

Symbol Codes

Mário S. Alvim
(msalvim@dcc.ufmg.br)

Information Theory

DCC-UFMG
(2017/02)

Symbol Codes - Introduction

- We have seen that the Shannon's Source Coding Theorem states that if
 1. you are encoding N symbols of a source X , and
 2. you are willing to accept a probability δ of error in the decompression, then
 3. the maximum achievable compression will use approximately $H(X)$ bits per symbol if the number of N of symbols being encoded is large enough.

Symbol Codes - Introduction

- Note that Shannon's Theorem assumes that:
 - symbols are encoded in blocks, and
 - some probability $\delta \geq 0$ of error is tolerated.
- In this lecture we will consider variable-length **symbol codes** in which
 - each symbol is encoded individually (as opposed to blocks of N symbols being encoded together), and
 - the encoding is lossless: it is guaranteed that compression and decompression occur without any errors.

Symbol Codes - Introduction

- As we already know, lossless compression implies that the shortening of some encodings is always accompanied by the lengthening of others.

Hence, the following questions rise naturally.

1. What are the implications if a symbol code is lossless?

(If some codewords are shortened, we want to determine by how much do other codewords have to be lengthened.)

2. How can we ensure that a symbol code is easy to decode?

(We want to make compression practical.)

3. How should we assign codelengths to achieve the best compression, and what is the best achievable compression?

(We want to find optimal symbol codes.)

Symbol Codes - Introduction

- At the end of this lecture, when the above questions have been studied and answered, we will have learned the following points.
 1. The optimal symbol coding uses about $H(X)$ bits per symbol encoded.
(So once more Shannon entropy arises as a measure of information.)
 2. The simple **Huffman coding** algorithm achieves optimal symbol-code compression.

- **Notation for alphabets.**

- \mathcal{A}^N denotes the set of ordered N -tuples of elements from the set \mathcal{A} , that is, strings of length N formed by symbols in \mathcal{A} .
- \mathcal{A}^+ denotes the set of all strings of finite length composed of elements from the set \mathcal{A} .

- For example:

① $\{0, 1\}^3 = \{000, 001, 010, 011, 100, 101, 110, 111\}.$

② $\{0, 1\}^+ = \{0, 1, 00, 01, 10, 11, 000, 001, \dots\}.$

Symbol Codes

Symbol Codes

- A **symbol code** C for an ensemble X is a mapping (i.e., a function) from

$$\mathcal{A}_X = a_1, \dots, a_I,$$

which is the range of x , to a set

$$\{0, 1, 2, 3, \dots, D-1\}^+.$$

- We will mostly be interested in binary codes, in which $D = 2$.

A **binary symbol code** C for an ensemble X is a mapping (i.e., a function) from $\mathcal{A}_X = a_1, \dots, a_I$, the range of x , to $\{0, 1\}^+$.

- $c(x)$ denotes the codeword corresponding to x .
- $\ell(x)$ denotes the length of the codeword corresponding to x , with $\ell_i = \ell(a_i)$.
- The **extended code** C^+ is a mapping from \mathcal{A}_X^+ to $\{0, 1\}^+$ obtained by concatenation, without punctuation, of the corresponding codewords:

$$c^+(x_1, x_2, \dots, x_N) = c(x_1)c(x_2) \cdots c(x_N)$$

Symbol Codes

- Example 1 Let X be the ensemble in which

$$\mathcal{A}_X = \{a, b, c, d\}, \quad \text{and} \quad \mathcal{P}_X = \left\{ \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{8} \right\}.$$

Consider the symbol code C_0 for the ensemble X defined as

	a_i	$c(a_i)$	l_i
C_0 :	a	1000	4
	b	0100	4
	c	0010	4
	d	0001	4

Using the extended code, we may encode $acdbac$ as

$$c^+(acdbac) = 100000100001010010000010.$$



- A code C is **uniquely decodeable** if, under the extended code C^+ , no two distinct strings have the same encoding, i.e.,

$$\forall x, y \in \mathcal{A}_X^+ : \quad x \neq y \quad \implies \quad c^+(x) \neq c^+(y).$$

- Example 2 Are the following codes uniquely decodeable?

