

15-853: Algorithms in the Real World

Data Compression: Lecture 2.5

Summary so far

Model generates probabilities, **Coder** uses them
Probabilities are related to **information**. The more you know, the less info a message will give.

More "skew" in probabilities gives lower **Entropy H** and therefore better compression

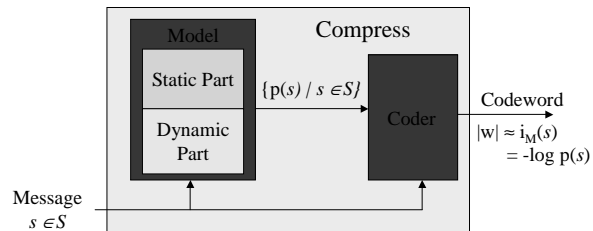
Context can help "skew" probabilities (lower H)
Average length l_a for **optimal prefix code** bound by

$$H \leq l_a < H + 1$$

Huffman codes are optimal prefix codes

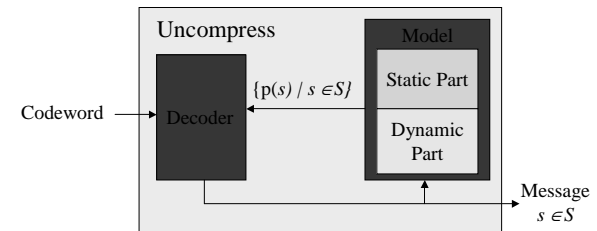
Arithmetic codes allow "blending" among messages

Encoding: Model and Coder



The **Static part** of the model is fixed
The **Dynamic part** is based on previous messages
The "optimality" of the code is relative to the probabilities.
If they are not accurate, the code is not going to be efficient

Decoding: Model and Decoder



The **probabilities** $\{p(s) \mid s \in S\}$ generated by the model need to be the same as generated in the encoder.

Note: consecutive "messages" can be from a different message sets, and the probability distribution can change

Codes with Dynamic Probabilities

Huffman codes:

Need to generate a new tree for new probabilities.

Small changes in probability, typically make small changes to the Huffman tree.

"**Adaptive Huffman codes**" update the tree without having to completely recalculate it.

Used frequently in practice

Arithmetic codes:

Need to recalculate the $f(m)$ values based on current probabilities.

Can be done with a balanced tree.

15-853

Page 5

Compression Outline

Introduction: Lossy vs. Lossless, Benchmarks, ...

Information Theory: Entropy, etc.

Probability Coding: Huffman + Arithmetic Coding

➔ **Applications of Probability Coding:** PPM + others

- Transform coding: move to front, run-length, ...

- Context coding: fixed context, partial matching

Lempel-Ziv Algorithms: LZ77, gzip, compress, ...

Other Lossless Algorithms: Burrows-Wheeler

Lossy algorithms for images: JPEG, MPEG, ...

Compressing graphs and meshes: BBK

15-853

Page 6

Applications of Probability Coding

How do we generate the probabilities?

Using character frequencies directly does not work very well (e.g. 4.5 bits/char for text).

Technique 1: transforming the data

- Run length coding (ITU Fax standard)
- Move-to-front coding (Used in Burrows-Wheeler)
- Residual coding (JPEG LS)

Technique 2: using conditional probabilities

- Fixed context (JBIG...almost)
- Partial matching (PPM)

15-853

Page 7

Run Length Coding

Code by specifying message value followed by the number of repeated values:

e.g. **abbbaacccca** => **(a,1),(b,3),(a,2),(c,4),(a,1)**

The characters and counts can be coded based on frequency.

This allows for small number of bits overhead for low counts such as 1.

15-853

Page 8

Facsimile ITU T4 (Group 3)

Standard used by all home Fax Machines
ITU = International Telecommunications Standard
Run length encodes sequences of black+white pixels
Fixed Huffman Code for all documents. e.g.

