Lecture 7 - Data compression

Jan Bouda

FI MU

April 22, 2010

Part I

Optimal Length of a Code

Message and Message Source

In our following analysis we will design various methods compressing input message unknown at the time of the design of the method. However, to design the method (algorithm) to be as efficient as possible we have to use all knowledge about the incoming message we have. In most cases the minimal information we have is the set of possible messages we may receive and a probability assigned to each message.

Following this analysis we model the **source** of information as a random variable X with all possible messages equal to $\mathbf{Im}(X)$. This source emits the message x with the probability $P(X = x)$. A sequence of messages is created by a sequence of independent trials described by X and hence is described by a random process $X_1, X_2, \ldots$ where X_i are independently and identically distributed. Such a source is called a **memoryless source**.

Message and Message Source; Code

We may naturally expect that source has a memory. This is modeled by a random process $X_1, X_2, \ldots$ with $\mathbf{Im}(X_i) = \mathbf{Im}(X_j), \forall i, j$, but we require neither independence nor identical distribution of X_i. In practice, this means that probability of a particular message being emitted at particular time depends on the history of the messages - it models a **source with memory**.

Definition

A **code** C for a random variable (memoryless source) X is a mapping $C : \mathbf{Im}(X) \rightarrow \mathbf{D}^*$, where $\mathbf{D}^*$ is the set of all finite length strings over the alphabet $\mathbf{D}$, with $|\mathbf{D}| = d$. $C(x)$ denotes the codeword assigned to x and $l_C(x)$ denotes the length of $C(x)$.

Code

Definition

The expected length $L_C(X)$ of a code C for a random variable X is given by

$$L_C(X) = \sum_{x \in \mathbf{Im}(X)} P(X = x) l_C(x). \quad (1)$$

In what follows we will assume (WLOG) that the alphabet is $\mathbf{D} = \{0, 1, \ldots, d-1\}$.

Code

Example

Let X and C be given by the following probability distribution and codeword assignment

$$\begin{aligned} P(X=1) &= 1/2, \text{ codeword } C(1) = 0 \\ P(X=2) &= 1/4, \text{ codeword } C(2) = 10 \\ P(X=3) &= 1/8, \text{ codeword } C(3) = 110 \\ P(X=4) &= 1/8, \text{ codeword } C(4) = 111 \end{aligned} \tag{2}$$

The entropy $H(X) = 1.75$ bits and the expected length $L_C(X) = E[l_C(X)] = 1.75$ too. Note that any encoded (not any!) sequence can be uniquely decoded to symbols $\{1, 2, 3, 4\}$, try e.g. 0110111100110.

Code

Example

Consider another example with

$$\begin{aligned} P(X=1) &= 1/3, \text{ codeword } C(1) = 0 \\ P(X=2) &= 1/3, \text{ codeword } C(2) = 10 \\ P(X=3) &= 1/3, \text{ codeword } C(3) = 11 \end{aligned} \tag{3}$$

The entropy in this case is $H(X) = log_2 3 = 1.58$ bits, but the expected length is $L_C(X) = 1.66$.

Non-singular Code

Definition

A code C is said to be **non-singular** if it maps every element in the range of X to different string in $\mathbf{D}^*$, i.e.

$$\forall x, y \in \mathbf{Im}(X) x \neq y \Rightarrow C(x) \neq C(y).$$

Non-singularity allows unique decoding of any single codeword, however, in practice we send a sequence of codewords and require the complete sequence to be uniquely decodable. We can use e.g. any non-singular code and use an extra symbol $\# \notin \mathbf{D}$ as a codeword separator. However, this is very inefficient and we can improve efficiency by designing uniquely decodable or prefix code.

Uniquely Decodable Code

Let $\mathbf{Im}(X)^+$ denotes the set of all nonempty strings over the alphabet $\mathbf{Im}(X)$.

Definition

An **extension** C^* of a code C is the mapping from $\mathbf{Im}(X)^+$ to $\mathbf{D}^*$ defined by

$$C^*(x_1x_2\ldots x_n) = C(x_1)C(x_2)\ldots C(x_n),$$

where $C(x_1)C(x_2)\ldots C(x_n)$ denotes concatenation of corresponding codewords.

Definition

A code is **uniquely decodable** iff its extension is non-singular.

In other words, a code is uniquely decodable if any encoded string has only one possible source string.

Prefix Code

Definition

A code is called **prefix (or instantaneous)** if no codeword is a prefix of any other codeword.

