Information-Theoretic Modeling

Lecture 7: Source Coding: Practice (continued)

Teemu Roos

Department of Computer Science, University of Helsinki

Fall 2009

Lecture 7: Source Coding: Practice (continued)

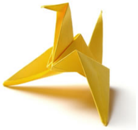
Concentric Circular Tower
(David Huffman)

[Photo: Tony Grant. Courtesy of the Huffman family.]

"Design with the help of binary code (0 and 1) the most efficient method to represent characters, figures and symbols."

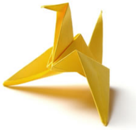
(Assignment at Prof. R.M. Fano's 1952 MIT Information Theory course.)

1. Symbol Codes
    - Entropy lower bound
    - Shannon-Fano Coding
    - Huffman



1. Symbol Codes
   - Entropy lower bound
   - Shannon-Fano Coding
   - Huffman

2. Beyond Symbols Codes
   - Problems with Symbol Codes
   - Two-Part Codes
   - Block Codes
   - Arithmetic Coding



Entropy Lower Bound

So what have we learned?

Entropy Lower Bound

So what have we learned? For decodable symbols codes:

1. $E[\ell(X)] - H(X) = D(p \parallel q) + \log_2 \frac{1}{c}$, where $q(x) = \dfrac{2^{-\ell(x)}}{c}$.

Entropy Lower Bound

So what have we learned? For decodable symbols codes:

1. $E[\ell(X)] - H(X) = D(p \parallel q) + \log_2 \frac{1}{c}$, where $q(x) = \dfrac{2^{-\ell(x)}}{c}$.
2. $E[\ell(X)] \geq H(X)$.

Entropy Lower Bound

So what have we learned? For decodable symbols codes:

1. $E[\ell(X)] - H(X) = D(p \parallel q) + \log_2 \frac{1}{c}$, where $q(x) = \dfrac{2^{-\ell(x)}}{c}$.
2. $E[\ell(X)] \geq H(X)$.
3. If $\ell(x) = \log_2 \frac{1}{p(x)}$, then $E[\ell(X)] = H(X)$. **Optimal!**

Entropy Lower Bound

So what have we learned? For decodable symbols codes:

1. $E[\ell(X)] - H(X) = D(p \parallel q) + \log_2 \frac{1}{c}$, where $q(x) = \dfrac{2^{-\ell(x)}}{c}$.
2. $E[\ell(X)] \geq H(X)$.
3. If $\ell(x) = \log_2 \frac{1}{p(x)}$, then $E[\ell(X)] = H(X)$. **Optimal!**

Note also that for a sequence $X_1, \ldots, X_n$ the expected codelength becomes

$$E[\ell(X_1, \ldots, X_n)] = E\left[\sum_{i=1}^{n} \ell(X_i)\right]$$

Entropy Lower Bound

So what have we learned? For decodable symbols codes:

1. $E[\ell(X)] - H(X) = D(p \parallel q) + \log_2 \frac{1}{c}$, where $q(x) = \dfrac{2^{-\ell(x)}}{c}$.
2. $E[\ell(X)] \geq H(X)$.
3. If $\ell(x) = \log_2 \frac{1}{p(x)}$, then $E[\ell(X)] = H(X)$. **Optimal!**

Note also that for a sequence $X_1, \ldots, X_n$ the expected codelength becomes

$$E[\ell(X_1, \ldots, X_n)] = E\left[\sum_{i=1}^{n} \ell(X_i)\right] = \sum_{i=1}^{n} E[\ell(X_i)]$$

Entropy Lower Bound

So what have we learned? For decodable symbols codes:

1. $E[\ell(X)] - H(X) = D(p \parallel q) + \log_2 \frac{1}{c}$, where $q(x) = \dfrac{2^{-\ell(x)}}{c}$.
2. $E[\ell(X)] \geq H(X)$.
3. If $\ell(x) = \log_2 \frac{1}{p(x)}$, then $E[\ell(X)] = H(X)$. **Optimal!**

Note also that for a sequence $X_1, \ldots, X_n$ the expected codelength becomes

$$E[\ell(X_1, \ldots, X_n)] = E\left[\sum_{i=1}^{n} \ell(X_i)\right] = \sum_{i=1}^{n} E[\ell(X_i)] = nH(X) \ .$$

Entropy Lower Bound

So what have we learned? For decodable symbols codes:

1. $E[\ell(X)] - H(X) = D(p \parallel q) + \log_2 \frac{1}{c}$, where $q(x) = \dfrac{2^{-\ell(x)}}{c}$.
2. $E[\ell(X)] \geq H(X)$.
3. If $\ell(x) = \log_2 \frac{1}{p(x)}$, then $E[\ell(X)] = H(X)$. **Optimal!**

Note also that for a sequence $X_1, \ldots, X_n$ the expected codelength becomes

$$E[\ell(X_1, \ldots, X_n)] = E\left[\sum_{i=1}^{n} \ell(X_i)\right] = \sum_{i=1}^{n} E[\ell(X_i)] = nH(X) \ .$$

! By Shannon's Noiseless Channel Coding Theorem, this is optimal among all codes, **not only symbol codes.**

Entropy Lower Bound

So what have we learned? For decodable symbols codes:

1. $E[\ell(X)] - H(X) = D(p \parallel q) + \log_2 \frac{1}{c}$, where $q(x) = \dfrac{2^{-\ell(x)}}{c}$.
2. $E[\ell(X)] \geq H(X)$.
3. If $\ell(x) = \log_2 \frac{1}{p(x)}$, then $E[\ell(X)] = H(X)$. **Optimal!**

Note also that for a sequence $X_1, \ldots, X_n$ the expected codelength becomes

$$E[\ell(X_1, \ldots, X_n)] = E\left[\sum_{i=1}^{n} \ell(X_i)\right] = \sum_{i=1}^{n} E[\ell(X_i)] = nH(X) \ .$$

! By Shannon's Noiseless Channel Coding Theorem, this is optimal among all codes, **not only symbol codes.** Fine print: only if X_i i.i.d.!

