Statistics for Computing MA4413

Lecture 18

Information Theory and File Compression

Kevin Burke

kevin.burke@ul.ie

Base-2

As *information* is stored using **bits**, i.e., binary numbers $\{0, 1\}$, working in **base-2** is central to **information theory**.

Example: to store the two symbols “a” and “b” $\Rightarrow$ use codes 0 and 1.

Note: **code length** here is 1 bit and we are storing $2 = 2^1$ symbols.

Symbols	Codes for Storage	Code Length
$2 = 2^1$	$\{0, 1\}$	1
$4 = 2^2$	$\{00, 10, 01, 11\}$	2
$8 = 2^3$	$\{000, 100, 010, 110, \ldots, 111\}$	3
$16 = 2^4$	$\{0000, 1000, 0100, 1100, \ldots, 1111\}$	4

Notation

- ***N*** is the **number of symbols** that we need to store.
- ℓ is the **code length** used, i.e., the number of bits.

Clearly ℓ bits *can be* used to store $2^\ell = N$ symbols (see previous slide).

This **fixed length** coding scheme maps each symbol to an ℓ-bit code.

We are giving each symbol equal weight. In a sense we are assuming that each symbol is *equally likely* to occur: $p(x_i) = \frac{1}{N}$.

(if symbol frequencies vary, we can do better - in terms of file size - by *varying* the individual code lengths)

log **base-2**

If we know N and wish to calculate ℓ, how do we go about this?

"What power ℓ makes 2^{ℓ} equal to N ?"

i.e., we wish to solve $2^{\ell} = N$ for ℓ.

The solution is $\ell = \log_2 N$ but, using the rules of logs, we can rewrite this as:

$$\ell = -\log_2 \tfrac{1}{N}.$$

Although this might seem slightly redundant here, it emphasises the assumption that each symbol is equally likely.

Fixed Code Length

$N = 4$ equally likely symbols $\Rightarrow \ell = -\log_2 \frac{1}{4} = \log_2 4 = 2$ bits.

$\Rightarrow$ the binary codes are $\{00, 10, 01, 11\}$.

$N = 6$ equally likely symbols $\Rightarrow \ell = -\log_2 \frac{1}{6} = \log_2 6 = 2.585$ bits. We cannot have 2.585 bits so we must use 3 bits.

$\Rightarrow$ use any 6 codes from $\{000, 100, 010, 110, 001, 101, 011, 111\}$.

Using Your Calculator

It is essential that you can calculate $\log_2 N$ on your calculator.

Modern calculators will all have the button $\boxed{\log_{[\ ]}[\ ]}$.

$$\Rightarrow \boxed{\log_2 N = \log_{[2]}[N]}$$

If your calculator does not have this button, you will use either

$$\log_2 N = \boxed{\log N} \ \boxed{\div} \ \boxed{\log 2}$$

or

$$\log_2 N = \boxed{\ln N} \ \boxed{\div} \ \boxed{\ln 2}$$

which comes from the fact that $\log_2 N = \frac{\log_{10} N}{\log_{10} 2} = \frac{\ln N}{\ln 2}$.

Information Content

The **information content**, $h(x_i)$, is a measure (in bits) of the amount of information contained in the occurrence of the event x_i.

It is defined as:

$$\boxed{h(x_i) = -\log_2 p(x_i)}$$

where $p(x_i)$ is the probability of x_i occurring.

Note: If there are N equally likely events then $h(x_i) = -\log_2 \frac{1}{N}$.

Information Content

It is easy to confirm that

- Common occurrences $\Rightarrow$ $h(x_i) \approx 0$ (since $p(x_i) \approx 1$)
- Rare occurrences $\Rightarrow$ $h(x_i) \approx \infty$ (since $p(x_i) \approx 0$)

In words, commonly occurring events provide little new information whereas rare events provide high information.

Example: Student Attendance

Assume that the attendance probabilities and, hence, information content for 2 students are:

$$\Pr(A) = 1.00 \quad \Rightarrow \quad h(A) = -\log_2 1.00 = 0 \text{ bits}$$

$$\Pr(B) = 0.02 \quad \Rightarrow \quad h(B) = -\log_2 0.02 = 5.644 \text{ bits}$$

We know nothing about the lecture if told that student A is present since he/she always attends $\Rightarrow$ information content is zero.

