COMP2610/6261 - Information Theory

Lecture 15: Shannon-Fano-Elias and Interval Coding

Robert C. Williamson

Research School of Computer Science

Australian National University

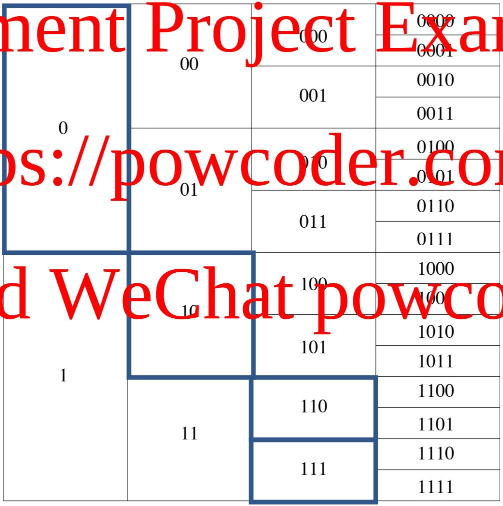
24 September, 2018

1. The Trouble with Huffman Coding

2. Interval Coding
   - Shannon-Fano-Elias Coding
   - Lossless property
   - The Prefix Property and Intervals
   - Decoding
   - Expected Length

Prefix Codes as Trees (Recap)

$$C_2 = \{0, 10, 110, 111\}$$



The Source Coding Theorem for Symbol Codes

Source Coding Theorem for Symbol Codes

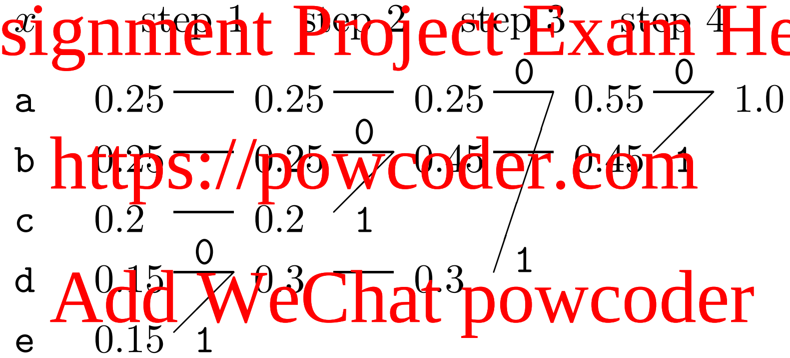
For any ensemble X there exists a *prefix code* C such that

$$H(X) \leq L(C, X) < H(X) + 1.$$

In particular, **Shannon codes** C — those with lengths $\ell_i = \left\lceil \log_2 \frac{1}{p_i} \right\rceil$ — have *expected code length within 1 bit of the entropy*.

Huffman Coding: Recap

$$\mathcal{A}_X = \{\texttt{a}, \texttt{b}, \texttt{c}, \texttt{d}, \texttt{e}\} \text{ and } \mathcal{P}_X = \{0.25, 0.25, 0.2, 0.15, 0.15\}$$



From Example 5.15 of MacKay

$$C = \{00, 10, 11, 010, 011\}$$

Huffman Coding: Advantages and Disadvantages

Advantages:

- Huffman Codes are provably optimal amongst prefix codes
- Algorithm is simple and efficient

Huffman Coding: Advantages and Disadvantages

Advantages:

- Huffman Codes are provably optimal amongst prefix codes

- Algorithm is simple and efficient

Disadvantages:

- Assumes a fixed distribution of symbols
- The extra bit in the SCT
  - If $H(X)$ is large – not a problem

  - If $H(X)$ is small (e.g., ~ 1 bit for English) codes are $2\times$ optimal

Huffman codes are the best possible symbol code
but symbol coding is not always the best type of code

This time

A different way of coding (interval coding)

Shannon-Fano-Elias codes

Worse guarantee than Huffman codes, but will lead us to the powerful arithmetic coding procedure

