Lecture 4:

Entropy & Huffman Codes

September 20, 2006

Reminder: Searching for Optimal Codes

- Last class we saw how to construct an instantaneously decodable code for any set of codeword lengths l_i satisfying $\sum_i 2^{-l_i} \leq 1$.
- We also saw that if $\sum_i 2^{-l_i} > 1$, no uniquely decodable code exists with those codeword lengths.
- If a source generates symbols X independently with probabilities p_i, the *expected codeword length* per symbol is $L = \sum_i p_i l_i$.
- Our goal is to look at the probabilities p_i and design the codeword lengths l_i to minimize L, while still ensuring that $\sum_i 2^{-l_i} \leq 1$.
- We say that a set of codeword lengths is *optimal* if no other lengths satisfying $\sum_i 2^{-l_i} \leq 1$ have a smaller value of $L = \sum_i p_i l_i$.

Bounding the optimal L

- Let $K = \sum_i 2^{-l_i}$. Assume $K \leq 1$ (otherwise the code is not UD).
- We can find a lower bound on L in terms of the p_i:

$$\begin{aligned}
\sum_i p_i l_i &\geq \sum_i p_i (l_i + \log K) \\
&= \sum_i p_i \log(2^{l_i} K) \\
&= \sum_i p_i \log\left(\frac{2^{l_i} K p_i}{p_i}\right) \\
&= \sum_i p_i \log \frac{1}{p_i} + \sum_i p_i \log(2^{l_i} K p_i)
\end{aligned}$$

$$\sum_i p_i l_i + \sum_i p_i \log\left(\frac{1}{2^{l_i} K p_i}\right) \geq \sum_i p_i \log \frac{1}{p_i}$$

Bounding the optimal L

- Using the fact that $\log(x) \leq \alpha(x-1) \quad \forall x > 0$, we can show that the second term on the left is always negative:

$$\begin{aligned}
\sum_i p_i \log\left(\frac{1}{2^{l_i} K p_i}\right) &\leq \alpha \sum_i p_i \left(\frac{1}{2^{l_i} K p_i} - 1\right) \\
&= \alpha \sum_i \frac{1}{2^{l_i} K} - \alpha \sum_i p_i \\
&= \alpha(1-1) \\
&= 0
\end{aligned}$$

- Thus, we can show that

$$\boxed{L = \sum_i p_i l_i \geq \sum_i p_i \log \frac{1}{p_i} = H}$$

Entropy

- The quantity $H(X) = \sum_i p_i \log \frac{1}{p_i}$ is called the *entropy* of the distribution $p(X = a_i) = p_i$ and is a fundamental quantity in the study of information theory.
- It represents an absolute lower bound on the average per-symbol codelength of *any* uniquely decodable code for a source $p(X)$ with symbol probabilities $p(X = a_i) = p_i$.
- The units of entropy depend on the base of the logarithm and tell us how many symbols are in the *encoding* alphabet $\mathcal{A}_Z$.
- For $\log_2$ (our default), the units are "bits" (binary digits), and $Z \in \{0, 1\}$. We could use another base, for example for $\log_{10}$, the units would be decimal digits and $Z \in \{0 - 9\}$.

Intuitions about Entropy

- A plausible proposal:
  The *amount of information* obtained when we learn that $X = a_i$ is $\log_2(1/p_i)$ bits, where $p_i = P(X = a_i)$. (Amount of surprise.)
- Example:
  We learn which of 64 equally-likely possibilities has occurred. The information content is $\log_2(64) = 6$ bits. This makes sense, since we could encode the result using codewords that are all 6 bits long, and we have no reason to favour one symbol over another by using a code of varying length.
- The above information content pertains to a single value of the random variable X. To find out how much information learning the value of X conveys *on average*, we find the expected value of the information content.
- For further intuitions about why this is a plausible measure of information, see Section 4.1 of MacKay's book.