Student B attending is a rare event which contains higher information about the nature of the lecture, i.e., it is likely to be a midterm exam.

Example: English Words

A word is picked out of the dictionary and you are told the first few letters are:

- "Th" $\Rightarrow$ not much information is contained in this since many words begin with "th".
- "Zeb" $\Rightarrow$ this contains high information about the word: it is either "zebra" or "zebu" (a South-Asian ox).
- "Xyl" $\Rightarrow$ the only word in the dictionary that this could be is "xylophone".

$\Rightarrow$ h("th") is small whereas h("zeb") and h("xyl") are large.

Additivity of Information

We may also define the **joint information content** for two events:

$$h(x_i, y_i) = -\log_2 p(x_i, y_i).$$

From the multiplication rule of probability and the properties of logs, it is straightforward to show that

$$\boxed{h(x_i, y_i) = h(x_i) + h(y_i \mid x_i)}.$$

$\Rightarrow$ Joint information content is the information contained in x_i plus the *extra* information contained in y_i given that we already know x_i.

Example: Finding the Prize

Assume that there are 4 boxes; we must guess which contains a prize.

Let G_1, G_2, G_3 be the events of finding the prize on guess 1, 2 or 3.

- $h(G_1) = -\log_2 \frac{1}{4} = 2$ bits of information $\Rightarrow$ we know where it is.
- $h(G_1^c) = -\log_2 \frac{3}{4} = 0.415$ bits $\Rightarrow$ we only know that it is *not* in the chosen box.
- $h(G_1^c, G_2) = h(G_1^c) - \log_2 \frac{1}{3} = 0.415 + 1.585 = 2$ bits $\Rightarrow$ we know where it is.
- $h(G_1^c, G_2^c) = h(G_1^c) - \log_2 \frac{2}{3} = 0.415 + 0.585 = 1$ bit $\Rightarrow$ we have ruled out two of the boxes.
- $h(G_1^c, G_2^c, G_3^c) = h(G_1^c, G_2^c) - \log_2 \frac{1}{2} = 1 + 1 = 2$ bits $\Rightarrow$ ruled out three boxes $\Rightarrow$ know where it is now.

Expected Value

We briefly review **expected value** which we met previously (Lecture6).

For some probability distribution, its expected value is

$$\boxed{E(X) = \sum x_i \, p(x_i)}.$$

In words: multiply each possible value of X by its probability value and then sum the results.

This is the value we would *expect* to get on average.

Example

Consider the probability distribution:

x_i	0	3	5	7
$p(x_i)$	0.1	0.3	0.4	0.2

$$\Rightarrow E(X) = 0(0.1) + 3(0.3) + 5(0.4) + 7(0.2)$$

$$= 0 + 0.9 + 2.0 + 1.4$$

$$= 4.3.$$

On average the value of X we observe will be 4.3.

Entropy

We have seen that information content, $h(x_i)$ is a measure of the information contained in a particular outcome, x_i.

Entropy is the *expected information content* for a distribution of values:

$$\boxed{H(X) = E[h(X)] = \sum h(x_i)\, p(x_i)},$$

i.e., the average amount of information from a particular source.

$H(X)$ is an overall measure of *uncertainty* or *disorder* associated to the value of X. (recall that standard deviation is another measure of uncertainty)

Entropy

It is relatively easy to show that $\boxed{0 \leq H(X) \leq \log N}$.

- Minimum entropy: $0 \Rightarrow$ there is only one outcome which is certain to occur $\Rightarrow$ no uncertainty.
- Maximum entropy: $\log N \Rightarrow$ each value is equally likely, i.e., $p(x_i) = \frac{1}{N} \Rightarrow$ maximum uncertainty.

Example

x_i	0	3	5	7
$p(x_i)$	0.1	0.3	0.4	0.2
$h(x_i) = -\log_2 p(x_i)$	3.322	1.737	1.322	2.322

$$\begin{aligned} H(X) = E[h(X)] &= 3.322(0.1) + 1.737(0.3) + 1.322(0.4) + 2.322(0.2) \\ &= 0.3322 + 0.5211 + 0.5288 + 0.4644 \\ &= 1.85 \text{ bits.} \end{aligned}$$

On average, 1.85 bits of information are gained.