1. The Trouble with Huffman Coding

2. Interval Coding
   - Shannon-Fano-Elias Coding
   - Lossless property
   - The Prefix Property and Intervals
   - Decoding
   - Expected Length

Coding via Cumulative Probabilities

Suppose X is an ensemble with probabilities $(p_1, \ldots, p_{|X|})$

Define the cumulative distribution function by

$$F(x) = \sum_{i \le x} p_i$$

Coding via Cumulative Probabilities

Suppose X is an ensemble with probabilities $(p_1, \ldots, p_{|X|})$

Define the cumulative distribution function by

$$F(x) = \sum_{i \le x} p_i$$

and the modified cumulative distribution function by

$$\overline{F}(x) = \sum_{i < x} p_i + \frac{1}{2} \cdot p(x) = F(x) - \frac{1}{2} \cdot p(x)$$

Coding via Cumulative Probabilities

Suppose X is an ensemble with probabilities $(p_1, \ldots, p_{|X|})$

Define the cumulative distribution function by

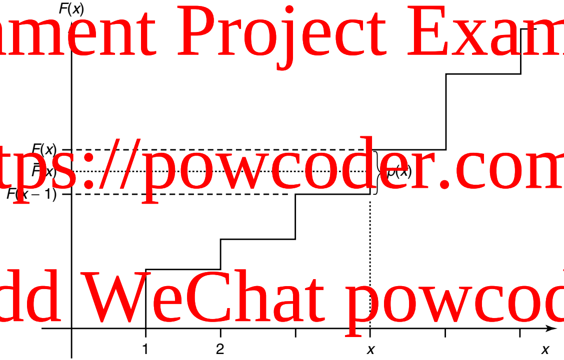
$$F(x) = \sum_{i \leq x} p_i$$

and the modified cumulative distribution function by

$$\overline{F}(x) = \sum_{i < x} p_i + \frac{1}{2} \cdot p(x) = F(x) - \frac{1}{2} \cdot p(x)$$

We can losslessly code outcomes based on $\overline{F}$!

Coding via Cumulative Probabilities



$\overline{F}(x)$ will uniquely determine each outcome x (lossless code)

Example

Suppose X has outcomes (a_1, a_2, a_3, a_4) and probabilities $(2/9, 1/9, 1/3, 1/3)$

Define the midpoint $\overline{F}(a_i) = F(a_i) - \frac{1}{2}p_i$

x	$p(x)$	$F(x)$	$\overline{F}(x)$
a_1	$2/9$	$2/9$	$1/9$
a_2	$1/9$	$1/3$	$5/18$
a_3	$1/3$	$2/3$	$1/2$
a_4	$1/3$	1	$5/6$

Example

Suppose X has outcomes (a_1, a_2, a_3, a_4) and probabilities $(2/9, 1/9, 1/3, 1/3)$

Define the midpoint $\overline{F}(a_i) = F(a_i) - \frac{1}{2}p_i$

x	$p(x)$	$F(x)$	$\overline{F}(x)$
a_1	$2/9$	$2/9$	$1/9$
a_2	$1/9$	$1/3$	$5/18$
a_3	$1/3$	$2/3$	$1/2$
a_4	$1/3$	1	$5/6$

How do we code $\overline{F}(x)$ in binary though?

Real Numbers in Binary

Real numbers are commonly expressed in decimal:

$$12_{10} \rightarrow 1 \times 10^1 + 2 \times 10^0$$

$$3.7_{10} \rightarrow 3 \times 10^0 + 7 \times 10^{-1}$$

$$0.94_{10} \rightarrow + 9 \times 10^{-1} + 4 \times 10^{-2}$$

Real Numbers in Binary

Real numbers are commonly expressed in decimal:

$$12_{10} \rightarrow 1 \times 10^1 + 2 \times 10^0$$

$$3.7_{10} \rightarrow 3 \times 10^0 + 7 \times 10^{-1}$$

$$0.94_{10} \rightarrow \quad + 9 \times 10^{-1} + 4 \times 10^{-2}$$

Some real numbers have infinite, repeating decimal expansions:

$$\frac{1}{3} = 0.33333\ldots_{10} = 0.\overline{3}_{10} \quad \text{and} \quad \frac{22}{7} = 3.14285714\ldots_{10} = 3.\overline{142857}_{10}$$