Codelengths and Probabilities

The only problem with the $\ell(x) = \log_2 \frac{1}{p(x)}$ codeword choice is the requirement that codeword lengths must be **integers** (try to think about a codeword with length 0.123, for instance), while the so obtained ℓ is not in general an integer.

Codelengths and Probabilities

The only problem with the $\ell(x) = \log_2 \frac{1}{p(x)}$ codeword choice is the requirement that codeword lengths must be **integers** (try to think about a codeword with length 0.123, for instance), while the so obtained ℓ is not in general an integer.

The simplest solution is to round upwards:

Shannon's Code

Given a pmf, the **Shannon code** has the codeword lengths

$$\ell(x) = \left\lceil \log_2 \frac{1}{p(x)} \right\rceil \quad \text{for all } x \in \mathcal{X}.$$

Alice in Wonderland

Shannon's code: Example

	X	$p(X)$	$\log_2 \frac{1}{p(X)}$	$\ell(X)$
	a	0.0644	3.9	4
	b	0.0108	6.5	7
	c	0.0178	5.8	6
	d	0.0359	4.7	5
	e	0.0991	3.3	4
	f	0.0147	6.0	7
	g	0.0184	5.7	6
	h	0.0535	4.2	5
	i	0.0551	4.1	5
	j	0.0011	9.8	10
	k	0.0083	6.8	7
	l	0.0343	4.8	5
		⋮		
	y	0.0165	5.9	6
	z	0.0005	10.7	11
		0.2111	2.2	3

$$H(X) = 4.03$$

Shannon's code: Example

	X	$p(X)$	$\log_2 \frac{1}{p(X)}$	$\ell(X)$
	a	0.0644	3.9	4
	b	0.0108	6.5	7
	c	0.0178	5.8	6
	d	0.0359	4.7	5
	e	0.0991	3.3	4
	f	0.0147	6.0	7
	g	0.0184	5.7	6
	h	0.0535	4.2	5
	i	0.0551	4.1	5
	j	0.0011	9.8	10
	k	0.0083	6.8	7
	l	0.0343	4.8	5
		⋮		
	y	0.0165	5.9	6
	z	0.0005	10.7	11
		0.2111	2.2	3

$H(X) = 4.03$

Shannon (1948):

Shannon's code: Example

X	$p(X)$	$\log_2 \frac{1}{p(X)}$	$\ell(X)$
a	0.0644	3.9	4
b	0.0108	6.5	7
c	0.0178	5.8	6
d	0.0359	4.7	5
e	0.0991	3.3	4
f	0.0147	6.0	7
g	0.0184	5.7	6
h	0.0535	4.2	5
i	0.0551	4.1	5
j	0.0011	9.8	10
k	0.0083	6.8	7
l	0.0343	4.8	5
	⋮		
y	0.0165	5.9	6
z	0.0005	10.7	11
	0.2111	2.2	3

$H(X) = 4.03$

Shannon (1948):

1. Sort by probability.

Shannon's code: Example

X	$p(X)$	$\log_2 \frac{1}{p(X)}$	$\ell(X)$
	0.2111	2.2	3
e	0.0991	3.3	4
t	0.0781	3.6	4
a	0.0644	3.9	4
o	0.0598	4.0	5
i	0.0551	4.1	5
h	0.0535	4.2	5
n	0.0516	4.2	5
s	0.0475	4.3	5
r	0.0401	4.6	5
d	0.0359	4.7	5
l	0.0343	4.8	5
	⋮		
x	0.0011	9.8	10
j	0.0011	9.8	10
z	0.0005	10.7	11

$H(X) = 4.03$

Shannon (1948):

1. Sort by probability.

Shannon's code: Example

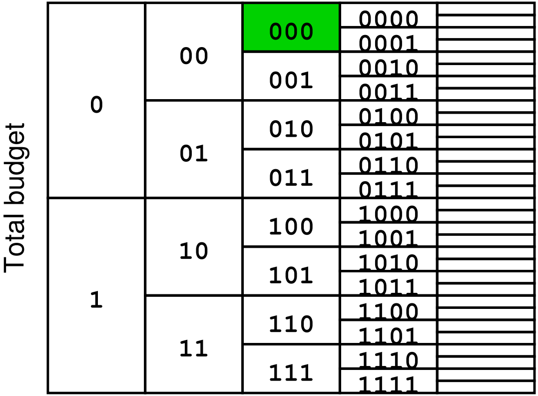
X	$p(X)$	$\log_2 \frac{1}{p(X)}$	$\ell(X)$
	0.2111	2.2	3
e	0.0991	3.3	4
t	0.0781	3.6	4
a	0.0644	3.9	4
o	0.0598	4.0	5
i	0.0551	4.1	5
h	0.0535	4.2	5
n	0.0516	4.2	5
s	0.0475	4.3	5
r	0.0401	4.6	5
d	0.0359	4.7	5
l	0.0343	4.8	5
	⋮		
x	0.0011	9.8	10
j	0.0011	9.8	10
z	0.0005	10.7	11

