# Information Theory and Networks

## Lecture 15: Stream Coding

Matthew Roughan
<matthew.roughan@adelaide.edu.au>
http://www.maths.adelaide.edu.au/matthew.roughan/Lecture_notes/InformationTheory/

School of Mathematical Sciences,
University of Adelaide

September 18, 2013

# Part I

## Stream Coding

Dr. Egon Spengler: There's something very important I forgot to tell you.
Dr. Peter Venkman: What?
Dr. Egon Spengler: Don't cross the streams.
Dr. Peter Venkman: Why?
Dr. Egon Spengler: It would be bad.

[some time later]

Dr. Egon Spengler: [hesitates] We'll cross the streams.
Dr. Peter Venkman: 'Scuse me Egon? You said crossing the streams was bad!

$\vdots$

Dr. Egon Spengler: Not necessarily. There's definitely a very slim chance we'll survive.

*Ghost Busters*

Section 1

# Problems and Solutions

# Problem of Block Huffman Coding

- Huffman coding needs probabilities
  - do you estimate them from the file – two passes?
  - or use generic probabilities – not quite accurate for a particular file?
- Blocks of length $n$ have $d^n$ possible "symbols"
  - estimating small probabilities is hard
  - do you include the (large) dictionary in the compressed file?
  - Huffman needs complete recalculation to change the block size

# Stream Coding

- **Block Coding:**
  - fixed set of input symbols (fixed block size)
  - same (possibly variable length) codes used through one file or transmission
  - transmitter and receiver need the same code dictionary
- **Steam Coding:**
  - process file (or transmission) as it comes
  - code dictionary adapts as it goes
  - decoder uses same method to construct dictionary as transmitter, and so they don't need to share this

Section 2

# Stream Compression Examples

# Arithmetic Coding

- RLE (see last lecture)
- Lempel-Ziv(-Welch)
- Arithmetic Coding

# Lempel-Ziv-Welch (LZW) [ZL78, Wel84]

- Simple version encodes series of 8-bit data (e.g., ASCII)
- 12 bit "codewords"
  - codes from 0-255 represents an 8-bit character (directly)
  - codes from 256-4095 refer to a dictionary, based on the data
- goal – replace long, repeated strings with a simple code (number)
  - construct the dictionary of strings as you go
  - as the file is processed, we get better and better compression (we hope)
- encoding
  - dictionary starts with all strings of length 1
  - repeat
    - find longest string W in dictionary that matches current input
    - put dictionary index for W in output, and remove W from input
    - add W followed by next symbol in the input to the dictionary
- decoding
  - iteratively translate and build the dictionary
  - don't need to transmit the dictionary

# Arithmetic Coding

- Use an adaptive Bayesian model for the probabilities
  - estimate it as we go along
- Encode with a Shannon-Fano-Elias-like code
- Decoder decodes symbols, and uses the same method to estimate probabilities, and hence derive the codes as you go along.

# Bayesian Model

- Take source alphabet $\Omega = \{a_1, a_2, \ldots, a_I\}$ where $a_I$ indicates "end of transmission"
- Source produces $X_1, X_2, \ldots \in \Omega$
- Both source coder, and receiver build a predictive probability distribution

$$p(X_n = a_i | X_{n-1}, X_{n-2}, \ldots, X_1)$$

- For example, use Bayesian estimates
  - fix probability of $a_I = 0.15$
  - iterate Bayes law to get estimates

# Shannon-Fano-Elias (SFE) Coding

- Lets do it for IID symbols:

$$p(X_n = a_i | X_{n-1}, X_{n-2}, \ldots, X_1) = p(X_n = a_i)$$

- Construct the CDF $F(x) = P(X \leq x)$

$$F(x) = \sum_{a<x} p(a)$$

and from this a new function

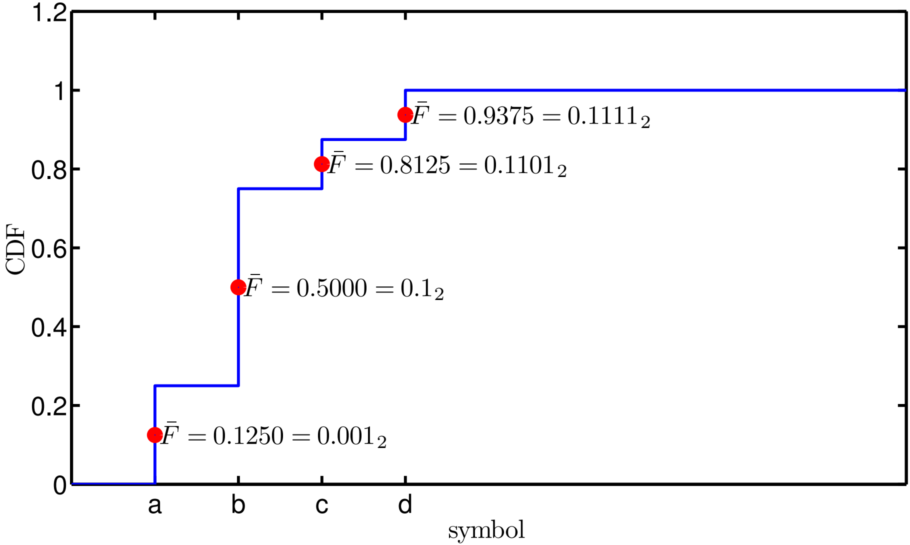
$$\bar{F}(x) = \sum_{a<x} p(a) + p(x)$$

- The values $F(a)$ make sense as codewords, except they may be infinite

# SFE Illustration 1

| $X$ | $p(x)$ | $F(x)$ | $\bar{F}(x)$ | $\bar{F}(x)$ in binary | $\ell(x)$ | codeword |
|---|---|---|---|---|---|---|
| a | 0.25 | 0.25 | 0.125 | $0.001_2$ | | |
| b | 0.5 | 0.75 | 0.5 | $0.1_2$ | | |
| c | 0.125 | 0.875 | 0.8125 | $0.1101_2$ | | |
| d | 0.125 | 1.0 | 0.9375 | $0.1111_2$ | | |
| $H(X)$ | 1.75 | | | | | |
| $E_p\ell$ | | | | | | |

# SFE Illustration 1



# SFE Coding

- Lets do it for IID symbols:

$$p(X_n = a_i | X_{n-1}, X_{n-2}, \ldots, X_1) = p(X_n = a_i)$$

  - Optimal codeword length approximation

$$\ell(a_i) = \left\lceil \log\left(\frac{1}{p(a_i)}\right) \right\rceil$$

  - Have one extra bit (see why in a second)

$$\ell(a_i) = \left\lceil \log\left(\frac{1}{p(a_i)}\right) \right\rceil + 1$$

  - Truncate the codes from $\bar{F}$ to this length

# SFE Illustration 1

| $X$ | $p(x)$ | $F(x)$ | $\bar{F}(x)$ | $\bar{F}(x)$ in binary | $\ell(x)$ | codeword |
|---|---|---|---|---|---|---|
| a | 0.25 | 0.25 | 0.125 | $0.001_2$ | 3 | 001 |
| b | 0.5 | 0.75 | 0.5 | $0.1_2$ | 2 | 10 |
| c | 0.125 | 0.875 | 0.8125 | $0.1101_2$ | 4 | 1101 |
| d | 0.125 | 1.0 | 0.9375 | $0.1111_2$ | 4 | 1111 |
| $H(X)$ | 1.75 | | | | | |
| $E_p\ell$ | | | | | | 2.75 bits per symbol |

# SFE Coding Theory

What are we trying to achieve?

- Code's related to $\bar{F}$ (we'll see why later)
- Optimal(ish) (code lengths given by $\simeq \ell(x)$ above)
- Prefix free (hence need for extra bit)

