A few bits of Information Theory

What do we mean by "Information"?
How do we measure it?

"*They were brusque to the world of manners, they had built their hope of heaven on the binary system and the computer, 1 and 0, Yes and No –* "

Norman Mailer
The Armies of the Night

A few points to be aware of

The material in this part is

A. **NOT** covered in the module textbook (see p. 470 for discussion).

B. Included for sake of interest:
Information Theory is fundamental to CS.

Some motivating background

At the centre of most computational activity there will (eventually) arise a need to **send** "data" (text) to another party.

Consider sending an e-mail:

A. We *hit* a **keypad** on a **keyboard** (eg '*S*')
B. This action results in *S* being added and **stored** in the text.
C. The message itself will be sent through some "**channel**": eg a mobile-network frequency, optical cable, copper wire.
D. On arrival the message is **processed** (the **content** is found).
E. The text is then "**displayed**" in some form.

What problems arise?

The characters (keypads) must be *encoded* (binary).
When a message is sent through a channel:
How much *time* will it take? ("transmission rate")
What if there is *interference*? ("noisy" channels)

Information Theory is the field of Computer Science that studies questions such as these.

What problems arise?

It offers robust solutions that address:

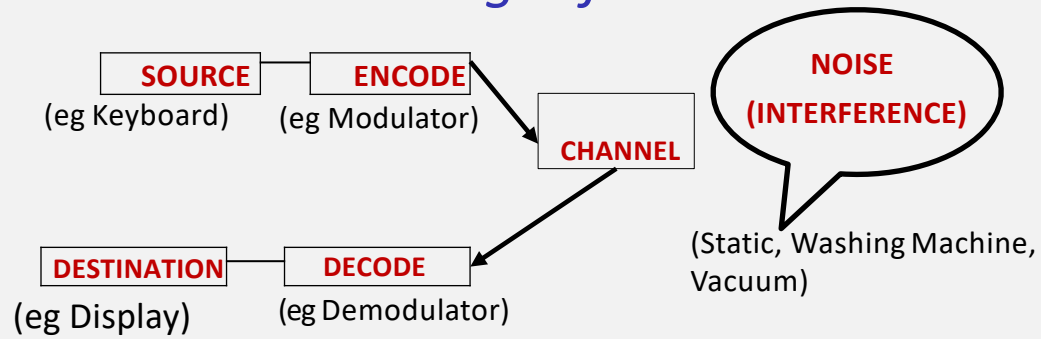
Defining *content* and *uncertainty* in information.

The *limits* on *speed* and data *compression*.

If data can be *reliably* sent when there is *noise*.

What can be done to get *correct data reception*.

The Shannon Paradigm for Communication



The model depicted is from
C. E. Shannon. A Mathematical Theory of Communication. *Bell Systems Technical Journal*, **27:379–423**,**623–656**, July, October 1948

This is the origin of Information Theory as a scientific discipline: it formulates the *key issues*, supporting *framework* and derives *results*.

Information & Uncertainty

"The fundamental problem of communication is that of reproducing at one point either exactly or approximately a message selected at another point."

C.E. Shannon, *op. cit.*, p. 379

What is information?

When we communicate (in speech, texting, or (proper) writing) the basic units are **words**. A word is just a *sequence* of *symbols* from an *alphabet*.

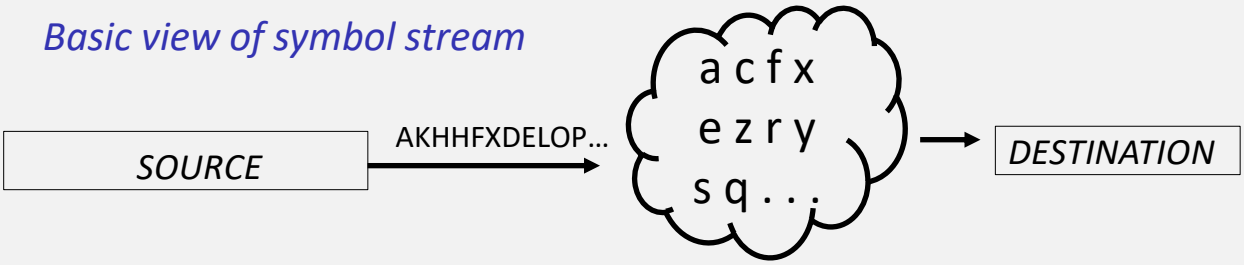
C1. "**communication**" involves "*sending*" "*information*"

C2. "**information**" is provided as a *sequence of symbols*.

In order effectively to communicate, the receiver must be able to recognise individual symbols in the stream of symbols that have been sent.

Information and the element of "surprise"

Basic view of symbol stream



The receiver will see a stream of symbols:
some will be "more common" than others;
some symbols will be quite rare.
For example, in written English the letters 'E' and 'T' occur often;
the letters 'Z' and 'J', however, are rarer.
A receiver will be "**less surprised**" to see some symbols and "**more surprised**" to see others.

Surprise is information gained

In English the letter ’Q’ is “usually” followed by the letter ’U’.

When a receiver sees the letter ’Q’ in an input stream of text, “almost surely” the next symbol seen will be ’U’.

Very few words in English begin with the letter ’X’.

If a symbol for the space character is seen (’ ’) a receiver would be “surprised” if the following symbol is an ’X’.

