Information-Theoretic Modeling

Teemu Roos

Department of Computer Science, University of Helsinki

Fall 2014

582650 Information-Theoretic Modeling

- An advanced studies course ("syventävät opinnot")
- *Algorithms and Machine Learning* sub-programme, optional.
- 5 credit units.
- Lectures: Sep 4–Oct 17, Wed & Fri 10–12 in C222.
- Exercises: Sep 11–Oct 16, Thu 16–18 in D122.
- Instructor: **Teemu Roos**, A322,
  teemu.roos at cs.helsinki.fi
  (no fixed office hours, make an appointment by e-mail).
- Teaching assistant: **Jussi Määttä**.
  jussi.maatta at helsinki.fi

Resources

There is no required textbook on the course, but the following are recommended.

- Highly recommended: Cover & Thomas, *Elements of Information Theory*,
- MacKay, *Information Theory, Inference and Learning Algorithms*,
- Grünwald, *The Minimum Description Length Principle*,
- Solomon, *Data Compression: The Complete Reference*.

582651 Project in Information-Theoretic Modeling

There is also a related project:

- 2 credit units.
- Period II
- This course is a prerequisite.
- Together they replace the old *Three Concepts: Information* course—both old and new version cannot be included in your degree.
- Groups of 2–3 persons.
- Task: compress data.
- Best compressor wins! Intermediate results annonced periodically.
- Programming + report.

Prerequisites

No formal prerequisites **but** you will need

- Calculus: integrals, derivatives, convergence, ...
- Probability theory: joint & conditional distributions, expectations, law of large numbers, ...
- Programming: language is up to you
  (but need to work in groups in project).

What do I need to do?

- Weekly exercises:
  - Mathematical problems.
  - Programming tasks.
- Course exam.

You do *not* have to attend the classes, unless otherwise stated. However, it is recommend that you do.

- Alternatively, you can just take a separate exam later (next one is on Nov 28).

Grading

The course grading is based on:

1. Exercises (40 %)
2. Exam (60 %)

Minimum 50 % of exercises have to be solved (or at least seriously attempted).

Feel free to discuss homework problems in groups.
However, each student is expected to

- independently write down his/her solutions to theory problems
- implement all programming tasks by him/herself.

What is Information?

- Etymology: *informare* = give form, 14th century.
- *knowledge [...], intelligence, news, facts, data, [...], (as nucleotides in DNA or binary digits in a computer program) [...], a signal [...], a numerical quantity that measures the uncertainty in the outcome of an experiment to be performed.* (source: Merriam-Webster).
- Data vs. Information vs. Knowledge.
- Information technology.
- Physical information.
- This course: measuring *the amount* of information in data, and using such measures for automatically buiding *models*.

Why Information?

- The amount of data around us is exploding – internet!
- Need to *store*, *transmit*, *and process* it efficiently.
- Wish to *understand* more and more complex phenomena.
- Computer science: make things automatic (intelligent).