Run length	White	Black
1	000111	010
2	0111	11
10	00111	0000100

Since alternate black and white, no need for values.

15-853

Page 9

Move to Front Coding

Transforms message sequence into sequence of integers, that can then be probability coded
Takes advantage of **temporal locality**

Start with values in a total order: e.g.: [a,b,c,d,...]

For each message

- output the position in the order
- move to the front of the order.

e.g.: b => output: 3, new order: [c,a,b,d,e,...]

a => output: 2, new order: [a,c,b,d,e,...]

Probability code the output.

The hope is that there is a bias for small numbers.

15-853

Page 10

Residual Coding

Typically used for message values that represent some sort of amplitude:
e.g. gray-level in an image, or amplitude in audio.

Basic Idea: guess next value based on current context. Output difference between guess and actual value. Use probability code on the output.

15-853

Page 11

JPEG-LS

JPEG Lossless (not to be confused with lossless JPEG)
Just completed standardization process.

Codes in Raster Order. Uses 4 pixels as context:



Tries to guess value of * based on W, NW, N and NE.

Works in two stages

15-853

Page 12

JPEG LS: Stage 1

Uses the following equation:

$$P = \begin{cases} \min(N, W) & \text{if } NW \geq \max(N, W) \\ \max(N, W) & \text{if } NW < \min(N, W) \\ N + W - NW & \text{otherwise} \end{cases}$$

Averages neighbors and captures edges. e.g.

40	3	*	30	40	*	3	3	*
40	3		20	30		40	40	

15-853

Page 13

JPEG LS: Stage 2

Uses 3 gradients: W-NW, NW-N, N-NE

Classifies each into one of 9 categories.

This gives $9^3=729$ contexts, of which only 365 are needed because of symmetry.

Each context has a bias term that is used to adjust the previous prediction

After correction, the residual between guessed and actual value is found and coded using a Golomb-like code. (Golomb codes are similar to Gamma codes)

15-853

Page 14

Using Conditional Probabilities: PPM

Use previous k characters as the context.

- Makes use of conditional probabilities

Base probabilities on counts:

e.g. if seen **th** 12 times followed by **e** 7 times, then the conditional probability $p(e|th) = 7/12$.

Need to keep k small so that dictionary does not get too large (typically less than 8).

Note that 8-gram Entropy of English is ≈ 2.3 bits/char while PPM does as well as 1.7bits/char

15-853

Page 15

PPM: Partial Matching

Problem: What do we do if we have not seen the context followed by the character before?

- Cannot code 0 probabilities!

The key idea of PPM is to reduce context size if previous match has not been seen.

- If character has not been seen before with current context of size 3, try context of size 2, and then context of size 1, and then no context

Keep statistics for each context size $< k$

15-853

Page 16

PPM: Changing between context

How do we tell the decoder to use a smaller context?

Send an **escape** message. Each escape tells the decoder to reduce the size of the context by 1.

The escape can be viewed as special character, but needs to be assigned a probability.

- Different variants of PPM use different heuristics for the probability.

15-853

Page 17

PPM: Example Contexts

Context	Counts	Context	Counts	Context	Counts
Empty	A = 4	A	C = 3	AC	B = 1
	B = 2		\$ = 1		C = 2
	C = 5	B	A = 2		\$ = 2
	\$ = 3		\$ = 1	BA	C = 1
		C	A = 1		\$ = 1
			B = 2	CA	A = 1
			C = 2		\$ = 1
			\$ = 3	CB	A = 2
					\$ = 1
				CC	A = 1
					B = 1
					\$ = 2

String = ACCBACCACBA

k = 2

15-853

Page 18

PPM: Other important optimizations

If context has not been seen before, automatically escape (no need for an escape symbol since decoder knows previous contexts)

Can exclude certain possibilities when switching down a context. This can save 20% in final length!

It is critical to use arithmetic codes since the probabilities are small.

15-853

Page 19