The advantage of prefix codes is not only their unique decodability, but also the fact that a codeword can be decoded as soon as we read its last symbol. See the following codes for comparison

X	Singular	Non-singular, but not uniquely decodable	Uniquely decodable, but not prefix	Prefix
1	0	0	10	0
2	0	010	00	10
3	0	01	11	110
4	0	10	110	111

Part II

Kraft Inequality

Kraft Inequality

In this section we concentrate on prefix codes of minimal expected length.

Theorem (Kraft inequality)

For any prefix code over an alphabet of size d, the codeword lengths (including multiplicities) $l_1, l_2, \ldots l_m$ satisfy the inequality

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

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

Kraft Inequality

Proof.

Consider a d–ary tree in which every inner node has d descendants. Each edge represents a choice of a code alphabet symbol at a particular position. In example, d edges emerging from the root represent d choices of the alphabet symbol at the first position of different codewords. Each codeword is represented by a node (some nodes are not codewords!) and the path from the root to a particular node (codeword) specifies the codeword symbols. The prefix condition implies that no codeword is an ancestor of other codeword on the tree. Hence, each codeword eliminates its possible descendants.

Let $l_{max} = \max\{l_1, l_2, \ldots, l_m\}$. Consider all nodes of the tree at the level l_{max}. Some of them are codewords, some of them are descendants of codewords, some of them are neither. □

Kraft Inequality

Proof.

A codeword at level l_i has $d^{l_{max}-l_i}$ descendants at level l_{max}. Sets of descendant of different codewords must be disjoint and the total number of nodes in all these sets must be at most $d^{l_{max}}$. Summing over all codewords we have

$$\sum_{i=1}^{m} d^{l_{max}-l_i} \leq d^{l_{max}}$$

and hence

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

Conversely, given any set of codeword lengths $l_1, l_2, \ldots, l_m$ satisfying the Kraft inequality we can always construct a tree described above. We may WLOG assume that $l_1 \leq l_2 \leq \cdots \leq l_m$. □

Kraft Inequality

Proof.

Label the first note of depth l_1 as the codeword 1 and remove its descendants from the tree. Then mark first remaining node of depth l_2 as the codeword 2. In this way you can construct prefix code with codeword lengths $l_1, l_2, \ldots, l_m$.

We may observe easily that this construction does not violate the prefix property. To do so, the new codeword should be placed either as a precedent, or an antecedent of an existing codeword, what is prevented by the construction. It remains to show that there is always enough nodes. Assume that for some $i \leq m$ there is no free node of level l_i when we want to add a new codeword of length l_i. This, however, means that all node at level l_i are either codewords, or descendants of a codeword, giving

$$\sum_{j=1}^{i-1} d^{l_i - l_j} = d^{l_i}$$

and we have $\sum_{j=1}^{i-1} d^{-l_j} = 1$, and, finally, $\sum_{j=1}^{i} d^{-l_j} > 1$ violating the initial assumption. □

McMillan Inequality

Kraft inequality holds also for codes with countably infinite number of codewords, however, we omit the proof here. There exist uniquely decodable codes that are not prefix codes, but, as established by the following theorem, the Kraft inequality applies to general uniquely decodable codes as well and, therefore, when searching for an optimal code it suffices to concentrate on prefix codes. General uniquely decodable codes offer no extra codeword lengths in contrast to prefix codes.

Theorem (McMillan inequality)

The codeword lengths of any uniquely decodable code must satisfy the Kraft inequality, i.e.

$$\sum_i d^{-l_i} \leq 1.$$

Conversely, given a set of codeword lengths that satisfy the inequality it is possible to construct a uniquely decodable code with these codeword lengths.

McMillan Inequality

McMillan inequality.

Consider the k-th extension C^k of a code C. By the definition of the unique decodability, C^k is non-singular for any k.
Observe that $l_{C^k}(x_1, \ldots, x_k) = \sum_{i=1}^{k} l_C(x_i)$. Let us calculate

$$\left(\sum_{x \in \mathbf{Im}(X)} d^{-l_C(x)} \right)^k = \sum_{x_1, x_2, \ldots, x_k \in \mathbf{Im}(X)} d^{-l_C(x_1)} d^{-l_C(x_2)} \ldots d^{-l_C(x_k)} \tag{4}$$

$$= \sum_{x_1, x_2, \ldots, x_k \in \mathbf{Im}(X)} d^{-l_{C^k}(x_1, x_2, \ldots, x_k)}.$$

McMillan Inequality

Proof.

We reorder the terms by word lengths to get

$$\sum_{x_1,x_2,\ldots,x_k \in \mathbf{lm}(X)} d^{-l_{Ck}(x_1,x_2,\ldots,x_k)} = \sum_{m=1}^{kl_{max}} a(m) d^{-m},$$

where l_{max} is the maximum codeword length and $a(m)$ is the number of k character source strings mapped to a codeword of length m. The code is uniquely decodable, i.e. there is at most one input being mapped on each codeword (of length m). The total number of such inputs is at most the same as the number of sequences of length m, i.e. at most d^m. □

McMillan Inequality

Proof.