- 1 The code $C_0 = \{1000, 0100, 0010, 0001\}$ from the previous example?

Yes.

- 2 The code $C'_0 = \{0, 1, 01, 10\}$?

No. The encoding 01 can be decoded as “ab” or simply as “c”.



Symbol Codes

- A code C is easier to decode if no codeword is a prefix of any other codeword.

This is because, in this case, as soon as we find a codeword that is in the code, we can decode it, as we know that it cannot be a prefix of any other codeword.

Such codes are called **prefix-free** codes, **instantaneous** codes, or **self-punctuating** codes.

- Prefix codes can always be represented as a tree.
- Example 3

 Check whether each of the codes below is prefix-free.

❶ $C_1 = \{0, 101\}.$

❷ $C_2 = \{1, 101\}.$

❸ $C_3 = \{00, 01, 10, 11\}.$

❹ $C_4 = \{0, 10, 110, 111\}.$



- The **expected length** $L(C, X)$ of a code C for an ensemble X is

$$L(C, X) = \sum_{x \in \mathcal{A}_X} p(x) \ell(x).$$

Denoting $|\mathcal{A}_X|$ by I we can also use

$$L(C, X) = \sum_{i=1}^I p_i \ell_i.$$

- Example 4

Let X be the ensemble in which

$$\mathcal{A}_X = \{a, b, c, d\}, \quad \text{and}$$

$$\mathcal{P}_X = \left\{ \frac{1}{2}, \frac{1}{4}, \frac{1}{8}, \frac{1}{8} \right\},$$

and consider the codes C_4 and C_5 .

(Note that C_4 is prefix-free, whereas C_5 isn't.)

The expected length of each code is as follows.

$$L(C_4, X) = \frac{1}{2} \cdot 2 + \frac{1}{4} \cdot 2 + \frac{1}{8} \cdot 2 + \frac{1}{8} \cdot 2 = 2 \text{ bits}$$

$$L(C_5, X) = \frac{1}{2} \cdot 1 + \frac{1}{4} \cdot 1 + \frac{1}{8} \cdot 2 + \frac{1}{8} \cdot 2 = 1.25 \text{ bits}$$

	C_4	C_5
a	00	0
b	01	1
c	10	00
d	11	11

- Example 5 Consider the code C_6 below.

C_6 :

a_i	$c(a_i)$	p_i	$h(p_i)$	l_i
a	0	$1/2$	1.0	1
b	01	$1/4$	2.0	2
c	011	$1/8$	3.0	3
d	111	$1/8$	3.0	3

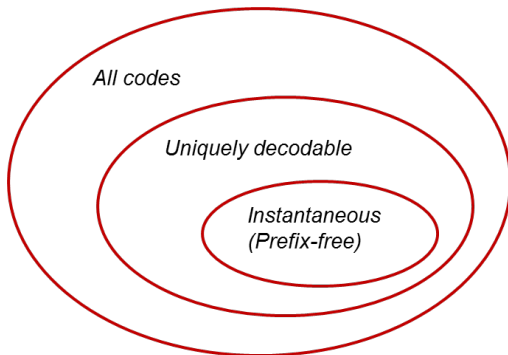
Is C_6 prefix-free? No!

Is C_6 uniquely decodeable? Yes! Why?

The expected code length of this code is $L(C_6, X) = 1.75$ bits.



- Types of codes



Symbol Codes

- Example 6 Consider the symbol codes below.

x	C_7 : Not uniquely decodable	C_8 : Uniquely decodable, but not instantaneous	C_9 : Instantaneous
a_1	0	10	0
a_2	010	00	10
a_3	01	11	110
a_4	10	110	111

Note that:

- In C_7 , 010 can be decoded as a_2 or a_1a_4 .
- In C_8 , 11011 is decoded uniquely as a_4a_3 , but you cannot be sure that the first three bits really represent a_4 until processing the codeword further (in this case, until the end).
- C_9 is instantaneous because it is prefix-free.



