15-583:Algorithms in the Real World

Data Compression I
- –Introduction
- –Information Theory
- –Probability Coding

Compression in the Real World

Generic File Compression
- files: gzip (LZ77), bzip (Burrows-Wheeler), BOA (PPM)
- archivers: ARC (LZW), PKZip (LZW+)
- file systems: NTFS

Communication
- Fax: ITU-T Group 3 (run-length + Huffman)
- Modems: V.42bis protocol (LZW) MNP5 (RL + Huffman)
- Virtual Connections

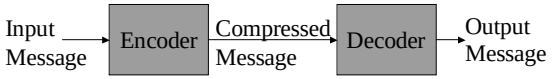
Multimedia
- Images: gif (LZW), jbig (context), jpeg-ls (residual), jpeg (transform+RL+arithmetic)
- TV: HDTV (mpeg-4)
- Sound: mp3

Compression Outline

Introduction: Lossy vs. Lossless, Benchmarks, …
Information Theory: Entropy, etc.
Probability Coding: Huffman + Arithmetic Coding
Applications of Probability Coding: PPM + others
Lempel-Ziv Algorithms: LZ77, gzip, compress, ...
Other Lossless Algorithms: Burrows-Wheeler
Lossy algorithms for images: JPEG, fractals, ...
Lossy algorithms for sound?: MP3, ...

Encoding/Decoding

Will use "message" in generic sense to mean the data to be compressed



The encoder and decoder need to understand common compressed format.

Lossless vs. Lossy

Lossless: Input message = Output message

Lossy: Input message ≈ Output message

Lossy does not necessarily mean loss of quality. In fact the output could be "better" than the input.

- Drop random noise in images (dust on lens)
- Drop background in music
- Fix spelling errors in text. Put into better form.

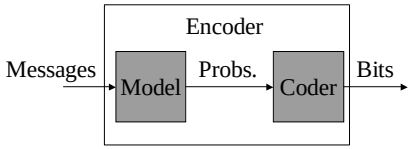
Writing is the art of lossy text compression.

How much can we compress?

For lossless compression, assuming all input messages are valid, if even one string is compressed, some other must expand.

Model vs. Coder

To compress we need a bias on the probability of messages. The model determines this bias



Example models:

- Simple: Character counts, repeated strings
- Complex: Models of a human face

Quality of Compression

Runtime vs. Compression vs. Generality

Several standard corpuses to compare algorithms

Calgary Corpus

- 2 books, 5 papers, 1 bibliography,
  1 collection of news articles, 3 programs,
  1 terminal session, 2 object files,
  1 geophysical data, 1 bitmap bw image

The **Archive Comparison Test** maintains a comparison of just about all algorithms publicly available

Comparison of Algorithms

Program	Algorithm	Time	BPC	Score
BOA	PPM Var.	94+97	1.91	407
PPMD	PPM	11+20	2.07	265
IMP	BW	10+3	2.14	254
BZIP	BW	20+6	2.19	273
GZIP	LZ77 Var.	19+5	2.59	318
LZ77	LZ77	?	3.94	?

Information Theory

An interface between modeling and coding

- **Entropy**
  - A measure of information content
- **Conditional Entropy**
  - Information content based on a context
- **Entropy of the English Language**
  - How much information does each character in "typical" English text contain?

Entropy (Shannon 1948)

For a set of messages S with probability $p(s)$, $s \in S$, the **self information** of s is:

$$i(s) = \log \frac{1}{p(s)} = -\log p(s)$$

Measured in bits if the log is base 2.

