10-704: Information Processing and Learning **Spring 2012**

Lecture 8: Source Coding Theorem, Huffman coding

Lecturer: Aarti Singh *Scribe: Dena Asta*

8.1 Codes

Codes are functions that convert strings over some alphabet into (typically shorter) strings over another alphabet. We recall different types of codes and bounds on the performance of codes satisfying various desirable properties, such as *unique decodability.*

8.1.1 Taxonomy of codes

Fix a random variable X having as its possible values entries of a set $\mathcal{X}$. Let Σ^* denote the set of strings whose characters are taken from an alphabet Σ. In the case Σ has two elements, we can interpret elements of Σ^* as bit strings, for example. Let $\mathcal{Y}$ denote a set of target symbols and C denote a *code*, a function of the form

$$\mathcal{X}^n \to \mathcal{Y}^*.$$

The *extension* of a *symbol code*, a code of the form $C : \mathcal{X} \to \mathcal{Y}^*$, is the function $C : \mathcal{X}^* \to \mathcal{Y}^*$ defined by

$$C(x_1x_2 \dots x_n) = C(x_1)C(x_2) \dots C(x_n), \quad n = 0, 1, \dots,\ x_1, x_2, \dots, x_n \in X.$$

Listed below are some terms used to describe codes.

codes	description of C	
block	$C : \mathcal{X}^n \to \mathcal{Y}^N$	fixed length (n, N fixed)
symbol	$C : \mathcal{X} \to \mathcal{Y}^*$	variable length
non-singular	C injective	loss-less compression
uniquely decodable	extension $\mathcal{X}^* \to \mathcal{Y}^*$ of a symbol code $\mathcal{X} \to \mathcal{Y}^*$ injective	
prefix/instantaneous/self-punctuating	no code word prefixes another: for all distinct $x', x'' \in \mathcal{X}$, $C(x')$ does not start with $C(x'')$	uniquely decodable

In other words, a *block code* translates n-length strings into N-length strings and a *symbol* code translates individual characters into various strings. For a given symbol code $C : \mathcal{X} \to \mathcal{Y}^*$, let x denote a source symbol, an element of X, $p(x)$ denote the probability $P(X = x)$, and $l(x)$ denote the length of the code $C(x)$.

Theorem 8.1 (Kraft-McMillan Inequality) *For any uniquely decodable code $C : \mathcal{X} \to \{1, 2, \dots, D\}^*$,*

$$\sum_x D^{-l(x)} \leq 1. \tag{8.1}$$

Conversely, for all sets $\{l(x)\}_{x\in\mathcal{X}}$ of numbers satisfying (8.1), there exists a prefix code $C : \mathcal{X} \to \{1, 2, \ldots, D\}^*$ such that $l(x)$ is the length of $C(x)$ for each x.

The idea behind the proof is to note that each uniquely decodable code (taking D possible values) corresponds to a finite D-ary tree having the code words as some of its leaves. Thus the second sentence of the theorem readily follows. The first sentence of the theorem follows from noting that we drew the tree so that its branches formed π/D radian angles and we scaled the tree so that the leaves hovered over the unit interval $[0, 1]$, then each leaf L hovers over the sum of the reciprocal lengths of paths to leaves left of L, i.e. each leaf maps to a disjoint subinterval of $[0, 1]$.

Proposition 8.2 *The ideal codelengths for a prefix code with smallest expected codelength are*

$$l^*(x) = \log_D \frac{1}{p(x)} \quad \textit{(Shannon information content)}$$

Proof: In last class, we showed that for all length functions l of prefix codes, $E[l^*(x)] = H_p(X) \leq E[l(x)]$. ■

While Shannon entropies are not integer-valued and hence cannot be the lengths of code words, the integers

$$\{\lceil \log_D \frac{1}{p(x)} \rceil\}_{x\in\mathcal{X}}$$

satisfy the Kraft-McMillan Inequality and hence there exists some uniquely decodable code C for which

$$H_p(x) \leq E[l(x)] < H_p(x) + 1, \quad x \in \mathcal{X} \tag{8.2}$$

by Theorem 8.1. Such a code is called **Shannon code**. Moreover, the lengths of code words for such a code C achieve the entropy for X asymptotically, i.e. if Shannon codes are constructed for strings of symbols x^n where $n \to \infty$, instead of individual symbols. Assuming $X_1, X_2, \ldots$ form an iid process, for all $n = 0, 1, \ldots$

$$\begin{aligned} H(X) &= \frac{H(X_1, X_2, \ldots, X_n)}{n} \\ &\leq \frac{E[l(x_1, \ldots, x_n)]}{n} \\ &< \frac{H(X_1, X_2, \ldots, X_n)}{n} + \frac{1}{n} = H(X) + \frac{1}{n} \end{aligned}$$

by (8.2), and hence $E[\frac{l(x_1,\ldots,x_n)}{n}] \xrightarrow[n\to\infty]{} H(X)$. If $X_1, X_2, \ldots$ form a startionary process, then a similar arugment shows that $E[\frac{l(x_1,\ldots,x_n)}{n}] \xrightarrow[n\to\infty]{} H(\mathcal{X})$, where $H(\mathcal{X})$ is the entropy rate of the process.