# SFE Coding Theory

Use the first $\ell(x)$ bits of $\bar{F}(x)$

- possible error in rounding off

$$\bar{F}(x) - \lfloor \bar{F}(x) \rfloor_{\ell(x)} < \frac{1}{2^{\ell(x)}}$$

- take

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

- then

$$\frac{1}{2^{\ell(x)}} < \frac{p(x)}{2} = \bar{F}(x) - F(x-1)$$

- thus

$$\bar{F}(x) - \lfloor \bar{F}(x) \rfloor_{\ell(x)} < \bar{F}(x) - F(x-1)$$

- thus the $\ell(x)$ length code is in the interval we want it to be in

# SFE Coding Theory

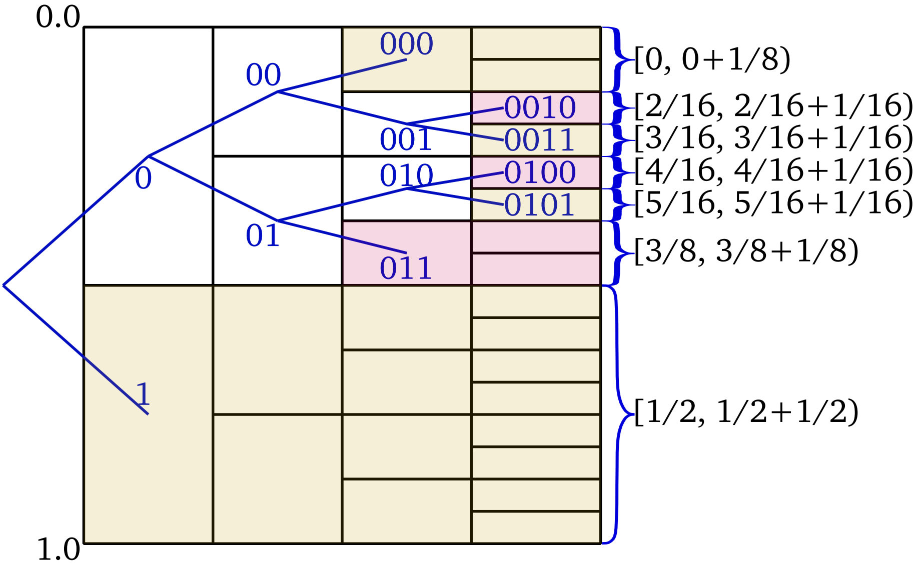
Test prefix free:

- Take a codeword $z_1 z_2 \ldots z_n$ to represent the interval

$$\left[0.z_1 z_2 \ldots z_n, \;\; 0.z_1 z_2 \ldots z_n + \frac{1}{2^n}\right)$$

  - the codes are prefix free iff the intervals are disjoint
  - to see that, think of the binary code tree
- From above, the intervals corresponding to the codewords must like entirely inside the interval $\big[F(x-1), F(x)\big)$, so they must be disjoint (see below)
  - given the choice of $\ell(x)$ above, the codewords will be on average 1 bit longer than similar Huffman code, but
  - if the codewords are 1 symbols less, then there is the potential for an overlap