Real Numbers in Binary

Real numbers are commonly expressed in decimal:

$$12_{10} \to 1 \times 10^1 + 2 \times 10^0$$

$$3.7_{10} \to 3 \times 10^0 + 7 \times 10^{-1}$$

$$0.94_{10} \to + 9 \times 10^{-1} + 4 \times 10^{-2}$$

Some real numbers have infinite, repeating decimal expansions:

$$\tfrac{1}{3} = 0.33333\ldots_{10} = 0.\overline{3}_{10} \quad \text{and} \quad \tfrac{22}{7} = 3.14285714\ldots_{10} = 3.\overline{142857}_{10}$$

Real numbers can also be similarly expressed in binary:

$$3_{10} = 11_2 \to 1 \times 2^1 + 1 \times 2^0$$

$$1.5_{10} = 1.1_2 \to 1 \times 2^0 + 1 \times 2^{-1}$$

$$0.75_{10} = 0.11_2 \to + 1 \times 2^{-1} + 1 \times 2^{-2}$$

$$\tfrac{1}{3} = 0.010101\ldots_2 = 0.\overline{01}_2 \quad \text{and} \quad \tfrac{22}{7} = 11.001001\ldots_2 = 11.\overline{001}_2$$

Converting Decimal Fractions to Binary

To convert a fraction (e.g. $3/4$) to binary:

1. Multiply the fraction by 2. Take the whole number part of the result; this is the first bit of the binary expansion.
2. Throw away the whole number part of the result, and just retain the part after the decimal point.
3. Repeat step 1. Stop when either:
   - what remains after the decimal point is zero, or
   - you detect an infinite loop

Converting Decimal Fractions to Binary

To convert a fraction (e.g. $3/4$) to binary:

1. Multiply the fraction by 2. Take the whole number part of the result; this is the first bit of the binary expansion.
2. Throw away the whole number part of the result, and just retain the part after the decimal point.
3. Repeat step 1. Stop when either:
   - what remains after the decimal point is zero, or
   - you detect an infinite loop

Example: for 0.625_{10},

- $2 \cdot 0.625 = 1.25$, so first bit is 1
- $2 \cdot 0.25 = 0.5$, so second bit is 0
- $2 \cdot 0.5 = 1.0$, so third bit is 1
- decimal part is zero, so stop

Shannon-Fano-Elias Coding: To Infinity and Beyond

$\bar{F}(x)$ will uniquely determine each outcome...

Shannon-Fano-Elias Coding: To Infinity and Beyond

$\overline{F}(x)$ will uniquely determine each outcome... but coding $\overline{F}(x)$ naively could need infinitely many bits!

- e.g. if $\overline{F}(x) = \frac{1}{3}$

Shannon-Fano-Elias Coding: To Infinity and Beyond

$\overline{F}(x)$ will uniquely determine each outcome... but coding $\overline{F}(x)$ naively could need infinitely many bits!