Theorem 8.3 (Shannon Source Coding Theorem) *A collection of n iid ranodm variables, each with entropy $H(X)$, can be compressed into $nH(X)$ bits on average with negligible loss as $n \to \infty$. Conversely, no uniquely decodable code can compress them to less than $nH(X)$ bits without loss of information.*

8.1.2 Non-singular vs. Uniquely decodable codes

Can we gain anything by giving up unique decodability and only requiring the code to be non-singular? First, the question is not really fair because we cannot decode sequence of symbols each encoded with a non-singular code easily. Second, (as we argue below) non-singular codes only provide a small improvement in expected codelength over entropy.

Theorem 8.4 *The length of a non-singular code satisifes*

$$\sum_x D^{-l(x)} \le l_{\max}$$

and for any probability distribution p on $\mathcal{X}$, the code has expected length

$$E[l(X)] = \sum_x p(x)l(x) \ge H_D(X) - \log_D l_{\max}.$$

Proof: Let a_l denote the number of unique codewords of length l. Then $a_l \le D^l$ since no codeword can be repeated due to non-singularity. Using this

$$\sum_x D^{-l(x)} = \sum_{l=1}^{l_{\max}} a_l D^{-l} \le \sum_{l=1}^{l_{\max}} D^l D^{-l} = l_{\max}.$$

The expected codelength can be obtained by solving the following optimization problem:

$$\min \sum_x p(x)l(x) \quad \text{subject to} \quad \sum_x D^{-l_x} \le l_{\max},$$

the convex non-singularity code constraint. Differentiating the Lagrangian $\sum_x p_x l_x + \lambda \sum_x D^{-l_x}$ with respect to l_x and noting that at the global minimum (λ^*, l_x^*) it must be zero, we get :

$$p_x - \lambda^* D^{-l_x^*} \ln D = 0$$

which implies that $D^{-l_x^*} = \frac{p_x}{\lambda^* \ln D}$.

Using complementary slackness, noting that $\lambda^* > 0$ for the above condition to make sense, we have :

$\sum_x D^{-l_x^*} = \sum_x \frac{p_x}{\lambda^* \ln D} = l_{\max}$ which implies $\lambda^* = 1/(l_{\max} \ln D)$ and hence $D^{-l_x^*} = p_x l_{\max}$, or the optimum length $l_x^* = -\log_D(p_x l_{\max})$.

This gives the expected minimum codelength for nonsingular codes as $\sum_x p_x l_x^* = -\sum_x p_x \log_D(p_x l_{\max}) = H_D(X) - \log_D l_{\max}$.

■

In last lecture, we saw an example of a non-singular code for a process which has expected length below entropy. However, this is only true when encoding individual symbols. As a direct corollary of the above result, if symbol strings of length n are encoded using a non-singular code, then

$$E[l(X^n)] \ge H(X^n) - \log_D(n l_{\max})$$

Thus, the expected length per symbol can't be much smaller than the entropy (for iid processes) or entropy rate (for stationary processes) asymptotically even for non-singular codes, since the second term divided by n is negligible.

Thus, non-singular codes don't offer much improvement over uniquely decodable and prefix codes. In fact, the following result shows that any non-singular code can be converted into a prefix code while only increasing the codelength per symbol by an amount that is negligible asymptotically.

Lemma 8.5 *For every non-singular code C such that $E[l_C(X)] = L_{NS}$, there exists a prefix code C' such that*

$$E[l_{C'}(X)] = L_{NS} + O(\sqrt{L_{NS}}).$$

8.1.3 Cost of using wrong distribution

We can use relative entropy to quantify the deviation from optimality that comes from using the wrong probability distribution $q \neq p$ on the source symbols. Suppose $l(x) = \lceil \log_D \frac{1}{q(x)} \rceil$, is the Shannon code assignment for a wrong distribution $q \neq p$. Then

$$H(p) + D(p\|q) \leq E_p[l(X)] < H(p) + D(p\|q) + 1.$$

Thus $D(p\|q)$ measures deviation from optimality in code lengths.
Proof: First, the upper bound:

$$\begin{aligned} E_p[l(X)] &= \sum_x p(x)l(x) = \sum_x p(x) \lceil \log_D \frac{1}{q(x)} \rceil \\ &< \sum_x p(x) \left(\log \frac{1}{q(x)} + 1 \right) = \sum_x p(x) \log \left(\frac{p(x)}{q(x)} \cdot \frac{1}{p(x)} \right) + 1 \\ &= D(p||q) + H(p) + 1 \end{aligned}$$

The lower bound follows similarly:

$$E_p[l(X)] \geq \sum_x p(x) \log \frac{1}{q(x)} = D(p||q) + H(p)$$

$\blacksquare$

8.1.4 Huffman Coding

Is there a prefix code with expected length shorter than Shannon code? The answer is yes. The optimal (shortest expected length) prefix code for a given distribution can be constructed by a simple algorithm due to Huffman.