(Note: Unlike the calculation of expected value, $E(X)$, the individual x_i values are not required in calculating $H(X)$; we only need the probabilities)

Question 1

Consider two text files which contain only four symbols with the following probabilities:

x_i		a	b	c	d
File 1:	$p(x_i)$	0.5	0.25	0.2	0.05
File 2:	$p(x_i)$	0.1	0.8	0.05	0.05

a) What is the maximum possible entropy here? How would this come about?

b) Calculate the entropy for file 1.

c) Calculate the entropy for file 2.

d) Comment on these values.

Symbol Codes

A **symbol code** maps a set of symbols to a set of codes.

Example:

x_i	$p(x_i)$	$c(x_i)$	$\ell(x_i)$
a	0.1	00	2
b	0.8	10	2
c	0.05	01	2
d	0.05	11	2

- $c(x_i)$ denotes the code for symbol x_i.
- $\ell(x_i)$ denotes the length of code $c(x_i)$.

Expected Code Length

The **expected code length** is given by

$$\boxed{E(L) = E[\ell(X)] = \sum \ell(x_i)\, p(x_i)}$$

Thus, for the example on the previous slide, we have:

$$\begin{aligned} E(L) &= 2(0.1) + 2(0.8) + 2(0.05) + 2(0.05) \\ &= 2(0.1 + 0.8 + 0.05 + 0.05) \\ &= 2(1) = 2 \text{ bits.} \end{aligned}$$

Of course we would have known this without any probability calculation in this case.

File Compression

In the previous example, we see that the symbol “b” appears 80% of the time in the file.

However, by assigning each symbol a code of length 2, we have not accounted for the frequency of occurrence in any way.

It is possible to take these probabilities into account so that we can **compress** the file, i.e., reduce the average code length, $E(L)$.

There are two compression types:

1. **Lossy:** reduces $E(L)$ but loses information.
2. **Lossless:** reduces $E(L)$ without loss of information.

Example: Lossy Compression

		Code 1		Code 2		Code 3	
x_i	$p(x_i)$	$c(x_i)$	$\ell(x_i)$	$c(x_i)$	$\ell(x_i)$	$c(x_i)$	$\ell(x_i)$
a	0.1	00	2	0	1	0	1
b	0.8	10	2	1	1	1	1
c	0.05	01	2	–	0	1	1
d	0.05	11	2	–	0	1	1

Code 1 has $E(L) = 2$ bits. We can reduce this via *lossy* compression:

- Code 2 (no code for “c” or “d”)

  $\Rightarrow E(L) = 1(0.1) + 1(0.8) + 0(0.05) + 0(0.05) = 0.9$ bits.

- Code 3 (“c” and “d” assigned the same code as “b”)

  $\Rightarrow E(L) = 1(0.1) + 1(0.8) + 1(0.05) + 1(0.05) = 1$ bit.

Example: Lossy Compression

Consider encoding the message: “abbbbcbabd”.

Code	Encoding	Length	Decoded As
Code 1	00|10|10|10|10|01|10|00|10|11	20	abbbbcbabd
Code 2	0 | 1 | 1 | 1 | 1 | | 1 | 0 | 1 |	8	abbbb bab
Code 3	0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1	10	abbbbbbabb

- Code 1: No compression and the message is fully decoded.
- Code 2: Compressed by eliminating “c” and “d”.
- Code 3: Compressed by replacing “c” and “d” with “b”.

Note on Lossy Compression

We saw two forms of lossy compression in the previous example:

- Eliminating data:
  - Data missing after decoding.
  - Useful for music files where, in this case, “c” and “d” are not letters but frequencies inaudible to the human ear.
- Replacing data:
  - Data replaced with other data after decoding.
  - Useful for image files where, in this case, “c” and “d” may may be colours virtually indistinguishable from “b”.

Lossless Compression

We have seen that trying to maintain *fixed* code lengths while achieving compression leads to *lossy* compression.

We now consider **lossless compression**, i.e., no loss of information.

Lossless compression can be achieved using **variable-length codes**, for example, $\{a, b, c\} \mapsto \{0, 10, 11\}$.

The idea is to assign shorter codes to frequently occurring symbols and longer codes to less frequently occurring symbols.

Important Properties

Three important properties of variable length codes are:

1. **Uniquely decodable:** each encoded message has only *one* possible interpretation, i.e., it cannot be decoded in multiple ways.

2. **Easy to decode.**

3. **High compression:** as much as possible but **cannot** compress below entropy without information loss.