- e.g. if $\overline{F}(x) = \frac{1}{3}$

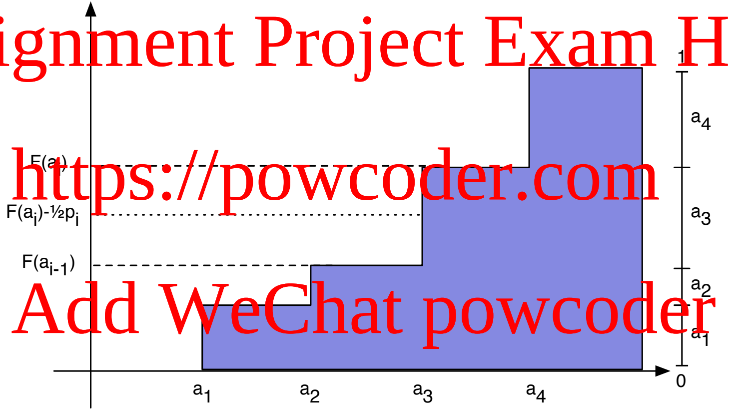
Fortunately, we can get away with only storing $\overline{F}(x)$ approximately

Shannon-Fano-Elias coding: code using the first $\ell(x) = \lceil \log_2 \frac{1}{p(x)} \rceil + 1$ bits of $\overline{F}(x)$

- (Almost) Constructive procedure for a Shannon code

Cumulative Distribution

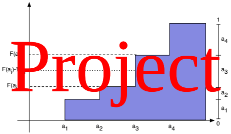
Example



Cumulative distribution for $\mathbf{p} = (\frac{2}{9}, \frac{1}{9}, \frac{1}{3}, \frac{1}{3})$

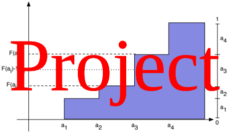
Shannon-Fano-Elias Coding

Example



Shannon-Fano-Elias Coding

Example



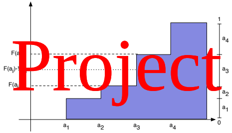
Define the midpoint $\overline{F}(a_i) = F(a_i) - \frac{1}{2}p_i$ and length $\ell(a_i) = \left\lceil \log_2 \frac{1}{p_i} \right\rceil + 1$.

Shannon-Fano-Elias Coding: code $x \in \mathcal{A}$ using first $\ell(x)$ bits of $\overline{F}(x)$.

x	$p(x)$	$F(x)$	$\overline{F}(x)$	$\overline{F}(x)_2$	$\ell(x)$	Code
a_1	$2/9$	$2/9$	$1/9$	$0.\overline{000111}_2$	4	0001
a_2	$1/9$	$1/3$	$5/18$	$0.01\overline{000111}_2$	5	01000
a_3	$1/3$	$2/3$	$1/2$	0.1_2	3	100
a_4	$1/3$	1	$5/6$	$0.1\overline{10}_2$	3	110

Shannon-Fano-Elias Coding

Example



Define the midpoint $\overline{F}(a_i) = F(a_i) - \frac{1}{2}p_i$ and length $\ell(a_i) = \left\lceil \log_2 \frac{1}{p_i} \right\rceil + 1$.

Shannon-Fano-Elias Coding: code $x \in \mathcal{A}$ using first $\ell(x)$ bits of $\overline{F}(x)$.

x	$p(x)$	$F(x)$	$\overline{F}(x)$	$\overline{F}(x)_2$	$\ell(x)$	Code
a_1	$2/9$	$2/9$	$1/9$	$0.\overline{000111}_2$	4	0001
a_2	$1/9$	$1/3$	$5/18$	$0.01\overline{000111}_2$	5	01000
a_3	$1/3$	$2/3$	$1/2$	0.1_2	3	100
a_4	$1/3$	1	$5/6$	$0.1\overline{10}_2$	3	110

Example: Sequence $\mathbf{x} = a_3 a_3 a_1$ coded as 100 100 0001.

Remaining questions

Encoding with a Shannon-Fano-Elias code is simple

But we have to check:

- is the code lossless?
- is the code prefix-free?
- how do we decode a given codeword?

1. The Trouble with Huffman Coding

2. Interval Coding
   - Shannon-Fano-Elias Coding
   - Lossless property
   - The Prefix Property and Intervals
   - Decoding
   - Expected Length

Shannon-Fano-Elias Coding: Is it lossless?

