What are the Ingredients of a Theory of Data Compression?

- A context for the problem.

  Eg, what are we trying to compress, and what are we compressing it into?

- A notion of what data compression schemes are *possible*.

  A data compression scheme must allow us to *encode* data, and then *decode* it, recovering the original data.

- A measure of *how good* a data compression scheme is.

  We will have to look at how good a scheme is *on average*, given some model for the source.

One Danger: If we don't formalize things well, we might eliminate data compression schemes that would have been practical.

What Do We Hope to Get From a Theory of Data Compression?

- Easier ways of telling whether a data compression scheme is possible, and if so, how good it is.
- A theorem that tells us how good a scheme can possibly be — the "theoretical limit".
- Some help in finding a scheme that approaches this theoretical limit.
- Insight into the nature of the problem, which may help for other problems.

One insight: Compression is limited by the *entropy* of the source, which is a measure of *information content* that has many other uses.

Formalizing the Source of Data

We'll assume that we are trying to compress data from a digital *source* that produces a sequence of symbols, $X_1, X_2, \ldots$. These will be viewed as *random variables*; particular values they take will be denoted by $x_1, x_2, \ldots$.

These *source symbols* come from a finite *source alphabet*, $\mathcal{A}_X = \{a_1, \ldots, a_I\}$.

Examples:

$$\mathcal{A}_X = \{A, B, \ldots, Z, \_\}$$
$$\mathcal{A}_X = \{0, 1, 2, \ldots, 255\}$$
$$\mathcal{A}_X = \{C, G, T, A\}$$
$$\mathcal{A}_X = \{0, 1\}$$

The source alphabet is known to the receiver — who may be us at a later time, for storage applications.

Formalizing What We Compress To

The output of the compression program is a sequence of *code symbols*, $Z_1, Z_2, \ldots$ from a finite *code alphabet*, $\mathcal{A}_Z$.

These symbols are sent through the *channel*, to the receiver. We assume for now that the channel is noise-free — the symbol received is always the symbol that was sent.

We'll almost always assume that the code alphabet is $\{0, 1\}$, since computer files and digital transmissions are usually binary, but the theory can easily be generalized to any finite code alphabet.

Possible Compression Programs

A compression program (ie, a *code*) defines a mapping of each source symbol to a finite sequence of code symbols (a *codeword*).

For example, suppose our source alphabet is

$$\mathcal{A}_X = \{C, G, T, A\}$$

One possible code is

$$\begin{aligned} C &\rightarrow \mathsf{0} \\ G &\rightarrow \mathsf{10} \\ T &\rightarrow \mathsf{110} \\ A &\rightarrow \mathsf{1110} \end{aligned}$$

We encode a sequence of source symbols by concatenating the codewords of each:

$$CCAT \rightarrow \mathsf{001110110}$$

We require that the mapping be such that we can *decode* this sequence.

Later, we'll see that the above formalization isn't really right...

What Codes are Decodable?

We intend to consider only codes that can be decoded. But what do we mean by that?

This may depend on how the channel behaves at the end of a transmission. Four possibilities:

- The end of the transmission is explicitly marked, say by "$":

  011101101$

- After the end of the transmission, subsequent symbols all have a single known value, say "0":

  0111011010000000000...

- After the end of the transmission, subsequent symbols are random garbage:

  0111011011100100101...

- There is no end to the transmission.

When Do We Need the Decoding?

Another possible issue is when we require that a decoded symbol be known. Possibilities:

- As soon as the codeword for the symbol has been received.

  If this is possible, the code is *instantaneously decodable*.

- With no more than a fixed delay after the codeword for the symbol has been received.

  If this is possible, the code is *decodable with bounded delay*.

- Not until the entire message has been received.

  Assuming that the end of transmission is explicitly marked, we then require only that the code be *uniquely decodable*.

How Much Difference Does it Make?

We could develop theories of data compression with various definitions of decodability.

Question: How much difference will it make?

Will we find that we can't compress data as much if we insist on using a code that is instantaneously decodable?

Or will we find that a single theory is "robust" — not sensitive to the exact details of the channel and decoding requirements.

Easiest: Assume the end of transmission is explicitly marked; don't require any symbols be decoded until the entire message is received.

Hardest: Require instantaneous decoding. (It then won't matter whether the end of transmission is marked, as far as decoding the symbols that were actually sent is concerned.)

Notation for Sequences & Codes

$\mathcal{A}_X$ and $\mathcal{A}_Z$ are the source and code alphabets.