Information, Entropy, and Codes

- How does this relate to designing codes for data compression?
- A vague idea: Since receipt of symbol a_i conveys $\log_2(1/p_i)$ bits of "information", this symbol "ought" to be encoded using a codeword with that many bits.
- A consequence: If this is done, then the expected codeword length will be equal to the entropy: $\sum_{i=1}^{I} p_i \log_2(1/p_i)$, which we know is the best we could ever do.
- At first we might think that for an optimal code, the expected codeword length ought to be equal to the entropy. But it's easy to see that this can't always be so. Consider $p_0 = 0.1$, $p_1 = 0.9$, so $H = 0.469$. But the optimal code for a symbol with only two values obviously uses codewords 0 and 1, with expected length of 1.
- The problem is that $\log_2(1/p_i)$ isn't always an integer!

Shannon-Fano Codes

- If we can't choose codewords with the "right" lengths, $\log_2(1/p_i)$, we can try to get close.
- Shannon-Fano codes are constructed so that the codewords for the symbols, with probabilities $p_1, \ldots, p_I$, have lengths
$$l_i = \lceil \log_2(1/p_i) \rceil$$
  Here, $\lceil x \rceil$ is the smallest integer greater than or equal to x.
- The Kraft inequality says such a code exists, since
$$\sum_{i=1}^{I} \frac{1}{2^{l_i}} \leq \sum_{i=1}^{I} \frac{1}{2^{\log_2(1/p_i)}} = \sum_{i=1}^{I} p_i = 1$$
- Example:

p_i:	0.4	0.3	0.2	0.1
$\log_2(1/p_i)$:	1.32	1.74	2.32	3.32
$l_i = \lceil \log_2(1/p_i) \rceil$:	2	2	3	4
Codeword:	00	01	100	1100

Expected Lengths of Shannon-Fano Codes

- The expected length of a Shannon-Fano code for X, if symbols have probabilities $p_1, \ldots, p_I$, is

$$\begin{aligned}\sum_{i=1}^{I} p_i l_i &= \sum_{i=1}^{I} p_i \lceil \log_2(1/p_i) \rceil \\ &< \sum_{i=1}^{I} p_i \left(1 + \log_2(1/p_i)\right) \\ &= \sum_{i=1}^{I} p_i \; + \; \sum_{i=1}^{I} p_i \log_2(1/p_i)) \\ &= 1 + H(X)\end{aligned}$$

- This gives an *upper bound* on the expected length of an optimal code for X. However, the Shannon-Fano code itself may not be optimal (though it sometimes is).

What Have We Shown?

- Combining the lower bound we derived as the source entropy and the upper bound we just proved for Shannon-Fano codes, we've now proved the following (Theorem 5.1 in MacKay's book):

  A source X can be encoded using an instantaneous code, C, with expected length, $L(C, X)$, satisfying

$$H(X) \;\leq\; L(C, X) \;<\; H(X) + 1$$

- Two main theoretical problems remain:
  1. Can we find *optimal* codes, which actually minimize L?
  2. Can we somehow close the gap between $H(X)$ and $H(X) + 1$ above, to show that the entropy is the exactly correct way of measuring the average information content of a source?

Huffman Codes

- The first piece of good news is that there is a very simple algorithm to construct symbol codes which are guaranteed to be optimal.
- The algorithm is called *Huffman's Algorithm* and the codes it generates are called *Huffman Codes*. They were developed by David A. Huffman when he was a Ph.D. student at MIT in 1952.
- We will prove later that Huffman codes are in fact optimal, but first let's see the actual algorithm.
- We'll concentrate on Huffman codes for a binary code alphabet. Non-binary Huffman codes are similar, but slightly messier.
- Huffman's algorithm is a recursive procedure which merges the two least probable symbols into one new "meta-symbol" as it descends into deeper levels of recursion and duplicates the resulting "meta-codeword" as it ascends.