Denote the Shannon-Fano-Elias code for an outcome x by

$$\lfloor \overline{F}(x) \rfloor_{\ell(x)},$$

where $\lfloor \cdot \rfloor_{\ell}$ means truncate to first ℓ bits

Shannon-Fano-Elias Coding: Is it lossless?

Denote the Shannon-Fano-Elias code for an outcome x by

$$\lfloor \overline{F}(x) \rfloor_{\ell(x)},$$

where $\lfloor \cdot \rfloor_\ell$ means truncate to first ℓ bits

Could it be true that $x \neq x'$ but $\lfloor \overline{F}(x) \rfloor_{\ell(x)} = \lfloor \overline{F}(x') \rfloor_{\ell(x')}$?

Shannon-Fano-Elias Coding: Is it lossless?

Denote the Shannon-Fano-Elias code for an outcome x by

$$\lfloor \overline{F}(x) \rfloor_{\ell(x)},$$

where $\lfloor \cdot \rfloor_\ell$ means truncate to first ℓ bits

Could it be true that $x \neq x'$ but $\lfloor \overline{F}(x) \rfloor_{\ell(x)} = \lfloor \overline{F}(x') \rfloor_{\ell(x')}$?

No, because (homework exercise!)

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

i.e. the codeword lies entirely in the interval between $x-1$ and x

- These intervals don't overlap for different outcomes
- The code is lossless!

1. The Trouble with Huffman Coding

2. Interval Coding
   - Shannon-Fano-Elias Coding
   - Lossless property
   - The Prefix Property and Intervals
   - Decoding
   - Expected Length

Prefixes and Binary Strings

What is the set of binary strings that begin with $\mathbf{b} = \mathtt{b}_1 \ldots \mathtt{b}_n$?

Prefixes and Binary Strings

What is the set of binary strings that begin with $\mathbf{b} = \mathtt{b}_1 \ldots \mathtt{b}_n$?

$\mathtt{b}_1 \ldots \mathtt{b}_n 0, \mathtt{b}_1 \ldots \mathtt{b}_n 1, \mathtt{b}_1 \ldots \mathtt{b}_n 01, \mathtt{b}_1 \ldots \mathtt{b}_n 11, \ldots$

Prefixes and Binary Strings

What is the set of binary strings that begin with $\mathbf{b} = \mathtt{b}_1 \dots \mathtt{b}_n$?

$$\mathtt{b}_1 \dots \mathtt{b}_n 0, \mathtt{b}_1 \dots \mathtt{b}_n 1, \mathtt{b}_1 \dots \mathtt{b}_n 01, \mathtt{b}_1 \dots \mathtt{b}_n 11, \dots$$

Basically, anything ranging from

$$\mathtt{b}_1 \dots \mathtt{b}_n 000 \dots \text{ to } \mathtt{b}_1 \dots \mathtt{b}_n 111 \dots$$

These are the strings having $\mathtt{b}_1 \dots \mathtt{b}_n$ as a prefix

Prefixes and Binary Strings

We could equally associate $\mathtt{b}_1 \ldots \mathtt{b}_n$ with the fraction $0.\mathtt{b}_1 \ldots \mathtt{b}_n$

What is the set of binary strings that begin with $\mathbf{b} = \mathtt{b}_1 \ldots \mathtt{b}_n$?

Prefixes and Binary Strings

We could equally associate $b_1 \ldots b_n$ with the fraction $0.b_1 \ldots b_n$

What is the set of binary strings that begin with $b = b_1 \ldots b_n$?

$0.b_1 \ldots b_n 0, 0.b_1 \ldots b_n 1, 0.b_1 \ldots b_n 01, 0.b_1 \ldots b_n 11, \ldots$

Prefixes and Binary Strings

We could equally associate $b_1 \ldots b_n$ with the fraction $0.b_1 \ldots b_n$

What is the set of binary strings that begin with $b = b_1 \ldots b_n$?

