Information Theory: Lecture Notes 4

zqy1018

2020. 3. 22

Contents

1 Source Code

1.1 Basic Definition

We first give a formal definition about code.

Definition. For a random variable X, a **source code** $C : \mathcal{X} \mapsto \mathcal{D}^*$ is a function mapping from $\mathcal{X}$ to the set of finite-length strings of symbols from a D-ary alphabet (i.e. the size of $\mathcal{D}$ is D). So for $x \in \mathcal{X}$, $C(x)$ denotes the **codeword** corresponding to x. And let $l(x)$ denote the length of $C(x)$. The **expected length** $L(C)$ of C is defined as

$$L(C) = \sum_{x \in \mathcal{X}} p(x) l(x)$$

Note. Without loss of generality, we let the D-ary alphabet be $\{0, 1, \cdots, D-1\}$.

Since we usually encode more than one character, we define the extension of C.

Definition. The **extension** $C^* : \mathcal{X}^* \mapsto \mathcal{D}^*$ of C is a mapping from finite-length strings of 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 $C(x_1)C(x_2)\cdots C(x_n)$ indicates concatenation of the corresponding codewords.

1.2 Types of Source Code

Definition. C is **nonsingular** if C is injective. That is,

$$\forall x \neq x', C(x) \neq C(x')$$

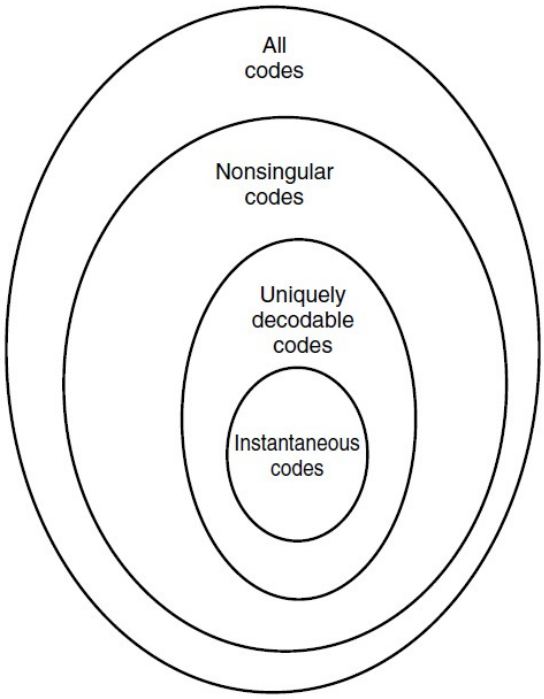
C is **uniquely decodable** if C^* is nonsingular.

Definition. A code is called a **prefix code** (or an **instantaneous code**) if no codeword is a prefix of any other codeword. Similarly we can define the **suffix code**.

Note. Here we use the concept "prefix" and "suffix" without defining it just for convenience.

Remark. The prefix code is called instantaneous because when a codeword stops, it can be decoded instantaneously. Also it is *self-punctuating*, i.e. we do not need to split the message with special characters so that it can be decoded.

The relationship between all the codes is shown below. We will focus on the prefix code first.



2 Representations of Prefix Code

We can represent a code as a D-ary tree or a D-ary floating number.

2.1 Tree Representation

Remark. The tree in this subsection is similar with "trie" in computer science.

Using a (complete) D-ary tree to represent a code, we use a path starting from the root to some node (or equivalently, use the node) to denote a codeword.

