CS258: Information Theory

Fan Cheng
Shanghai Jiao Tong University

http://www.cs.sjtu.edu.cn/~chengfan/
chengfan@sjtu.edu.cn
Spring, 2020

Outline

- Kraft inequality
- Optimal codes
- Huffman coding
- Shannon-Fano-Elias coding
- Generation of discrete distribution
- Universal source coding

Random Variable Generation

Heads Vs. Tails

- We are given a **sequence of fair coin tosses $Z_1, Z_2, \ldots,$** and we wish to generate X on $\mathcal{X} = \{1, 2, \ldots, m\}$ with probability mass function $\mathbf{p} = (p_1, \ldots, p_m)$.
- Let the random variable T denote the number of coin flips used in the algorithm.

Generate a random variable according the outcome of **fair coin flips**:

HHHH, TTTTT, HTHTHT, THTHTH

If $X = \{0, 1, 2\}, p(X) = (\frac{1}{2}, \frac{1}{4}, \frac{1}{4})$

- H: $X = 0$
- TH: $X = 1$
- TT: $X = 2$

- How many fair coin flips to generate X?

The entropy of X

$$H(X) = 1.5$$

The expected number of coin flips

$$E(T) = 1.5$$

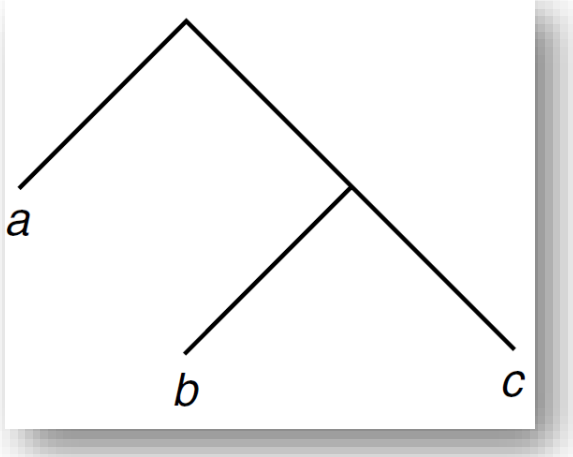
Random Variable Generation

Representation of a generation algorithm

- We can describe the algorithm mapping strings of bits $Z_1, Z_2, \ldots$, to possible outcomes X by a binary tree
- The leaves of the tree are marked by output symbols X, and the path to the leaves is given by the sequence of bits produced by the fair coin

The tree representing the algorithm must satisfy certain properties:

- The tree should be complete (i.e., every node is either a leaf or has two descendants in the tree). The tree may be infinite, as we will see in some examples.
- The probability of a leaf at depth k is 2^{-k}. Many leaves may be labeled with the same output symbol—the total probability of all these leaves should equal the desired probability of the output symbol.
- The expected number of fair bits ET required to generate X is equal to the expected depth of this tree.



Tree for generation of the distribution $(\frac{1}{2}, \frac{1}{4}, \frac{1}{4})$

Intuition: Each coin tossing generates 1 bit.

$$E(T) \geq H(X)$$

Random Variable Generation

Let $\mathcal{Y}$ denote the set of leaves of a complete tree. Consider a distribution on the leaves such that the probability of a leaf at depth k on the tree is 2^{-k}. Let Y be a random variable with this distribution.

(Lemma). For any complete tree, consider a probability distribution on the leaves such that the probability of a leaf at depth k is 2^{-k}. Then **the expected depth of the tree is equal to the entropy of this distribution ($H(Y) = ET$).**

- The expected depth of the tree

$$E(T) = \sum_{y \in \mathcal{Y}} k(y) 2^{-k(y)}$$

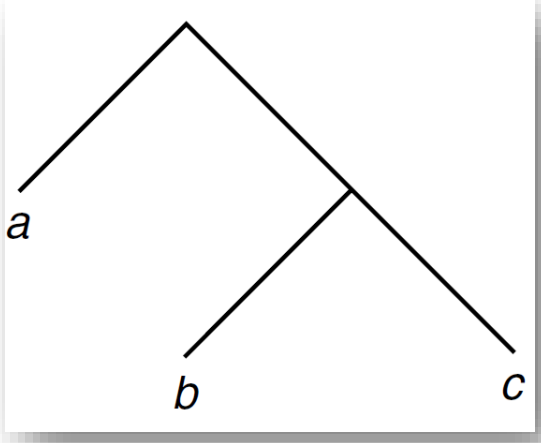
- The entropy of the distribution of Y is

$$H(Y) = -\sum_{y \in \mathcal{Y}} \frac{1}{2^{k(y)}} \log \frac{1}{2^{k(y)}}$$

$$= \sum_{y \in \mathcal{Y}} k(y) 2^{-k(y)}$$

where $k(y)$ denotes the depth of leaf y. Thus,

$$H(Y) = ET$$



$H(Y) = ET$

Random Variable Generation

(Theorem). For any algorithm generating X, the expected number of fair bits used is greater than the entropy $H(X)$, that is,