Information vs. Complexity

One way of thinking about this is

$$\textit{Complexity} = \textit{Information} + \textit{Noise}$$

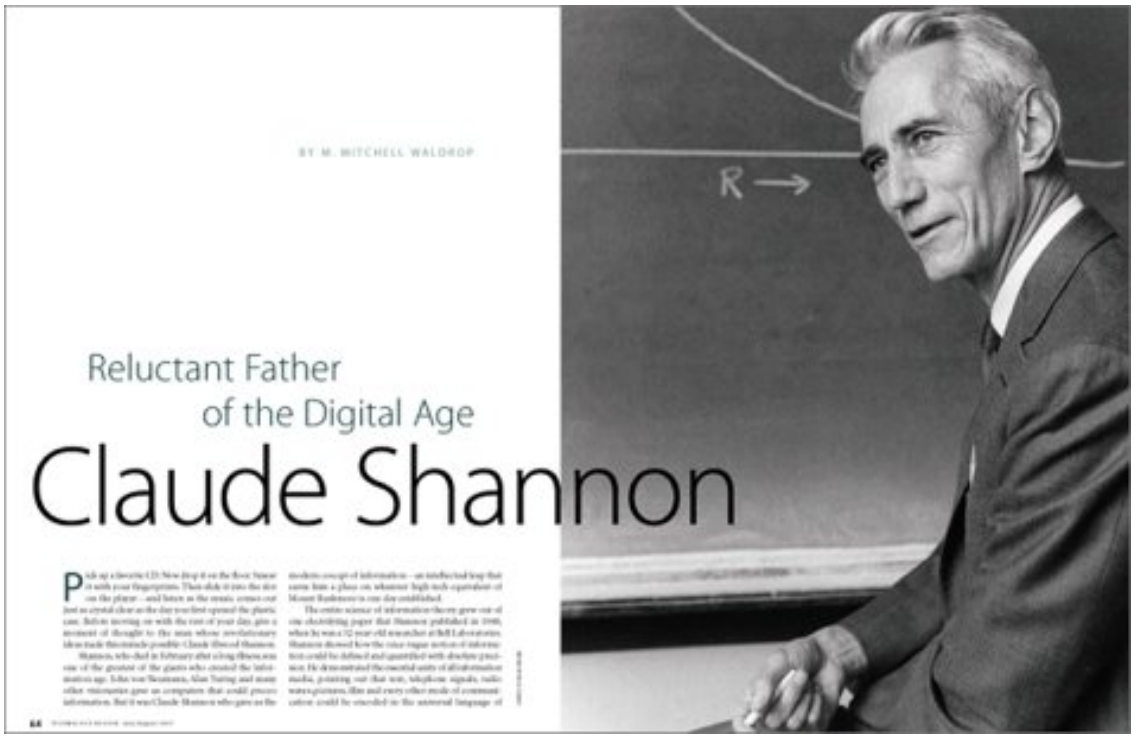
so that for example random strings are complex, but the complexity mainly comes from noise, not information. From a bit different points of view,

$$\textit{Complexity} = \textit{Regularity} + \textit{Randomness}$$

or

$$\textit{Complexity} = \textit{Algorithm} + \textit{Compressed file}.$$

Information Theory



"The real birth of modern information theory can be traced to the publication in 1948 of Claude Shannon's *"The Mathematical Theory of Communication"* in the Bell System Technical Journal. " (Encyclopædia Britannica)

Course Topics

Information Theory:

- entropy and information, bits,
- compression,
- error correction.

Fundamental limits (mathematical and statistical) and practice (computer science).

Modeling:

- statistical models,
- complexity (in data and models),
- over-fitting, Occam's Razor, and MDL Principle.

Coding Game

Form groups of 3–4 persons. Each group constructs a *code* for the letters A–Z by using as *code-words* unique sequences of dots • and dashes (—) like "•", "— •", " — • — —", etc.

A	_____	G	_____	M	_____	S	_____	Y	_____
B	_____	H	_____	N	_____	T	_____	Z	_____
C	_____	I	_____	O	_____	U	_____		
D	_____	J	_____	P	_____	V	_____		
E	_____	K	_____	Q	_____	W	_____		
F	_____	L	_____	R	_____	X	_____		

Coding Game

Use your code to *encode* the message
"WHAT DOES THIS HAVE TO DO WITH INFORMATION".