Using $a(m) \leq d^m$ we get

$$\left(\sum_{x \in \mathbf{Im}(X)} d^{-l_C(x)} \right)^k = \sum_{m=1}^{kl_{max}} a(m) d^{-m} \tag{5}$$

$$\leq \sum_{m=1}^{kl_{max}} d^m d^{-m} = kl_{max}$$

implying

$$\sum_i d^{-l_i} \leq (kl_{max})^{1/k} .$$

McMillan Inequality

Proof.

This inequality holds for any k and observing $\lim_{k\to\infty} \left(k l_{max}\right)^{1/k} = 1$ we have

$$\sum_i d^{-l_i} \leq 1.$$

The opposite implication follows from the Kraft inequality. □

Part III

Optimal Codes

Optimal Codes

In the previous part we derived necessary and sufficient condition on lengths of codewords for prefix (uniquely decodable) codes. Now we will use them to find a prefix code with the minimum expected length.

Theorem

The expected length of any prefix d–ary code C for a random variable X is greater than or equal to the entropy $H_d(X)$ (d is the base of the logarithm), i.e.

$$L_C(X) \geq H_d(X)$$

with equality if and only if for all x_i $P(X = x_i) = p_i = d^{-l_i}$ for some integer l_i.

Optimal Codes

Proof.

We write the difference between the expected length and the entropy as

$$
\begin{aligned}
L_C(X) - H_d(X) &= \sum_i p_i l_i + \sum_i p_i \log_d p_i = \\
&= \sum_i p_i \log_d d^{l_i} + \sum_i p_i \log_d p_i = \\
&= \sum_i p_i \log_d \frac{p_i}{d^{-l_i}} = \\
&= \sum_i p_i \log_d \frac{p_i}{d^{-l_i}} + \sum_i p_i \log_d \left(\sum_j d^{-l_j} \right) \\
&\quad - \sum_i p_i \log_d \left(\sum_j d^{-l_j} \right) .
\end{aligned}
\tag{6}
$$

Optimal Codes

Proof.

We put $r_i = d^{-l_i} / \sum_j d^{-l_j}$ and $c = \sum_i d^{-l_i}$ to get

$$\begin{aligned} L_C(X) - H_d(X) &= \sum_i p_i \log_d \frac{p_i}{d^{-l_i}} + \sum_i p_i \log_d \frac{d^{-l_i}}{r_i} - \log_d c \\ &= \sum_i p_i \log_d \frac{p_i}{r_i} - \log_d c \qquad (7) \\ &= D(p\|r) + \log_d \frac{1}{c} \geq 0 \end{aligned}$$

by the nonnegativity of the relative entropy and the fact that $c \leq 1$ (Kraft inequality). Hence, $L_C(X) \geq H_d(X)$ with equality if and only if for all i $p_i = d^{-l_i}$, i.e. $-\log_d p_i$ is an integer. □

Optimal Codes

Definition

A probability distribution is called d–adic if each of the probabilities is equal to d^{-n} for some integer n.

Proof of the previous theorem shows that the expected length is equal to the entropy if and only if the probability distribution of X is d–adic. It also suggests a method to find a code with optimal length in case the probability distribution is not d–adic.

Optimal Codes

1. Find a d–adic distribution that is the closest to the distribution of X in the relative entropy. This distribution defines the set of codeword lengths.
2. Use the technique described in the proof of the Kraft inequality to construct the code.

Note that this procedure is not easy, since the search for the closest d–adic distribution is not obvious.

Part IV

Bounds on the Optimal Code Length

Bounds on the Optimal Code Length

Let us consider a code that achieves the expected description length within 1 bit of the lower bound, i.e.

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

Our basic setup is to minimize $\sum_i p_i l_i$ with the restriction $\sum_i d^{-l_i} \leq 1$. We have shown that optimal solution for probability distribution that is not d–adic is the d–adic probability distribution closest in the relative entropy, i.e. finding d–adic distribution $\vec{r}$, $r_i = d^{-l_i} / \sum_j d^{-l_j}$ minimizing

$$L_C(X) - H_d(X) = D(p\|r) - \log_d \left(\sum_i d^{-l_i} \right) \geq 0. \tag{8}$$

Bounds on the Optimal Code Length

The choice of word lengths $l_i = \log_d \frac{1}{p_i}$ gives $L = H_d(X)$. Since it may not equal an integer, we round it up to get