$\mathcal{A}_X^+$ and $\mathcal{A}_Z^+$ denote sequences of one or more symbols from the source or code alphabets.

A symbol code, C, is a mapping $\mathcal{A}_X \to \mathcal{A}_Z^+$. We use $c(x)$ to denote the codeword C maps x to.

We can use concatenation to extend this to a mapping for the *extended code*, $C^+ : \mathcal{A}_X^+ \to \mathcal{A}_Y^+$:

$$c^+(x_1 x_2 \cdots x_N) \;=\; c(x_1)c(x_2)\cdots c(x_N)$$

Ie, we code a string of symbols by just stringing together the codes for each symbol.

We sometimes also use C to denote the set of codewords: $\{w \mid w = C(a) \text{ for some } a \in \mathcal{A}_X\}$.

Formalizing Uniquely Decodable and Instantaneous Codes

We can now define a code to be *uniquely decodable* if the mapping $C^+ : \mathcal{A}_X^+ \to \mathcal{A}_Z^+$ is one-to-one. In other words:

For all $\boldsymbol{x}$ and $\boldsymbol{x}'$ in $\mathcal{A}_X^+$, $\boldsymbol{x} \neq \boldsymbol{x}'$ imples $c^+(\boldsymbol{x}) \neq c^+(\boldsymbol{x}')$

A code is obviously not uniquely decodable if two symbols have the same codeword — ie, if $c(a_i) = c(a_j)$ for some $i \neq j$ — so we'll usually assume that this isn't the case.

We define a code to be *instantaneously decodable* if any source sequences $\boldsymbol{x}$ and $\boldsymbol{x}'$ in $\mathcal{A}^+$ for which $\boldsymbol{x}$ is not a prefix of $\boldsymbol{x}'$ have encodings $\boldsymbol{z} = C(\boldsymbol{x})$ and $\boldsymbol{z}' = \mathcal{C}(\boldsymbol{x}')$ for which $\boldsymbol{z}$ is not a prefix of $\boldsymbol{z}'$. (Since otherwise, after receiving $\boldsymbol{z}$, we wouldn't yet know whether the message starts with $\boldsymbol{z}$ or with $\boldsymbol{z}'$.)

Examples

Examples with $\mathcal{A}_X = \{a, b, c\}$ and $\mathcal{A}_Z = \{0, 1\}$:

	Code A	Code B	Code C	Code D
a	10	0	0	0
b	11	10	01	01
c	111	110	011	11

Code A: Not uniquely decodable
Both bbb and cc encode as 111111

Code B: Instantaneously decodable
End of each codeword marked by 0

Code C: Decodable with one-symbol delay
End of codeword marked by *following* 0

Code D: Uniquely decodable, but with unbounded delay:
011111111111111 decodes as $accccccc$
01111111111111 decodes as $bcccccc$

How to Check Whether a Code is Uniquely Decodable

The *Sardinas-Patterson Theorem* tells us how to check whether a code is uniquely decodable.

Let C be the set of codewords.

Define $C_0 = C$.

For $n > 0$, define

$$C_n \;=\; \{w \in \mathcal{A}_X^+ \mid uw = v \text{ where } u \in C,\ v \in C_{n-1} \text{ or } u \in C_{n-1},\ v \in C\}$$

Finally, define

$$C_\infty \;=\; C_1 \cup C_2 \cup C_3 \cup \cdots$$

The code C is uniquely decodable if and only if C and C_∞ are disjoint.

We won't both much with this theorem, since as we'll see it isn't of much practical use.

How to Check Whether a Code is Instantaneously Decodable

A code is instantaneous if and only if no codeword is a prefix of some other codeword.

Proof:

($\Rightarrow$) If codeword $\mathcal{C}(a_i)$ is a prefix of codeword $\mathcal{C}(a_j)$, then the encoding of the sequence $\boldsymbol{x} = a_i$ is obviously a prefix of the encoding of the sequence $\boldsymbol{x}' = a_j$.

($\Leftarrow$) If the code is not instantaneous, let $\boldsymbol{z} = \mathcal{C}(\boldsymbol{x})$ be an encoding that is a prefix of another encoding $\boldsymbol{z}' = \mathcal{C}(\boldsymbol{x}')$, but with $\boldsymbol{x}$ not a prefix of $\boldsymbol{x}'$, and let $\boldsymbol{x}$ be as short as possible.

The first symbols of $\boldsymbol{x}$ and $\boldsymbol{x}'$ can't be the same, since if they were, we could drop these symbols and get a shorter instance. So these two symbols must be different, but one of their codewords must be a prefix of the other.