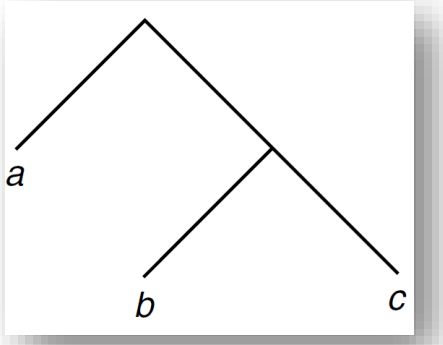
$$\boldsymbol{E(T) \geq H(X)}$$

- Any algorithm generating X from fair bits can be represented by a complete binary tree. Label all the leaves of this tree by distinct symbols $y \in Y = \{1, 2, \dots\}$. If the tree is infinite, the alphabet Y is also infinite.
- Now consider the random variable Y defined on the leaves of the tree, such that for any leaf y at depth k, the probability that $Y = y$ is 2^{-k}. The expected depth of this tree is equal to the entropy of Y:

$$ET = H(Y)$$

- Now the random variable X is a function of Y (one or more leaves map onto an output symbol), and hence we have

$$H(X) \leq H(Y)$$

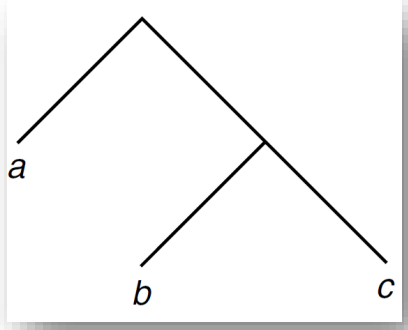


$$\boldsymbol{ET \geq H(X)}$$

Random Variable Generation

(Theorem). Let the random variable X have a **dyadic distribution**. The optimal algorithm to generate X from fair coin flips requires an expected number of coin tosses precisely equal to the entropy:

$$ET = H(X)$$



- For the constructive part, we use the Huffman code tree for X as the tree to generate the random variable. Each $X = x$ will correspond to a leaf
- For a dyadic distribution, the **Huffman code is the same as the Shannon code** and achieves the entropy bound.

$$l_i = \log D^{-n_i} = n_i$$

- For any $x \in \mathcal{X}$, the depth of the leaf in the code tree corresponding to x is the length of the corresponding codeword, which is $\log \frac{1}{p(x)}$. Hence, when this code tree is used to generate X, the leaf will have a probability

$$2^{-\log \frac{1}{p(x)}} = p(x) \ .$$

- The expected number of coin flips is the expected depth of the tree, which is equal to the entropy (because the distribution is dyadic). Hence, for a dyadic distribution, the optimal generating algorithm achieves

$$ET = H(X).$$

Random Variable Generation

- If the distribution is not dyadic? In this case we cannot use the same idea, since **the code tree for the Huffman code will generate a dyadic distribution on the leaves, not the distribution** with which we started
- Since all the leaves of the tree have probabilities of the form 2^{-k}, it follows that **we should split any probability p_i that is not of this form into atoms of this form**. We can then allot these atoms to leaves on the tree

$$p(x) = \frac{7}{8} = \frac{1}{2} + \frac{1}{4} + \frac{1}{8}$$

- **Finding the binary expansions of the probabilities $p_i's$**. Let the binary expansion of the probability p_i be

$$p_i = \sum_{j \geq 1} p_i^{(j)},$$

where $p_i^{(j)} = 2^{-j}$ or 0. Then the atoms of the expansion are the $\{p_i^{(j)} : i = 1,2, \dots, m, j \geq 1\}$.

- Since $\sum_i p_i = 1$, **the sum of the probabilities of these atoms is 1.** We will allot an atom of probability 2^{-j} to a leaf at depth j on the tree.
- The **depths (j) of the atoms satisfy the Kraft inequality**, we can always construct such a tree with all the atoms at the right depths.