$H(X) = 4.03$

Shannon (1948):

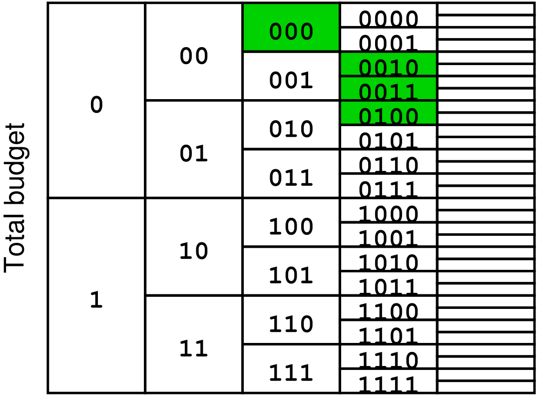
1. Sort by probability.
2. Choose codewords in order, avoiding prefixes. ("Kraft table"!)

Shannon's code: Example



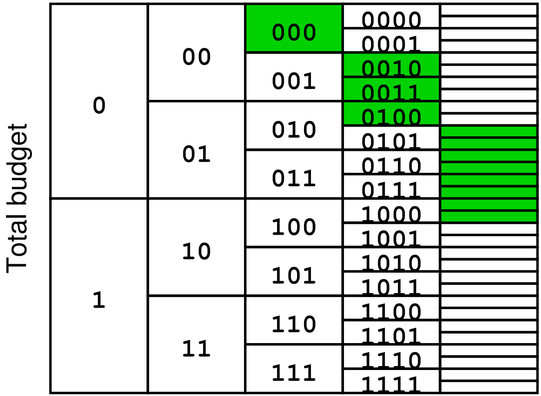
Codeword lengths $({\color{red}3}, 4, 4, 4, 5, 5, 5, 5, \ldots, 10, 10, 11)$

Shannon's code: Example



Codeword lengths $(3, 4, 4, 4, 5, 5, 5, 5, \ldots, 10, 10, 11)$

Shannon's code: Example



Codeword lengths $(3, 4, 4, 4, 5, 5, 5, 5, \ldots, 10, 10, 11)$

Shannon's code: Example

X	$p(X)$	$\log_2 \frac{1}{p(X)}$	$\ell(X)$	$C(X)$
	0.2111	2.2	3	000
e	0.0991	3.3	4	0010
t	0.0781	3.6	4	0011
a	0.0644	3.9	4	0100
o	0.0598	4.0	5	01010
i	0.0551	4.1	5	01011
h	0.0535	4.2	5	01100
n	0.0516	4.2	5	01101
s	0.0475	4.3	5	01110
r	0.0401	4.6	5	01111
d	0.0359	4.7	5	10000
l	0.0343	4.8	5	10001
	⋮			
x	0.0011	9.8	10	1010111101
j	0.0011	9.8	10	1010111110
z	0.0005	10.7	11	10101111110

Shannon's code: Example

X	$p(X)$	$\log_2 \frac{1}{p(X)}$	$\ell(X)$	$C(X)$
	0.2111	2.2	3	000
e	0.0991	3.3	4	0010
t	0.0781	3.6	4	0011
a	0.0644	3.9	4	0100
o	0.0598	4.0	5	01010
i	0.0551	4.1	5	01011
h	0.0535	4.2	5	01100
n	0.0516	4.2	5	01101
s	0.0475	4.3	5	01110
r	0.0401	4.6	5	01111
d	0.0359	4.7	5	10000
l	0.0343	4.8	5	10001
	⋮			
x	0.0011	9.8	10	1010111101
j	0.0011	9.8	10	1010111110
z	0.0005	10.7	11	10101111110

$$H(X) = 4.03$$

$$E[\ell(X)] = 4.60$$

$$E[\ell(X)] - H(X) = 0.57$$

Shannon's code

The expected codeword length of Shannon's code is

$$E\left[\ell(X)\right] = E\left[\left\lceil \log_2 \frac{1}{p(X)} \right\rceil\right] < E\left[\log_2 \frac{1}{p(X)} + 1\right] = H(X) + 1 \ .$$

Shannon's code

The expected codeword length of Shannon's code is

$$E\left[\ell(X)\right] = E\left[\left\lceil \log_2 \frac{1}{p(X)} \right\rceil\right] < E\left[\log_2 \frac{1}{p(X)} + 1\right] = H(X) + 1 \ .$$

In the Alice example we had

$$E[\ell(X)] - H(X) = 4.60 - 4.03 = 0.57 < 1 \ .$$

Fano: Example

X	$p(X)$	$\log_2 \frac{1}{p(X)}$
a	0.0644	3.9
b	0.0108	6.5
c	0.0178	5.8
d	0.0359	4.7
e	0.0991	3.3
f	0.0147	6.0
g	0.0184	5.7
h	0.0535	4.2
i	0.0551	4.1
j	0.0011	9.8
k	0.0083	6.8
l	0.0343	4.8
	⋮	
y	0.0165	5.9
z	0.0005	10.7
	0.2111	2.2

Fano: Example

X	$p(X)$	$\log_2 \frac{1}{p(X)}$
a	0.0644	3.9
b	0.0108	6.5
c	0.0178	5.8
d	0.0359	4.7
e	0.0991	3.3
f	0.0147	6.0
g	0.0184	5.7
h	0.0535	4.2
i	0.0551	4.1
j	0.0011	9.8
k	0.0083	6.8
l	0.0343	4.8
	⋮	
y	0.0165	5.9
z	0.0005	10.7
	0.2111	2.2