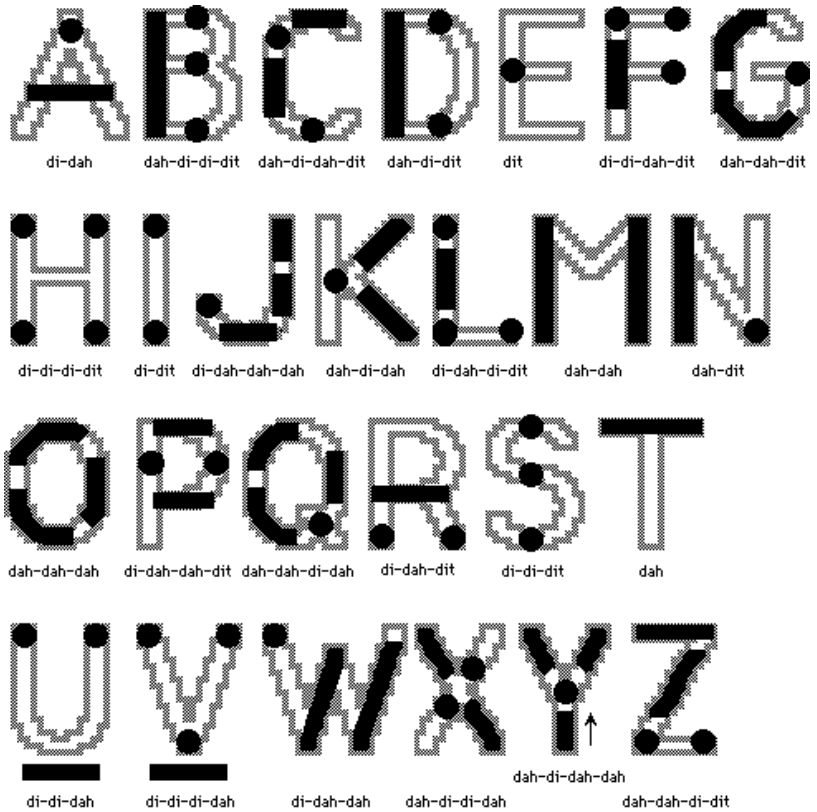
Now count how long the encoded message is using the rule:

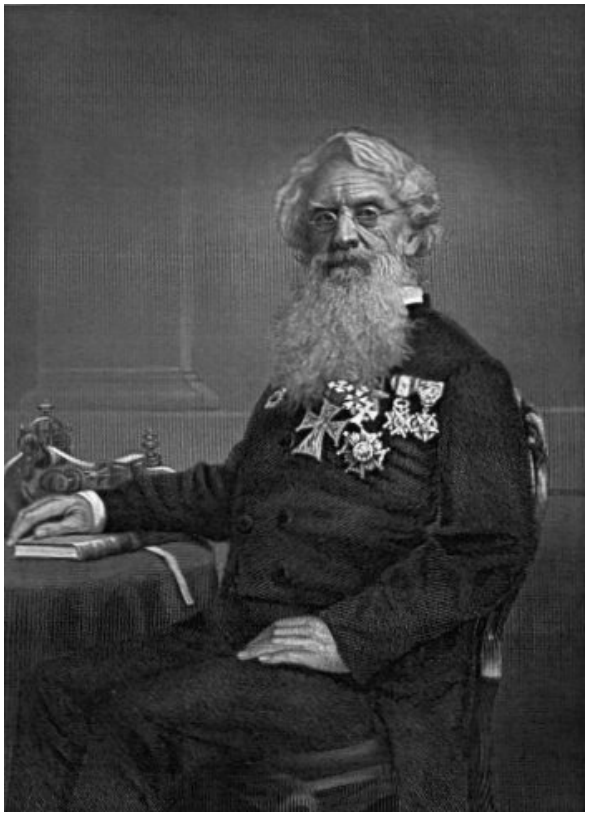
- A dot •: 1 units.
- A dash —: 2 units.
- A space between words: 2 units.

• • • — — — • • •: $1+1+1+2+2+2+1+1+1=12$.

The *coding rate* of your code is the length of the encoded message divided by the length of the original message, including spaces (42).

Coding Game





Samuel F.M. Morse (1791–1872)

Coding Game

WHAT DOES THIS HAVE TO DO WITH INFORMATION

.-- .... .- - -.. --- . ... - ....
.... .- ...- . - --- -.. --- .-- .. - ....
.. -. ..-. --- .-. -- .- - .. --- -.