$0.b_1 \ldots b_n 0, 0.b_1 \ldots b_n 1, 0.b_1 \ldots b_n 01, 0.b_1 \ldots b_n 11, \ldots$

Basically, anything ranging from

$$0.b_1 \ldots b_n 000 \ldots \text{ to } 0.b_1 \ldots b_n 111 \ldots$$

i.e.

$$0.b_1 \ldots b_n \text{ to } 0.b_1 \ldots b_n \overline{1}$$

Prefixes and Binary Strings

We could equally associate $\mathtt{b_1 \dots b_n}$ with the fraction $0.\mathtt{b_1 \dots b_n}$

What is the set of binary strings that begin with $\mathbf{b} = \mathtt{b_1 \dots b_n}$?

$0.\mathtt{b_1 \dots b_n}0, 0.\mathtt{b_1 \dots b_n}1, 0.\mathtt{b_1 \dots b_n}01, 0.\mathtt{b_1 \dots b_n}11, \dots$

Basically, anything ranging from

$$0.\mathtt{b_1 \dots b_n}000\dots \text{ to } 0.\mathtt{b_1 \dots b_n}111\dots$$

i.e.

$$0.\mathtt{b_1 \dots b_n} \text{ to } 0.\mathtt{b_1 \dots b_n}\overline{1}$$

Note that

$$0.\mathtt{b_1 \dots b_n}\overline{1} = 0.\mathtt{b_1 \dots b_n} + \frac{1}{2^n} = 0.\mathtt{b_1 \dots b_n} + 0.\mathtt{0 \dots 1},$$

just like $0.1\overline{9}_{10} = 0.2$

Intervals: Definition

It will be useful to analyse the prefix property in terms of intervals

An interval $[a, b)$ is the set of all the numbers at least as big as a but smaller than b. That is,

$$[a, b) = \{x : a \leq x < b\}.$$

Examples: $[0, 1)$, $[0.3, 0.6)$, $[0.2, 0.4)$.

Intervals in Binary

The set of numbers in $[0, 1)$ that start with a given sequence of bits $\mathbf{b} = b_1 \dots b_n$ form the interval

$$\left[0.\mathtt{b_1} \dots \mathtt{b_n}, 0.\mathtt{b_1} \dots \mathtt{b_n} + \frac{1}{2^n}\right) = [0.\mathtt{b_1} \dots \mathtt{b_n}, 0.\mathtt{b_1} \dots \mathtt{b_n} + 0.\mathtt{0} \dots \mathtt{1})$$

- $\mathtt{1} \to [0.\mathtt{1}, 1.0)$ $[0.5, 1]_{10}$
- $\mathtt{01} \to [0.\mathtt{01}, 0.10)$ $[0.25, 0.5)_{10}$
- $\mathtt{1101} \to [0.\mathtt{1101}, 0.1110)$ $[0.8125, 0.875)_{10}$

Prefix Property and Intervals

Prefix property (tree form): Once you pick a node in the binary tree, you cannot pick any of its descendants

Prefix property (interval form): Once you pick a codeword $\mathtt{b_1 b_2 \ldots b_n}$ you cannot pick any codeword in

$$\left[0.\mathtt{b_1 b_2 \ldots b_n}, 0.\mathtt{b_1 b_2 \ldots b_n} + \frac{1}{2^n}\right)$$

Why? This contains all binary strings for which $\mathtt{b_1 b_2 \ldots b_n}$ is a prefix

e.g. If we pick $\mathtt{0110}$, we cannot pick anything from

$$\begin{aligned}[0.0110, 0.0111) &= [0.0110\overline{0}, 0.0110\overline{1}) \\ &= \{0.0110, 0.01101, 0.011001, 0.011011, \ldots,\}\end{aligned}$$