(Shannon-)Fano code:

Fano: Example

X	$p(X)$	$\log_2 \frac{1}{p(X)}$
a	0.0644	3.9
b	0.0108	6.5
c	0.0178	5.8
d	0.0359	4.7
e	0.0991	3.3
f	0.0147	6.0
g	0.0184	5.7
h	0.0535	4.2
i	0.0551	4.1
j	0.0011	9.8
k	0.0083	6.8
l	0.0343	4.8
	⋮	
y	0.0165	5.9
z	0.0005	10.7
	0.2111	2.2

(Shannon-)Fano code:

1. Sort by probability.

Fano: Example

X	$p(X)$	$\log_2 \frac{1}{p(X)}$
	0.2111	2.2
e	0.0991	3.3
t	0.0781	3.6
a	0.0644	3.9
o	0.0598	4.0
i	0.0551	4.1
h	0.0535	4.2
n	0.0516	4.2
s	0.0475	4.3
r	0.0401	4.6
d	0.0359	4.7
l	0.0343	4.8
	⋮	
x	0.0011	9.8
j	0.0011	9.8
z	0.0005	10.7

(Shannon-)Fano code:

1. Sort by probability.

Fano: Example

X	$p(X)$	$\log_2 \frac{1}{p(X)}$
	0.2111	2.2
e	0.0991	3.3
t	0.0781	3.6
a	0.0644	3.9
o	0.0598	4.0
i	0.0551	4.1
h	0.0535	4.2
n	0.0516	4.2
s	0.0475	4.3
r	0.0401	4.6
d	0.0359	4.7
l	0.0343	4.8
	⋮	
x	0.0011	9.8
j	0.0011	9.8
z	0.0005	10.7

(Shannon-)Fano code:

1. Sort by probability.
2. Divide in two equally probable parts (as equal as possible)

Fano: Example

X	$p(X)$	$\log_2 \frac{1}{p(X)}$
	0.2111	2.2
e	0.0991	3.3
t	0.0781	3.6
a	0.0644	3.9
o	0.0598	4.0
i	0.0551	4.1
h	0.0535	4.2
n	0.0516	4.2
s	0.0475	4.3
r	0.0401	4.6
d	0.0359	4.7
l	0.0343	4.8
	⋮	
x	0.0011	9.8
j	0.0011	9.8
z	0.0005	10.7

(Shannon-)Fano code:

1. Sort by probability.
2. Divide in two equally probable parts (as equal as possible)
3. Add '0' to the codewords in the first part, '1' to the others.

Fano: Example

X	$p(X)$	$\log_2 \frac{1}{p(X)}$
	0.2111	2.2
e	0.0991	3.3
t	0.0781	3.6
a	0.0644	3.9
o	0.0598	4.0
i	0.0551	4.1
h	0.0535	4.2
n	0.0516	4.2
s	0.0475	4.3
r	0.0401	4.6
d	0.0359	4.7
l	0.0343	4.8
	⋮	
x	0.0011	9.8
j	0.0011	9.8
z	0.0005	10.7

(Shannon-)Fano code:

1. Sort by probability.
2. Divide in two equally probable parts (as equal as possible)
3. Add '0' to the codewords in the first part, '1' to the others.
4. Repeat recursively for both parts.

Fano: Example

	X	$p(X)$	$\log_2 \frac{1}{p(X)}$	$\ell(X)$	$C(X)$
		0.2111	2.2	2	0
	e	0.0991	3.3	4	0
	t	0.0781	3.6	4	0
	a	0.0644	3.9	4	0
	o	0.0598	4.0	4	0
	i	0.0551	4.1	4	1
	h	0.0535	4.2	4	1
	n	0.0516	4.2	4	1
	s	0.0475	4.3	5	1
	r	0.0401	4.6	5	1
	d	0.0359	4.7	5	1
	l	0.0343	4.8	5	1
		⋮			
	x	0.0011	9.8	10	1
	j	0.0011	9.8	10	1
	z	0.0005	10.7	10	1

Fano: Example

	X	$p(X)$	$\log_2 \frac{1}{p(X)}$	$\ell(X)$	$C(X)$
		0.2111	2.2	2	00
	e	0.0991	3.3	4	01
	t	0.0781	3.6	4	01
	a	0.0644	3.9	4	01
	o	0.0598	4.0	4	01
	i	0.0551	4.1	4	10
	h	0.0535	4.2	4	10
	n	0.0516	4.2	4	10
	s	0.0475	4.3	5	10
	r	0.0401	4.6	5	10
	d	0.0359	4.7	5	11
	l	0.0343	4.8	5	11
		⋮			
	x	0.0011	9.8	10	11
	j	0.0011	9.8	10	11
	z	0.0005	10.7	10	11

Fano: Example

X	$p(X)$	$\log_2 \frac{1}{p(X)}$	$\ell(X)$	$C(X)$
	0.2111	2.2	2	00
e	0.0991	3.3	4	010
t	0.0781	3.6	4	010
a	0.0644	3.9	4	011
o	0.0598	4.0	4	011
i	0.0551	4.1	4	100
h	0.0535	4.2	4	100
n	0.0516	4.2	4	101
s	0.0475	4.3	5	101
r	0.0401	4.6	5	101
d	0.0359	4.7	5	110
l	0.0343	4.8	5	110
	⋮			
x	0.0011	9.8	10	111
j	0.0011	9.8	10	111
z	0.0005	10.7	10	111