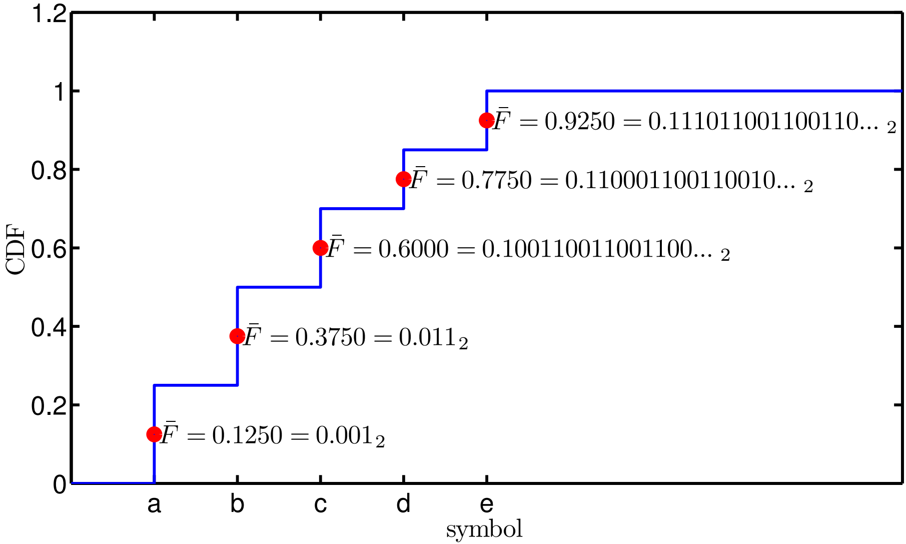
# Disjoint intervals = prefix free



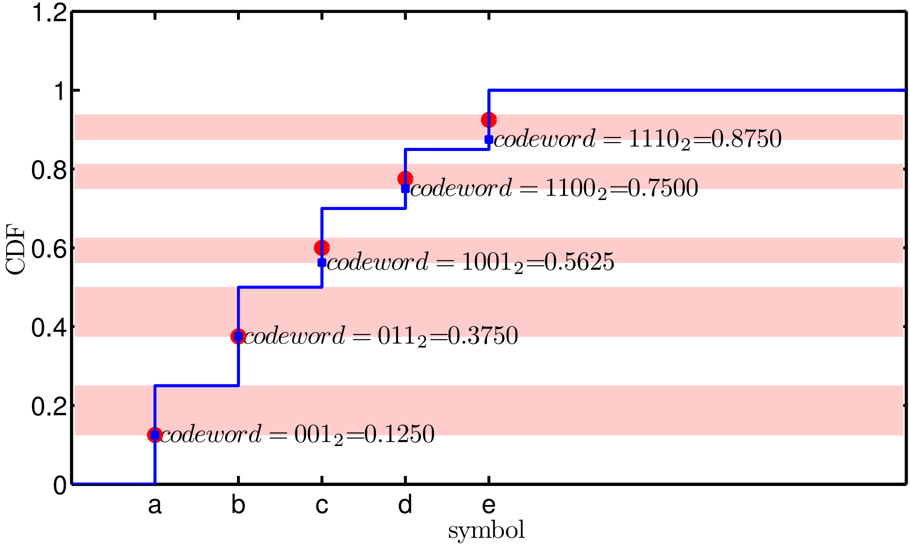
# SFE Illustration 2

| $X$ | $p(x)$ | $F(x)$ | $\bar{F}(x)$ | $\bar{F}(x)$ in binary | $\ell(x)$ | codeword |
|---|---|---|---|---|---|---|
| a | 0.25 | 0.25 | 0.125 | $0.001_2$ | 3 | 001 |
| b | 0.25 | 0.5 | 0.375 | $0.011_2$ | 3 | 011 |
| c | 0.2 | 0.7 | 0.6 | $0.1\overline{0011}_2$ | 4 | 1001 |
| d | 0.15 | 0.85 | 0.775 | $0.1100\overline{011}_2$ | 4 | 1100 |
| e | 0.15 | 1.0 | 0.925 | $0.1110\overline{110}_2$ | 4 | 1110 |
| $H(X)$ | 2.2855 | | | | | |
| $E_p\ell$ | | | | | | 3.5 bits per symbol |

# SFE Illustration 2



# SFE Illustration 2



# Arithmetic Coding

## The Coding Step

- We can see the SFE coding isn't the most efficient, but it has the huge advantage that we can build hierarchical codes in a similar way.
- Assume we can estimate

$$p(X_n | X_{n-1}, X_{n-2}, \ldots, X_1)$$

- Imagine we could construct the SFE code for this
  - if the first bit of that code would result in an interval that is entirely inside the step, then we can use it
  - if not, keep that bit in mind, and then divide the current step into components according to the next probability distribution
  - eventually, we can start fixing some of the old bits
- Iteratively perform these operations, along with probability estimation.

# Others

- DEFLATE: png (images), gzip and zip
- Apple Lossless (ALAC - Apple Lossless Audio Codec)
- Free Lossless Audio Codec (FLAC)
- WMA Lossless (Windows Media Lossless)

# Further reading I


- Thomas M. Cover and Joy A. Thomas, *Elements of information theory*, John Wiley and Sons, 1991.
- David J. MacKay, *Information theory, inference, and learning algorithms*, Cambridge University Press, 2011.
- Terry Welch, *A technique for high-performance data compression*, IEEE Computer (1984), 819, http://ieeexplore.ieee.org/xpls/abs_all.jsp?tp=&arnumber=1659158&isnumber=34743&tag=1.
- Jacob Ziv and Abraham Lempel, *Compression of individual sequences via variable-rate coding*, IEEE Transactions on Information Theory (1978).