Predictive Coding

- The next symbol can be statistically predicted from the past.
  - Code with context
  - Code the difference
  - Move to front, then code

- Goal of prediction
  - The prediction should make the distribution of probabilities of the next symbol as skewed as possible
  - After prediction there is no way to predict more so we are in the first order entropy model

Bad and Good Prediction

- From information theory – The lower the information the fewer bits are needed to code the symbol.
  \[ \text{inf}(a) = \log_2 \frac{1}{P(a)} \]
- Examples:
  - \( P(a) = \frac{1024}{1024} \), \( \text{inf}(a) = .000977 \)
  - \( P(a) = \frac{1}{2} \), \( \text{inf}(a) = 1 \)
  - \( P(a) = \frac{1}{1024} \), \( \text{inf}(a) = 10 \)

Entropy

- Entropy is the expected number of bit to code a symbol in the model with \( a_i \) having probability \( P(a_i) \).
  \[ H = \sum_{i=1}^{n} P(a_i) \log_2 \left( \frac{1}{P(a_i)} \right) \]
- Good coders should be close to this bound.
  - Arithmetic
  - Huffman
  - Golomb
  - Tunstall

PPM

- Prediction with Partial Matching
  - Cleary and Witten (1984)
  - Tries to find a good context to code the next symbol
  [Table showing contexts and their associated counts]
  - Uses adaptive arithmetic coding for each context

JBIG

- Coder for binary images
  - documents
  - graphics
- Codes in scan line order using context from the same and previous scan lines.
  [Diagram showing context and next bit to be coded]
- Uses adaptive arithmetic coding with context
**JBIG Example**

\[
\begin{array}{c|cc}
\text{next bit} & 0 & 1 \\
\hline
\text{frequency} & 100 & 10 \\
\end{array}
\]

\[
H = -\frac{100}{110} \log_2(100/110) -\frac{10}{110} \log_2(10/110) = .44
\]

\[
\begin{array}{c|cc}
\text{next bit} & 0 & 1 \\
\hline
\text{frequency} & 15 & 50 \\
\end{array}
\]

\[
H = -\frac{15}{65} \log_2(15/65) -\frac{50}{65} \log_2(50/65) = .78
\]

**Issues with Context**

- **Context dilution**
  - If there are too many contexts then too few symbols are coded in each context, making them ineffective because of the zero-frequency problem.

- **Context saturation**
  - If there are too few contexts then the contexts might not be good as having more contexts.

- **Wrong context**
  - Again poor predictors.

**Prediction by Differencing**

- **Used for Numerical Data**
- **Example:** 2 3 4 5 6 7 8 7 6 5 4 3 2

  \[
  \begin{array}{c|c|c|c|c|c}
  \text{next bit} & 0 & 1 & 2 & 3 & 4 \\
  \hline
  \text{frequency} & 10 & 10 & 10 & 10 & 10 \\
  \end{array}
  \]

- **Transform to 2 1 1 1 1 1 1 1 1 1 1**
  - much lower first-order entropy

**General Differencing**

- **Let** \( x_1, x_2, \ldots, x_n \) **be some numerical data that is correlated, that is** \( x_i \) **is near** \( x_{i+1} \)
- **Better compression can result from coding**
  \[ x_1, x_2 - x_1, x_3 - x_2, \ldots, x_i - x_{i-1} \]
- **This idea is used in**
  - signal coding
  - audio coding
  - video coding
- **There are fancier prediction methods based on linear combinations of previous data, but these may require training.**

**Move to Front Coding**

- **Non-numerical data**
- The data have a relatively small working set that changes over the sequence.
- **Example:** \( a \ b \ b \ a \ b \ c \ b \ b \ c \ c \ c \ b \ c \ b \ c \ b \)
- **Move to Front algorithm**
  - Symbols are kept in a list indexed 0 to \( m-1 \)
  - To code a symbol output its index and move the symbol to the front of the list