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

These lengths satisfy the Kraft inequality since

$$\sum_i d^{-\left\lceil \log \frac{1}{p_i} \right\rceil} \leq \sum_i d^{-\log \frac{1}{p_i}} = \sum_i p_i = 1.$$

The choice of codeword lengths satisfies

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

Bounds on the Optimal Code Length

Taking expectation over p_i on both sides we get

$$H_d(X) \leq L_C(X) < H_d(X) + 1. \tag{9}$$

The optimal code can do only better and we have

Theorem

Let $l_1^*, l_2^*, \ldots, l_m^*$ be the optimal codeword lengths for a source distribution $\{p_i\}_i$ and a d–ary alphabet and let L^* be the associated expected length of the optimal code, i.e. $L^* = \sum_i p_i l_i^*$. Then

$$H_d(X) \leq L^* < H_d(X) + 1.$$

Bounds on the Optimal Code Length

Proof.

Let $l_i = \lceil \log_d \frac{1}{p_i} \rceil$. Then l_i satisfies the Kraft inequality and from (9) we have

$$H_d(X) \leq L_C(X) = \sum_i p_i l_i < H_d(X) + 1. \tag{10}$$

But since our code is optimal, $L^* \leq L = \sum_i p_i l_i$ and since $L^* \geq H_d(X)$ we have the result. □

The non-integer expressions $\log_d(1/p_i)$ cause in the previous theorem overhead at most 1 bit per symbol. We can further reduce it by spreading it over a number of symbols. Let us consider a system in which we send a sequence of symbols emitted by source X, where all symbols are drawn independently according to an identical distribution. We can consider n such symbols to be a supersymbol from alphabet $\mathbf{Im}(X)^n$.

Bounds on the Optimal Code Length

Let us define L_n as the expected codeword length per input symbol, i.e.

$$L_n = \frac{1}{n} \sum p(x_1, x_2, \ldots, x_n) l(x_1, x_2, \ldots, x_n) = \frac{1}{n} E[l(X_1, X_2, \ldots, X_n)].$$

Using the bounds derived above we have

$$H(X_1, X_2, \ldots, X_n) \leq E[l(X_1, X_2, \ldots, X_n)] < H(X_1, X_2, \ldots, X_n) + 1.$$

Since $X_1, X_2, \ldots, X_n$ are independently and identically distributed, we have $H(X_1, X_2, \ldots, X_n) = nH(X)$ and dividing by n we get

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

Using large blocks allows us to arbitrarily approach the optimal length - the entropy.

Sources with memory

An analogous argument can be applied even when $X_1, X_2, \ldots, X_n$ are no independently and identically distributed. We still have

$$H(X_1, X_2, \ldots, X_n) \leq E[I(X_1, X_2, \ldots, X_n)] < H(X_1, X_2, \ldots, X_n) + 1$$

and dividing by n we obtain

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

Definition

The **entropy rate** of a random process $X_1, X_2, \ldots$ is

$$H = \lim_{n \to \infty} \frac{1}{n} H(X_1, X_2, \ldots, X_n).$$

Sources with memory

For strictly stationary process the entropy rate always exists and equals to

$$H = \lim_{n\to\infty} H(X_n|X_{n-1}, X_{n-2}, \ldots, X_1).$$

Therefore we have

Theorem (We omit the proof)

The minimum expected codeword length per symbol satisfies

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

and if $X_1, X_2, \ldots$ is a strictly stationary process,

$$L_n^* \to H,$$

where H is the entropy rate of the process.

Shannon coding and relative entropy

Let us return to memoryless channels. The relative entropy allows us to quantify inefficiency caused by wrong input probability distribution estimation.

Theorem

The expected length under $p(x)$ of the code assignment $l(x) = \lceil \log \frac{1}{q(x)} \rceil$ satisfies

$$H(p) + D(p\|q) \leq E[l(X)] < H(p) + D(p\|q) + 1. \tag{11}$$

Shannon coding and relative entropy

Proof.

$$
\begin{aligned}
E[l(X)] &= \sum_x p(x) \left\lceil \log \frac{1}{q(x)} \right\rceil \\
&< \sum_x p(x) \left(\log \frac{1}{q(x)} + 1 \right) \\
&= \sum_x p(x) \log \left(\frac{p(x)}{q(x)} \frac{1}{p(x)} \right) + 1 \\
&= \sum_x p(x) \log \frac{p(x)}{q(x)} + \sum_x p(x) \log \frac{1}{p(x)} + 1 \\
&= D(p\|q) + H(p) + 1.
\end{aligned} \tag{12}
$$

The lower bound can be proven analogously. □

Part V

Huffman codes

Huffman codes

Let us introduce the d–ary Huffman codes for the source described by the random variable X with probability distribution $p_1, p_2, \ldots, p_m$. Then the d–**ary Huffman code** for X is constructed as

- Add redundant input symbols with probability 0 to the distribution so that the distribution has $1 + k(d-1)$ symbols for some k.
- Find d smallest probabilities $p_{i_1}, \ldots, p_{i_d}$ and replace them with $p_{i_1,\ldots,i_d} = \sum_{j=1}^{d} p_{i_j}$.
- Repeat the previous step until we end with the probability distribution having only single nonzero probability - equal to 1.