Fano: Example

X	$p(X)$	$\log_2 \frac{1}{p(X)}$	$\ell(X)$	$C(X)$
	0.2111	2.2	2	00
e	0.0991	3.3	4	0100
t	0.0781	3.6	4	0101
a	0.0644	3.9	4	0110
o	0.0598	4.0	4	0111
i	0.0551	4.1	4	1000
h	0.0535	4.2	4	1001
n	0.0516	4.2	4	1010
s	0.0475	4.3	5	1011
r	0.0401	4.6	5	1011
d	0.0359	4.7	5	1100
l	0.0343	4.8	5	1100
	⋮			
x	0.0011	9.8	10	1111
j	0.0011	9.8	10	1111
z	0.0005	10.7	10	1111

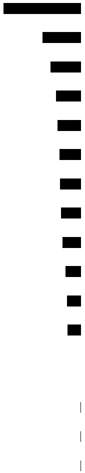
Fano: Example

	X	$p(X)$	$\log_2 \frac{1}{p(X)}$	$\ell(X)$	$C(X)$
		0.2111	2.2	2	00
	e	0.0991	3.3	4	0100
	t	0.0781	3.6	4	0101
	a	0.0644	3.9	4	0110
	o	0.0598	4.0	4	0111
	i	0.0551	4.1	4	1000
	h	0.0535	4.2	4	1001
	n	0.0516	4.2	4	1010
	s	0.0475	4.3	5	10110
	r	0.0401	4.6	5	10111
	d	0.0359	4.7	5	11000
	l	0.0343	4.8	5	11001
		⋮			
	x	0.0011	9.8	10	11111
	j	0.0011	9.8	10	11111
	z	0.0005	10.7	10	11111

Fano: Example

X	$p(X)$	$\log_2 \frac{1}{p(X)}$	$\ell(X)$	$C(X)$
	0.2111	2.2	2	00
e	0.0991	3.3	4	0100
t	0.0781	3.6	4	0101
a	0.0644	3.9	4	0110
o	0.0598	4.0	4	0111
i	0.0551	4.1	4	1000
h	0.0535	4.2	4	1001
n	0.0516	4.2	4	1010
s	0.0475	4.3	5	10110
r	0.0401	4.6	5	10111
d	0.0359	4.7	5	11000
l	0.0343	4.8	5	11001
	$\vdots$			
x	0.0011	9.8	10	111111
j	0.0011	9.8	10	111111
z	0.0005	10.7	10	111111

Fano: Example



X	$p(X)$	$\log_2 \frac{1}{p(X)}$	$\ell(X)$	$C(X)$
	0.2111	2.2	2	00
e	0.0991	3.3	4	0100
t	0.0781	3.6	4	0101
a	0.0644	3.9	4	0110
o	0.0598	4.0	4	0111
i	0.0551	4.1	4	1000
h	0.0535	4.2	4	1001
n	0.0516	4.2	4	1010
s	0.0475	4.3	5	10110
r	0.0401	4.6	5	10111
d	0.0359	4.7	5	11000
l	0.0343	4.8	5	11001
	⋮			
x	0.0011	9.8	10	1111111101
j	0.0011	9.8	10	1111111110
z	0.0005	10.7	10	1111111111

Fano: Example

X	$p(X)$	$\log_2 \frac{1}{p(X)}$	$\ell(X)$	$C(X)$
	0.2111	2.2	2	00
e	0.0991	3.3	4	0100
t	0.0781	3.6	4	0101
a	0.0644	3.9	4	0110
o	0.0598	4.0	4	0111
i	0.0551	4.1	4	1000
h	0.0535	4.2	4	1001
n	0.0516	4.2	4	1010
s	0.0475	4.3	5	10110
r	0.0401	4.6	5	10111
d	0.0359	4.7	5	11000
l	0.0343	4.8	5	11001
	⋮			
x	0.0011	9.8	10	1111111101
j	0.0011	9.8	10	1111111110
z	0.0005	10.7	10	1111111111

$$H(X) = 4.03$$

$$E[\ell(X)] = 4.07$$

$$E[\ell(X)] - H(X) = 0.04$$

Shannon-Fano Code

The expected codeword length of the Shannon-Fano code is

$$E\left[\ell(X)\right] \leq H(X) + 1 \ .$$

[corrected on Oct 6, 2009]

Shannon-Fano Code

The expected codeword length of the Shannon-Fano code is

$$E\left[\ell(X)\right] \leq H(X) + 1 \quad .$$

[corrected on Oct 6, 2009]

In the Alice example we had

$$E[\ell(X)] - H(X) = 4.06 - 4.03 = 0.04 \leq 1 \quad .$$

Shannon-Fano Code

The expected codeword length of the Shannon-Fano code is

$$E\left[\ell(X)\right] \leq H(X) + 1 \quad .$$ [corrected on Oct 6, 2009]

In the Alice example we had

$$E[\ell(X)] - H(X) = 4.06 - 4.03 = 0.04 \leq 1 \quad .$$

Is this optimal?

Shannon-Fano Code

The expected codeword length of the Shannon-Fano code is

$$E\left[\ell(X)\right] \leq H(X) + 1 \quad .$$

[corrected on Oct 6, 2009]

In the Alice example we had

$$E[\ell(X)] - H(X) = 4.06 - 4.03 = 0.04 \leq 1 \quad .$$

Is this optimal? Not necessarily — Huffman!