51 dots, 36 dashes, 7 spaces: $51 + 72 + 14 = 137$ units.

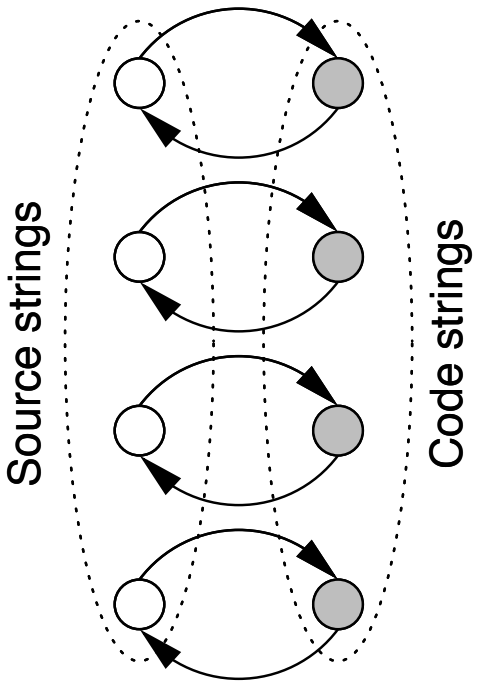
Morse code

Coding rate: $\frac{137}{42} \approx 3.26$

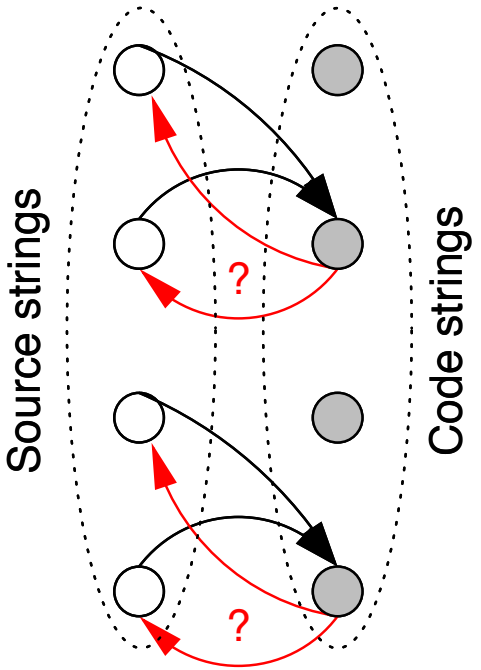
Did you do better or worse? Why?