Note: **Prefix codes** are both *uniquely decodable and easy to decode*.

Unique Decodability

A symbol code is **uniquely decodable** if an encoding can be interpreted in only *one* possible way.

- Uniquely decodable
  - $\{a, b, c\} \mapsto \{0, 10, 11\}$.
  - $\{a, b, c, d\} \mapsto \{00, 10, 01, 11\}$.
- *Not* uniquely decodable
  - $\{a, b, c\} \mapsto \{0, 1, 11\}$: “bb” $=$ 11 $=$ “c”.
  - $\{a, b, c, d\} \mapsto \{0, 10, 01, 11\}$: “ada” $=$ 0110 $=$ “cb”.

Prefix Code

A **prefix code** is a code where no codeword is a prefix (i.e., the initial segment) of another codeword.

Prefix codes are **uniquely decodable**.

- Prefix code
  - $\{a, b, c, d\} \mapsto \{00, 10, 01, 11\}$.
  - $\{a, b, c, d\} \mapsto \{0, 10, 110, 111\}$.
- *Not* a prefix code
  - $\{a, b, c, d\} \mapsto \{0, 10, 01, 11\}$: since 0 is a prefix of 01.
  - $\{a, b, c, d\} \mapsto \{0, 01, 011, 111\}$ since 0 is a prefix of both 01 and 011. Also, 01 is a prefix of 011.

Prefix Code

Prefix codes are **easy to decode**; each codeword is identified as soon as it arrives $\Rightarrow$ also known as **instantaneous codes**.

Example: $\{a, b, c, d\} \mapsto \{0, 10, 110, 111\}$: consider receiving the message "adb"$=$ 0-111-10 $=$ 011110 one bit at a time:

1. 0 $\Rightarrow$ a
2. 0-1 $\Rightarrow$ a b, a c, a d . . . ?
3. 0-11 $\Rightarrow$ a c, a d . . . ?
4. 0-111 $\Rightarrow$ ad
5. 0-111-1 $\Rightarrow$ ad b, ad c, ad d . . . ?
6. 0-111-10 $\Rightarrow$ adb

Non-Prefix Code

Some *non*-prefix codes may be uniquely decodable - *just not as easily*.

Example: $\{a, b, c, d\} \mapsto \{0, 01, 011, 111\}$: consider receiving the message "adb"$=$ 0-111-01 $=$ 011101 one bit at a time:

1. 0 $\Rightarrow$ a, b, c . . . ?
2. 01 $\Rightarrow$ ad, b, c . . . ?
3. 011 $\Rightarrow$ ad, bd, c . . . ?
4. 0111 $\Rightarrow$ ad, bd, cd . . . ?
5. 0-111-0 $\Rightarrow$ ad a, ad b, ad c . . . ?

6.A 0-111-01 $\Rightarrow$ adb *(if we know this is the end of message)*

6.B 0-111-01 $\Rightarrow$ ad ad, ad b, ad c . . . ? *(if we do not know)*

Kraft Inequality

We may wish to know if a *non*-prefix code is uniquely decodable.

In order for a symbol code to be uniquely decodable, the individual code-lengths *must* satisfy the **Kraft inequality:**

$$\boxed{\sum 2^{-\ell(x_i)} \leq 1}$$

where $\ell(x_i)$ is the length of a codeword.

Kraft Inequality

The Kraft inequality can only tell us if a code is *not* uniquely decodable.

Once we calculate the Kraft value, $K = \sum 2^{-\ell(x_i)}$, we have:

- $K > 1 \Rightarrow$ code is *not* uniquely decodable.
- $K \leq 1 \Rightarrow$ code may or may not be uniquely decodable.

The *Sardinas-Patterson algorithm* can be used to determine if a code is uniquely decodable - not covered on this course.

Kraft Inequality: Example 1

Consider the symbol code $\{a, b, c, d\} \mapsto \{0, 10, 01, 11\}$ whose code-lengths are $\{1, 2, 2, 2\}$. The Kraft value is:

$$K = \frac{1}{2^1} + \frac{1}{2^2} + \frac{1}{2^2} + \frac{1}{2^2}$$

$$= \frac{1}{2} + \frac{3}{4} = 1.25 \quad > 1.$$

$\Rightarrow$ Symbol code with lengths $\{1, 2, 2, 2\}$ *cannot* be uniquely decodable.