Huffman Code

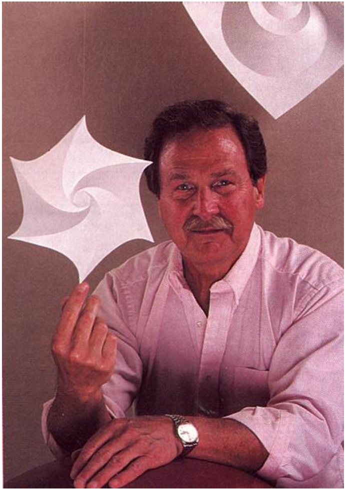
So the Shannon-Fano code is not the optimal symbol code. This is where Professor Fano and a student called David Huffman enter:

Huffman Code

So the Shannon-Fano code is not the optimal symbol code. This is where Professor Fano and a student called David Huffman enter:

`"Design with the help of binary code (0 and 1) the most efficient method to represent characters, figures and symbols."`

David Huffman (1925–1999)



Huffman Code: Algorithm

Huffman's algorithm proceeds as follows:

Huffman Code: Algorithm

Huffman's algorithm proceeds as follows:

1. Sort all symbols by their probabilities p_i.

Huffman Code: Algorithm

Huffman's algorithm proceeds as follows:

1. Sort all symbols by their probabilities p_i.
2. Join the two least probable symbols, i and j, and remove them from the list. Add a new *pseudosymbol* whose probability is $p_i + p_j$.

Huffman Code: Algorithm

Huffman's algorithm proceeds as follows:

1. Sort all symbols by their probabilities p_i.
2. Join the two least probable symbols, i and j, and remove them from the list. Add a new *pseudosymbol* whose probability is $p_i + p_j$.
3. If there is more than one symbol left, go to Step 1.

Huffman Code: Algorithm

Huffman's algorithm proceeds as follows:

1. Sort all symbols by their probabilities p_i.
2. Join the two least probable symbols, i and j, and remove them from the list. Add a new *pseudosymbol* whose probability is $p_i + p_j$.
3. If there is more than one symbol left, go to Step 1.
4. Use the resulting binary tree to define the codewords.

Huffman Code: Algorithm

Huffman's algorithm proceeds as follows:

1. Sort all symbols by their probabilities p_i.
2. Join the two least probable symbols, i and j, and remove them from the list. Add a new *pseudosymbol* whose probability is $p_i + p_j$.
3. If there is more than one symbol left, go to Step 1.
4. Use the resulting binary tree to define the codewords.

See the demo at
`www.cs.auckland.ac.nz/software/AlgAnim/huffman.html`

Huffman Code: Optimality

The reason why the Huffman code is the optimal symbol code (shortest expected codelength) is roughly as follows:

Huffman Code: Optimality

The reason why the Huffman code is the optimal symbol code (shortest expected codelength) is roughly as follows:

It can be shown that there is an optimal code (not necessarily unique) such that

1. If $p(x) > p(y)$, then $\ell(x) \leq \ell(y)$.

Huffman Code: Optimality

The reason why the Huffman code is the optimal symbol code (shortest expected codelength) is roughly as follows:

It can be shown that there is an optimal code (not necessarily unique) such that

1. If $p(x) > p(y)$, then $\ell(x) \leq \ell(y)$.
2. The longest two codewords have the same length.

Huffman Code: Optimality

The reason why the Huffman code is the optimal symbol code (shortest expected codelength) is roughly as follows:

It can be shown that there is an optimal code (not necessarily unique) such that

1. If $p(x) > p(y)$, then $\ell(x) \leq \ell(y)$.
2. The longest two codewords have the same length.
3. The longest two codewords differ only at the last bit and correspond to the two least probable symbols.

Huffman Code: Optimality

The reason why the Huffman code is the optimal symbol code (shortest expected codelength) is roughly as follows:

It can be shown that there is an optimal code (not necessarily unique) such that

1. If $p(x) > p(y)$, then $\ell(x) \leq \ell(y)$.
2. The longest two codewords have the same length.
3. The longest two codewords differ only at the last bit and correspond to the two least probable symbols.

Points 2 & 3 suggest the first step of Huffman's algorithm. Any subtree must satisfy the same conditions $\Rightarrow$ Induction.

Huffman Code: Optimality

The reason why the Huffman code is the optimal symbol code (shortest expected codelength) is roughly as follows:

It can be shown that there is an optimal code (not necessarily unique) such that

1. If $p(x) > p(y)$, then $\ell(x) \leq \ell(y)$.
2. The longest two codewords have the same length.
3. The longest two codewords differ only at the last bit and correspond to the two least probable symbols.

Points 2 & 3 suggest the first step of Huffman's algorithm. Any subtree must satisfy the same conditions $\Rightarrow$ Induction.

Note that since Shannon-Fano gives $E[\ell(X)] \leq H(X) + 1$, and Huffman is optimal, Huffman must satisfy the same bound.

Problems with Symbol Codes

Now we have found the optimal symbols code with expected codelength $E[\ell(X)] \leq H(X) + 1$. Are we done?

Problems with Symbol Codes

Now we have found the optimal symbols code with expected codelength $E[\ell(X)] \leq H(X) + 1$. Are we done?

No. (At least) three problems remain:

Problems with Symbol Codes

Now we have found the optimal symbols code with expected codelength $E[\ell(X)] \leq H(X) + 1$. Are we done?