**Example**

- **Example:** \( a \ b \ b \ a \ b \ c \ b \ b \ c \ c \ c \ b \ c \ b \ b \)

  \[
  \begin{array}{c|c|c|c}
  \text{next bit} & 0 & 1 & 2 \\
  \hline
  \text{frequency} & 1 & 2 & 3 \\
  \end{array}
  \]

- **Transform to 0 1 2 3**
  - \( a \ b \ c \ d \)
Example
- Example: \( a b a b a b c b b b c b c b c b c b c \)
  
  \[
  \begin{array}{cccc}
  0 & 1 & 2 & 3 \\
  a & b & c & d \\
  \downarrow \\
  0 & 1 & 2 & 3 \\
  b & a & c & d \\
  \end{array}
  \]

Example
- Example: \( a b a b a b c b b b c b c b c b c b c \)
  
  \[
  \begin{array}{cccc}
  0 & 1 & 2 & 3 \\
  b & a & c & d \\
  \downarrow \\
  0 & 1 & 2 & 3 \\
  a & b & c & d \\
  \end{array}
  \]

Example
- Example: \( a b a b a b c b b b c b c b c b c b c \)
  
  \[
  \begin{array}{cccc}
  0 & 1 & 2 & 3 \\
  a & b & c & d \\
  \downarrow \\
  0 & 1 & 2 & 3 \\
  b & a & c & d \\
  \end{array}
  \]

Example
- Example: \( a b a b a b c b b b c b c b c b c b c \)
  
  \[
  \begin{array}{cccc}
  0 & 1 & 2 & 3 \\
  a & b & c & d \\
  \downarrow \\
  0 & 1 & 2 & 3 \\
  b & a & c & d \\
  \end{array}
  \]
Example
- Example: \( \text{a b a b a b c b c c b c b c c b c b c} \)
  \[
  \begin{array}{cccc}
  0 & 1 & 2 & 3 \\
  b & a & c & d \\
  \downarrow \\
  0 & 1 & 2 & 3 \\
  c & b & a & d \\
  \end{array}
  \]

Example
- Example: \( \text{a b a b a b c b b c c c b c b c c b c b c c} \)
  \[
  \begin{array}{cccc}
  0 & 1 & 2 & 3 \\
  0 & 1 & 1 & 1 & 0 & 1 & 2 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 3 & 1 & 2 & 0 \\
  c & b & d & a \\
  \end{array}
  \]

Example
- Example: \( \text{a b a b a b c b c c b c b c c b c b c c b c b c c} \)

Example
- Example: \( \text{a b a b a b c b c c b c b c c b c b c c b c b c c} \)

Example
- Example: \( \text{a b a b a b c b c c b c b c c b c b c c b c b c c} \)

Example
- Example: \( \text{a b a b a b c b c c b c b c c b c b c c b c b c c} \)

Example
- Example: \( \text{a b a b a b c b c c b c b c c b c b c c b c b c c} \)

Example
- Example: \( \text{a b a b a b c b c c b c b c c b c b c c b c b c c} \)

Example
- Example: \( \text{a b a b a b c b c c b c b c c b c b c c b c b c c} \)

Example
- Example: \( \text{a b a b a b c b c c b c b c c b c b c c b c b c c} \)

Example
- Example: \( \text{a b a b a b c b c c b c b c c b c b c c b c b c c} \)

Example
- Example: \( \text{a b a b a b c b c c b c b c c b c b c c b c b c c} \)

Example
- Example: \( \text{a b a b a b c b c c b c b c c b c b c c b c b c c} \)

Burrows-Wheeler Transform
- Burrows-Wheeler, 1994
- BW Transform creates a representation of the data which has a small working set.
- The transformed data is compressed with move to front compression.
- The decoder is quite different from the encoder.
- The algorithm requires processing the entire string at once (it is not on-line).
- It is a remarkably good compression method.