Codes as Mappings

Lossless compression:
injective mapping

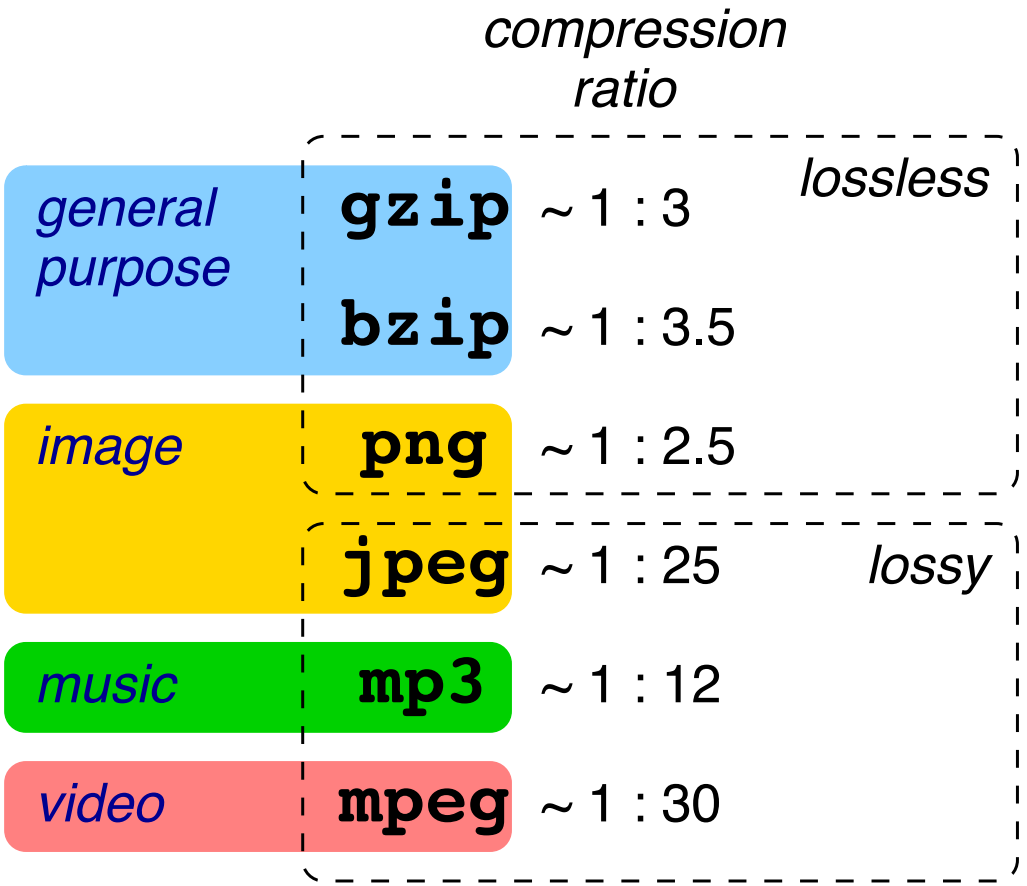


Lossy compression:
non-injective mapping



Only *lossless* codes are *uniquely decodable*.

Examples


compression ratio
general purpose
gzip
~ 1 : 3
bzip
~ 1 : 3.5
lossless
image
png
~ 1 : 2.5
jpeg
~ 1 : 25
lossy
music
mp3
~ 1 : 12
video
mpeg
~ 1 : 30

Compression

Is it always possible to compress data?

Theorem

The proportion of binary strings compressible by more than k bits is less than 2^{-k}.

Proof. For all $n \geq 1$, the number of binary strings of length n is 2^n.The number of binary code strings of length less than $n-k$ is $2^0 + 2^1 + 2^2 + \ldots + 2^{n-k-1} = 2^{n-k} - 1$. Thus the ratio is

$$\frac{2^{n-k} - 1}{2^n} < \frac{2^{n-k}}{2^n} = 2^{-k}.$$

□

Less than 50 % of files are compressible by more than one bit.

Compression

Is it always possible to compress data?

Theorem

The proportion of binary strings compressible by more than k bits is less than 2^{-k}.

Proof. For all $n \geq 1$, the number of binary strings of length n is 2^n.The number of binary code strings of length less than $n-k$ is $2^0 + 2^1 + 2^2 + \ldots + 2^{n-k-1} = 2^{n-k} - 1$. Thus the ratio is

$$\frac{2^{n-k} - 1}{2^n} < \frac{2^{n-k}}{2^n} = 2^{-k}.$$

□

Less than 1 % of files are compressible by more than 7 bits.

Compression

Is it always possible to compress data?

Theorem

The proportion of binary strings compressible by more than k bits is less than 2^{-k}.

Proof. For all $n \geq 1$, the number of binary strings of length n is 2^n.The number of binary code strings of length less than $n-k$ is $2^0 + 2^1 + 2^2 + \ldots + 2^{n-k-1} = 2^{n-k} - 1$. Thus the ratio is

$$\frac{2^{n-k} - 1}{2^n} < \frac{2^{n-k}}{2^n} = 2^{-k}.$$

□

Less than 0.00000000000000000000000000001 % of files are compressible by 100 bits.

How is it possible?

Why was the compression ratio greater than one in all the examples we saw?