The lower the probability, the higher the information

Entropy is the weighted average of self information.

$$H(S) = \sum_{s \in S} p(s) \log \frac{1}{p(s)}$$

Entropy Example

$$p(S) = \{.25, .25, .25, .125, .125\}$$
$$H(S) = 3 \cdot .25 \log 4 + 2 \cdot .125 \log 8 = 2.25$$

$$p(S) = \{.5, .125, .125, .125, .125\}$$
$$H(S) = .5 \log 2 + 4 \cdot .125 \log 8 = 2$$

$$p(S) = \{.75, .0625, .0625, .0625, .0625\}$$
$$H(S) = .75 \log(4/3) + 4 \cdot .0625 \log 16 = 1.3$$

Conditional Entropy

The **conditional probability** $p(s|c)$ is the probability of s in a context c. The **conditional self information** is

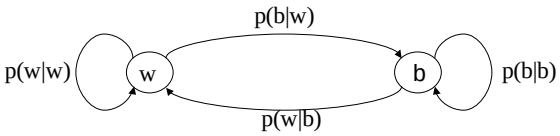
$$i(s|c) = -\log p(s|c)$$

The conditional information can be either more or less than the unconditional information.

The **conditional entropy** is the average of the conditional self information a

$$H(S|C) = \sum_{c \in C} p(c) \sum_{s \in S} p(s|c) \log \frac{1}{p(s|c)}$$

Example of a Markov Chain



Entropy of the English Language

How can we measure the information per character?

- ASCII code = 7
- Entropy = 4.5 (based on character probabilities)
- Huffman codes (average) = 4.7
- Unix Compress = 3.5
- Gzip = 2.5
- BOA = 1.9 (current close to best text compressor)

Must be less than 1.9.

Shannon's experiment

Asked humans to predict the next character given the whole previous text. He used these as conditional probabilities to estimate the entropy of the English Language.

The number of guesses required for right answer:

# of guesses	1	2	3	3	5	> 5
Probability	.79	.08	.03	.02	.02	.05

From the experiment he predicted
H(English) = .6-1.3

Coding

How do we use the probabilities to code messages?

- **Prefix codes and relationship to Entropy**
- **Huffman codes**
- **Arithmetic codes**
- **Implicit probability codes**

Assumptions

Communication (or file) broken up into pieces called messages.

Adjacent messages might be of a different types and come from a different probability distributions

We will consider two types of coding:

- **Discrete**: each message is a fixed set of bits
  - Huffman coding, Shannon-Fano coding
- **Blended**: bits can be "shared" among messages
  - Arithmetic coding

Uniquely Decodable Codes

A **variable length code** assigns a bit string (codeword) of variable length to every message value

e.g. `a = 1, b = 01, c = 101, d = 011`

What if you get the sequence of bits
`1011` ?

Is it `aba, ca,` or, `ad`?

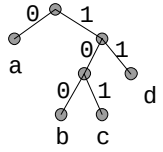
A **uniquely decodable code** is a variable length code in which bit strings can always be uniquely decomposed into its codewords.

Prefix Codes

A **prefix code** is a variable length code in which no codeword is a prefix of another word

e.g a = 0, b = 110, c = 111, d = 10

Can be viewed as a binary tree with message values at the leaves and 0 or 1s on the edges.



Some Prefix Codes for Integers

n	Binary	Unary	Split
1	..001	0	1|
2	..010	10	10|0
3	..011	110	10|1
4	..100	1110	110|00
5	..101	11110	110|01
6	..110	111110	110|10

Many other fixed prefix codes:
Golomb, phased-binary, subexponential, ...

Average Length

For a code *C* with associated probabilities *p(c)* the **average length** is defined as

$$l_a(C) = \sum_{c \in C} p(c) l(c)$$

We say that a prefix code *C* is **optimal** if for all prefix codes *C'*, $l_a(C) \leq l_a(C')$