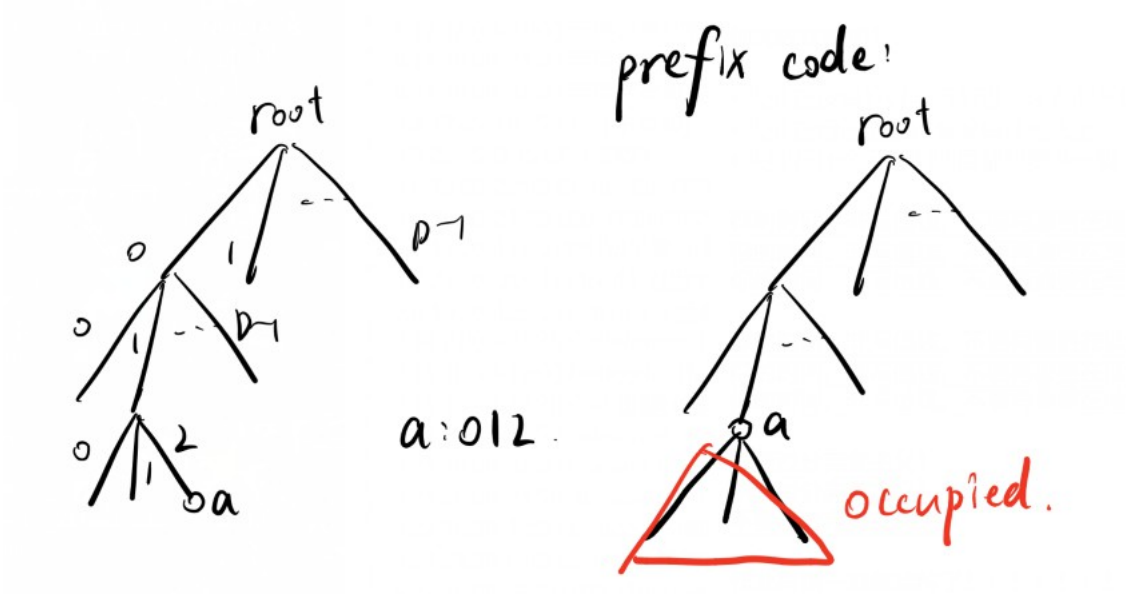


Figure 1: An example. Here the codeword for a is 012.

For a prefix code, it satisfies that for different codewords $C(x_1), C(x_2)$, the corresponding paths p_1, p_2 *will not contain each other*. Or equivalently, if a node a is used, then the subtree with root a is *occupied* (i.e. the nodes in the subtree cannot be used anymore).

This representation will be useful in the proof of Kraft inequality.

2.2 Floating Number Representation

For a codeword $d_1 d_2 \cdots d_n$, use a D-ary floating number $0.d_1 d_2 \cdots d_n$ to represent it. Then it maps a codeword to a floating number $\in [0, 1)$.

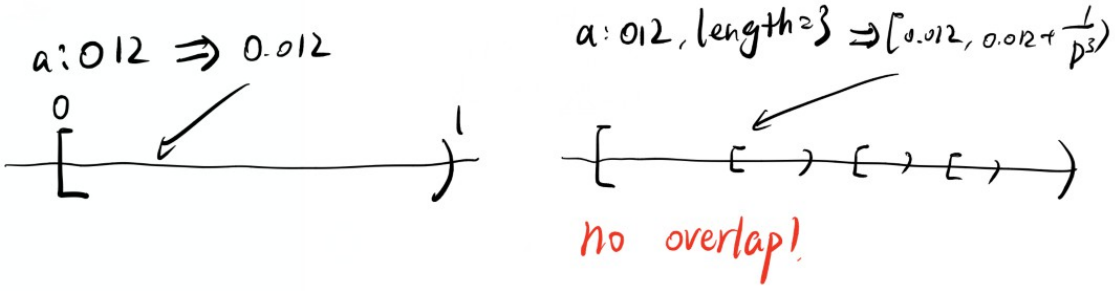


Figure 2: An example. Here the codeword for a is 012.

For prefix codes, we can step further, to represent codewords as *intervals*. For a codeword $d_1 d_2 \cdots d_n$, use $[0.d_1 d_2 \cdots d_n, 0.d_1 d_2 \cdots d_n + D^{-n})$ to represent it.

Then it satisfies that the intervals *will not overlap*, since $[0.d_1d_2\cdots d_n, 0.d_1d_2\cdots d_n + D^{-n})$ contains all the codewords with $d_1d_2\cdots d_n$ as prefix.