Limits of instantaneous codes

Symbol Codes - Kraft inequality

- If we want to use instantaneous codes only, what type of property we must ensure that our codes have?
- **Theorem Kraft inequality.** For any instantaneous (prefix-free) code $C(X)$ over an alphabet having D symbols, the codelengths must satisfy

$$\sum_{i=1}^I D^{-\ell_i} \leq 1,$$

where $I = |\mathcal{A}_X|$.

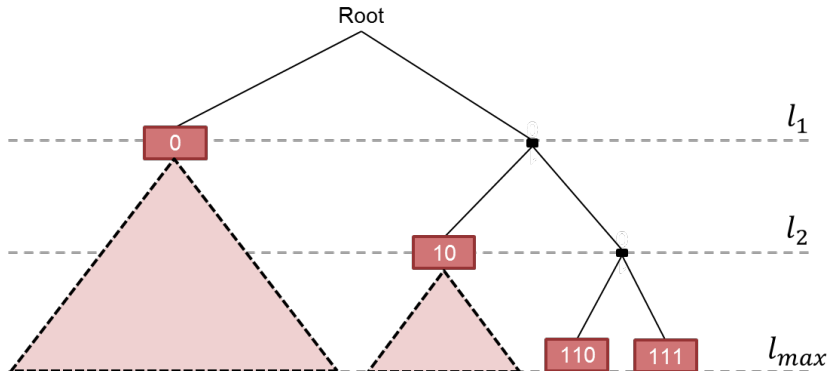
Note that if our code uses a binary alphabet $\{0, 1\}$, we have $D = 2$ and the inequality becomes

$$\sum_{i=1}^I 2^{-\ell_i} \leq 1.$$

Symbol Codes - Kraft inequality

- **Proof.** If ℓ_{\max} is the maximum length of a codeword in our code, the code tree can have at most $D^{\ell_{\max}}$ leaves.

Since the code is prefix-free, each codeword eliminates all of its descendants from the tree.



Symbol Codes - Kraft inequality

- **Proof.** (Continued)

A codeword at level ℓ_i eliminates $D^{\ell_{\max}-\ell_i}$ leaves.

The sum of the leaves eliminated by each codeword cannot exceed the maximum number $D^{\ell_{\max}}$ of leaves in the tree.

Hence

$$\sum_i D^{\ell_{\max}-\ell_i} \leq D^{\ell_{\max}},$$

and dividing both sides by $D^{\ell_{\max}}$ we get

$$\sum_i D^{-\ell_i} \leq 1.$$



Symbol Codes - Kraft inequality

- We just saw that if a code C is instantaneous, then the choices of lengths in this code must satisfy the Kraft inequality.
- The next theorem shows that the converse is also true: if we choose the lengths ℓ_i of codewords so that they satisfy the Kraft inequality, we are guaranteed to be able to find a prefix-code for our source.
- **Theorem Kraft inequality and prefix codes.** Given a set of codeword lengths that satisfy the Kraft inequality, there exists a uniquely decodable prefix code with these codeword lengths.

Symbol Codes - Kraft inequality

- **Proof.** (Sketch)

0	00	000	0000	The total symbol code budget
			0001	
		001	0010	
			0011	
	01	010	0100	
			0101	
		011	0110	
			0111	
1	10	100	1000	
			1001	
		101	1010	
			1011	
	11	110	1100	
			1101	
		111	1110	
			1111	

Symbol Codes - Kraft inequality

• Proof. (Continued)

$$C_0$$

0	00	000	0000
			0001
		001	0010
			0011
	01	010	0100
			0101
		011	0110
			0111
1	10	100	1000
			1001
		101	1010
			1011
	11	110	1100
			1101
		111	1110
			1111





$$C_3$$

0	00	000	0000
			0001
		001	0010
			0011
	01	010	0100
			0101
		011	0110
			0111
	10	100	1000
			1001
		101	1010
			1011
1	11	110	1100
			1101
		111	1110
			1111

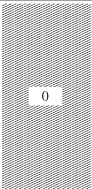



Symbol Codes - Kraft inequality

• Proof. (Continued)

C_4

0		000	0000
			0001
		001	0010
			0011
		010	0100
			0101
		011	0110
			0111
1		100	1000
			1001
		101	1010
			1011
		110	1100
			1101
		111	1110
			1111

C_6

	00	000	0000
			0001
		001	0010
			0011
		010	0100
			0101
			0110
			0111
1	10	100	1000
			1001
		101	1010
			1011
	11	110	1100
			1101
			1110
			1111

The maximum compression possible

Symbol Codes - Optimal codes

- An **optimal code** C for an ensemble X has an average codelength $L(C, X)$ that is no greater than any other alternative code C' for the same ensemble.

