ECE 7680
Lecture 6 – Data Compression

Objective: We will apply what we know of entropy to the problem of data compression. We will introduce and prove the important **Kraft inequality**, Shannon codes, and Huffman codes.

Reading:

1. Read Chapter 5.

We are now ready to use the tools we have been building over the last few weeks to work on the problem of efficient representation of data: data compression. In order the made usable coding representations, we introduce a type of codes known as *instantaneous* codes, which can be decoded without any backtracking. We present the Kraft inequality, which is an important result on the lengths of codewords. Then we show how to achieve a lower bound and introduce Huffman coding.

1 Some simple codes

Definition 1 A **source code** C for a random variable X is a mapping from $\mathcal{X}$ to $\mathcal{D}^*$, the set of finite-length strings of symbols from a D-ary alphabet. Let $C(x)$ denote the codeword corresponding to x and let $l(x)$ denote the length of $C(x)$. □

Example 1 Suppose that $D = 2$, $\mathcal{D} = \{0, 1\}$, and $\mathcal{X} = \{\text{red,blue,green,black,purple}\}$. Then a code is:

- $C(\text{red}) = 1$
- $C(\text{blue}) = 11$
- $C(\text{green}) = 0$
- $C(\text{black}) = 01$
- $C(\text{purple}) = 10$

Suppose we want to send the string of symbols "green red black purple". This would be coded as 010110. Can this be uniquely decoded at the receiver? □

Example 2 Let X be a r.v. with the following distribution and coding assignment:

- $P(X = 1) = 1/2$. Codeword: $C(1) = 0$
- $P(X = 2) = 1/4$. Codeword: $C(2) = 10$
- $P(X = 3) = 1/8$. Codeword: $C(3) = 110$
- $P(X = 4) = 1/8$. Codeword: $C(4) = 111$

The entropy is $H(X) = 1.75$, and the expected codeword length is 1.75 bits, the same as the entropy. Also note that we can uniquely decode a sequence of bits. □

Example 3 Now let the code be assigned as

- $P(X = 1) = 1/2$. Codeword: $C(1) = 0$
- $P(X = 2) = 1/4$. Codeword: $C(2) = 0$

- $P(X = 3) = 1/8$. Codeword: $C(3) = 11$
- $P(X = 4) = 1/8$. Codeword: $C(4) = 111$

In this case, we cannot distinguish between $X = 1$ and $X = 2$. □

Definition 2 A code is said to be **non-singular** is every element of the range of X maps into a different string in $\mathcal{D}^*$. That is, if $x_j \neq x_i$ then $C(x_i) \neq C(x_j)$. □

The last example is a code that is a singular code.

We have met the idea of stringing together a bunch of codes in succession. This has a definition:

Definition 3 An **extension** C^* of a code C is a mapping from finite length strings of $\mathcal{X}$ to finite length strings of $\mathcal{D}$ defined by

$$C(x_1 x_2 \cdots x_n) = C(x_1)C(x_2)\cdots C(x_n),$$

where the RHS is the **concatenation** of the codewords. □

Example 4 If $C(x_1) = 00$ and $C(x_2) = 11$, then $C(x_1 x_2) = 0011$. □

Definition 4 A code is called **uniquely decodable** if its extension is uniquely decodable. □

That is, if we string together a bunch of codewords, we want to be able to tell where one codeword leaves off and another begins. The first example code presented is *not* uniquely decodable.

There may be codes which are uniquely decodable, but in order to do the decoding, the decoder may have to do some look-ahead and some backtracking in order to come up with a unique sequence. In practice, this means the decoding hardware is more complicated, and these kinds of codes are avoided where possible.

Definition 5 A code is called a **prefix code** or an **instantaneous code** if no codeword is a prefix of any other codeword. □

An instantaneous codeword can be decoded without look-ahead, since the end of a codeword is immediately recognizable (it is not the beginning of any other codeword). Instantaneous codes are "self-punctuating."

Example 5 The table below illustrates three different codes assigned to the r.v. X.