To construct the code, we keep expanding the sum of probabilities and create the codewords assigned to probabilities, i.e.

- We assign ε, i.e. the empty codeword, to the probability $p_{1,\ldots,1+k(d-1)}$.
- Let w be a codeword assigned to $p_{i_1,\ldots,i_d}$. We assign the codewords $w0, w1, \ldots, w(d-1)$ to probabilities to $p_{i_1}, \ldots, p_{i_d}$, respectively.
- We keep expanding until we end with the original probability distribution.

Huffman codes

Example

Let us consider a random variable with outcomes $1, 2, \ldots, 6, r$ and corresponding probabilities $0.25, 0.25, 0.2, 0.1, 0.1, 0.1, 0$, where we included one redundant symbol r to obtain $7 = 1 + 3(3 - 1)$ symbols and construct 3-ary code.

- We add $0 + 0.1 + 0.1 = 0.2$ corresponding to $5, 6, r$.
- We add $0.2 + 0.1 + 0.2 = 0.5$ corresponding to $(5, 6, r), 4, 3$.
- We add $0.5 + 0.25 + 0.25 = 1$ corresponding to $((5, 6, r), 4, 3), 2, 1$.
- We assign ε to $((5, 6, r), 4, 3), 2, 1$.
- We assign $0; 1; 2$ to $((5, 6, r), 4, 3); 2; 1$, respectively.
- We assign $00; 01; 02$ to $(5, 6, r); 4; 3$, respectively.
- We assign $000; 001; 002$ to $5; 6; r$, respectively.

Huffman Codes

Example (Continuing)

Therefore we end with the code $2, 1, 02, 01, 000, 001$ (redundant symbol would have the codeword 002).

Example

Let us compare an example of the Huffman and the Shannon code. We have two source symbols with probabilities 0.0001 and 0.9999. Using the Shannon code we get two codewords with lengths $1 = \lceil \log \frac{1}{0.9999} \rceil$ and $14 = \lceil \log \frac{1}{0.0001} \rceil$ bits. The optimal code obviously uses 1 bit codeword for both symbols as it is with the Huffman code.

Is it true that an optimal code uses always codewords of length not larger than $\lceil \log \frac{1}{p_i} \rceil$?

Optimality of Huffman Codes

Lemma

For any distribution X with $P(X = x_i) = p_i$ there exists an optimal prefix code that satisfies the following properties:

1. *If $p_j > p_k$ then $l_j \leq l_k$.*
2. *Two longest codewords have the same length.*
3. *Two longest codewords differ only in the last bit and correspond to the least likely symbols.*

Optimality of Huffman Codes

Proof.

Let us consider an optimal code C.

1. Let us suppose that $p_j > p_k$. Consider C' with the codewords j and k interchanged (comparing to C). Then

$$\begin{aligned} L_{C'}(X) - L_C(X) &= \sum_i p_i l'_i - \sum_i p_i l_i \\ &= p_j l_k + p_k l_j - p_j l_j - p_k l_k \\ &= (p_j - p_k)(l_k - l_j). \end{aligned} \tag{13}$$

We know that $p_j - p_k > 0$ and since C is optimal we have $L_{C'}(X) - L_C(X) \geq 0$. Hence, we have $l_k \geq l_j$.

Optimality of Huffman Codes

Proof.

2. If two longest codewords have different length, then we can delete the last bit of the longer one to get shorter code while preserving the prefix property, what contradicts our assumption that C is optimal. Therefore, two longest codewords have the same length.
3. If there is a codeword of maximal length without a sibling then we can delete the last bit of the codeword and still maintain the prefix property, what contradicts optimality of the code. In case the two siblings do not correspond to two least likely symbols, we simply exchange them with codewords corresponding to the two least likely symbols to obtain a better code.

Optimality of Huffman Codes

Theorem

Huffman code is optimal, i.e. if C is the Huffman code for X and C' is any other code, then $L_{C'}(X) \geq L_C(X)$.

Proof.

Let us suppose that the probabilities are ordered starting with the biggest one. This proof is limited to the case of a binary code, general n-ary code is analogous.