$\Rightarrow$ The code $\{0, 10, 01, 11\}$ is not uniquely decodable; this is easy to verify since "ada" $= 0110 =$ "cb".

Kraft Inequality: Example 2

Now consider the codes $\{0, 10, 110, 111\}$ and $\{0, 00, 110, 111\}$ which have the same code lengths: $\{1, 2, 3, 3\}$. The Kraft value is:

$$K = \frac{1}{2^1} + \frac{1}{2^2} + \frac{1}{2^3} + \frac{1}{2^3}$$

$$= \frac{1}{2} + \frac{1}{4} + \frac{2}{8} = 1 \quad \leq 1.$$

$\Rightarrow$ A uniquely decodable symbol code with lengths $\{1, 2, 3, 3\}$ exists.
(note: this doesn't mean that either of the codes above are uniquely decodable)

- $\{0, 10, 110, 111\}$ is uniquely decodable since it is a prefix code.
- $\{0, 00, 110, 111\}$ is *not* uniquely decodable since "aa" $= 00 =$ "b".

Question 2

x_i	$p(x_i)$	C_1	C_2	C_3	C_4	C_5	C_6
a	0.5	00	0	0	0	0	10
b	0.3	10	1	10	10	10	110
c	0.1	01	01	11	110	010	0
d	0.1	11	11	001	111	111	111

a) Identify which codes are prefix codes - these are uniquely decodable.

b) For the non-prefix codes, calculate their Kraft value, $K = \sum 2^{-\ell(x_i)}$. Which codes are definitely not uniquely decodable?

c) For any remaining codes, see if you can figure out whether or not they are uniquely decodable.

Shannon's Source Coding Theorem

Shannon's source coding theorem: *no source can be compressed below its entropy without information loss.*

$\Rightarrow$ For lossless compression, the average code length will *always* be greater than the entropy:

$$\boxed{E(L) \geq H(X)}.$$

$\Rightarrow$ We can define the **efficiency** of a symbol code:

$$\boxed{e = \frac{H(X)}{E(L)}}.$$

Maximum efficiency is 100% which occurs when $E(L) = H(X)$.

Example

We established in Q2 that C_1, C_4 and C_6 are uniquely decodable.

To calculate efficiencies, we need the entropy, $H(X) = E[h(X)]$:

$$H(X) = 1.000(0.5) + 1.737(0.3) + 3.322(0.1) + 3.322(0.1) = 1.6855.$$

x_i	$p(x_i)$	$h(x_i)$	C_1	C_2	C_3	C_4	C_5	C_6
a	0.5	1.000	00	0	0	0	0	10
b	0.3	1.737	10	1	10	10	10	110
c	0.1	3.322	01	01	11	110	010	0
d	0.1	3.322	11	11	001	111	111	111
$E(L) = E[\ell(X)]$			2	1.2	1.6	1.7	1.7	2.3
$\Rightarrow e = \frac{H(X)}{E(L)}$			0.84	1.4	1.05	0.99	0.99	0.73

Example

C_4 is the most efficient; it is almost at the entropy limit with $e_4 = 99\%$.

(note: C_4 is a *Huffman code* - see next section)

C_2 and C_3 compress below entropy $\Rightarrow e_2 = 140\%$, $e_3 = 105\%$.

It is not possible to compress below entropy without information loss; we already know these codes are not uniquely decodable.

Huffman Coding

Clearly a prefix codes with optimal efficiency is desirable.

The **Huffman coding algorithm** achieves this by building a binary tree from the leaves up:

1. Sort the symbol probabilities from highest to lowest.

2. Repeat the following steps until the tree root is reached:
   (i) Select the two lowest probabilities and add them.
   (ii) Combine these two nodes by extending branches to a new node.