X	non-singular, but uniquely decodable	uniq. decod. but not not instant.	instantaneous
1	0	10	0
2	010	00	10
3	01	11	110
4	10	110	111

Take the uniquely-decodable but non-instantaneous code: if the first two bits are 11, then we must look at following bits. If the next bit is a 1 then the first symbol is 3. If the length of the string of 0s following the 11 is odd, then the first codeword must be 110 and the first source symbol must be 4. If the length of the string of 0s is even, the first source symbol must be 3. □

2 Kraft Inequality

In this section we develop an inequality on the lengths of the codewords that is necessary and sufficient for a code to be an instantaneous code.

Theorem 1 *(Kraft inequality) For any instantaneous code over an alphabet of size D, the codeword lengths $l_1, l_2, \ldots, l_m$ must satisfy*

$$\sum_{i=1}^{m} D^{-l_i} \leq 1.$$

Conversely, given a set of codeword lengths that satisfy this inequality there exists an instantaneous code with these word lengths.

Proof Consider a D-ary tree representing the codewords: the path down the tree is the sequence of symbols, and each leaf of the tree (with its unique associate path) corresponds to a codeword. The prefix condition implies that no codeword is an ancestor of any other codeword on the tree: each codeword eliminates its descendants as possible codewords.

Let $l_{\max}$ be the length of the longest codeword. Of all the possible nodes at a level of $l_{\max}$, some may be codewords, some may be descendants of codewords, and some may be neither. A codeword (leaf node) at level l_i has $D^{l_{\max}-l_i}$ descendants at level $l_{\max}$. Each of the descendant sets must be disjoint (because of the tree structure). The total number of possible leaf nodes at level $l_{\max}$ is $D^{l_{\max}}$. Hence, summing over all codewords,

$$\sum_{\text{all codewords}} D^{l_{\max}-l_i} \leq D^{l_{\max}}.$$

That is,

$$\sum D^{-l_i} \leq 1.$$

Conversely, given any set of codewords $l_1, l_2, \ldots, l_m$ which satisfy the inequality, we can always construct a tree. $\square$

3 Optimal codes

We deem a code to be optimal if it has the shortest average codeword length. The goal, after all, is to use the smallest number of bits to send the information. This may be regarded as an optimization problem. In designing the code, we must select the codeword lengths $l_1, l_2, \ldots, l_m$ so that that average length

$$L = \sum_i p_i l_i$$

is as short as possible (less than any other prefix code), subject to the constraint that the lengths satisfy the Kraft inequality (so it will be a prefix code). That is, minimize

$$L = \sum_i p_i l_i$$

subject

$$\sum D^{-l_i} \leq 1.$$

We will make two simplifying assumptions to get started: (1) we will neglect integer constraints on the codelengths; and (2) we will assume Kraft holds with equality. Then we can write a Lagrange-multiplier problem

$$J = \sum_i p_i l_i + \lambda \sum_i D^{-l_i}.$$

Taking derivative with respect to l_j and equating to zero

$$\frac{\partial J}{\partial l_j} = p_j - \lambda D^{-l_j} \log D = 0$$

leads to

$$D^{-l_j} = \frac{p_j}{\lambda \log D}$$

Substituting into the constraint,

$$\sum_i \frac{p_i}{\lambda \log D} = 1$$

so $\lambda = 1/\log D$, and

$$p_i = D^{-l_i}$$

and the optimal codelengths are $l_i^* = -\log_D p_i$. (The * denotes the optimal value.) Under this solution, the minimal average codeword length is

$$L^* = \sum_i p_i l_i^* = H_D(X).$$

(The subscript $_D$ denotes the log with respect to D.)

Of course, in practice the codeword lengths must be integer values, so the result just obtained is a *lower bound* on the average codeword length. We will validate this lower bound in the following theorem:

Theorem 2 *The expected length L of any instantaneous D-ary code for a random variable X satisfies*

$$L \geq H_D(X)$$

with equality if and only if $D^{-l_i} = p_i$.

