CS 229r Information Theory in Computer Science Feb. 7, 2019

Lecture 4

Instructor: Madhu Sudan *Scribe: Bill Zhang*

1 Overview

1.1 Logistical Notes

1. **Scribing:** Reminder to sign up for scribing, and the first scribing draft is due within 24 hours of lecture.
2. **Problem Set 1:** Due Friday, February 8 at 8:00 PM. Reminder that each student has 3 late days, and may use up to 2 per problem set. Email Madhu if you plan to use late day(s).
3. **Office Hours:** Madhu will hold office hours after lectures, in MD 339. Mitali will hold office hours on Friday from 4:30 to 5:30 PM.

1.2 Outline for Today

1. Single-shot compression
2. Universal compression
3. Markovian sources

2 Single-Shot Compression

The general structure of an encoding problem in real life, rather than the theoretical definition we've been dealing with, revolves around a sender and receiver, both of which know of a distribution $P = (P_1 \dots P_m)$. The sender has an $X \sim P$ and encodes it as $E(X)$ using the encoding function $E : [m] \to \{0,1\}^*$ to send to the receiver. Meanwhile, the receiver uses a decoding function $D : \{0,1\}^* \to [m] \cup \{?\}$ to unpack the encoding. We include the ? because its possible for the decoding function to return multiple possible inputs for the same encoded output.

Thus, our goal is represented by the following:

$$\min_E \left\{ \mathop{\mathbb{E}}_{X \sim P} [|E(X)|] \right\}$$

Where we are trying to minimize this over all E, and this naturally leads us to consider Huffman encoding, which as it turns out can "solve" this problem optimally! In addition, we'll cover Shannon encoding, which solves this problem in $\leq H(X) + 1$ bits. Recall that we've seen the Shannon lower bound, which asserts that the optimal encoding is $\geq H(X)$. Thus, it follows that Huffman encoding will fall somewhere between these two.

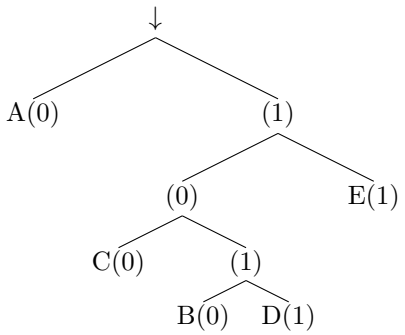
2.1 Huffman encoding

The underlying concept of Huffman encoding is a binary tree representation of the support Ω.

Example 1. Suppose we have the following prefix-free mapping:

$$A \to 0 \quad B \to 1011 \quad C \to 100 \quad D \to 1010 \quad E \to 11$$

We construct a tree as follows:



Note that the prefix-free property requires that no encoding is an ancestor of another. Thus, our algorithm (not really pseudocode but just to give the intuition) for Huffman encoding is as follows:

Huffman Encoding $(P_1, \ldots, P_m)$:
- sort/relabel P_i s.t. $P_1 \geq P_2 \geq \cdots \geq P_m$
- merge P_m and P_{m-1}, giving us $Q_1, \ldots, Q_{m-1}$, where $Q_i = P_i$ except $Q_{m-1} = P_{m-1} + P_m$
- let E' be the encoding for $(Q_1, \ldots, Q_{m-1})$, then set $E(i) = E'(i)$ except for $E(m-1) = E'(m-1) + \text{`0'}$, and $E(m) = E'(m-1) + \text{`1'}$

Proof of Huffman encoding. The proof of optimality for Huffman encoding also follows a recursive pattern: WLOG assume $P_1 \geq P_2 \geq \cdots \geq P_m$, and let $\ell_i = |E(i)|$ for some optimal encoding, then first note that $\ell_1 \leq \ell_2 \leq \cdots \leq \ell_m$. To see why this must be true, consider $P_i \geq P_j$ but $\ell_i \geq \ell_j$, in which case if we swap the encodings $E(i)$ and $E(j)$ then we reduce $\sum p_i \ell_i$ by $(p_i - p_j)(\ell_i - \ell_j)$, thus contradicting the optimality of the encoding.

Here, we observe that node m must be a deepest leaf, with depth ℓ_m, and it must have a sibling (otherwise just move m up a level and the encoding will still be prefix-free). Thus, nodes m and $m-1$ are siblings, meaning if we merge them, we get our recursive optimality with $Q_1, \ldots, Q_{m-1}$ as well (if this merged version is not optimal, then neither would our original with $P_1, \ldots, P_m$). Thus, by induction, we can eventually formalize this sketch and prove that Huffman encoding is optimal.

2.2 Shannon encoding

Shannon asserted that, to encode i, we could just use $\lceil \log \frac{1}{p_i} \rceil = \ell_i$ bits, and further noted that:

$$\sum p_i \leq 1 \Rightarrow \sum 2^{-\ell_i} \leq p_i \leq 1$$

Although Shannon didn't formulate an actual encoding strategy, you know from the problem set that an encoding function exists, thus we can just use that.

In particular, we evaluate the performance as follows:

$$\mathop{\mathbb{E}}_{X \sim P}[E(X)] = \sum_i p_i \ell_i = \sum_i p_i \left\lceil \log \frac{1}{p_i} \right\rceil \leq \sum_i p_i \left(\log \frac{1}{p_i} + 1 \right) = H(X) + 1$$

Thus, Huffman encoding is bounded above by this $H(X) + 1$, since it must be more optimal than Shannon encoding.