This will be useful in the proof of extended Kraft inequality.

3 Kraft Inequality and Its Extension

Kraft inequality shows the constraint a prefix code must satisfy. It bounds below the codeword lengths of a prefix code.

3.1 Kraft Inequality

Theorem 1. (Kraft Inequality) For any prefix code over an alphabet of size D, the codeword lengths $l_1, l_2, \cdots, l_m$ satisfies the inequality

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

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

Proof. We first prove the inequality. Without loss of generality, let $l_1 \leq l_2 \leq \cdots \leq l_m$. Let the height of the tree be l_m.

Then with the tree representation, we know that for l_i, it occupies $D^{l_m - l_i}$ leaves.

Since this is a prefix code, $\forall i \neq j$, l_i and l_j will not occupy the same leaf. And there are totally D^{l_m} leaves. So

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

To construct a prefix code with $l_1, l_2, \cdots, l_m$, first let $l_1 \leq l_2 \leq \cdots \leq l_m$, and then assign nodes with depth from l_1 to l_m. Easy to check the validity. □

3.2 Extended Kraft Inequality

If $\mathcal{X}$ is countably infinite, will the Kraft hold?

Theorem 2. (Extended Kraft Inequality, or McMillan Inequality) For any countably infinite set of codewords that form a prefix code, the codeword lengths satisfy the extended Kraft inequality,

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

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

Proof. We first prove the inequality. Using the interval representation, we know that the intervals are disjoint. Since the length of the i-th interval is D^{-l_i}, we have $\sum_{i=1}^{\infty} D^{-l_i} \leq 1$.

To construct the code, we can first sort $\{l_i\}$ increasingly and then assign the intervals in order from the low end on $[0, 1)$. □

4 Optimal Code

We can find the optimal prefix code using Kraft inequality.

4.1 Construction

Mainly we want to solve an optimization problem:

$$\min L = \sum_i p_i l_i$$
$$\text{s.t.} \sum_i D^{-l_i} \leq 1$$

This can be solved using Lagrange multipliers. Let $\mathcal{L} = \sum_i p_i l_i + \lambda(\sum_i D^{-l_i} - 1)$ be the Lagrangian. Then

$$\frac{\partial \mathcal{L}}{\partial l_i} = p_i - \lambda D^{-l_i} \ln D = 0 \implies D^{-l_i} = \frac{p_i}{\lambda \ln D}$$

Basically we want l_i to be as small as possible. So we let $\sum_i D^{-l_i} = 1$. Then we have $\lambda = \frac{1}{\ln D}$, and hence

$$p_i = D^{-l_i} \implies l_i^* = -\log_D p_i$$

This yield the *optimal length*. For these optimal lengths (although they may be nonintegers), $L^* = -\sum_i p_i \log_D p_i = H_D(X)$ exactly.

So in general, when the lengths are integers, we have

$$L^* \geq H_D(X)$$

To make the optimal lengths be integers, we can use **Shannon code**, where $l_i = \lceil -\log_D p_i \rceil$. It satisfies Kraft inequality.

4.2 Bound

Theorem 3. $H_D(X) \leq L^* < H_D(X) + 1$.

Proof. For Shannon code,

$$l_i < -\log_D p_i + 1 \implies \sum_i p_i l_i < -\sum_i p_i \log_D p_i + 1 = H_D(X) + 1$$

Since $H_D(X) \leq L^* \leq \sum_i p_i l_i$, we have $H_D(X) \leq L^* < H_D(X) + 1$. □

Can we eliminate the 1 here?

4.3 Eliminating 1

Actually, the answer is yes, when the message is long enough. We can encode n symbols $X_1, \cdots, X_n$ together, where $X_1, \cdots, X_n$ are i.i.d. and obey $p(x)$.

Theorem 4.

5 Wrong Code

6 From Prefix to Uniquely Decodable

7 Review

Acknowledgment

The contents are mainly based on the course materials of CS258, Shanghai Jiao Tong University and *Elements of Information Theory* by Thomas M. Cover.