Huffman Procedure for Binary Codes

procedure Huffman:

inputs: symbols $a_1, \ldots, a_I$
probabilities $p_1, \ldots, p_I$

output: a code mapping $a_1, \ldots, a_I$ to codewords

if $I = 2$:

Return the code $a_1 \mapsto 0,\; a_2 \mapsto 1$.

else

Let $j_1, \ldots, j_I$ be a permutation of $1, \ldots, I$ for which $p_{j_1} \geq \cdots \geq p_{j_I}$.

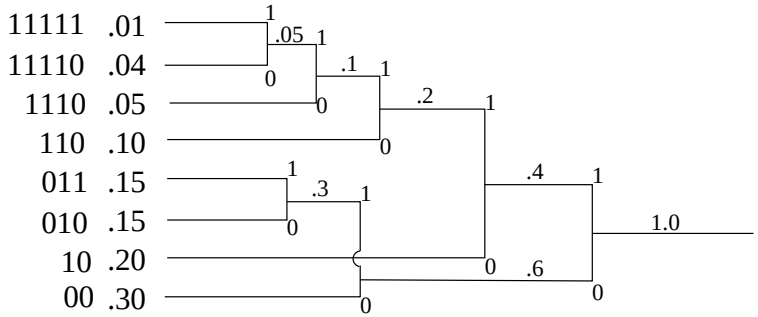
Create a new symbol a', with associated probability $p' = p_{j_{I-1}} + p_{j_I}$.

Recursively call Huffman to find a code for $a_{j_1}, \ldots, a_{j_{I-2}}, a'$ with probabilities $p_{j_1}, \ldots, p_{j_{I-2}}, p'$.
Let the codewords for $a_{j_1}, \ldots, a_{j_{I-2}}, a'$ in this code be $w_1, \ldots, w_{I-2}, w'$.

Return the code $a_{j_1} \mapsto w_1,\; \ldots,\; a_{j_{I-2}} \mapsto w_{I-2},\; a_{j_{I-1}} \mapsto w'0,\; a_{j_I} \mapsto w'1$.

Huffman Coding Example

Consider these probabilities: $\{.01, .04, .05, .10, .15, .15, .20, .30\}$.
The entropy of this source is $H = 2.607...$ bits.



The codeword lengths are $\{5, 5, 4, 3, 3, 3, 2, 2\}$.
The average codeword length is $L = 2.65$ bits.
In comparison, Shannon-Fano coding would give lengths $\{7, 5, 5, 4, 3, 3, 3, 2\}$, which would have given an average length of $L = 3.02$ bits.

Is Huffman the Answer to Everything?

- Huffman codes seem to have solved the main practical problem: We can now construct an optimal symbol code for any source.
- But: This code is optimal *only* if the assumptions we made in formalizing the problem match the real situation. Often they don't:
  - Symbol probabilities may vary over time.
  - Symbols may not be independent.
  - There is usually no reason to require that X_1, X_2, X_3, ... be encoded one symbol at a time, as $c(X_1)c(X_2)c(X_3)\cdots$, unless we really need instantaneous decoding.

Old Example: Black-and-White Images

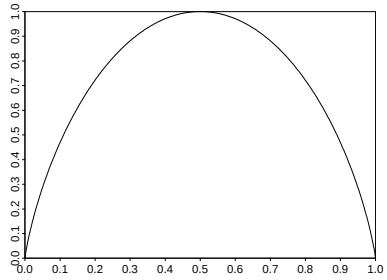
- Recall the example from the first lecture, of black-and-white images.

```
..................................................
..XX...XX...........XX.....XX.....................
..XX...XX...........XXX...XXX.....................
..XX...XX...X.......XX.XXX.XX.....................
..XXXXXXX...........XX..X..XX.....................
..XXXXXXX...X.......XX.....XX....XX....XXXXX.XXX..
..XX...XX...X.......XX.....XX...X..X...XX...X...X.
..XX...XX...X.......XX.....XX...X..X...XX...X...X.
..XX...XX...X.......XX.....XX....XX....XX...X...X.
..................................................
```

  There are only two symbols — "white" and "black".
  The Huffman code is white $\mapsto 0$, black $\mapsto 1$.
- This is just the obvious code. But we saw that various schemes such as run length coding can do better than this.
- Partly, this is because the pixels are not independent. Even if they were independent, however, we would expect to be able to compress the image if black pixels are much less common than white pixels.