Prefix Property and Intervals

If $\mathbf{b}'$ is a prefix of $\mathbf{b}$, the interval for $\mathbf{b}$ is contained in the interval for $\mathbf{b}'$

e.g. $\mathbf{b}' = 01$ is prefix of $\mathbf{b} = 0101$ so $\underbrace{[0.0101, 0.0110)}_{[0.3125,0.375)_{10}} \subset \underbrace{[0.01, 0.10)}_{[0.25,0.5)_{10}}$

Why? Because interval for $\mathbf{b}'$ contains all strings for which $\mathbf{b}'$ is a prefix

- And if $\mathbf{b}$ has $\mathbf{b}'$ as a prefix, so does anything having $\mathbf{b}$ as a prefix

Prefix Property and Intervals

If $\mathbf{b}'$ is a prefix of $\mathbf{b}$, the interval for $\mathbf{b}$ is contained in the interval for $\mathbf{b}'$

e.g. $\mathbf{b}' = 01$ is prefix of $\mathbf{b} = 0101$ so $\underbrace{[0.0101, 0.0110)}_{[0.3125, 0.375)_{10}} \subset \underbrace{[0.01, 0.10)}_{[0.25, 0.5)_{10}}$

Why? Because interval for $\mathbf{b}'$ contains all strings for which $\mathbf{b}'$ is a prefix

- And if $\mathbf{b}$ has $\mathbf{b}'$ as a prefix, so does anything having $\mathbf{b}$ as a prefix

Implication: If intervals for $\mathbf{b}, \mathbf{b}'$ are disjoint, one cannot be a prefix of another

Shannon-Fano-Elias Coding is Prefix-Free

We already know $\lfloor \overline{F}(x) \rfloor_{\ell(x)} > F(x-1)$. We also have

$$\begin{aligned}
\lfloor \overline{F}(x) \rfloor_{\ell(x)} + \frac{1}{2^{\ell}} &\leq \overline{F}(x) + \frac{1}{2^{\ell}} \\
&\leq \overline{F}(x) + \frac{p(x)}{2} \\
&= F(x)
\end{aligned}$$

and so

$$\left[\lfloor \overline{F}(x) \rfloor_{\ell(x)}, \lfloor \overline{F}(x) \rfloor_{\ell(x)} + \frac{1}{2^{\ell}} \right) \subset [F(x-1), F(x))$$

The intervals for each codeword are thus trivially disjoint, since we know each of the $[F(x-1), F(x))$ intervals is disjoint

The SFE code is prefix-free!

Two Types of Interval

The **symbol interval** for some outcome x_i is (assuming $F(x_0) = 0$)

$$[F(x_{i-1}), F(x_i))$$

These intervals are disjoint for each outcome

The **codeword interval** for some outcome x_i is

$$\left[\lfloor \overline{F}(x_i) \rfloor_{\ell(x_i)}, \lfloor \overline{F}(x_i) \rfloor_{\ell(x_i)} + \frac{1}{2^{\ell(x_i)}} \right)$$

This is a strict subset of the symbol interval

All strings in the codeword interval start with the same prefix

- This is **not true** in general for the symbol interval

1. The Trouble with Huffman Coding

2. Interval Coding
   - Shannon-Fano-Elias Coding
   - Lossless property
   - The Prefix Property and Intervals
   - Decoding
   - Expected Length

Shannon-Fano-Elias Decoding

To decode a given bitstring:

1. start with the first bit, and compute the corresponding binary interval

2. if the interval is strictly contained within that of a codeword:
    1. output the codeword

    2. skip over any redundant bits for this codeword
    3. repeat (1) for the rest of the bitstring