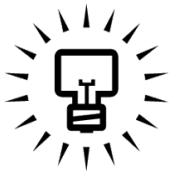
What are those rare files that are compressible?

Why are the files we use in practice so often compressible?

Compression

`echo <x> | gzip - | wc -c` # multiply by 8 for bits

Source string, x		$\ell(C(x))$	**ratio**
$aaa\ldots a$	$(10000 \times a)$	368	27.2 : 1.
$aabaabbbbabbbbb\ldots$	(10000 random letters)	13456	0.74 : 1
$abababab\ldots ab$	$(5000 \times ab)$	368	27.2 : 1
$aaa\ldots abbb\ldots b$	$(5000 \times a, 5000 \times b)$	376	26.6 : 1
$abbaababba\ldots$	$(1000 \times abbaababba)$	488	20.5 : 1



Strings following a rule are compressible?

Compression

`echo <x> | gzip - | wc -c` # multiply by 8 for bits

Source string, x		$\ell(C(x))$	**ratio**
$aaa\ldots a$	$(10000 \times a)$	368	27.2 : 1.
$aabaabbbbabbbbb\ldots$	(10000 random letters)	13456	0.74 : 1
$abababab\ldots ab$	$(5000 \times ab)$	368	27.2 : 1
$aaa\ldots abbb\ldots b$	$(5000 \times a, 5000 \times b)$	376	26.6 : 1
$abbaababba\ldots$	$(1000 \times abbaababba)$	488	20.5 : 1
$aaabbabbabb\ldots$	$(\pi,\ 0\text{–}4 \mapsto a,\ 5\text{–}9 \mapsto b)$	13416	0.74 : 1

π follows a rule but isn't compressed!

Maybe it's just `gzip`? It would be possible to create to *special program* to compress π into a short file.

But what does it mean to compress an *individual* string?

Information

An individual string is "simple" (as opposed to "complex") if it can be compressed into a small file by a *prespecified* program.

But which program? `gzip` is not good for images (or for π).

We can use several compressors if we prefix the code string by an index of the used program.

How about new compressors? *Self-extracting files!*

Can it be made automatic? Find the shortest program to print x.
No. *Kolmogorov complexity.*

Project

Next on course

Basics of (statistical) information theory

- introductory examples
- mathematical background
- basic concepts: entropy, mutual information etc.

Source coding

Alice, Bob, Cecilia and David play frequent tennis tournaments. We want to encode the sequence of winners using the least possible number of bits.
For example, denoting by A the event that Alice wins, etc., we could choose codes

$$C(\mathrm{A}) = 00 \qquad C(\mathrm{B}) = 01$$
$$C(\mathrm{C}) = 10 \qquad C(\mathrm{D}) = 11.$$

A sequence where first Bob wins, then Alice twice in row, and then David, would be encoded as 01000011.
The code length is four bits per result.

Source coding (cont.)

Suppose now that we additionally know the winning probabilities are as follows: Alice wins 50% of the time, Bob 25%, and Cecilia and David 12.5% each.
That is,

$$P(\mathrm{A}) = 1/2 \qquad P(\mathrm{B}) = 1/4$$
$$P(\mathrm{C}) = 1/8 \qquad P(\mathrm{D}) = 1/8.$$

Consider the following encoding:

$$C(\mathrm{A}) = 0 \qquad C(\mathrm{B}) = 10$$
$$C(\mathrm{C}) = 110 \qquad C(\mathrm{D}) = 111.$$

Code lengths are not constant any more. However sequences of code words still have a unique meaning. For example, 11011110 can only mean the sequence CDB.

Source coding (cont.)