Entropy of a Binary Source

- For a binary source, with symbol probabilities p and $1 - p$, the entropy as a function of p looks like this:



- If $p = 0.1$, $H(0.1) = 0.469$, so we hope to compress a binary source with symbol probabilities of 0.1 and 0.9 by more than a factor of 2.
- We obviously can't do that if we encode symbols one at a time.

Solution: Coding Blocks of Symbols

- We can do better by using Huffman codes to encode *blocks* of adjacent symbols.
- Suppose our source probabilities are 0.7 for white and 0.3 for black. Assuming pixels are independent, the probabilities for blocks of two pixels will be

white white	$0.7 \times 0.7 = 0.49$
white black	$0.7 \times 0.3 = 0.21$
black white	$0.3 \times 0.7 = 0.21$
black black	$0.3 \times 0.3 = 0.09$

- Here's a Huffman code for these blocks:

$$WW \mapsto 0,\ WB \mapsto 10,\ BW \mapsto 110,\ BB \mapsto 111$$

The average length for this code is 1.81, which is less than the two bits needed to encode blocks in the obvious way.

Extensions of a Source

- We formalize the notion of encoding symbols in blocks by defining the N-th *extension* of a source, in which we look at sequences of symbols, written as $(X_1, \ldots, X_N)$ or X^N.
- If our original source alphabet, $\mathcal{A}_X$, has I symbols, the source alphabet for its N-th extension, $\mathcal{A}_X^N$, will have I^N symbols — all possible blocks of N symbols from $\mathcal{A}_X$.
- If the probabilities for symbols in $\mathcal{A}_X$ are $p_1, \ldots, p_q$, the probabilities for symbols in $\mathcal{A}_X^N$ are found by multiplying the p_i for all the symbols in the block.
  (This is appropriate when symbols are independent.)
- For instance, if $N = 3$:

$$P((X_1, X_2, X_3) = (a_i, a_j, a_k)) \;=\; p_i\, p_j\, p_k$$

Entropy of an Extension

We can easily prove that $H(X^N) = NH(X)$:

$$\begin{aligned}
H(X^N) &= \sum_{i_1=1}^{I} \cdots \sum_{i_N=1}^{I} p_{i_1} \cdots p_{i_N} \log\left(\frac{1}{p_{i_1} \cdots p_{i_N}}\right) \\
&= \sum_{i_1=1}^{I} \cdots \sum_{i_N=1}^{I} p_{i_1} \cdots p_{i_N} \sum_{j=1}^{N} \log\left(\frac{1}{p_{i_j}}\right) \\
&= \sum_{j=1}^{N} \sum_{i_1=1}^{I} \cdots \sum_{i_N=1}^{I} p_{i_1} \cdots p_{i_N} \log\left(\frac{1}{p_{i_j}}\right) \\
&= \sum_{j=1}^{N} \sum_{i_j=1}^{I} \sum_{i_k \text{ for } k \neq j} p_{i_1} \cdots p_{i_N} \log\left(\frac{1}{p_{i_j}}\right) \\
&= \sum_{j=1}^{N} \sum_{i_j=1}^{I} p_{i_j} \log\left(\frac{1}{p_{i_j}}\right) \times \sum_{i_k \text{ for } k \neq j} p_{i_1} \cdots p_{i_{j-1}} p_{i_{j+1}} \cdots p_{i_N} \\
&= \sum_{j=1}^{N} \sum_{i_j=1}^{I} p_{i_j} \log\left(\frac{1}{p_{i_j}}\right) \;=\; NH(X)
\end{aligned}$$

(Or just use the fact that $E(U + V) = E(U) + E(V)$.)