Proof Write

$$\begin{aligned}
L - H_D(X) &= \sum_i p_i l_i - \sum_i p_i \log_D \frac{1}{p_i} \\
&= -\sum_i p_i \log_D D^{-l_i} + \sum_i p_i \log_D p_i \\
&= \sum_i p_i \log_D \frac{p_i}{D^{-l_i}} \\
&= \sum_i p_i \log_D \frac{p_i}{D^{-l_i}(\sum_j D^{-l_j})/(\sum_j D^{-l_j})}
\end{aligned}$$

Now let $r_i = D^{-l_i}/\sum_j D^{-l_j}$ and $c = \sum_i D^{-l_i}$ we have

$$\begin{aligned}
L - H_D(X) &= \sum_i p_i \log_D \frac{p_i}{r_i} - \log_D c \\
&= D(\mathbf{p} \| \mathbf{r}) + \log_D \frac{1}{c} \\
&\geq 0
\end{aligned}$$

since relative entropy is nonnegative and $c \leq 1$ (Kraft inequality). □

4 Bounds on the optimal code

The theorem just proved shows that the length must be greater than $H_D(X)$. We can now prove that a physically implementable instantaneous code (that is, a code with integer codeword lengths), we can find an upper bound on the code length:

$$H(X) \leq L < H(X) + 1.$$

That is, the overhead due to the integer codeword length it not more than one bit.

The codeword lengths are found by

$$l_i = \lceil \log_D \left(\frac{1}{p_i}\right) \rceil$$

where $\lceil x \rceil$ is the smallest integer $\geq x$. These codeword lengths satisfy the Kraft inequality:

$$\sum_i D^{-l_i} = \sum_i D^{-\lceil \log \frac{1}{p_i} \rceil} \leq \sum_i D^{-\log \frac{1}{p_i}} = \sum_i p_i = 1$$

The codewords lengths satisfy

$$\log_D \frac{1}{p_i} \leq l_i < \log_d \frac{1}{p_i} + 1$$

Taking the expectation through we get

$$H_D(X) \leq L < H_D(X) + 1.$$

The next trick is to reduce the overhead (of up to one bit) by spreading it over several symbols. Suppose we are sending a sequence of symbols, drawn independently according to the distribution $p(x)$. A sequence of n symbols can be regarded as a symbol from the alphabet $\mathcal{X}^n$.

Let L_n be the expected codeword length per input symbol:

$$L_n = \sum_{\mathbf{x} \in \mathcal{X}^n} p(x_1, x_2, \ldots, x_n) l(x_1, x_2, \ldots, x_n) = \frac{1}{n} E(X_1, X_2, \ldots, X_n).$$

Applying the inequality to the codelengths:

$$H(X_1, X_2, \ldots, X_n) \leq El(X_1, X_2, \ldots, X_n) \leq H(X_1, X_2, \ldots, X_n) + 1$$

Since the symbols are i.i.d., $H(X_1, X_2, \ldots, X_n) = nH(X_1)$. Dividing through by n we obtain

$$H(X) \leq L_n \leq H(X) + \frac{1}{n}.$$

By choosing the block size sufficiently large, the average code length can be made arbitrarily close to the entropy.

The next observation is that if the symbols are *not* independent, we can still write

$$H(X_1, X_2, \ldots, X_n) \leq El(X_1, X_2, \ldots, X_n) \leq H(X_1, X_2, \ldots, X_n) + 1.$$

Dividing through by n we obtain

$$\frac{H(X_1, X_2, \ldots, X_n)}{n} \leq L_n \leq \frac{H(X_1, X_2, \ldots, X_n)}{n} + \frac{1}{n}.$$

If X is a stationary stochastic process, then taking the limit yields

$$L_n \to H(\mathcal{X}).$$

Another question is what if the distribution used to design the codes is not the same as the actual distribution? Consider the code designed by $l(x) = \lceil \frac{1}{q(x)} \rceil$, for the distribution $q(x)$, while the true distribution is $p(x)$.