Random Variable Generation

Let X have distribution

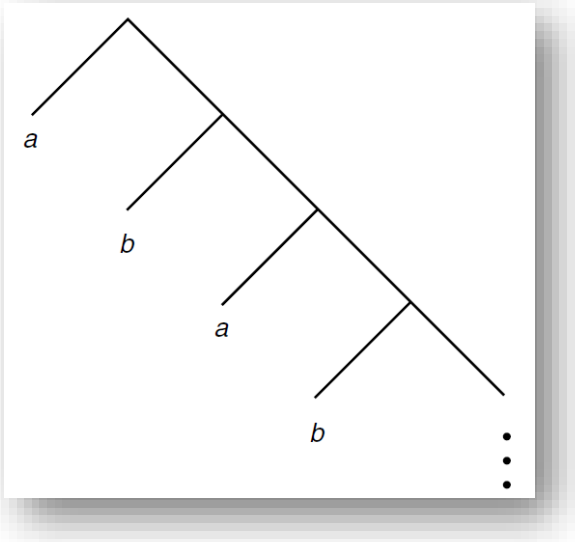
$$X = \begin{cases} a \text{ with prob. } \frac{2}{3} \\ b \text{ with prob. } \frac{1}{3} \end{cases}$$

We find the binary expansions of these probabilities:

$$\frac{2}{3} = 0.10101010\ldots_2$$
$$\frac{1}{3} = 0.01010101\ldots_2$$

Hence, the atom for the expansion are:

$$\frac{2}{3} \to \left(\frac{1}{2}, \frac{1}{8}, \frac{1}{32}, \ldots\right)$$
$$\frac{1}{3} \to \left(\frac{1}{4}, \frac{1}{16}, \frac{1}{64}, \ldots\right)$$



Tree to generate a $(\frac{2}{3}, \frac{1}{3})$ distribution

- This procedure yields a tree that generates the random variable X. We have argued that this procedure is optimal (gives a tree of minimum expected depth)
- **(Theorem)** The expected number of fair bits required by the optimal algorithm to generate a random variable X lies between $H(X)$ and $H(X) + 2$:

$$H(X) \leq ET < H(X) + 2$$

Universal Source Coding

Challenge: For many practical situations, however, the probability distribution underlying the source may be unknown

- One possible approach is to wait until we have seen all the data, estimate the distribution from the data, use this distribution to construct the best code, and then go back to the beginning and compress the data using this code.
  - This two-pass procedure is used in some applications where there is a fairly small amount of data to be compressed.
- In yet other cases, there is no probability distribution underlying the data—all we are given is an individual sequence of outcomes. How well can we compress the sequence?
  - If we do not put any restrictions on the class of algorithms, we get a meaningless answer—there always exists a function that compresses a particular sequence to one bit while leaving every other sequence uncompressed. This function is clearly "overfitted" to the data.

- Assume we have a random variable X drawn according to a distribution from the family $\{p_\theta\}$, **where the parameter $\theta \in \{1, 2, 3, \ldots, m\}$ is unknown**
- We wish to find an efficient code for this source

Minmax Redundancy

- **If we know θ**, we can construct a code with codeword length $l(x) = \log\frac{1}{p_\theta(x)}$

$$\min_{l(x)} E_p[l(X)] = E_p\left[\log\frac{1}{p_\theta(X)}\right] = H(p_\theta)$$

- What happens if **we do not know the true distribution p_θ**, yet wish to code as efficiently as possible? In this case, using a code with codeword lengths $l(x)$ and implied probability $q(x) = 2^{-l(x)}$, we define the redundancy of the code as the difference between the expected length of the code and the lower limit for the expected length:

$$R(p_\theta, q) = E_{p_\theta}[l(x)] - E_{p_\theta}\left[\log\frac{1}{p_\theta(X)}\right] = \sum_x p_\theta(x)\left(l(x) - \log\frac{1}{p_\theta(x)}\right)$$

$$= \sum_x p_\theta(x)\left(\log\frac{1}{q(x)} - \log\frac{1}{p_\theta(x)}\right) = D(p_\theta||q)$$

- We wish to find a code that does well irrespective of the true distribution p_θ, and thus we define the **minimax redundancy as**

$$\boldsymbol{R^* = \min_q \max_{p_\theta} R = \min_q \max_{p_\theta} D(p_\theta||q)}$$

