}} 2^{-l(p)} \leq 1$

Proof This is fairly simple, and based on the proof of Kraft's inequality in "Elements of information theory".

Take any finite set of strings p, s.t. $U(p)$ halts. We will call these strings "valid" in the proof to follow. We can view the valid strings as nodes in a full binary tree, where each string is located at the node we get by tracing out the sequence of bits in the string from the root.

Let l_{max} be the maximal length of any of the valid strings, i.e. the maximal depth of the tree where there is a node that represents a valid string. Consider all notes at depth l_{max} of the tree. Each of them either denotes a valid string, a descendant of a valid string, or neither.

A valid string at level l has at most $2^{l_{max}-l}$ descendants at level l_{max}. Each of these descendant sets are disjoint, since the valid strings are prefix-free. So, $\sum_{\text{p:p is a valid string}} 2^{l_{max}-l(p)} \leq 2^{l_{max}}$, i.e. $\sum_{\text{p:p is a valid string}} 2^{-l(p)} \leq 1$, as we need.

Now, since this holds for any finite subset of strings p, s.t. $U(p)$ halts, it holds in the infinite case as well. ■

Lemma 9. *If x_i are iid random variables, $\frac{1}{n}\mathbb{E}[K(x_1, x_2, \ldots, x_n)] \to H(x)$, as $x \to \infty$*

This will not be proven here – only the following partial result:

Theorem 10. *Let $x_1, x_2, \ldots, x_n$ be $B_{1/2}$; Then, $\Pr[K(x_1, x_2, \ldots, x_n) < n - k] < 2^{-k}$*

Proof $\Pr[K(x_1, x_2, \ldots x_n) < n-k] = |\{x_1 x_2 \ldots x_n : K(x_1, x_2, \ldots, x_n) < n-k\}| \cdot 2^{-n} < 2^{n-k} \cdot 2^{-n} = 2^{-k}$. ■

Finally, we give a few definitions to inform the reader how one might generalize these notions to infinite strings.

Definition 11. *A sequence $x_1, x_2 \ldots x_n$ is algorithmically random if $K(x_1, x_2, \ldots, x_n|n) \geq n$.*

Definition 12. *An infinite string is incompressible if $\lim_{n\to\infty} \frac{K(x_1, x_2, \ldots, x_n|n)}{n} = 1$.*

3 Universal Probability

In this section, we define universal probability, which defines a probability distribution on all binary strings, favoring the ones with short programs outputting them. The name universal is clear here: in nature usually we like shorter and simpler explanations (i.e. Occam's Razor), and this notion captures exactly that.
Equivalently, we also capture the following idea. If a monkey were to start hitting a typewriter, under the assumption that the text he types out is a valid program, what output strings of these programs are more likely to be produced?

Definition 13. *The universal probability $P_U(x)$ of a string x is defined as*

$$\sum_{p:U(p)=x} 2^{-l(p)} = \Pr_{p:U(p) \text{ halts}}[U(p) = x]$$

Note that by claim 3, $P_U(x) \geq C'_A \cdot P_A(x)$ for any other universal computer A.

3.1 Kolmogorov Complexity vs Universal Probability

We prove now the main result regarding universal probability, relating it to Kolmogorov complexity.

Theorem 14. *$2^{-K(x)} < P_U(x) \leq C \cdot 2^{-K(x)}$, for some constant C.*

Proof The proof here is based on the exposition in "Elements of Information Theory". The part $P_U(x) > 2^{-K(x)}$ is trivial – just by definition $P_U(x)$ includes the term $2^{-K(x)}$.
We can rewrite the second part as $K(x) \leq \log(\frac{1}{P_U(x)}) + C$. In other words, we wish to find a short program for strings with high $P_U(x)$.
The problem is that $P_U(x)$ is not computable, so despite the fact that we know it will be large for simple programs, we can't just brute force run all programs of length $\leq \log(\frac{1}{P_U(x)})$, and find one. So we have to be somewhat clever.
The program for $K(x)$ will try to simulate the universal computer U on all programs in lexicographic order, and keep an estimate for $P_U(x')$ as it goes along for all strings x'. Then, we will specify a particular short string, which will allow the program to get x from these estimates.
We can simulate U on all programs as follows. Let $p_1, p_2, \ldots$ be the programs in lexicographic order, and the corresponding outputs after running them are $x_1, x_2, \ldots$ (if they terminate of course). We run the simulation in stages. In stage i, we run programs $p_1, p_2, \ldots p_i$ for i steps each. That way, each program that terminates will be run until it terminates.
Furthermore, we handle the terminations of programs as follows. If the program p_k terminates in exactly i steps in the i-th stage (i.e. we see it terminate for the first time), we increase the estimate $\tilde{P}_U(x_k)$ by $2^{-l(p_k)}$. Then, we calculate $n_k = \lceil \log(\frac{1}{\tilde{P}_U(x_k)}) \rceil$. Now, we build a binary tree in whose nodes triplets (p_k, n_k, x_k) will