We introduce an optimal symbol code, called a *Huffman code*, that admits a simple algorithm for its implementation. We fix $\mathcal{Y} = \{0, 1\}$ and hence consider binary codes, although the procedure described here readily adapts for more general $\mathcal{Y}$. Simply, we define the *Huffman code* $C : X \to \{0, 1\}$ as the coding scheme that builds a binary tree from leaves up - takes the two symbols having the least probabilities, assigns them equal lengths, merges them, and then reiterates the entire process. Formally, we describe the code as follows. Let

$$\mathcal{X} = \{x_1, \ldots, x_N\}, \quad p_1 = p(x_1), p_2 = p_2(x_2), \ldots p_N = p(x_N).$$

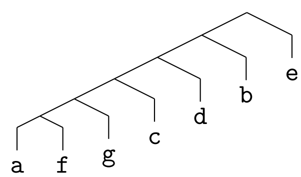
The procedure Huff is defined as follows:

Huff $(p_1, \ldots, p_N)$:
  if $N > 2$ **then**
    $C(1) \leftarrow 0$, $C(2) \leftarrow 1$
  else
    sort $p_1 \geq p_2 \geq \ldots p_N$
    $C' \leftarrow$ Huff$(p_1, p_2, \ldots, p_{N-2}, p_{N-1} + p_N)$
    for each i
      if $i \leq N - 2$ **then** $C(i) \leftarrow C'(i)$
      else if $i = N - 1$ **then** $C(i) \leftarrow C'(N-1) \cdot 0$
      else $C(i) \leftarrow C'(N-1) \cdot 1$
  return C

For example, consider the following probability distribution:

symbol	a	b	c	d	e	f	g
p_i	0.01	0.24	0.05	0.20	0.47	0.01	0.02
Huffman code	000000	01	0001	001	1	000001	00001

The Huffman tree is build using the procedure described above. The two least probable symbols at the first iteration are '`a`' and '`f`', so they are merged into one new symbol '`af`' with probability $0.01 + 0.01 = 0.02$. At the second iteration, the two least probable symbols are '`af`' and '`g`' which are then combined and so on. The resulting Huffman tree is shown below.



The Huffman code for a symbol x in the alphabet $\{\mathtt{a},\mathtt{b},\mathtt{c},\mathtt{d},\mathtt{e},\mathtt{f},\mathtt{g}\}$ can now be read starting from the root of the tree and traversing down the tree until x is reached; each leftwards movement suffixes a 0 bit and each rightwards movement adds a trailing 1, resulting in the code shown above in the table.

Remark 1: If more than two symbols have the same probability at any iteration, then the Huffman coding may not be unique (depending on the order in which they are merged). However, all Huffman codings on that alphabet are optimal in the sense they will yield the same expected codelength.

Remark 2: One might think of another alternate procedure to assign small codelengths by building a tree top-down instead, e.g. divide the symbols into two sets with almost equal probabilities and repeating. While intuitively appealing, this procedure is suboptimal and leads to a larger expected codelength than the Huffman encoding. You should try this on the symbol distribution described above.

Remark 3: For a D-ary encoding, the procedure is similar except D least probable symbols are merged at each step. Since the total number of symbols may not be enough to allow D variables to be merged at each step, we might need to add some dummy symbols with 0 probability before constructing the Huffman tree. How many dummy symbols need to be added? Since the first iteration merges D symbols and then each iteration combines D-1 symbols with a merged symbols, if the procedure is to last for k (some integer number of) iterations, then the total number of source symbols needed is $1 + k(D - 1)$. So before beginning the Huffma procedure, we add enough dummy symbols so that the total number of symbols look like $1+k(D-1)$ for the smallest possible value of k.

Now we will show that the Huffman procedure is indeed optimal, i.e. it yields the smallest expected codelength for any prefix code. Since there can be many optimal codes (e.g. flipping bits in a code still leads to a code with same codelength, also exchanging source symbols with same codelength still yields an optimal code) and Huffman coding only finds one of them, lets first characterize some properties of optimal codes.

Assume the source symbols $x_1, \dots, x_N \in \mathcal{X}$ are ordered so that $p_1 \geq p_2 \geq \cdots \geq p_N$. For brevity, we write l_i for $l(x_i)$ for each $i = 1, \dots, N$. We first observe some properties of general optimal prefix codes.