Redundancy and Capacity

How to compute R^*: Take $\{p_\theta : 1 \le \theta \le m\}$ as a transition a matrix

$$\boldsymbol{\theta} \to \begin{bmatrix} \dots \boldsymbol{p_1(x)} \dots \\ \dots \boldsymbol{p_2(x)} \dots \\ \vdots \\ \dots \boldsymbol{p_\theta(x)} \dots \\ \dots \boldsymbol{p_m(x)} \dots \end{bmatrix} \to X$$

This is a channel $\{\theta, p_\theta(x), \mathcal{X}\}$. The capacity of this channel is given by

$$C = \max_{\pi(\theta)} I(\theta; X) = \max_{\pi(\theta)} \sum_\theta \pi(\theta) p_\theta(x) \log \frac{p}{q}$$

where $q_\pi(x) = \sum_\theta \pi(\theta) p_\theta(x)$.

(Theorem) The capacity of a channel $\boldsymbol{p(x|\theta)}$ with rows $\boldsymbol{p_1, p_2, \dots, p_m}$ is given by

$$\boldsymbol{C = R^* = \min_q \max_\theta D(p_\theta || q)}$$

Channel capacity is well understood.

Shannon-Fano-Elias → Arithmetic Coding

Shannon-Fano-**Elias** Coding: $F(a) = \Pr(x \le a)$

$$l(x) = \left\lceil \frac{1}{p(x)} \right\rceil + 1$$

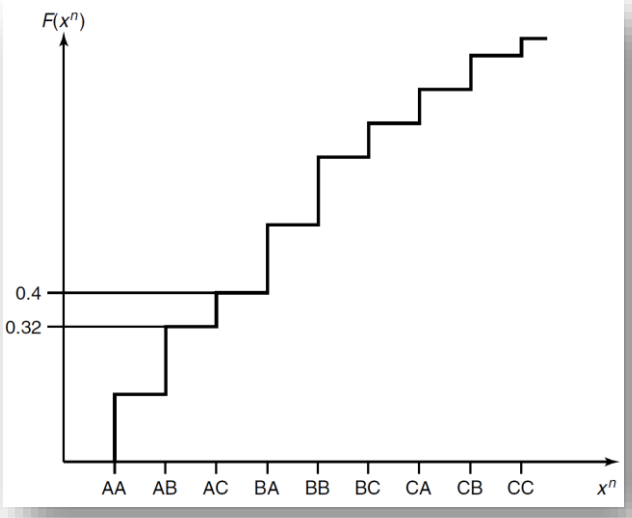
$$H(X) \le E\big(l(x)\big) < H(X) + 2$$

Motivation: using intervals to represent symbols

Consider a random variable X with a ternary alphabet $\{A, B, C\}$, with probabilities 0.4, 0.4, and 0.2, respectively. $F(x) = (0.4, 0.8, 1.0)$.

Let the sequence to be encoded by **ACAA**

- A $\rightarrow [0,\ 0.4)$
- AC $\rightarrow$ [0.32, 0.4) (scale with ratio $(0.4, 0.8, 1.0)$)
- ACA $\rightarrow$ [0.32, 0.352)
- ACAA $\rightarrow$ [0.32, 0.3328)



Combination of x_1x_2, $x_1x_2 \in \{A, B, C\}$

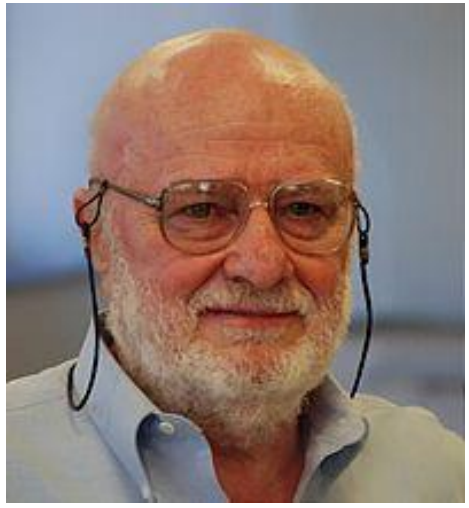
- **The procedure is incremental and can be used for any blocklength**
- **Coding by intervals: new insight**

"火车刚发明的时候比马车还慢"

Lempel-Ziv Coding: Introduction

- Use **dictionaries** for compression dates back to the invention of the telegraph.
  - "25: Merry Christmas"
  - "26: May Heaven's choicest blessings be showered on the newly married couple."