For a code C_m with m codewords we define the 'merged' code C_{m-1} of $(m-1)$ codewords the way that we take the common prefix of the two longest codewords (corresponding to the two least likely symbols) and assign it to a new symbol with probability $p_{m-1} + p_m$. □

Optimality of Huffman Codes

Proof.

Let us denote $l_i = l_C(x_i)$, $l'_i = l_{C'}(x_i)$, $p'_i = p_i$ for $i = 1 \dots m-2$ and $p'_{m-1} = p_{m-1} + p_m$. The expected length of the code C_m is

$$\begin{aligned} L_{C_m}(X) &= \sum_{i=1}^{m} p_i l_i \\ &= \sum_{i=1}^{m-2} p_i l'_i + p_{m-1}(l'_{m-1} + 1) + p_m(l'_{m-1} + 1) \\ &= \sum_{i=1}^{m-1} p'_i l'_i + p_{m-1} + p_m \\ &= L_{C_{m-1}}(X) + p_{m-1} + p_m. \end{aligned} \tag{14}$$

Optimality of Huffman Codes

Proof.

Important is that the expected length of C_m differs from expected length of C_{m-1} only by fixed amount that is independent of C_{m-1} and depends only on the probability distribution of the source. Therefore, to minimize the length of C_m it suffices to minimize the length of C_{m-1}, i.e. to find minimal code for the distribution $p_1, p_2, \ldots, p_{m-1} + p_m$.

The code C_{m-1} satisfies the previous lemma and, therefore, we can apply this procedure iteratively. In this way we reduce our problem to two symbol source, where the obvious optimal solution assigns codewords 0 and 1. Since every step preserves optimality, we have optimal construction for m item probability distribution. This is precisely the construction of the Huffman code. □

Part VI

Competitive Optimality of Shannon Codes

Competitive Optimality of Shannon Codes

We have already proven that Huffman codes are optimal from the average length point of view. Now we will define the optimality in a bit different way. Let us consider the following game of two players:

- Two players are given a probability distribution and encouraged to design a code for the distribution.
- Then a source symbol is drawn according to this distribution and the payoff of each player is 1 or -1 depending whether his codeword for this symbol is shorter or longer than the codeword of the other player.
- In case the length of both codewords is the same, both payoffs are 0.

To prove optimality of Huffman codes in this setting is difficult since we have no explicit formula for codeword lengths. Instead, we will prove it for Shannon codes where we have explicit codeword lengths.

Competitive Optimality of Shannon Codes

Theorem

Let us consider a source X distributed according $p(x)$. Let $l(x)$ denotes the length of a particular codeword in the Shannon code and $l'(x)$ length of the corresponding codeword in an arbitrary (fixed) other code. Then

$$P(l(X) \geq l'(X) + c) \leq \frac{1}{2^{c-1}}.$$

Competitive Optimality of Shannon Codes

Proof.

$$
\begin{aligned}
P(l(X) \geq l'(X) + c) &= P\left(\left\lceil \log \frac{1}{p(X)} \right\rceil \geq l'(X) + c\right) \\
&\leq P\left(\log \frac{1}{p(X)} \geq l'(X) + c - 1\right) \\
&= P\left(p(X) \leq 2^{-l'(X)-c+1}\right) \\
&= \sum_{x:p(x)\leq 2^{-l'(X)-c+1}} p(x) \\
&\leq \sum_{x:p(x)\leq 2^{-l'(X)-c+1}} 2^{-l'(X)-c+1} \\
&\leq \sum_{x} 2^{-l'(X)-c+1} \leq 2^{-(c-1)}
\end{aligned} \tag{15}
$$

since $\sum_x 2^{-l'(x)} \leq 1$ by Kraft inequality. □

Part VII

Data compression in practice

Huffman and Shannon Codes in Practice

- In practice we have to consider a number of various problems.
- Usually it is very hard to determine probability distribution of the source. Even when we determine it correctly, the actual sequence generated can be different from what we expected.
- In case we want to compress e.g. a general file, the strategy adopted is to calculate probabilities as the relative frequency of 'symbols' (e.g. a sequence of bytes) in the file. This assures optimal coding (relatively to the chosen set of symbols!), but we have to generate a codeword table that has to be stored together with the compressed file.
- When measuring practical efficiency we have to judge both size of the compressed file and size of the table.

Adaptive Coding

- In extreme, we can consider the whole file to be one symbol and it is then compressed to a single-bit message. However, the coding table is as long as the original file.
- Another restriction is that symbols are fixed for the whole file (message).
- A nice and elegant solution is **adaptive coding**, where the list of symbols and codewords is generated 'on the fly' without the need to store the codeword table.
- An asymptotically optimal coding is e.g. the Lempel-Ziv coding.