Formally:

$$C \text{ is optimal for } X \quad \text{iff} \quad \forall C' : L(C, X) \leq L(C', X).$$

- We can set a goal of
 1. finding an optimal code to achieve the best compression possible, and
 2. that our optimal code be easy to decode (for that we must ensure that our code satisfies the Kraft inequality).

The problem of finding an optimal code can be framed as an optimization problem.

Symbol Codes - Optimal codes

- An optimal binary code can be found as the solution of the following optimization problem.

Minimize the choice of ℓ_i in

$$L = \sum_i p_i \ell_i$$

subjected to

$$\sum_i 2^{-\ell_i} \leq 1.$$

The solution of this problem is...

$$\ell_i = \log_2 \frac{1}{p_i}.$$

Symbol Codes - Optimal codes

- That means that the optimal solution is to make the length of a codeword ℓ_i equal to the amount of information

$$\ell_i = \log_2 \frac{1}{p_i} = h(a_i)$$

carried by the symbol being compressed!

In other words, the optimal choice is to attribute:

- longer codewords to symbols that carry a lot of information, and
 - shorter codewords to symbols that carry little information!
- As a consequence, we get that the expected code length $L(C, X)$ of any code C must respect

$$L(C, X) = \sum_i p_i \ell_i \geq \sum_i p_i \log_2 \frac{1}{p_i} = H(X),$$

which means that it cannot be smaller than the entropy $H(X)$ of the source!

Symbol Codes - Optimal codes

- Now we know that an optimal code achieves compression using $H(X)$ bits per symbol, but for that it must attribute a codewords of length exactly

$$\ell_i = \log_2 \frac{1}{p_i}$$

to each symbol.

- But it is not always possible to attribute a codelength of $\ell_i = \log_2 1/p_i$ to every symbol (damn fractions!).
- Taking that into consideration, what is an upper bound on the compression we can actually get?

As we will see, we can always build a code that uses at most $H(X) + 1$ bits per symbol!

Symbol Codes - Optimal codes

- **Theorem Source coding theorem for symbol codes.** For an ensemble X there exists a prefix code C with expected length satisfying

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

The average length is equal to the entropy $H(X)$ only if the codelength for each outcome is equal to its Shannon information content.

Proof. We already showed that for every code C we must have $L(C, X) \geq H(X)$.

To show that there is a C that satisfies $L(C, X) < H(X) + 1$, let us show that if we pick for every symbol a_i a codeword with length $\ell_i = \lceil \log_2 \frac{1}{p_i} \rceil$, we can achieve a compression of at most $H(X) + 1$ bits per symbol.

Symbol Codes - Optimal codes

- **Proof.** (Continued)

To show this, we follow two steps:

Step 1. We see that our choice of codelengths actually gives us a prefix-free code by verifying that it satisfies the Kraft inequality:

$$\sum_i 2^{-\lceil \log_2 \frac{1}{p_i} \rceil} \leq \sum_i 2^{-\log_2 \frac{1}{p_i}} = \sum_i p_i = 1.$$

Step 2. We verify that our code indeed has codelength no greater than $H(X) + 1$:

$$L(C, X) = \sum_i p_i \left\lceil \log_2 \frac{1}{p_i} \right\rceil < \sum_i p_i \left(\log_2 \frac{1}{p_i} + 1 \right) = H(X) + 1.$$



Optimal symbol codes: Huffman coding

- So far we have seen that:
 1. Every symbol code C must have codelength $L(C, X)$ of at least $H(X)$ bits per symbol.
 2. For every ensemble X there exists a prefix-free code that uses at most $H(X) + 1$ bits per symbol.
- Now we will present an algorithm to find a code achieving this level of compression: the **Huffman code**.