No. (At least) three problems remain:

1. The one extra bit, $H(X) + 1$.

Problems with Symbol Codes

Now we have found the optimal symbols code with expected codelength $E[\ell(X)] \leq H(X) + 1$. Are we done?

No. (At least) three problems remain:

1. The one extra bit, $H(X) + 1$.
   - Can make all the difference if $H(X)$ is small.

Problems with Symbol Codes

Now we have found the optimal symbols code with expected codelength $E[\ell(X)] \leq H(X) + 1$. Are we done?

No. (At least) three problems remain:

1. The one extra bit, $H(X) + 1$.
   - Can make all the difference if $H(X)$ is small.
2. Shannon-Fano and Huffman codes require that the distribution generating the source symbols is known.

Problems with Symbol Codes

Now we have found the optimal symbols code with expected codelength $E[\ell(X)] \leq H(X) + 1$. Are we done?

No. (At least) three problems remain:

1. The one extra bit, $H(X) + 1$.
   - Can make all the difference if $H(X)$ is small.
2. Shannon-Fano and Huffman codes require that the distribution generating the source symbols is known.
   - We can of course first estimate the distribution from the data to be compressed, but how about the decoder?

Problems with Symbol Codes

Now we have found the optimal symbols code with expected codelength $E[\ell(X)] \leq H(X) + 1$. Are we done?

No. (At least) three problems remain:

1. The one extra bit, $H(X) + 1$.
   - Can make all the difference if $H(X)$ is small.
2. Shannon-Fano and Huffman codes require that the distribution generating the source symbols is known.
   - We can of course first estimate the distribution from the data to be compressed, but how about the decoder?
3. Distribution is not i.i.d.: Dependence and changes.

Two-Part Codes

Solution to problem 2:

2. The Shannon-Fano and Huffman codes require that the distribution generating the source symbols is known.
    - We can of course first estimate the distribution from the data to be compressed, but how about the decoder?

Two-Part Codes

Write the distribution (or code) in the beginning of the file.

Two-Part Codes

Solution to problem 2:

2. The Shannon-Fano and Huffman codes require that the distribution generating the source symbols is known.
   - We can of course first estimate the distribution from the data to be compressed, but how about the decoder?

Two-Part Codes

Write the distribution (or code) in the beginning of the file.

Usually the overhead is minor compared to the total file size.

Block Codes

Solution to problems 1 & 3:

1. The one extra bit, $H(X) + 1$.
   - Can make all the difference if $H(X)$ is small.

3. Distribution is not i.i.d.: Dependence and changes.

Block Codes

Combine successive symbols into blocks and treat blocks as symbols. $\Rightarrow$ One extra bit per block.

Block Codes

Solution to problems 1 & 3:

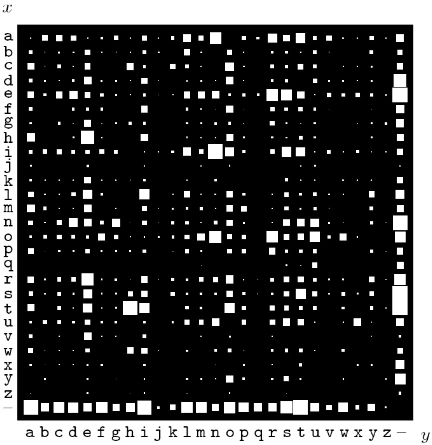
1. The one extra bit, $H(X) + 1$.
   - Can make all the difference if $H(X)$ is small.
3. Distribution is not i.i.d.: Dependence and changes.

Block Codes

Combine successive symbols into blocks and treat blocks as symbols. $\Rightarrow$ One extra bit per block.

Allows modeling of dependence.

Block Codes


x
a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s
t
u
v
w
x
y
z
–
a b c d e f g h i j k l m n o p q r s t u v w x y z –
y

Block Codes

Combining solutions to problems 1–3, we get **two-part block codes**: Write first the joint distribution of blocks of N symbols, and then encode using blocks of length N.

Block Codes

Combining solutions to problems 1–3, we get **two-part block codes**: Write first the joint distribution of blocks of N symbols, and then encode using blocks of length N.

The size of the first part (distribution/code) grows with N, but the performance of the block code get better.

Block Codes

Combining solutions to problems 1–3, we get **two-part block codes**: Write first the joint distribution of blocks of N symbols, and then encode using blocks of length N.

The size of the first part (distribution/code) grows with N, but the performance of the block code get better.

Complexity Tradeoff

Find suitable balance between complexity of the model (increases with N) and codelength of data given model (decreases with N).
$\Rightarrow$ **MDL/MML Principle**

Adaptive Codes

Alternative Solution to Problems 2 & 3:

Adaptive Codes

For each symbol (or a block of symbols), we can construct a code based on the probability $p(x_{\text{new}} \mid x_1, \ldots, x_n)$.

Adaptive Codes

Alternative Solution to Problems 2 & 3:

Adaptive Codes

For each symbol (or a block of symbols), we can construct a code based on the probability $p(x_{\text{new}} \mid x_1, \ldots, x_n)$.

This may lead to computational problems since the code tree has to be constantly updated.

Adaptive Codes

Alternative Solution to Problems 2 & 3:

Adaptive Codes

For each symbol (or a block of symbols), we can construct a code based on the probability $p(x_{\mathrm{new}} \mid x_1, \ldots, x_n)$.

This may lead to computational problems since the code tree has to be constantly updated.