Information gained is uncertainty reduced

We have an **alphabet**, $S = \{s_1, s_2, \ldots, s_n\}$ of n **symbols**.

A **message** is a sequence of symbols.

Two questions:

Q1. How to measure the "*uncertainty*" ("degree of surprise" or "content")

Q2. How to "*encode*" messages (in binary) with desirable properties?

*Messages as "sequences of **events**" I*

In a stream of symbols (as noted earlier) we might expect to see some symbols "very frequently" and others "only rarely".
We can think of attaching a **probability**, P_i, to each symbol s_i.
When the receiver sees s_i "**information**" has been "**gained**".
This is because from a state of being "*unsure*" what the next symbol would be, the observer's "*uncertainty*" has **decreased**.

$$u_i = -\log_2 P_i$$

informally "*the decrease in uncertainty when s_i is seen*"
"*the information gained from seeing s_i*"

*Messages as "sequences of **events**" II*

If we have an alphabet in which one symbol (s_1 say) is the *only* symbol that ever appears then $P_1 = 1$, so that $u_1 = 0$.

There is **no uncertainty** (only s_1 will appear)

and

no increase in information (if anything is seen it can only be s_1).

Shannon's Uncertainty Formula & Limits on Source Coding

What is meant by "Source Coding"?

The symbols we use cannot be "*sent directly*".

These need to be **encoded** as *binary sequences*.

Various standards (ASCII, UNICODE etc) are used for this.

The **source coding problem** has three principal questions.

Shannon's Uncertainty Formula & Limits on Source Coding

1. Is it **possible** to *compress* the number of bits?
2. On *average*, how many bits are **needed**?
3. What are good algorithms that maximise compression.

   A *high level* of compression increases "**throughput**" (more data sent in shorter time)

   but

   "*compression*" must allow the receiver to **decode** the text "*easily*".

The Uncertainty Formula & Source Entropy

We introduced P_i as

"*the probability of seeing s_i in a stream*".

For the "simplified" development being given we assume that these are "**independent**": should s_i appear at time T the probability of seeing s_j at time $T + 1$ is **still** P_j.

Shannon derives (**NOT** "defines") the

"**Uncertainty in a data source from alphabet** *S*"

The Uncertainty Formula & Source Entropy

$$H(S) = -\sum_{i=1}^{n} p_i \log_2 p_i$$

H(*S*) is sometimes referred to as the **entropy** of *S*.

Entropy depends on the **distribution** p_i not the **alphabet** *S*.

The Source Coding Theorem

L(*S*) ("*average number of bits to encode a symbol*"): *L*(*S*) ≥ *H*(*S*).

A small example

$S = \{A, B, C, D\}$

If $P_i = 0.25$ for each symbol (each is equally likely) then

$$H(S) = -4(0.25)\log_2(0.25) = -\log_2(0.25) = 2$$

We need *at least two* bits for *some* symbol. Could use

$$A = 00 \; ; \; B = 01 \; ; \; C = 10 \; ; \; D = 11$$

A stream of 104,857,600 characters (100Mega) needs 200*Mbits*

Example (continued)

What if $P_A = 0.5$, $P_B = 0.25$, $P_C = P_D = 0.125$?

We *could* use the **same** coding scheme.

We **know**, however, that the entropy is **now**:

$$(-0.5 \log_2 0.5) + (-0.25 \log_2 0.25) + 2(-0.125 \log_2 0.125)$$

A B C and D

which is 1.75 (so *less than 2*)

Exploiting the reduction in entropy

What if we now use the code

$$A = 1 \; ; \; B = 01 \; ; \; C = 000 \; ; \; D = 001$$

Given the symbol probabilities, we "*expect*" there to be:

104,857,600/2 = 52,428,800 As (50Megabits);

104,857,600/4 = 26,214,400 Bs ($2 \times 26{,}214{,}400 \sim$ 50Megabits)

26,214,400 Cs and Ds ($3 \times 26{,}214{,}400 \sim$ 75Megabits)

In total we expect to transmit 175Megabits.

Exploiting the reduction in entropy

In order to be effective, the estimate of the *relative frequency* of different symbols must be accurate.

For a “large” data file this can be determined exactly allowing data **compression** with no loss of information.
It is assumed, of course, the receiver will be informed (or know) of the code scheme used.

A Bit more on Source Coding Schemes

The Source Coding Theorem gives a lower bound on the number of bits that might be needed for some symbol (on average).

We need to use *at least* as many bits (on average) as the **Source Entropy**.

This lower bound is **optimal** and (to within a small additive term) actually **achievable**.

To do so involves careful study of **coding schemes**.

A Bit more on Source Coding Schemes

Coding schemes

A binary code, C , is a set of **codewords**

$$C = \{c_1, c_2, \ldots, c_t\}$$

We think of individual **alphabet** symbols s being assigned (*coded*) using a particular $C(s) = c_i$ in C

Example Coding Schemes

Original ASCII

Lower and upper case alphabetic characters, digits, punctuation symbols, so-called \CONTROL (CTRL)" characters (128 in total) are mapped to 7-bit binary codes.

A **1000001** | 0 **0110000**
a **1100001** | 9 **0111001**
! **0100001** | % **0100101**
ESC **0011011** | ‘ ’ (SPACE) **0100000**