10-704: Information Processing and Learning **Fall 2016**

Lecture 10: Oct 3

Lecturer: Aarti Singh

Note: *These notes are based on scribed notes from Spring15 offering of this course. LaTeX template courtesy of UC Berkeley EECS dept.*

In last lecture, we introduced source coding and defined non-singular, uniquely decodable and prefix-free (aka instantaneous or self-punctuating) codes. We quickly summarize these. Let X be a random variable characterizing a source taking values in a set $\mathcal{X}$. A symbol code C is a mapping from an alphabet or symbol in $\mathcal{X}$ to another D-ary alphabet. We overload notation and also call C a source code by defining its *extension* over strings of alphabets/symbols as

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

We will use the shorthand notation x^n to denote $x_1x_2\dots x_n$ and let ℓ_x denote the length of the codeword $C(x)$ and $\ell(x^n)$ denote the length of the codeword $C(x^n)$.

Listed below are the types of source codes we saw in class.

Codes	**Description of C**
Nonsingular	C injective – i.e. $\forall x, x' \in \mathcal{X}, x \neq x' \implies C(x) = C(x')$
Uniquely Decodable	The extension of the code is nonsingular.
Prefix/Instantaneous/Self-punctuating	No code word prefixes another: for all distinct $x', x'' \in \mathcal{X}$, $C(x')$ does not start with $C(x'')$

Notice that prefix-free codes are necessary to allow decoding on-the-fly i.e. as soon as a codeword is received, it can be decoded without waiting for the entire sequence of codewords in a transmission to be received.

We also stated the source coding theorem.

Theorem 10.1 (Shannon Source Coding Theorem) *(i) Let x^n be iid drawn from a source distribution p_X. There exists a code that maps sequence x^n of length n into binary strings such that the mapping is one-to-one and $\mathbb{E}[\ell(X^n)/n] \leq H(X) + \epsilon$ where $\epsilon \to 0$ as $n \to \infty$.*
(ii) The expected length of any prefix-free or uniquely decodable D-ary code for a random variable X is greater than or equal to its entropy, i.e. $\mathbb{E}[\ell(X)] \geq H(X)$, with equality iff $D^{-\ell_x} = p_x$.

In last lecture, we saw a proof of the first part (i) on achievability of the source coding limit using a random codebook that mapped sequences in the typical set using short codewords and the sequences not in the typical set by longer codewords. However, notice that

1. This is an asymptotic statement as the entropy limit is achieved only when $n \to \infty$.
2. This code is non-singular for each n, but its extension is not necessarily uniquely decodable. Indeed the code may be completely different for each n since the typical sequences may change. Essentially we have constructed a sequence of codes (one for each n) that achieves the Shannon Rate.

3. This code requires maintaining look up tables exponential in n so this is not necessarily the most practical code construction.

We will consider more practical codes a bit later, but first we show that no uniquely decodable or prefix-free code can achieve an expected codelength shorter than entropy i.e. part (ii) of the source coding theorem.

$$\begin{aligned}
\mathbb{E}[\ell(X)] - H_D(X) &= \sum_x p_x \ell_x - \sum_x p_x \log_D 1/p_x \\
&= -\sum_x p_x \log_D D^{-\ell_x} + \sum_x p_x \log_D p_x = \sum_x p_x \log_D \frac{p_x}{D^{-\ell_x}} \\
&= \sum_x p_x \log \frac{p_x}{q_x} + \sum_x p_x \log_D \frac{1}{\sum_x D^{-\ell_x}} \quad \text{where } q_x = \frac{D^{-\ell_x}}{\sum_x D^{-\ell_x}} \\
&= D(p||q) + \log_D \frac{1}{\sum_x D^{-\ell_x}} \geq 0
\end{aligned}$$

The last step follows using Gibbs inequality and assuming the condition that $\sum_x D^{-\ell_x} \leq 1$. We will show below that this condition is satisfied by both prefix codes and uniquely decodable codes.

To see when equality is achieved, notice that we need both $D(p||q) = 0$ (i.e. $p_x = q_x$) and $\log_D \frac{1}{\sum_x D^{-\ell_x}} = 0$. The latter implies $\sum_x D^{-\ell_x} = 1$ for the optimal codelength (achieving expected codelength equal to entropy) and this, coupled with the former, implies that $p_x = D^{-\ell_x}$.