Lemma 8.6 *For any distribution, an optimal prefix code exists that satisfies:*

1. *if $p_j > p_k$, then $l_j \leq l_k$.*
2. *The two longest codewords have the same length and correspond to the two least likely symbols.*

Proof: The collection of prefix codes is well-ordered under expected lengths of code words. Hence there exists a (not necessarily unique) optimal prefix code. To see (1), suppose C is an optimal prefix code. Let C' be the code interchanging $C(x_j)$ and $C(x_k)$ for some $j < k$ (so that $p_j \geq p_k$). Then

$$\begin{aligned} 0 &\leq L(C') - L(C) \\ &= \sum_i p_i l'_i - \sum_i p_i l_i \\ &= p_j l_k + p_k l_j - p_j l_j - p_k l_k \\ &= (p_j - p_k)(l_k - l_j) \end{aligned}$$

and hence $l_k - l_j \geq 0$, or equivalently, $l_j \leq l_k$.

To see (2), note that if the two longest codewords had differing lengths, a bit can be removed from the end of the longest codeword while remaining a prefix code and hence have strictly lower expected length. An application of (1) yields (2) since it tells us that the longest codewords correspond to the least likely symbols. ■

We claim that Huffman codes are optimal, at least among all prefix codes. Because our proof involves multiple codes, we avoid ambiguity by writing $L(C)$ for the expected length of a code word coded by C, for each C.

Proposition 8.7 *Huffman codes are optimal prefix codes.*

Proof: Define a sequence $\{\mathcal{A}_N\}_{N=2,\ldots,|\mathcal{X}|}$ of sets of source symbols, and associated probabilities $\mathcal{P}_N = \{p_1, p_2, \ldots, p_{N-1}, p_N + p_{N+1} + \cdots + p_{|\mathcal{X}|}\}$. Let C_N denote a huffman encoding on the set of source symbols $\mathcal{A}_N$ with probabilities $\mathcal{P}_N$.

We induct on the size of the alphabets N.

1. For the base case $N = 2$, the Huffman code maps x_1 and x_2 to one bit each and is hence optimal.
2. Inductively assume that the Huffman code C_{N-1} is an optimal prefix code.
3. We will show that the Huffman code C_N is also an optimal prefix code.
   Notice that the code C_{N-1} is formed by taking the common prefix of the two longest codewords (least-likely symbols) in $\{x_1, \ldots, x_N\}$ and allotting it to a symbol with expected length $p_{N-1} + p_N$. In other words, the Huffman tree for the merged alphabet is the merge of the Huffman tree for the original alphabet. This is true simply by the definition of the Huffman procedure. Let l_i denote the length of the codeword for symbol i in C_N and let l'_i denote the length of symbol i in C_{N-1}. Then

   $$\begin{aligned} L(C_N) &= \sum_{i=1}^{N-2} p_i l_i + p_{N-1} l_{N-1} + p_N l_N \\ &= \underbrace{\sum_{i=1}^{N-2} p_i l'_i + (p_{N-1} + p_N) l'_{N-1}}_{L(C_{N-1})} + (p_{N-1} + p_N) \end{aligned}$$

   the last line following from the Huffman construction. Suppose, to the contrary, that C_N were not optimal. Let $\bar{C}_N$ be optimal (existence is guaranteed by previous Lemma). We can take $\bar{C}_{N-1}$ to be obtained by merging the two least likely symbols which have same length by Lemma 8.6. But then

   $$L(\bar{C}_N) = L(\bar{C}_{N-1}) + (p_{N-1} + p_N) \geq L(C_{N-1}) + (p_{N-1} + p_N) = L(C_N)$$

   where the inequality holds since C_{N-1} is optimal. Hence, C_N had to be optimal.

■

Remarks 8.8 *The numbers $p_1, p_2, \ldots, p_N$ need not be probabilities - just weights $\{w_i\}$ taking arbitrary non-negative values. Huffman encoding in this case results in a code with minimum $\sum_i p_i w_i$.*

Remarks 8.9 *Since Huffman codes are optimal prefix codes, they satisfy $H(X) \leq E[l(X)] < H(X) + 1$, same as Shannon code. However, expected length of Huffman codes is never longer than that of a Shannon code, even though for any given individual symbol either Shannon or Huffman code may assign a shorter codelength.*

Remarks 8.10 *Huffman codes are often undesirable in practice because they cannot easily accomodate changing source distributions. We often desire codes that can incorporate refined information on the probability distributions of strings, not just the distributions of individual source symbols (e.g. English language.) The huffman coding tree needs to be recomputed for different source distributions (e.g. English vs French). In next class, we will discuss arithmetic codes which address this issue.*