Lempel-Ziv Coding

The source sequence is parsed into strings that did not appear before. In example, if the input is $1011010100010\ldots$, it is parsed as $1, 0, 11, 01, 010, 00, 10, \ldots$. After determining each phrase we search for the shortest string that did not appear before. The coding follows:

- Parse the input sequence as above and count the number of codewords. This will be used to determine the length of the bit string referring to a particular codeword.
- We code each phrase by specifying the *id* of its longest prefix (it certainly already appeared and was parsed) and the extra bit. The empty prefix we usually assign index 0.
- Our example will be coded as $(000, 1)(000, 0)(001, 1)(010, 1)(100, 0)(010, 0)(001, 0)$.
- The length of the code can be further optimized, e.g. at the beginning of the coding process the length of the bit string describing the codeword can be shorter than at the end. Note that if fact we do not need commas and parentheses, it suffices to specify the length of the bit string identifying the prefix.

Part VIII

Generating Discrete Distribution Using Fair Coin

Discrete Distribution and Fair Coin

Example

Suppose we want to simulate a source described by a random variable X with the distribution

$$X = \begin{cases} a & \text{with probability } \frac{1}{2} \\ b & \text{with probability } \frac{1}{4} \\ c & \text{with probability } \frac{1}{4} \end{cases}$$

using a sequence of fair coin tosses. The solution is pretty easy - if the outcome of the first coin toss is 0, we set $X = a$, otherwise we perform another coin toss and set $X = b$ if the outcomes were 10 and $X = c$ if the outcomes were 11.

The average number of fair coin tosses is 1.5 what equals to the entropy of X.

Discrete Distribution and Fair Coin

The general formulation of the problem is that we have a sequence of fair coin tosses $Z_1, Z_2, \ldots$ and we want to generate a discrete random variable X with the probability distribution $\vec{p} = (p_1, p_2, \ldots, p_m)$. Let the random variable T denotes the number of coin flips used by the algorithm.

We can describe the algorithm mapping outcomes of $Z_1, Z_2, \ldots$ to outcomes of X by a binary tree. Leaves of the tree are marked by outcomes of X and the path from the root to a particular leaf represents the sequence of coin toss outcomes.

Discrete Distribution and Fair Coin

The tree should satisfy:

1. It is complete, i.e. every node is either leaf, or it has two descendants in the tree. The tree may be infinite.
2. The probability of a leaf at depth k is 2^{-k}. There can be more leaves labeled by the same outcome of X. The sum of their probabilities is the probability of this outcome.
3. The expected number of fair bits $E(T)$ required to generate X is equal to the expected depth of this tree.

Discrete Distribution and Fair Coin

Lemma

Let $\mathbf{Y}$ *denotes the set of leaves of a complete binary tree and* Y *random variable with distribution on* $\mathbf{Y}$*, where the probability of a leaf of the depth* k *is* 2^{-k}*. The expected depth of this tree is equal to the entropy of* Y*.*

Proof.

The expected depth of the tree is

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

where $k(y)$ denotes the depth of y. □

Discrete Distribution and Fair Coin

Proof.

The entropy of Y is

$$H(Y) = -\sum_{y \in \mathbf{Y}} \frac{1}{2^{k(y)}} \log \frac{1}{2^{k(y)}}$$
$$= \sum_{y \in \mathbf{Y}} k(y) 2^{-k(y)} = E(T).$$

Discrete Distribution and Fair Coin

Theorem

For any algorithm generating X, the expected number of fair bits used is at least the entropy $H(X)$, i.e.

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

Proof.

Any algorithm generating X from fair bits can be represented by a binary tree. Label all leaves by distinct symbols $\mathbf{Y}$. The tree may be infinite. Consider the random variable Y defined on the leaves of the tree such that for any leaf of depth k the probability is $P(Y = y) = 2^{-k}$. By the previous lemma we get $E(T) = H(Y)$. The random variable X is a function of Y and hence we have $H(X) \leq H(Y)$. Combining we get that for any algorithm $H(X) \leq E(T)$. □

Discrete Distribution and Fair Coin

Theorem

Let X be a random variable with a dyadic distribution. The optimal algorithm to generate X from fair coin flips requires an expected number of coin tosses equal to the entropy $H(X)$.

Proof.