The above yields that optimal codelengths are $\ell_x = \log_D p_x$, which is the information content of an outcome that occurs with probability p_x. However, since $\log_D p_x$ may not be integer, one can consider the codelengths $\ell_x = \lceil \log_D p_x \rceil$. . Notice that these codelengths satisfy $H_D(X) \leq \mathbb{E}[\ell(X)] \leq H_D(X) + 1$. Later, we will argue that there exists a prefix code called **Shannon code** with these codelengths. This suggests that it is possible to achieve compression within 1 bit of entropy limit, however notice that this overhead is per symbol. If we are encoding lots of symbols, then this overhead adds up quickly. However, the overhead of 1 bit can be spread over multiple symbols by encoding blocks of symbols together as we get:

$$H_D(X^n) \leq \mathbb{E}[\ell(X^n)] \leq H_D(X^n) + 1$$

Or in terms of average expected codelength

$$\frac{H_D(X^n)}{n} \leq \frac{\mathbb{E}[\ell(X^n)]}{n} \leq \frac{H_D(X^n)}{n} + \frac{1}{n}$$

If X^n are n iid draws, then average expected codelength converges to entropy $H_D(X)$. And if X^n are samples from a stationary process, then it converges to the entropy rate of the process $H_D(\mathcal{X})$.

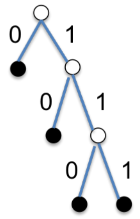
We now argue that the condition $\sum_x D^{-\ell_x} \leq 1$ is satisfied by both prefix codes (Kraft's Inequality) and uniquely decodable codes (McMillian Inequality).

Lemma 10.2 Kraft's Inequality: *If a prefix free code has lengths $\{\ell_x\}_{x\in\mathcal{X}}$ and is D-ary, then*

$$\sum_x D^{-\ell_x} \leq 1$$

Proof: Any prefix code can be thought of as a D-ary tree with code words represented by its leaves and traced by following the path from root to a leaf, as shown in the figure below. No internal nodes can correspond to code words because that would violate the prefix property, since the code for the internal node will be the prefix for the code of any leaf that is a descendant of the internal node. If depth of the tree is $\ell_{\max}$, then if a symbol appears at depth ℓ_x , it rules out $D^{\ell_{\max}-\ell_x}$ possible leaves in the full tree, so $\sum_x D^{\ell_{\max}-\ell_x} \leq D^{\ell_{\max}}$ where $D^{\ell_{\max}}$ is the maximum number of leaves at depth $\ell_{\max}$.

Prefix code
0
10
110
111



∎

We also present an alternate proof: **Proof:** Drop 1 unit of dust at the root of the tree and split it evenly at each split, i.e send half down the left branch and half down the right branch at every split. If a leaf is at depth ℓ_x, it accumulates $D^{-\ell_x}$ dust and by conservation since we only added one unit at the beginning at the root, necessarily $\sum_x D^{-\ell_x} \leq 1$

The converse of the inequality is also true i.e. if we have codelengths satisfying Kraft inequality, then there exists a prefix code with those codelengths. To see this, if $\sum_x D^{-\ell_x} \leq 1$ then build a D-ary tree of depth $\ell_{\max}$ and put the symbols at the correct levels. ∎

Lemma 10.3 McMillan's Inequality *If a uniquely decodable code has lengths $\{\ell_x\}_{x\in\mathcal{X}}$ and is D-ary, then*

$$\sum_x D^{-\ell_x} \leq 1$$

Proof: Since we have already shown that we can construct a prefix code given ℓ_x with $\sum_x D^{-\ell_x} \leq 1$, and prefix-free codes ⊆uniquely decodable codes, we have already shown the converse of McMillan inequality.