Huffman coding

- The **Huffman coding algorithm** builds a prefix-free code as a tree in a bottom-up manner (from the leaves to the root).

The algorithm follows two steps:

1. Take the two least probable symbols in the alphabet.

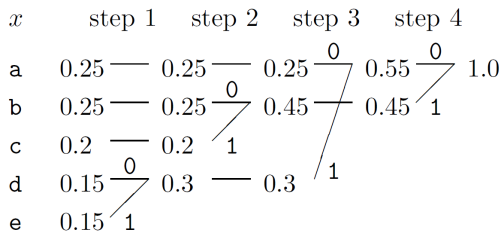
These two symbols will be given the longest codewords, which will have equal length, and differ only in the last digit.

2. Combine these two symbols into a single symbol, and repeat.

Huffman coding

- Example 7 Let

$$\mathcal{A}_X = \{a, b, c, d, e\} \quad \text{and} \quad \mathcal{P}_X = \{0.25, 0.25, 0.2, 0.15, 0.15\}.$$



a_i	p_i	$h(p_i)$	l_i	$c(a_i)$
a	0.25	2.0	2	00
b	0.25	2.0	2	10
c	0.2	2.3	2	11
d	0.15	2.7	3	010
e	0.15	2.7	3	011

The Huffman coding achieves an average code length of $L = 2.30$ bits per symbol, whereas the Shannon entropy of the source is $H = 2.28$ bits per symbol.



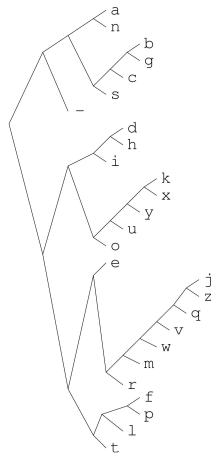
Huffman coding

- Example 8

 For the English language, a Huffman has average length $L = 4.15$ bits, whereas the entropy of the source is $H = 4.11$ bits.

a_i	p_i	$\log_2 \frac{1}{p_i}$	l_i	$c(a_i)$
a	0.0575	4.1	4	0000
b	0.0128	6.3	6	001000
c	0.0263	5.2	5	00101
d	0.0285	5.1	5	10000
e	0.0913	3.5	4	1100
f	0.0173	5.9	6	111000
g	0.0133	6.2	6	001001
h	0.0313	5.0	5	10001
i	0.0599	4.1	4	1001
j	0.0006	10.7	10	1101000000
k	0.0084	6.9	7	1010000
l	0.0335	4.9	5	11101
m	0.0235	5.4	6	110101
n	0.0596	4.1	4	0001

a_i	p_i	$\log_2 \frac{1}{p_i}$	l_i	$c(a_i)$
o	0.0689	3.9	4	1011
p	0.0192	5.7	6	111001
q	0.0008	10.3	9	110100001
r	0.0508	4.3	5	11011
s	0.0567	4.1	4	0011
t	0.0706	3.8	4	1111
u	0.0334	4.9	5	10101
v	0.0069	7.2	8	110100001
w	0.0119	6.4	7	1101001
x	0.0073	7.1	7	1010001
y	0.0164	5.9	6	101001
z	0.0007	10.4	10	1101000001
-	0.1928	2.4	2	01



Huffman coding

- As we have seen, the Huffman coding algorithm works bottom-up, building the code tree from the leaves up to the root.

In our first examples (e.g., the weighing problem) we created trees top-down (from the root to the leaves).

- Could we find an optimal code by building the code tree *top-down*?

More precisely, in a top-down approach we start from the root and at each step:

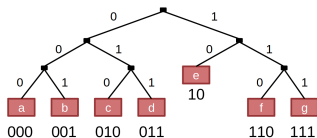
1. we divide the set of possible symbols into two sets with as even probability as possible;
2. we then create a subtree for each set, labeling one with 0 and another with 1;
3. we repeat the process at each subtree.

The the following example shows that the top-down approach is not guaranteed to be optimal.