Adaptive codes also avoid another problem with block codes: the first symbol can be read only after the whole block is decoded.

Adaptive Codes

Alternative Solution to Problems 2 & 3:

Adaptive Codes

For each symbol (or a block of symbols), we can construct a code based on the probability $p(x_{\text{new}} \mid x_1, \ldots, x_n)$.

This may lead to computational problems since the code tree has to be constantly updated.

Adaptive codes also avoid another problem with block codes: the first symbol can be read only after the whole block is decoded.

Arithmetic coding avoids "all problems": adaptive, spreads the one additional bit over the whole sequence, and can be decoded instantaneously.

Arithmetic coding

The basic idea is to represent source strings as intervals $[a, b) \subseteq [0, 1)$.

Arithmetic coding

The basic idea is to represent source strings as intervals $[a, b) \subseteq [0, 1)$.

The end-points, a, b, cannot be encoded with infinite precision. Fortunately, it is enough to specify *any* value $x \in [a, b)$ within the interval.

Arithmetic coding

The basic idea is to represent source strings as intervals $[a, b) \subseteq [0, 1)$.

The end-points, a, b, cannot be encoded with infinite precision. Fortunately, it is enough to specify *any* value $x \in [a, b)$ within the interval.

We represent the value x as a sequence of binary digits, e.g., $x = 0.0110100$.

Arithmetic coding

The basic idea is to represent source strings as intervals $[a, b) \subseteq [0, 1)$.

The end-points, a, b, cannot be encoded with infinite precision. Fortunately, it is enough to specify *any* value $x \in [a, b)$ within the interval.

We represent the value x as a sequence of binary digits, e.g., $x = 0.0110100$.

- For a *wide* interval we need a *short* binary string.

Arithmetic coding

The basic idea is to represent source strings as intervals $[a, b) \subseteq [0, 1)$.

The end-points, a, b, cannot be encoded with infinite precision. Fortunately, it is enough to specify *any* value $x \in [a, b)$ within the interval.

We represent the value x as a sequence of binary digits, e.g., $x = 0.0110100$.

- For a *wide* interval we need a *short* binary string.
- For a *narrow* interval we need a *long* binary string.

Arithmetic coding

We assign wide intervals for probable symbols, and narrow intervals for improbable symbols.

Arithmetic coding

We assign wide intervals for probable symbols, and narrow intervals for improbable symbols.

Arithmetic coding is an example of *"stream coding"*: each symbol in the source string is used to divide the interval $[a, b)$ into smaller sub-intervals.

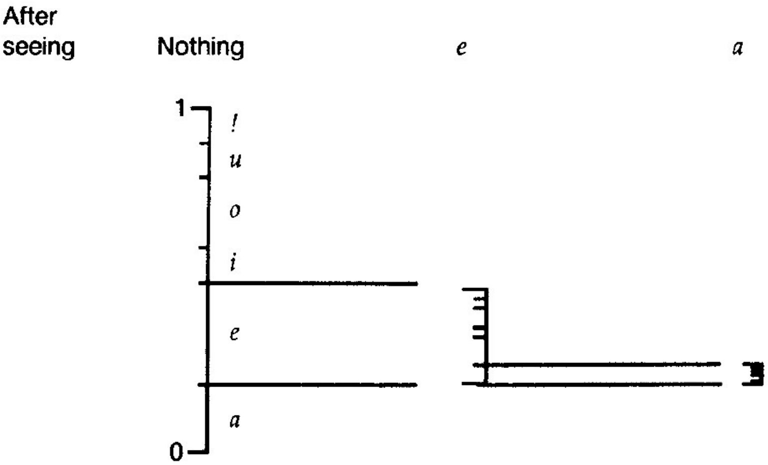
Arithmetic coding

We assign wide intervals for probable symbols, and narrow intervals for improbable symbols.

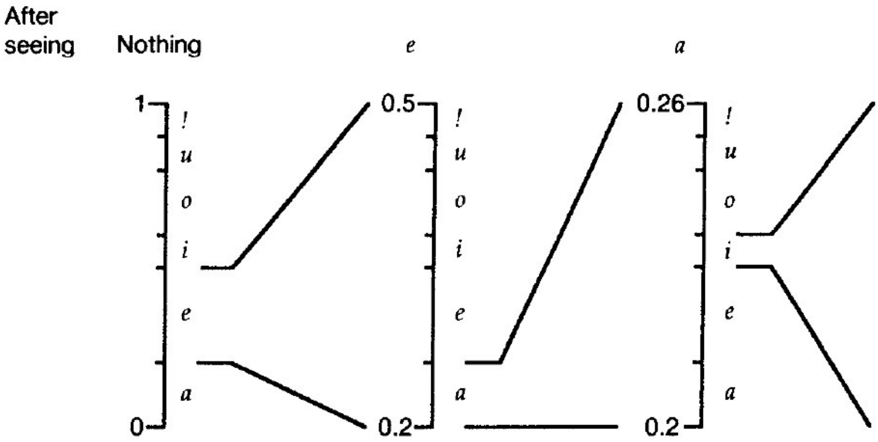
Arithmetic coding is an example of *"stream coding"*: each symbol in the source string is used to divide the interval $[a, b)$ into smaller sub-intervals.

We continue dividing recursively until the whole source string is encoded.

Arithmetic coding



Arithmetic coding



Arithmetic coding

Next Friday:

- Universal Coding.

Arithmetic coding

Next Friday:

- Universal Coding.
- Read the material on arithmetic coding (Witten, Neal & Cleary)