The previous theorem shows that we need at least $H(X)$ bits to generate X. We use the Huffman code tree to generate the variable. For dyadic distribution Huffman code coincides with Shannon code, has codewords of length $\log \frac{1}{p(x)}$ and the probability of such a codeword is $p(x) = 2^{\log p(x)}$. The expected depth of the tree is $H(X)$. □

Discrete Distribution and Fair Coin

To deal with a general (non-dyadic) distribution we have to find the binary expansion of each probability, i.e.

$$p_i = \sum_{j \geq 0} p_i^{(j)},$$

where $p_i^{(j)}$ is either 2^{-j} or 0. Now we will assign to each nonzero $p_i^{(j)}$ a leaf of depth j in a binary tree. Their depths satisfy the Kraft inequality, because $\sum_{i,j} p_i^{(j)} = 1$, and therefore we can always do this.

Theorem

The expected number of fair bits $E(T)$ required by the optimal algorithm to generate a random variable X is bounded as $H(X) \leq E(T) < H(X) + 2$.

Discrete Distribution and Fair Coin

Proof.

The lower bound has been already established, it remains to prove the upper bound.

Let us start with the initial distribution $(p_1, p_2, \ldots, p_m)$ and expand each of the probabilities using the dyadic coefficients, i.e.

$$p_i = p_i^{(1)} + p_i^{(2)} + \cdots \tag{16}$$

with $p_j^{(j)} \in \{0, 2^{-j}\}$. Let us consider new random variable Y with the probability distribution $p_1^{(1)}, p_1^{(2)}, \ldots, p_2^{(1)}, p_2^{(2)}, \ldots, p_m^{(1)}, p_m^{(2)}, \ldots$. We construct the binary tree T for the dyadic probability distribution Y. Recall that the expected depth of T, i.e. the expected number of coin tosses, is $H(Y)$. □

Discrete Distribution and Fair Coin

Proof.

X is a function of Y giving

$$H(Y) = H(Y, X) = H(X) + H(Y|X). \tag{17}$$

It remains to show that $H(Y|X) < 2$.
Let us expand the entropy of Y as

$$H(Y) = -\sum_{i=1}^{m} \sum_{j \geq 1} p_i^{(j)} \log p_i^{(j)} = \sum_{i=1}^{m} \sum_{j : p_i^{(j)} > 0} j 2^{-j}. \tag{18}$$

Let T_i denotes the sum of the addends corresponding to p_i, i.e.

$$T_i = \sum_{j : p_i^{(j)} > 0} j 2^{-j}. \tag{19}$$

Discrete Distribution and Fair Coin

Proof.

We can find n such that $2^{-(n-1)} > p_i \geq 2^{-n}$. This is equivalent to

$$n - 1 < -\log p_i \leq n. \tag{20}$$

We have that $p_i^{(j)} > 0$ only if $j \geq n$ and we rewrite T_i as

$$T_i = \sum_{j: j \geq n, p_i^{(j)} > 0} j 2^{-j}. \tag{21}$$

Recall that

$$p_i = \sum_{j: j \geq n, p_i^{(j)} > 0} 2^{-j}. \tag{22}$$

Discrete Distribution and Fair Coin

Proof.

Next, we will show that $T_i < -p_i \log p_i + 2p_i$. Let us expand

$$\begin{aligned} T_i + p_i \log p_i &\overset{(*)}{<} T_i - p_i(n-1) - 2p_i = T_i - (n-1+2)p_i = \\ &\sum_{j:j\geq n, p_i^{(j)}>0} j2^{-j} - (n+1) \sum_{j:j\geq n, p_i^{(j)}>0} 2^{-j} = \\ &\sum_{j:j\geq n, p_i^{(j)}>0} (j-n-1)2^{-j} = \\ &- 2^{-n} + 0 + \sum_{j:j\geq n+2, p_i^{(j)}>0} (j-n-1)2^{-j} \overset{(**)}{=} \\ &- 2^{-n} + \sum_{k:k\geq 1, p_i^{(k+n+1)}>0} k2^{-(k+n+1)} \end{aligned} \tag{23}$$

Discrete Distribution and Fair Coin

Proof.

We get

$$-2^{-n} + \sum_{k:k\geq 1, p_i^{(k+n+1)}>0} k2^{-(k+n+1)} \leq -2^{-n} + \sum_{k:k\geq 1} k2^{-(k+n+1)}, \quad (24)$$

since on the right hand side we only increase the number of addends. Finally,

$$-2^{-n} + \sum_{k:k\geq 1} k2^{-(k+n+1)} = -2^{-n} + 2^{-(n+1)}2 = 0 \quad (25)$$

using the formula for infinite geometric series summation. □

Discrete Distribution and Fair Coin

Proof.

Using $E(T) = \sum_i T_i$ and $T_i < -p_i \log p_i + 2p_i$ we obtain the desired result

$$E(T) = \sum_i T_i < -\left(\sum_i p_i \log p_i\right) + 2\sum_i p_i = H(X) + 2. \tag{26}$$

□