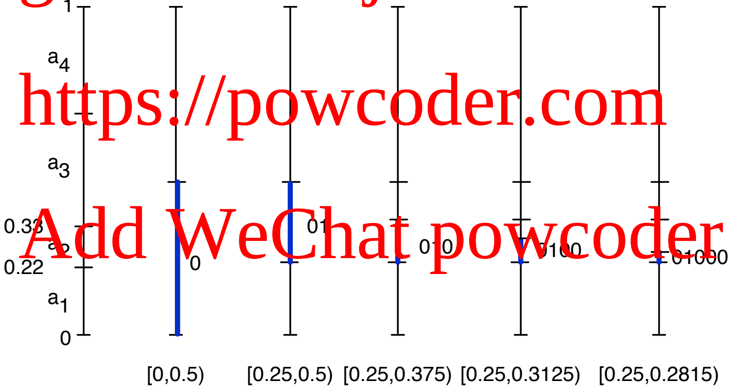
3. else include next bit, and compute the corresponding binary interval
4. ⋮

We might be able to stop early owing to redundancies in SFE

Shannon-Fano-Elias Decoding

Let $\mathbf{p} = \{\frac{2}{9}, \frac{1}{9}, \frac{1}{3}, \frac{1}{3}\}$. Suppose we want to *decode* 01000:

Find symbol interval containing codeword interval for $\texttt{01000} = [0.25, 0.28125)_{10}$



We could actually stop once we see $\texttt{0100}$, since $[0.25, 0.3125) \subset [0.22, 0.33]$

1. The Trouble with Huffman Coding

2. Interval Coding
   - Shannon-Fano-Elias Coding
   - Lossless property
   - The Prefix Property and Intervals
   - Decoding
   - Expected Length

Expected Code Length of SFE Code

The extra bit for the code lengths is because we code $\frac{p_i}{2}$ and

$$\log_2 \frac{2}{p_i} = \log_2 \frac{1}{p_i} + \log_2 2 = \log_2 \frac{1}{p_i} + 1$$

What is the expected length of a SFE code C for ensemble X with probabilities $\mathbf{p}$?

$$\begin{aligned} L(C, X) = \sum_{i=1}^{K} p_i \ell(a_i) &= \sum_{i=1}^{K} p_i \left(\left\lceil \log_2 \frac{1}{p_i} \right\rceil + 1 \right) \\ &\leq \sum_{i=1}^{K} p_i \left(\log_2 \frac{1}{p_i} + 2 \right) \\ &= H(X) + 2 \end{aligned}$$

Similarly, $H(X) + 1 \leq L(C, X)$ for the SFE codes.

Why bother?

Let X be an ensemble, C_{SFE} be a Shannon-Fano-Elias code for X and C_H be a Huffman code for X

$$\underbrace{H(X) \leq L(C_H, X)}_{\text{Source Coding Theorem}} \leq H(X) + 1 \leq L(C_{SFE}, X) \leq H(X) + 2$$

so why not just use Huffman codes?

SFE is a stepping stone to a more powerful type of codes

- Roughly, try to apply SFE to a block of outcomes

Main points:

- Problems with Huffman coding symbol distribution

- Binary strings to/from intervals in [0, 1]
- Shannon-Fano-Elias Coding:
  - Code C via cumulative distribution function for p
  - $H(X) + 1 \le L(C, X) \le H(X) + 2$

- Extra bit guarantees interval containment

Summary and Reading

Main points:

- Problems with Huffman coding symbol distribution
- Binary strings to/from intervals in $[0, 1]$
- Shannon-Fano-Elias Coding:
  - Code C via cumulative distribution function for p
  - $H(X) + 1 \le L(C, X) \le H(X) + 2$
- Extra bit guarantees interval containment

Reading:

- Interval coding: MacKay §6.1 and §6.2
- Shannon-Fano-Elias Coding: Cover & Thomas §5.9

Summary and Reading

Main points:

- Problems with Huffman coding symbol distribution
- Binary strings to/from intervals in $[0, 1]$

- Shannon-Fano-Elias Coding:
  - Code C via cumulative distribution function for p
  - $H(X) + 1 \le L(C, X) \le H(X) + 2$

- Extra bit guarantees interval containment

Reading:

- Interval coding: MacKay §6.1 and §6.2

- Shannon-Fano-Elias Coding: Cover & Thomas §5.9

Next time:

Extending SFE Coding to sequences of symbols