15:853: Algorithms in the Real World
Fall 2019

Lecture 9

October 3 2019

Lecturer: Rashmi Vinayak *Scribe: Leslie Rice*

9.1 Recap: Entropy

For a set of messages S with probability $p(s), s \in S$, the **self information** of s is:

$$i(s) = log\frac{1}{p(s)} = -log(p(s))$$

Entropy is the weighted average of self information.

$$H(S) = \sum_{s \in S} p(s) \log\left(\frac{1}{p(s)}\right)$$

Conditional entropy is the weighted average of the conditional self information.

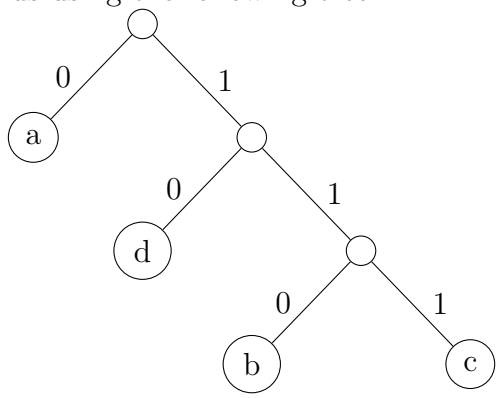
$$H(S|C) = \sum_{c \in C}\left(p(c)\sum_{s \in S} p(s|c) \log\frac{1}{p(s|c)}\right)$$

9.2 Uniquely Decodable Codes

Consider the codewords $a = 1, b = 01, c = 101, d = 011$. The decoder receives 1011. What is the sequence decoded to? The sequence could be decoded to be *ad*, *ca*, or *aba*. This is a problem in **variable-length** codes, in which message symbols are mapped to a variable number of code bits. In this case, the codeword is not *uniquely decodable*.

How can we make variable-length codes uniquely decodable? The problem in the previous example is due to the fact that $a = \underline{1}$ is a prefix of $c = \underline{1}01$, and $b = \underline{01}$ is a prefix of $d = \underline{01}1$. The solution is a **prefix code**, which requires that no codeword is a prefix of another code word, and therefore is uniquely decodable. We can represent prefix codes using trees.

Consider the codewords $a = 0, b = 110, c = 111, d = 10$. We can represent these codewords using the following tree:



There are no codewords at intermediate nodes. Each codeword is represented by a leaf of the tree, with values along the path from the root to the leaf.

9.3 Average Length

We define $l(c)$ as the length of a codeword, and $l_a(c)$ as the average length of a codeword. We have bounds on the average length of a codeword.

Theorem (lower bound): For any probability distribution $p(S)$ with associated uniquely decodable code C,

$$H(S) \leq l_a(C)$$

.

Theorem (upper bound): For any probability distribution $p(S)$ with associated *optimal* prefix code C,

$$l_a(c) \leq H(S) + 1$$

9.4 Kraft McMillan Inequality

The Kraft-McMillan inequality gives a necessary and sufficient condition for the existence of a uniquely decodable code.

Theorem (Kraft-McMillan): For any *uniquely decodable code* C,

$$\sum_{c \in C} 2^{-l(c)} \leq 1$$

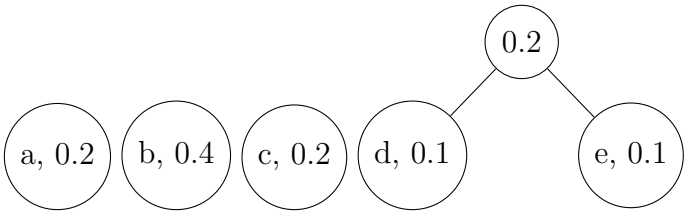
.

Conversely, if we have a set of codelengths that satisfy the inequality, we can construct a prefix code with the codelengths.

9.5 Huffman Codes

Huffman codes are a particular type of optimal prefix code. The Huffman algorithm starts with a forest of trees each consisting of a single vertex corresponding to a message s with weight $p(s)$. The algorithm selects two trees with minimum weight roots p_1 and p_2 and joins them into a single tree by adding root with weight $p_1 + p_2$. The algorithm repeats this step until one tree is left. To encode, start at a leaf of the Huffman tree and follow the path to the root, and then reverse the order of bits. To decode, start at the root of the Huffman tree, and take the corresponding branch for each bit received, until at a leaf.

Consider the example $p(a) = 0.2$, $p(b) = 0.4$, $p(c) = 0.2$, $p(d) = 0.1$, $p(e) = 0.1$. The first two trees we would join would be $p(d)$ and $p(e)$, as shown below.



Next, we have more than one choice when selecting two trees with minimum weight roots, in this case $p = 0.2$. The result will be codes of the same average length, but potentially different variance. In order to reduce variance, whenever there is a choice to combine, choose the node that was created earliest (so, in this case, nodes a and c). These codes are referred to as **minimum variance Huffman codes**.

9.6 Arithmetic Coding

The problem with Huffman coding is that it assigns a codeword to a particular message, independently of the rest of the messages (see slide 26 from Lecture 2 on Compression for an example where this can produce a very inefficient encoding of a sequence of messages). This is also referred to as a **discrete** method - each message is a fixed set of bits. A **blended** method allows sharing of bits among messages. *Arithmetic coding* allows blending of bits in a message sequence. We can bound the total bits required based on sum of self information:

$$L \leq 2 + \sum i(s)$$

Arithmetic coding maps messages to intervals within $[0, 1)$, and codes a message sequence by composing intervals, where the final interval is called a sequence interval. An example of this is shown on the slides from today's lecture.