3. Left branches are labelled “0”; right branches are labelled “1”.

4. Create codewords by traversing the tree from its root to its leaves.

Example

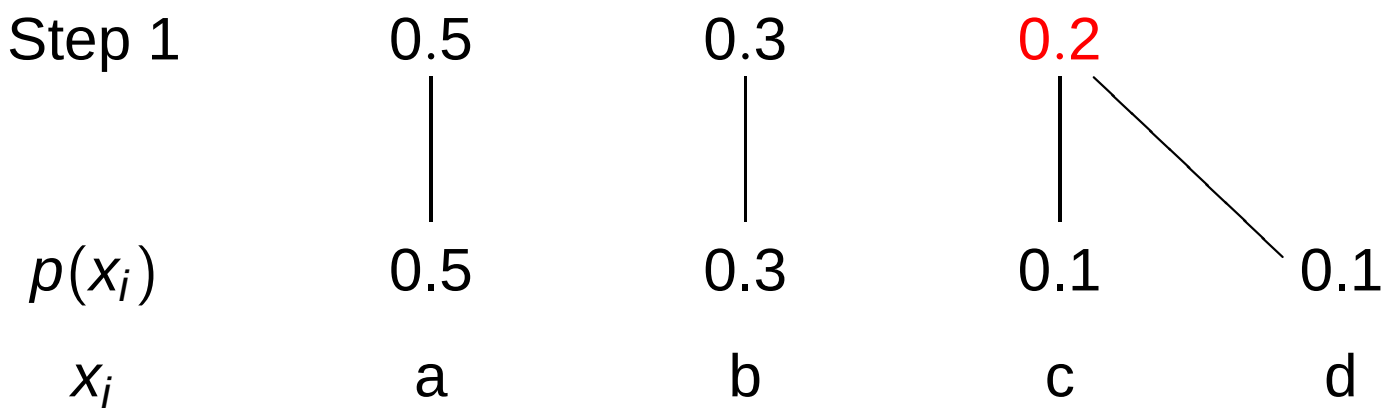
The Huffman coding algorithm is most easily understood by example.

Thus, consider again the symbols from the previous example:

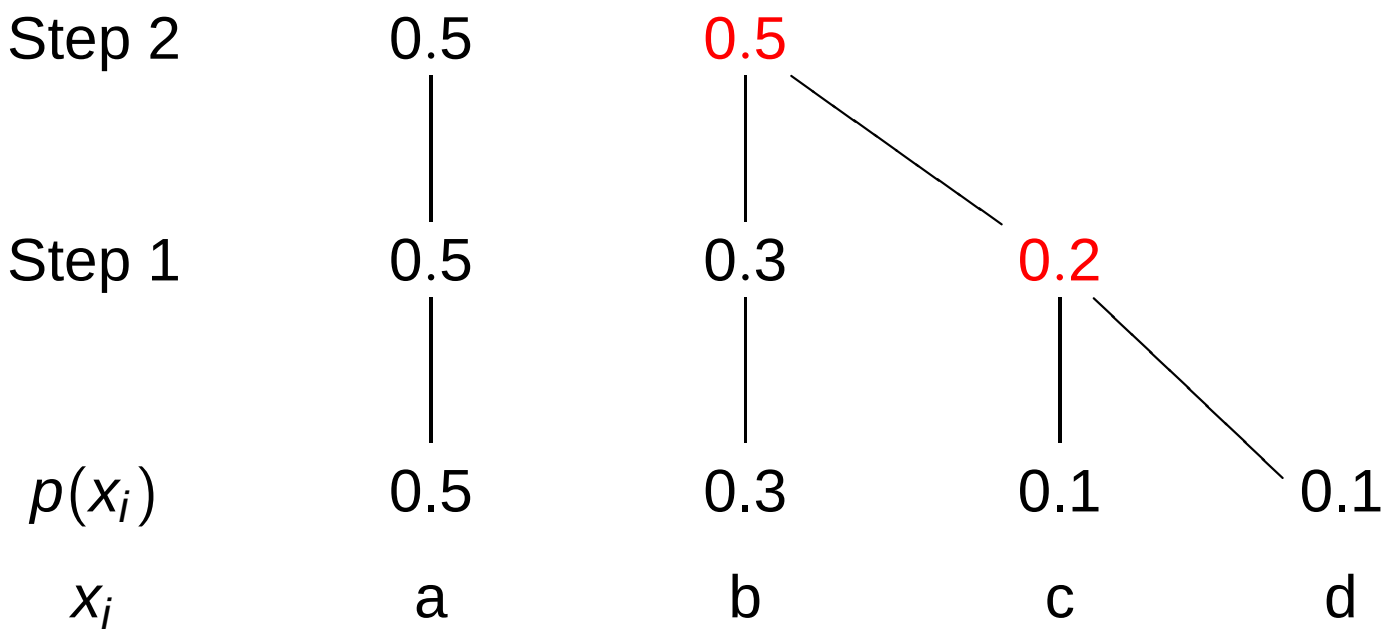
x_i	$p(x_i)$
a	0.5
b	0.3
c	0.1
d	0.1

Note that these are already in order from highest to lowest probability.