For the other direction, we have We will use the key fact that, for uniquely decodable codes, the number of codewords of length ℓ is $n_\ell \leq D^\ell$ by uniqueness. We also note that this is not true for non-singular codes. As a result, we have

$$(\sum_x D^{-\ell(x)})^k = (\sum_{x_1} D^{-\ell(x_1))})(\cdots)(\sum_{x_k} D^{-\ell(x_k))}) =$$

$$= \sum_{x_1}\cdots\sum_{x_k} D^{-\ell(x_1)}\cdots D^{-\ell(x_k)} = \sum_{x^k} D^{-\ell(x^k)} = \sum_{l=1}^{kl_{\max}} n_l D^{-l} \leq \sum_{l=1}^{kl_{\max}} D^l D^{-l} = kl_{\max}$$

Hence $\sum_x D^{-\ell(x))} \leq (kl_{max})^{\frac{1}{k}}$ and this holds for any k. Since the left hand side does not depend on k, it holds in the limit and since $\lim_{k\to\infty}(kl)^{\frac{1}{k}} = 1$, we conclude that

$$\sum_x D^{-\ell(x)} \leq 1$$

∎

We can check that the integers

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

satisfy the Kraft-McMillan Inequality and hence there exists some prefix code C with these codelenghts for which

$$H(X) \leq \mathbb{E}[\ell(X)] < H(X) + 1. \tag{10.1}$$

Such a code is called **Shannon code**.

10.1 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 consider binary codes first, although the procedure described here readily adapts to D-ary setting. 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

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

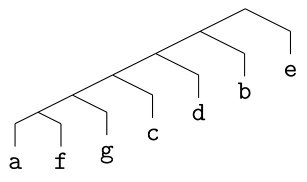
The procedure Huff is defined as follows:

```
**Huff** $(p_1, \dots, p_N)$:
    **if** $N > 2$ **then**
        $C(1) \leftarrow 0$, $C(2) \leftarrow 1$
    **else**
        sort $p_1 \geq p_2 \geq \dots p_N$
        $C' \leftarrow$ Huff$(p_1, p_2, \dots, p_{N-2}, p_{N-1} + p_N)$
        **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 $x_1, \ldots, x_N \in \mathcal{X}$ 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 $i = 1, \ldots, N$. We first observe some properties of general optimal prefix codes.

Lemma 10.4 *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.*
3. *The two longest codewords only differ in the last two bits.*

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 have 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 second part of (2) since it tells us that the longest codewords correspond to the least likely symbols. (3) follows since if there is a maximal-length codeword without a sibling, we can delete the last bit of the codeword and still satisfy the prefix property. This reduces the average codeword length and contradicts the optimality of the code. ■

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 10.5 *Huffman codes are optimal prefix codes.*

Proof: Let $\mathcal{A}_{|\mathcal{X}|} = \{x_1, x_2, \ldots, x_{\mathcal{X}}\}$ be a set of source symbols sorted by probabilities as $p_1 \geq p_2 \geq \ldots p_{|\mathcal{X}|-1} \geq p_{|\mathcal{X}|}$. Define a sequence $\{\mathcal{A}_N\}_{N=2,\ldots,|\mathcal{X}|}$ of sets of source symbols, and associated probabilities $\mathcal{P}_N$ defined recursively starting from $\mathcal{P}_{|\mathcal{X}|} = \{p_1, p_2, \ldots, p_{|\mathcal{X}|}\}$ and then $\mathcal{P}_{N-1} = \{p_{\{1\}}, p_{\{2\}}, \ldots, p_{\{N-2\}}, p_{\{N-1\}} + p_{\{N\}}\}$ for $N = |\mathcal{X}| - 1, \ldots, 3, 2$ where $p_{\{1\}} \geq p_{\{2\}} \geq \cdots \geq p_{\{N-1\}} \geq p_{\{N\}}$ were the ordered probabilities of the set of source symbols in $\mathcal{A}_N$. 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 is formed by taking the longest codeword in C_{N-1} and appending a 0 and 1 to it to get the codewords for the two least likely symbols in $\mathcal{A}_N$. 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 the above observation implies that $l_{N-1} = l_N = l'_{N-1} + 1$. And we have

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

   the last line follows from the definition of the set $\mathcal{A}_{N-1}$ and associated probabilities $\mathcal{P}_{N-1}$. Suppose, that C_N were not optimal. Let $\bar{C}_N$ be optimal. We can take $\bar{C}_{N-1}$ to be obtained by merging the two least likely symbols which have same length by Lemma 10.4. 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. This is a contradiction to the assumed optimality of $\bar{C}_N$. Hence, C_N has to be optimal.

■

Remarks 10.6 *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 10.7 *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 10.8 *Huffman codes are often undesirable in practice because they cannot easily accommodate 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).*