The *expected* code length per result is now

$$\begin{aligned}
&\sum_{x \in \{\,\mathrm{A,B,C,D}\,\}} p(x)\ell(C(x)) \\
&\quad = \frac{1}{2}\cdot\ell(0) + \frac{1}{8}\cdot\ell(10) + \frac{1}{8}\cdot\ell(110) + \frac{1}{2}\cdot\ell(111) \\
&\quad = \frac{1}{2}\cdot 1 + \frac{1}{4}\cdot 2 + \frac{1}{8}\cdot 3 + \frac{1}{8}\cdot 3 = \frac{7}{4}.
\end{aligned}$$

The expected code length $7/4$ bits per results is actually the same as the *entropy* of the distribution from which the results come.

This is because we chose code lengths such that $\ell(C(x)) = \log_2(1/P(x))$, which makes the code in some sense optimal for this source.

Source coding (cont.)

Question: Can we do better?
Answer: Perhaps, *if* we make additional assumptions.

Question: Can this be implemented efficiently?
Answer: Yes.

Question: What if $\log_2(P(X))$ is not an integer?
Answer: In this context we can give a reasonable interpretation to fractions of a bit.

We'll return to all these questions later.

Noisy channel coding

Suppose we want to transmit a sequence of symbols from the 26-letter alphabet A, B, C, ..., Z. They are transmitted over a noisy channel where with probability 1/2, there is no error, but with probability 1/2, the symbol gets corrupted to the next one in the alphabet (with Z rolling over to A).

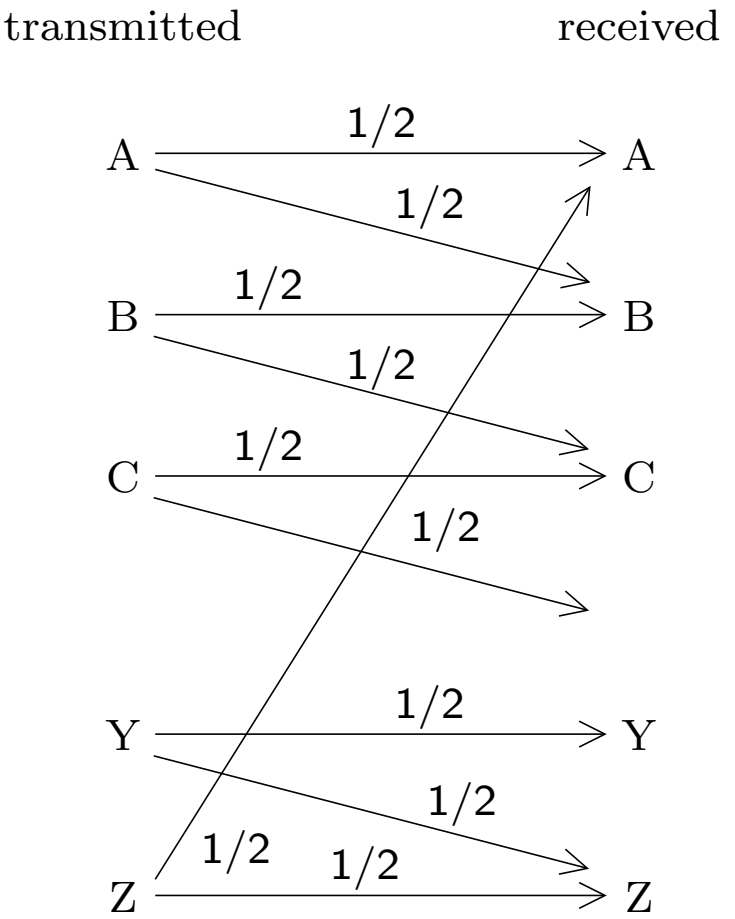
Noisy channel coding (cont.)

Denoting the transmitted symbol by X and the received symbol by $\hat{X}$, we thus have

$$P(\hat{X} = \mathrm{A} \mid X = \mathrm{A}) = 1/2 \qquad P(\hat{X} = \mathrm{B} \mid X = \mathrm{A}) = 1/2$$
$$P(\hat{X} = \mathrm{B} \mid X = \mathrm{B}) = 1/2 \qquad P(\hat{X} = \mathrm{C} \mid X = \mathrm{B}) = 1/2$$
$$\ldots \qquad \ldots$$
$$P(\hat{X} = \mathrm{Z} \mid X = \mathrm{Z}) = 1/2 \qquad P(\hat{X} = \mathrm{A} \mid X = \mathrm{Z}) = 1/2.$$

For example, transmission INFORMATION might be received as IOGPRMBTIOO.