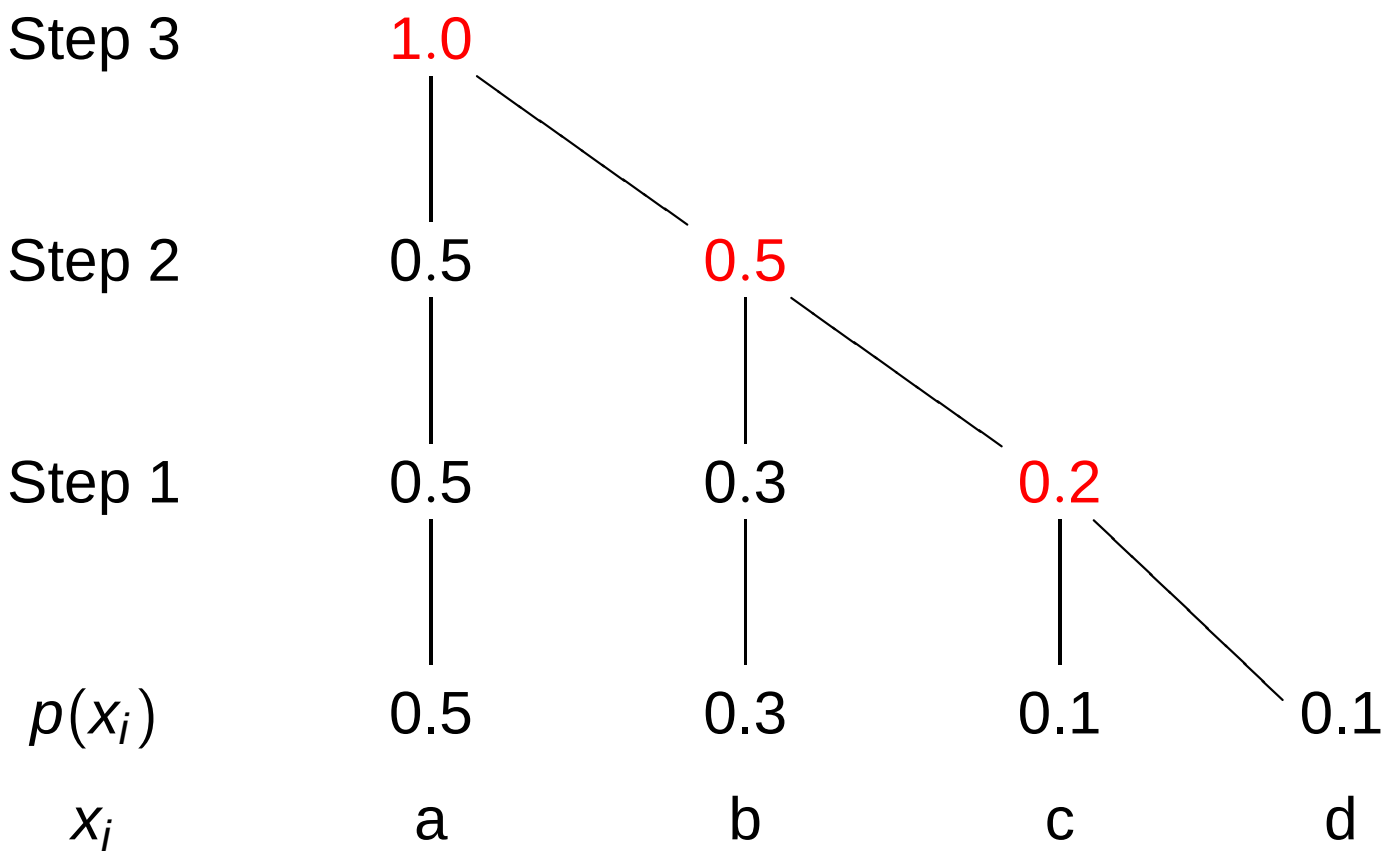
Example



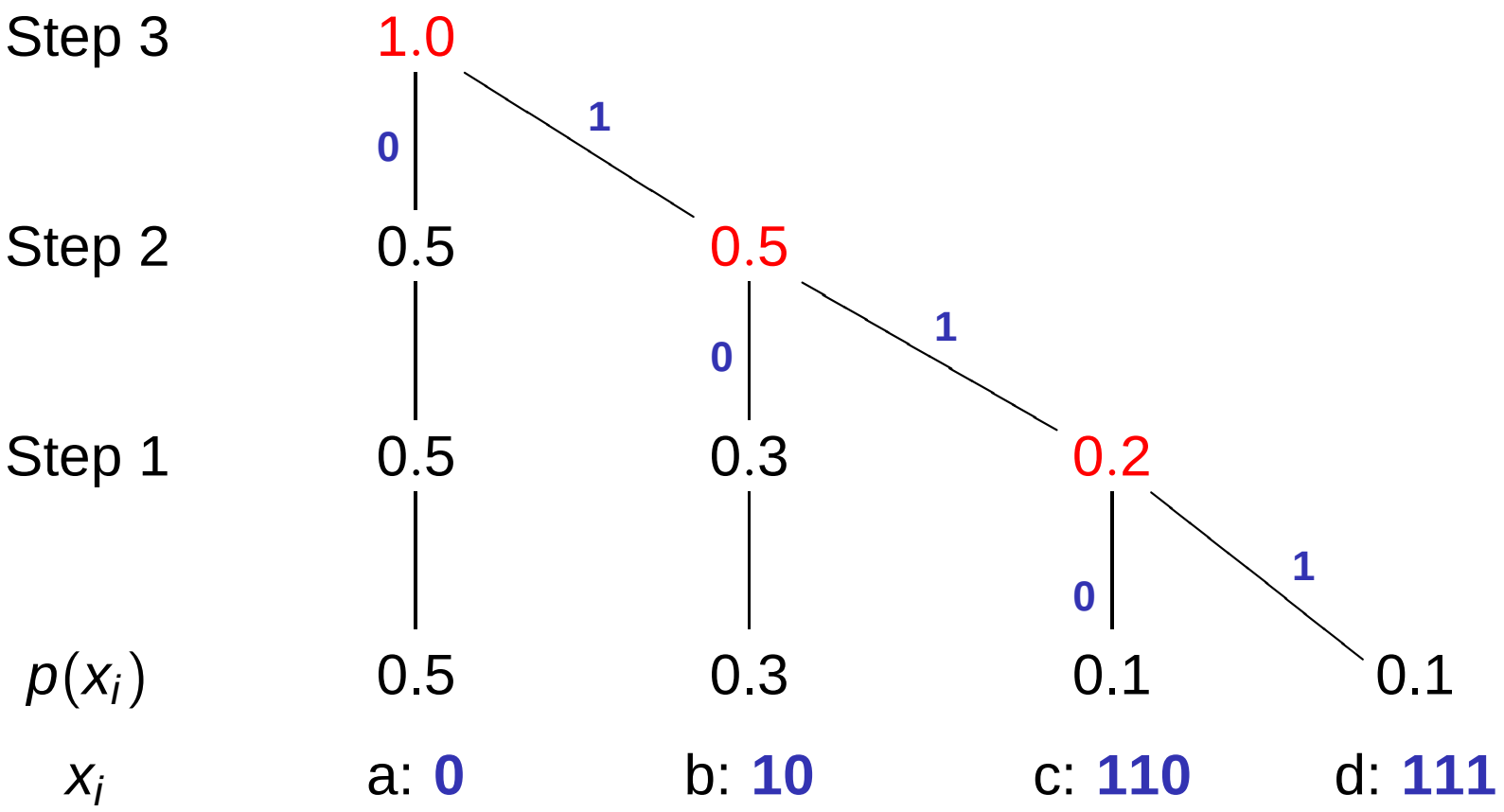
Example



Example



Example


Step 3
Step 2
Step 1
$p(x_i)$
x_i
1.0
0.5
0.5
0.5
0.3
0.2
0.5
0.3
0.1
0.1
0
1
0
1
0
1
a: 0
b: 10
c: 110
d: 111

Question 3

A file contains the following characters:

x_i	a	b	c	d	e
$p(x_i)$	0.2	0.1	0.15	0.2	0.35

a) Calculate the entropy for this file.

b) Construct a Huffman code.

c) Calculate the expected length of the code.

d) Calculate the efficiency of the code.