Theorem 3 *The expected length under $p(x)$ of the code designed under $l(x) = \lceil \frac{1}{q(x)} \rceil$ satisfies*

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

That is, the mistaken distribution costs us an extra $D(p\|q)$ bits per symbol to code.
Proof

$$\begin{aligned}
El(X) &= \sum_x p(x) \lceil \log \frac{1}{q(x)} \rceil \\
&< \sum_x p(x) \left(\log \frac{1}{q(x)} + 1 \right) \\
&= \sum_x p(x) \log \frac{p(x)}{q(x)} \frac{1}{p(x)} + 1 \\
&= p(x) \log \frac{p(x)}{q(x)} + \sum_x p(x) \log \frac{1}{p(x)} + 1 \\
&= D(p\|q) + H(p) + 1.
\end{aligned}$$

The lower bound is similar. □

5 Kraft inequality for uniquely decodable codes

The set of instantaneous codes is smaller than the set of uniquely decodable codes, so we might think that we might be able to obtain a lower average codeword L for uniquely decodable codes. However, the point of this section is that this is not the case. Hence, we may as well just use instantaneous codes, since they are easier to decode.

6 Huffman codes

Huffman codes are the optimal prefix codes for a given distribution. What's more, if we know the distribution, Huffman codes are easy to find. The code operates from the premise of assigning longer codewords to less-likely symbols, and doing it in a tree-structured way so that the codes obtained are prefix-free.
Example 6 Consider the distribution X taking values in the set $\mathcal{X} = \{1, 2, 3, 4, 5\}$ with probabilities $.25, .25, .2, .15, .15$, respectively.

At each stage of the development, we combine the two least-probable symbols (in this case, for a binary code) into one symbol.

1. At the first round, then, the .15 and .15 are combined to form a symbol with probability .3. The set of probabilities (in ordered form) is .3, .25, .25, .2.

2. Now combine the lowest two probabilities: $.2 + .25 = .45$. The ordered list of probabilities is .45, .3, .25.

3. Combine the two lowest probabilities: .25 + .3 = .55. The ordered list of probabilities is .55, ..45.

4. Combine these to obtain the total probability of 1. We are done!

Now assign codewords on the tree. The average codelength is 2.3 bits. □

Codes with more than $D = 2$ symbols can also be built, as described in the book.

Proving the optimality of the Huffman code begins with the following simple lemma:

Lemma 1 *For any distribution, there exists an optimal instantaneous code (of shortest average length) that satisfies the following properties:*

1. *If $p_j > p_k$, then $l_j \leq l_k$.*

2. *The two longest codewords have the same length.*

3. *The two longest codewords differ only in the last bit, and correspond to the two least-likely symbols.*

Proof (Sketch)

1. Simply swap lengths.

2. If the two longest are not of the same length, trim the longer: still a prefix code.

3. If not siblings on the tree (i.e., do not differ just in one bit), then we can remove a bit from the longest code, which contradicts the optimality property.

□

For a code on m symbols, assume (w.o.l.o.g.) that the probabilities are ordered $p_1 > p_2 > \cdots > p_m$. Define the merged code on $m - 1$ symbols by merging the two least probable symbols p_m, p_{m-1}. The codeword on this merged symbol is the common prefix on the two least-probable (longest) codewords, which, by the lemma, exists. The expected length of the code C_m is

$$\begin{aligned} L(C_m) &= \sum_{i=1}^{m} p_i l_i \\ &= \sum_{i=1}^{m-2} p_i l_i + p_{m-1}(l_{m-1} + 1) + p_m(l_m + 1) \\ &= \sum_{i=1}^{m-2} p_i l_i + p_{m-1}(l_{m-1} + 1) + p_m(l_m + 1) \\ &= \sum_{i=1}^{m-1} p_i l_i + p_m(l_m + 1) \\ &= L(C_{m-1}) + p_m(l'_m + 1). \end{aligned}$$

The optimization problem on m symbols has been reduced to an optimization problem on $m - 1$ symbols. Proceeding inductively, we get down to two symbols, for which the optimal code is obvious: 0 or 1.

7 Arithmetic coding

Go over idea of arithmetic coding. Start with $p_0 = .75, p_1 = .25$ and explain procedure for encoding and decoding. Then take case of $p_0 = 0.7$ and $p_1 = 0.3$. Generalize to multiple (≥ 2) symbols.

Discuss numerical problems.

