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.

---

<table>
<thead>
<tr>
<th>Formalizing the Source of Data</th>
<th>Formalizing What We Compress To</th>
</tr>
</thead>
<tbody>
<tr>
<td>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, $A_X = {a_1, \ldots, a_I}$. Examples: $A_X = {A, B, \ldots, Z, _}$ $A_X = {0, 1, 2, \ldots, 255}$ $A_X = {C, G, T, A}$ $A_X = {0, 1}$</td>
<td></td>
</tr>
<tr>
<td>The source alphabet is known to the receiver — who may be us at a later time, for storage applications.</td>
<td></td>
</tr>
<tr>
<td>The output of the compression program is a sequence of code symbols, $Z_1, Z_2, \ldots$ from a finite code alphabet, $A_Z$. 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.</td>
<td></td>
</tr>
</tbody>
</table>
**Possible Compression Programs**

A compression program (i.e., 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

\[ \mathcal{X} = \{C, G, T, A\} \]

One possible code is

- \( C \rightarrow 0 \)
- \( G \rightarrow 10 \)
- \( T \rightarrow 110 \)
- \( A \rightarrow 1110 \)

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

\[ CCAT \rightarrow 0011101110 \]

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”:

  \[ 01110110100000000000… \]

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

  \[ 011101101110100101… \]

- 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 \( \mathcal{A}_X \rightarrow \mathcal{A}_Z^+ \). 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, \( C^+ : \mathcal{A}_X^+ \rightarrow \mathcal{A}_Z^+ \):

\[
c^+(x_1 x_2 \ldots x_N) = c(x_1) c(x_2) \ldots c(x_N)
\]

I.e., 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 \( C^+ : \mathcal{A}_X^+ \rightarrow \mathcal{A}_Z^+ \) is one-to-one. In other words:

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

A code is obviously not uniquely decodable if two symbols have the same codeword — i.e., 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 \( x \) and \( x' \) in \( \mathcal{A}_X^+ \) for which \( x \) is not a prefix of \( x' \) have encodings \( z = C(x) \) and \( z' = C(x') \) for which \( z \) is not a prefix of \( z' \). (Since otherwise, after receiving \( z \), we wouldn't yet know whether the message starts with \( z \) or with \( z' \).)

Examples

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

<table>
<thead>
<tr>
<th>Code</th>
<th>Code A</th>
<th>Code B</th>
<th>Code C</th>
<th>Code D</th>
</tr>
</thead>
<tbody>
<tr>
<td>( a )</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>( b )</td>
<td>11</td>
<td>10</td>
<td>01</td>
<td>01</td>
</tr>
<tr>
<td>( c )</td>
<td>111</td>
<td>110</td>
<td>011</td>
<td>11</td>
</tr>
</tbody>
</table>

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:
\( 01111111111111 \) decodes as \( aaccccc \)
\( 011111111111111 \) decodes as \( bccccc \)

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

\[
C_{\infty} = C_1 \cup C_2 \cup C_3 \cup \ldots
\]

The code \( C \) is uniquely decodable if and only if \( C \) and \( C_{\infty} \) 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:

(⇒) If codeword \( C(a_i) \) is a prefix of codeword \( C(a_j) \), then the encoding of the sequence \( x = a_i \) is obviously a prefix of the encoding of the sequence \( x' = a_j \).

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

The first symbols of \( x \) and \( x' \) 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.