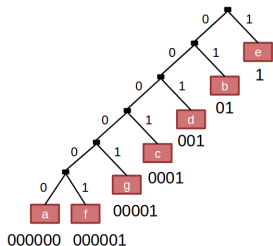
Huffman coding

- Example 9** Let $\mathcal{A}_X = \{a, b, c, d, e, f, g\}$, and $\mathcal{P}_X = \{0.01, 0.24, 0.05, 0.20, 0.47, 0.01, 0.02\}$.

Top-down code:



Huffman code:



a_i	p_i	Greedy	Huffman
a	.01	000	000000
b	.24	001	01
c	.05	010	0001
d	.20	011	001
e	.47	10	1
f	.01	110	000001
g	.02	111	00001

The top-down code has an average bit-length 2.53, whereas the Huffman code has an average bit-length 1.97.

Huffman coding - Limitations

- The Huffman coding assumes the distribution p on the ensemble X (the source) is unchanging over time.

If the frequency of symbols varies over time, the code is no longer optimal.

- The Huffman code achieves a code length $H(X) \leq L(C, X) < H(X) + 1$, so it may waste almost a bit per symbol.

If the entropy $H(X)$ of the source is low enough (in some texts it is lower than 1 bit per symbol), the coding may be wasting a large proportion of the bits per symbol used.

- The Huffman code is an optimal symbol code.

It is, hence, a good choice among symbol codes.

But in general we don't want symbol codes, and we will cover stream codes in the next lecture.

Kullback-Liebler divergence, a.k.a. Relative entropy

Kullback-Liebler divergence a.k.a. Relative entropy

- We saw that the optimal coding attributes a codeword with length

$$\ell_i = \log_2 \frac{1}{p_i}$$

to each symbol.

Note that to create such an optimal code we must know the probability distribution p on symbols on the source.

But what if we don't know p exactly?

If we believe that the probability distribution on symbols is q and build a code in which

$$\ell_i = \log_2 \frac{1}{q_i},$$

when in fact the real distribution of the source is p , we will create a code that is not optimal.

Let us calculate how bad this code is.

Kullback-Liebler divergence a.k.a. Relative entropy

- If we assume the probability distribution on symbols is q when it is actually p , we will attribute length $\ell_i = \log_2 \frac{1}{q_i}$ to each symbol and obtain a code C in which

$$\begin{aligned} L(C, X) &= \sum_i p_i \log_2 \frac{1}{q_i} = \sum_i p_i \log_2 \frac{p_i}{p_i q_i} \\ &= \sum_i p_i \log_2 \frac{1}{p_i} + \sum_i p_i \log_2 \frac{p_i}{q_i} \\ &= H(X) + \sum_i p_i \log_2 \frac{p_i}{q_i}, \end{aligned}$$

where the last term in the sum,

$$D_{KL}(p \parallel q) = \sum_i p_i \log_2 \frac{p_i}{q_i},$$

is named the **Kullback-Liebler divergence** (KL-divergence) or **relative entropy** between two probability distributions p and q .

Kullback-Liebler divergence a.k.a. Relative entropy

- As a sanity check, we can derive the KL-divergence by computing the difference between the expected length $L(C, X)$ when C is constructed according to the wrong distribution q and the optimal code length $H(X)$ following the real distribution p :

$$\begin{aligned} D_{KL}(p \parallel q) &= L(C, X) - H(X) \\ &= \sum_i p_i \log_2 \frac{1}{q_i} - \sum_i p_i \log_2 \frac{1}{p_i} \\ &= \sum_i p_i \log_2 \frac{p_i}{q_i}. \end{aligned}$$

Kullback-Liebler divergence a.k.a. Relative entropy

- The KL-divergence between two probability distributions p and q represents the wasted bits when you encode an ensemble believing it has distribution q when it actually has distribution p .

- **Gibb's inequality:** $D_{KL}(p \parallel q)$ is always non-negative:

$$D_{KL}(p \parallel q) \geq 0,$$

with equality $D_{KL}(p \parallel q) = 0$ only when $p = q$.

- Also the KL-divergence is not symmetric, since in general:

$$D_{KL}(p \parallel q) \neq D_{KL}(q \parallel p).$$

- The KL-divergence appears in various problems in information theory, and we will run into it a few more times in this course.