Relationship to Entropy

Theorem (lower bound): *For any probability distribution p(S) with associated uniquely decodable code C,*

$$H(S) \leq l_a(C)$$

Theorem (upper bound): *For any probability distribution p(S) with associated optimal prefix code C,*

$$l_a(C) \leq H(S) + 1$$

Kraft McMillan Inequality

Theorem (Kraft-McMillan): *For any uniquely decodable code C,*

$$\sum_{c \in C} 2^{-l(c)} \leq 1$$

Also, for any set of lengths L such that

$$\sum_{l \in L} 2^{-l} \leq 1$$

there is a prefix code C such that

$$l(c_i) = l_i (i = 1, \ldots, |L|)$$

Proof of the Upper Bound (Part 1)

Assign to each message a length $l(s) = \lceil \log(1/p(s)) \rceil$

We then have

$$\begin{aligned} \sum_{s \in S} 2^{-l(s)} &= \sum_{s \in S} 2^{-\lceil \log(1/p(s)) \rceil} \\ &\leq \sum_{s \in S} 2^{-\log(1/p(s))} \\ &= \sum_{s \in S} p(s) \\ &= 1 \end{aligned}$$

So by the Kraft-McMillan ineq. there is a prefix code with lengths *l(s)*.

Proof of the Upper Bound (Part 2)

Now we can calculate the average length given *l(s)*

$$\begin{aligned} l_a(S) &= \sum_{s \in S} p(s) l(s) \\ &= \sum_{s \in S} p(s) \cdot \lceil \log(1/p(s)) \rceil \\ &\leq \sum_{s \in S} p(s) \cdot (1 + \log(1/p(s))) \\ &= 1 + \sum_{s \in S} p(s) \log(1/p(s)) \\ &= 1 + H(S) \end{aligned}$$

And we are done.

Another property of optimal codes

Theorem: *If C is an optimal prefix code for the probabilities* $\{p_1, \ldots, p_n\}$ *then* $p_i < p_j$ *implies* $l(c_i) \leq l(c_j)$

Proof: (by contradiction)

Assume $l(c_i) > l(c_j)$. Consider switching codes c_i and c_j. If l_a is the average length of the original code, the length of the new code is

$$\begin{aligned} l_a' &= l_a + p_j(l(c_i) - l(c_j)) + p_i(l(c_j) - l(c_i)) \\ &= l_a + (p_j - p_i)(l(c_i) - l(c_j)) \\ &< l_a \end{aligned}$$

This is a contradiction since l_a is not optimal

Huffman Codes

Invented by Huffman as a class assignment in 1950.

Used in many, if not most compression algorithms

- gzip, bzip, jpeg (as option), fax compression,…

Properties:

- Generates optimal prefix codes
- Cheap to generate codes
- Cheap to encode and decode
- $l_a = H$ if probabilities are powers of 2

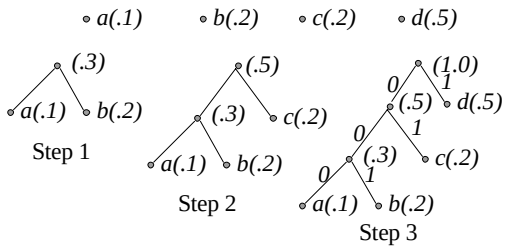
Huffman Codes

Huffman Algorithm

- Start with a forest of trees each consisting of a single vertex corresponding to a message *s* and with weight *p(s)*
- ***Repeat:***
  - Select two trees with minimum weight roots p_1 and p_2
  - Join into single tree by adding root with weight $p_1 + p_2$

Example

p(a) = .1, p(b) = .2, p(c) = .2, p(d) = .5



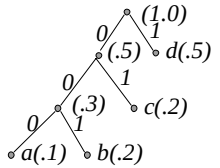
a=000, b=001, c=01, d=1

Encoding and Decoding

Encoding: Start at leaf of Huffman tree and follow path to the root. Reverse order of bits and send.

Decoding: Start at root of Huffman tree and take branch for each bit received. When at leaf can output message and return to root.

There are even faster methods that can process 8 or 32 bits at a time



Huffman codes are optimal

Theorem: *The Huffman algorithm generates an optimal prefix code.*