Exercise 1. *Is this gap of 1 tight? In particular, note that:*

$$H(X) \leq \textit{Huffman} \leq \textit{Shannon} \leq H(X) + 1$$

Where does the $+1$ occur?

It's remarkable that there is only a tiny additive difference between the optimal and maximal encodings!

Exercise 2. *(Additional) Verify that Huffman encoding is indeed bounded above by $H(X)+1$. What distributions can you construct which define the best and worst cases for Huffman encoding?*

3 Universal Encoding

Back when we were building fax machines to communicate information, the first compression mechanisms used a random piece of paper to determine a distribution on which to build the encoding and compression. However, nowadays the big difference between techniques we just learned (Huffman encoding, etc.) and tools we often use (GZip, etc.) is that we don't know the probability distribution for these compression tools.

Thus, we come to the topic of universal encoding, where we have an input of $W \in \Sigma^n$, and our task is to compress to $\{0,1\}^*$. We'll example the Lempel Ziv algorithm, which has empirically performed well, and later we'll consider other algorithms for certain classes of probabilistic sources of W.

3.1 Lempel Ziv algorithm

The algorithm takes the following approach: first decompose the string W into small pieces, written as a composition $W = S_0 \circ S_1 \circ S_2 \circ \cdots \circ S_m$, where $S_0 = \lambda$ is empty. Then, for all i, we write $S_i = S_{j_i} \circ b_i$, where $j_i < i$ and $b_i \in \Sigma$, and we try to take the longest S_{j_i} possible such that $S_i \neq S_j$ for $j < i$. The general intuition is we express later strings as earlier ones plus some character b_i such that they remain unique. The encoding would thus be $E(W) = (j_1, b_1), (j_2, b_2), \ldots, (j_m, b_m)$, and if we have a lot of these j_i then that's not good, but if we only have a few then that's great!

Example 2. Visually, a possible decomposition looks like:

$$W = 010111001101111101$$

$$= [0]\ \ [1]\ \ [01]\ \ [11]\ \ [00]\ \ [110]\ \ [111]\ \ [1101]$$

$$\Rightarrow (\lambda, 0), (\lambda, 1), (A, 1), (B, 1), (A, 0), (D, 0), (D, 1), (F, 1)$$

Here, we would ultimately encode $\lambda, A, B, \ldots$ in $\sim \log n$ bits.

Warning: Small strings can possibly get longer after being compressed. It takes much longer strings for Lempel Ziv to "get going" and result in actual reductions in length.

Exercise 3. *Find a prefix-free encoding of $\mathbb{Z}^+$ that encodes n in $\log n + O(\log \log n)$ bits.*

We now analyze the performance of Lempel Ziv; if we consider $W = W_1, \ldots, W_n$ with $W_i \sim P_X$ i.i.d. then as $n \to \infty$, with high probability the length of compression approaches $(H(X) + o(1)) \cdot n$.

Exercise 4. *Apply Lempel Ziv to the following sequence:*

$$W = 0100110001110000111100000011111$$

In addition, what would you conjecture are the worst and best case strings for the effectiveness of Lempel Ziv compression?

3.2 Markovian sources

We will now observe that Lempel Ziv also successfully compresses Markovian sources, which aren't i.i.d. like what we've considered so far.

Definition 5 (Markov chain)**.** a (time invariant) *Markov chain* is a sequence $Z_1, Z_2, \ldots, Z_n, \ldots$ such that:

1. $Z_n|Z_1 \dots Z_{n-1} \sim Z_n|Z_{n-1} \quad \forall n$, and

2. $Z_n|Z_{n-1} \sim Z_m|Z_{m-1}$, transitions always happen with same probability distribution.

Definition 6 (k-state Markov chain)**.** If we consider $Z_i \in \Gamma = \{1, \dots, k\}$, then this forms a *k-state Markov chain*, which is given by a $k \times k$ matrix M, where $M_{ij} = \Pr[Z_2 = j|Z_1 = i]$. In addition, we will only consider Markov chains that satisfy:

1. irreducibility, meaning the chain is strongly connected, and there exists a path between every pair of vertices, and

2. aperiodicity, meaning GCD(cycle lengths) = 1.

And in particular we know that for all chains M that satisfy these conditions, there exists a *stationary distribution* $\Pi : \Pi(M)$ such that if $Z_i \sim \Pi$, then $Z_{i+1} \sim \Pi$, or that in any point of time, each edge is equally likely to be traversed.

Thus, we can calculate the entropy of the chain M as:

$$H(M) \triangleq \lim_{n\to\infty} H(Z_n|Z_1 \dots Z_{n-1})$$

This can be interpreted as, given the chain goes on forever, how much do I need to say about the current state to get the next state. Note that we can thus transform this into:

$$H(M) \triangleq \lim_{n\to\infty} H(Z_n|Z_1 \dots Z_{n-1}) = \lim_{n\to\infty} H(Z_n|Z_{n-1}) = H(Z_2|Z_1)$$

Note that there are some Markov chains that we can't do anything about, particularly those where we cannot tell where we started given the current state of the chain. However, there is a nice class of models for which Lempel Ziv works well on.

Definition 7 (Hidden Markov Models)**.** Given a model with $X_1 \dots X_n$ such that $X_i \sim P_{Z_i}$, then there exists an underlying Markov chain $Z_1 \dots Z_n$ upon which the model's states are drawn. The sequence $\{X_1\}$ is what's observed, and the Markov chain exists in the background.

Lempel Ziv can compress these Hidden Markov Models, which is quite impressive!