Handouts.

8 Run-length coding

Suppose you want to encode the information on a regular fax machine. There are two possible outputs, white or black. Most paper consists of a white background with black lettering, so the proportion of white letters tends to be quite large. However, when black appears, it often appears as a run.

One potential way of doing data compression on such a source is by means of run-length coding. Sequences of runs are encoded into a count. We have to worry about how long the sequences can be, so there is a maximum run-length allowable.

What fax machines actually do is run-length coding following by Huffman coding. This is called Modified Huffman Coding.

9 Lempel-Ziv Coding

Lempel-Ziv coding (and its variants) is a very common data compression scheme. It is based upon the source building up a dictionary of previously-seen strings, and transmitting only the "innovations" while creating new strings.

For the first example, we start off with an empty dictionary, and assume that we know (somehow!) that the dictionary will not contain more than 8 symbols. Suppose we are given the string

1011010100010...

The source stream is parsed until the shortest string is encountered that has not been encountered before. Since this is the shortest such string, all of its prefixes must have been sent before. The string can be coded by sending the index from the dictionary of the prefix string and the new bit. This string is then added to the dictionary.

To illustrate, 1 has not been seen before, so we send the index of its prefix (set at 000), then the number 1. We add the sequence 1 to the dictionary. Then 0 has not been seen before, so we send the index of its prefix (000) and the number 0. We add the sequence 0 to the dictionary. The sequence 11 has a prefix string of 1, so we send its index, and the number 1. Proceeding this way, the dictionary looks like this:

The Lempel-Ziv string dictionary

index	contents
000	null
001	1
010	0
011	11
100	01
101	010
110	00
111	10

The encoded sequence is

$$(000,1)(000,0)(001,1)(010,10)(100,0)(010,0)(001,0)$$

Observe that we haven't compressed the data: 13 bits went in, and 28 bits came out. But if we had a longer stream of data (and a bigger dictionary), data compression could result. In fact, the point that made fame and fortune for Lempel and Ziv is that they proved that asymptotically, the output rate approaches the entropy rate for a stationary source. (We could spend a year working through their proof.) The bottom line is the following theorem:

Theorem 4 *Let $\{X_i\}_{-\infty}^{\infty}$ be a stationary ergodic stochastic process. Let $l(X_1, X_2, \ldots, X_n)$ be the Lempel-Ziv codeword length associated with $X_1, X_2, \ldots, X_n$. Then*

$$\limsup_{n \to infty} \frac{1}{n} l(X_1, X_2, \ldots, X_n) \leq H(\mathcal{X}) \quad \textit{with probabilility } 1.$$

An obvious improvement on this basic explanatory example is to use fewer bits at the beginning to send the index information, because the table is known not to need that many bits yet.

There are several implementation issues to refine the performance, including:

- Compression performance vs. table size.
- Dictionary initialization: start empty, or start with the alphabet.
- Adaptive index sizing
- Dictionary organization and search strategies (speed vs. storage)
- Adaptation: throw away sequences that are rarely used.

Here is another example on a three-symbol alphabet $\{a, b, c\}$. In this case, the dictionary is assumed to be initialized to contain the alphabet of the source. The sequence is

$$ababcbababaaaaa...$$

The output is

$$(1, b)(4, c)(2a)(6, b)(1, a)(8, a)...$$

and the dictionary is

The Lempel-Ziv string dictionary

index	contents
1	a
2	b
3	c
4	ab
5	abc
6	ba
7	bab
8	aa
9	aaa

10 Adaptive Huffman coding

The Huffman coding algorithm works when the source is stationary and the probabilities are known. In the circumstance in which the source is non-stationary or the probabilities are not known in advance, then the adaptive Huffman coding algorithm is a possibility. In this case, the relative probabilities of the symbols are estimated by keeping counts of the occurrences of the source symbols. When the counts reach a point that the tree is no longer optimal, it is shifted to provide a new Huffman code. Since the operation can take place simultaneously at both the transmitter and the receiver, decoding can take place. A paper will be handed out describing the technique. (This might make a good paper project.) It is believed (by me) that the method is flawed.