live. We will assign the triplets to the nodes incrementally. Namely, every time a program p_k outputs a string x_k, we check whether the new estimate of n_k, after updating $\tilde{P_U}(x_k)$ has changed. If that is so, then we find any unassigned node on level $n_k + 1$, and assign it the triplet (p_k, n_k, x_k).
There are two things we need to prove. One is that using this construction, we get a program of the required size outputting x. The other is that it is always possible to place the pairs as specified above (i.e. if we need to put a node on some level, there is always at least one unassigned node on that level).
For the first part, the encoding of x can be the path to the lowest depth node (i.e. closest to the root) containing a triplet (p_k, n_k, x_k), $x_k = x$. By construction, the depth will be $\leq \lceil \log(\frac{1}{P_U(x)}) \rceil + 1 \leq \log(\frac{1}{P_U(x)}) + 2$ – so $K(x) \leq \log(\frac{1}{P_U(x)}) + C$, as we need.
For the second, the proof resembles the Kraft inequality proof mentioned above. We sketch the proof here as well, because the idea of the proof is simple, but elegant, and used in other places as well. We state it in the form of a general lemma.

Lemma 15. *Let $r_1, r_2, \ldots$ be a sequence of requests to place index i at depth r_i of a prefix-free binary coding tree. Then, it's possible to satisfy all of the requests* online *if* $\sum_{i=1}^{n} 2^{-r_i} \leq 1$.

Proof We can intuitively thing of think of 2^{-r_i} as the "volume" of a request i, as it blocks all of the nodes at depth higher than r_i. So, really, what the statement of the lemma says is that the obvious condition the volumes need to satisfy is also sufficient.
We prove that the trivial strategy of placement, greedy leftmost actually works. In other words, when we get a new request r_i, we place the element in the leftmost available spot at depth r_i.
The proof is by induction on n.
The case $n = 1$ is trivial.
So, suppose the case holds when the number of requests is $< n$.
Now, suppose we get n requests. Suppose also, for the sake of contradiction, we get stuck inserting r_n. Let k be the depth of the deepest element in the right subtree R (WLOG assume R is not empty, otherwise the claim is trivial: we can just insert the request in the right subtree – so we get a contradiction immediately). We prove that in this case, both subtrees have to be fairly full, and the volume constraint is violated.
An element on depth k and r_n didn't fit into left subtree L. Indeed, otherwise in the former case, we could have put all the elements on depth k in the left subtree. In the latter, we could have satisfied r_n by putting the element in the left subtree. Let's define $l := max(k, r_n)$. Let $d_1, d_2, \ldots d_m$ be the depths of the nodes assigned to the left subtree L. Then, since neither k nor r_n fit into the left subtree, $w(L) = \sum_{i=1}^{m} 2^{-d_i} + 2^{-l} > \frac{1}{2}$.
Similarly, since r_n didn't fit in R, $w(R) + 2^{-r_n} > \frac{1}{2}$. Furthermore, all elements of R are at depth $\leq l$, therefore $w(R)$ is an integer multiple of 2^{-l}. Hence, $w(R) + 2^{-r_n} \geq \frac{1}{2} + 2^{-l}$.
Putting everything together, $w(L) + w(R) + 2^{-r_n} > \frac{1}{2} - 2^{-l} + \frac{1}{2} + 2^{-l} = 1$, which is a contradiction. Hence, the claim follows by induction. ■

With the lemma above, the last piece of the proof is almost complete. Once we verify that the sequence of requests in our particular case satisfy the constraints of Lemma 15, we are done. But this is easy. In our particular case, we get

$$\sum_{i=1}^{n} 2^{-r_i} =$$

$$= \sum_{x \in \{0,1\}^* \text{ there is a request to insert } x \text{ at level } n_k} \sum 2^{-(n_k+1)} =$$

$$= \sum_{x \in \{0,1\}^*} 2^{-1} \sum_{\text{there is a request to insert } x \text{ at level } n_k} 2^{-n_k} =$$

But now, since we insert a new triplet if the new estimate of n_k changes, and the estimate will stop growing once the estimated value $P_{\tilde{U}}(x)$ reaches it's actual value, $P_U(x)$, the above summation is at most:

$$\sum_{x\in\{0,1\}^*} 2^{-1} \sum_{i=0}^{\infty} 2^{-\lceil \log 1/P_U(x)\rceil - i} =$$

$$= \sum_{x\in\{0,1\}^*} 2^{-1} 2^{-\lceil \log 1/P_U(x)\rceil} \sum_{i=0}^{\infty} 2^{-i} =$$

$$= \sum_{x\in\{0,1\}^*} 2^{-1} 2^{-\lceil \log 1/P_U(x)\rceil} 2 \leq$$

$$\leq \sum_{x\in\{0,1\}^*} P_U(x) \leq 1$$

where the last inequality comes from Lemma 8. This finishes the proof. ■