Noisy channel coding (cont.)



Noisy channel coding (cont.)

Suppose now we are willing to increase the number of transmissions in order to allow error correction.

First notice that if we restrict our transmissions to symbols A, C, E, G, ..., Y, then we can always deduce the correct symbol.

For example, if we receive D, we know that it's a corrupted C, since D itself is not on the list of symbols we use.

Noisy channel coding (cont.)

We use this to devise the following encoding, where we use two symbols to encode each original symbol:

$$C(\mathrm{A}) = \mathrm{AA} \qquad C(\mathrm{B}) = \mathrm{AC}$$
$$C(\mathrm{C}) = \mathrm{CA} \qquad C(\mathrm{D}) = \mathrm{CC}$$
$$C(\mathrm{E}) = \mathrm{EA} \qquad C(\mathrm{F}) = \mathrm{EC}$$
$$\ldots \qquad \ldots$$

Now in each block of two symbols in the code, if the first received symbol is for example E or F, we know that E or F was transmitted, and the second symbol tells which one.

Noisy channel coding (cont.)

Thus we can have full error correction at the cost of doubling the lenght of the transmissions. Can we do better?

If the alphabet size is 2^b, then one symbol takes b bits.

Let's assume for the moment this somehow also makes sense if the alphabet size is not a power of 2.

This can be made more precise (and practical) by encoding longer blocks of symbols together.

Noisy channel coding (cont.)

Thus for alphabet size 26, we need to transmit $\log_2 26 \approx 4.7$ bits per symbol.

We just argued that by transmitting one symbol over the noisy channel, we can communicate without error one symbol from a set of 13. This is worth $\log_2 13 \approx 3.7$ bits.

So intutively, we should be able to correct all errors by increasing the message length by a factor of $\log_2 26 / \log_2 13 \approx 1.27$.

This would be much better than the factor 2 of the previous encoding.

Noisy channel coding (cont.)

To get an idea how this could be done, we consider coding blocks of 3 symbols using 4 symbols per block, giving ratio $4/3 \approx 1.33$.

Consider a block $s_1 s_2 s_3$ consisting of three symbols from alphabet $\{ \mathrm{A}, \mathrm{B}, \mathrm{C}, \ldots, \mathrm{Z} \}$. We start by encoding it as a block $t_1 t_2 t_3 t_4 t_5 t_6$ of six symbols from $\{ \mathrm{A}, \mathrm{C}, \mathrm{E}, \ldots, \mathrm{Y} \}$ using the encoding C.

For example, if $s_1 s_2 s_3 = \mathrm{CBZ}$, then $t_1 t_2 t_3 t_4 t_5 t_6 = \mathrm{CAACYC}$.

Notice that t_2, t_4 and t_6 take only values A and C. There are only 8 possible combinations $t_2 t_4 t_6$. We encode these using the symbols A, C, E, G, I, K, M and O. Let's denote this code for $t_2 t_4 t_6$ by t_7.

If we now transmit $t_1 t_3 t_5 t_7$, we can recover $s_1 s_2 s_3$ error-free.

Noisy channel coding (cont.)

The channel coding idea can be generalised to situations where the noise does not have such a nice structure.

It turns out that (asymptotically), the achievable error-free transmission rate depends on a quantity called *mutual information* between the transmitted and received symbol.

These two examples illustrate the role of *redundancy* in communication theory.

In the tennis tournament example, we used the properties of the information source to *remove* redundancy to compress the data.

In the noisy typewriter example, we *added* redundancy using the properties of the channel to allow error correction.