The idea of adaptive dictionary-based schemes was not explored until Ziv and Lempel wrote their papers in 1977 and 1978. The two papers describe two distinct versions of the algorithm. We refer to these versions as LZ77 or sliding window Lempel–Ziv and LZ78 or tree-structured Lempel–Ziv.



Abraham Lempel

Yaakov Ziv

Gzip, pkzip, compress in unix, GIF

Lempel-Ziv Coding: Sliding Window

The key idea of the Lempel–Ziv algorithm is to **parse** the string into **phrases** and to replace phrases by pointers to where the same string has occurred in the past.

Sliding Window Lempel–Ziv Algorithm

- We assume that we have a string $x_1, x_2, \ldots$ to be compressed from a finite alphabet. A ***parsing*** S of a string $x_1 x_2 \cdots x_n$ is a division of the string into phrases, separated by commas. Let W be the **length of the window**.
- Assume that we have compressed the string until time $i-1$. Then to find the next phrase, find the largest k such that for some $j, i-W \le j \le i-1$, the string of length k starting at x_j is equal to the string (of length k) starting at x_i (i.e., $x_{j+l} = x_{i+l}$ for all $0 \le l < k$). The next phrase is then of length k (i.e., $x_i \ldots x_{i+k-1}$) and is represented by the pair (P, L), where P is the location of the beginning of the match and L is the length of the match.
- If a match is not found in the window, the next character is sent uncompressed.

0101010101010101011010101010101101, W = 7

0101010101010101011010101010101101

0101010101010101011010101010101101

Find the maximum repeated substring inside *W*

Lempel-Ziv Coding: Sliding Window

0101010101010101011010101010101101, W = 6
0101010101010101011010101010101101
0101010101010101011010101010101101
Find the maximum repeated substring inside *W*

$W = 4$, ABBABBABBBAABABA

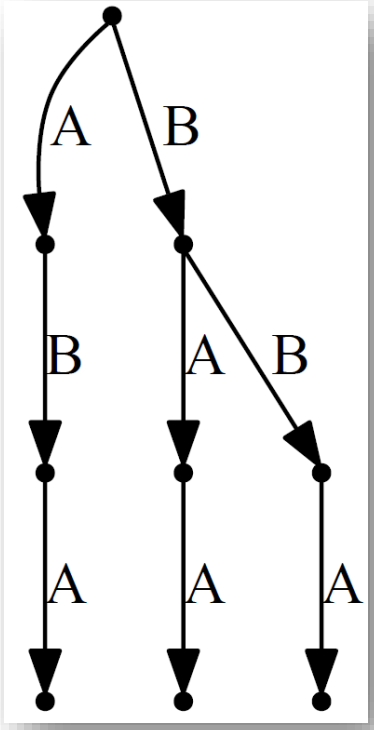
	ABBABBABBBAABABA
A	BBABBABBBAABABA
A, B	BABBABBBAABABA
A, B, B	ABBABBBAABABA
A, B, B, ABBABB	BAABABA
A, B, B, ABBABB, BA	ABABA
A, B, B, ABBABB, BA, A	BABA
A, B, B, ABBABB, BA, A, BA	BA
A, B, B, ABBABB, BA, A, BA, BA	

Lempel-Ziv Coding: Tree-Structured

- In the 1978 paper, Ziv and Lempel described an algorithm that parses a string into phrases, **where each phrase is the shortest phrase not seen earlier**.
- This algorithm can be viewed as building a dictionary in the form of a tree, where the nodes correspond to phrases seen so far.
- Find a string in a set of strings: **Trie**

ABBABBABBBAABABAA

	ABBABBABBBAABABAA
A	BBABBABBBAABABAA
A, B	BABBABBBAABABAA
A, B, BA	BBABBBAABABAA
A, B, BA, BB	ABBBAABABAA
A, B, BA, BB, AB	BBAABABAA
A, B, BA, BB, AB, BBA	ABABAA
A, B, BA, BB, AB, BBA, ABA	BAA
A, B, BA, BB, AB, BBA, ABA, BAA	



Optimality of LZ77, LZ78
Ref. Ch. 13.5 T. Cover

Summary

Cover: 5.11, 13.1, 13.3, 13.4, 13.5