Encoding Example
- abracadabra
  1. Create all cyclic shifts of the string.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Output</td>
<td>abracadabra</td>
<td>bracadabra</td>
<td>cabradabra</td>
<td>dbraabrac</td>
<td>erebradba</td>
<td>fadbabrca</td>
<td>gcbadraac</td>
<td>haidbraca</td>
<td>ibjadrac</td>
<td>kcaradbac</td>
<td>lcaabradbac</td>
</tr>
</tbody>
</table>
## Encoding Example

2. Sort the strings alphabetically into array A

| 0  | abracadabra          |
| 1  | abracadabra          |
| 2  | abracadabra          |
| 3  | abracadabra          |
| 4  | abracadabra          |
| 5  | abracadabra          |
| 6  | abracadabra          |
| 7  | abracadabra          |
| 8  | abracadabra          |
| 9  | abracadabra          |

| 10 | abracadabra          |

## Encoding Example

3. L = the last column

| 0  | abracadabra          |
| 1  | abracadabra          |
| 2  | abracadabra          |
| 3  | abracadabra          |
| 4  | abracadabra          |
| 5  | abracadabra          |
| 6  | abracadabra          |
| 7  | abracadabra          |
| 8  | abracadabra          |
| 9  | abracadabra          |

| 10 | abracadabra          |

## Encoding Example

4. Transmit X the index of the input in A and L (using move to front coding).

| 0  | abracadabra          |
| 1  | abracadabra          |
| 2  | abracadabra          |
| 3  | abracadabra          |
| 4  | abracadabra          |
| 5  | abracadabra          |
| 6  | abracadabra          |
| 7  | abracadabra          |
| 8  | abracadabra          |
| 9  | abracadabra          |

| 10 | abracadabra          |

## Why BW Works

- Ignore decoding for the moment.
- The prefix of each shifted string is a context for the last symbol.
  - The last symbol appears just before the prefix in the original.
- By sorting similar contexts are adjacent.
  - This means that the predicted last symbols are similar.

## Decoding Example

- We first decode assuming some information. We then show how compute the information.
- Let $A^*$ be $A$ shifted by 1

| 0  | abracadabra          |
| 1  | abracadabra          |
| 2  | abracadabra          |
| 3  | abracadabra          |
| 4  | abracadabra          |
| 5  | abracadabra          |
| 6  | abracadabra          |
| 7  | abracadabra          |
| 8  | abracadabra          |
| 9  | abracadabra          |

| 10 | abracadabra          |

## Decoding Example

- Assume we know the mapping $T[i]$ is the index in $A^*$ of the string $i$ in $A$.
- $T = [2 5 6 7 8 9 10 4 1 0 3]$

| 0  | abracadabra          |
| 1  | abracadabra          |
| 2  | abracadabra          |
| 3  | abracadabra          |
| 4  | abracadabra          |
| 5  | abracadabra          |
| 6  | abracadabra          |
| 7  | abracadabra          |
| 8  | abracadabra          |
| 9  | abracadabra          |

| 10 | abracadabra          |

| 10 | abracadabra          |

| 10 | abracadabra          |

| 10 | abracadabra          |
Decoding Example

• Let $F$ be the first column of $A$, it is just $L$, sorted.

$$F = \begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ a & a & a & a & b & b & b & b & c & c & c \\ \end{bmatrix}$$

$$T = \begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 2 & 5 & 6 & 7 & 8 & 9 & 10 & 4 & 1 & 0 & 3 \\ \end{bmatrix}$$

• Follow the pointers in $T$ in $F$ to recover the input starting with $X$.

Decoding Example

$$F = \begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ a & a & a & a & b & b & c & d & r & r \\ \end{bmatrix}$$

$$T = \begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 2 & 5 & 6 & 7 & 8 & 9 & 10 & 4 & 1 & 0 & 3 \\ \end{bmatrix}$$

• Why does this work?


Decoding Example

• How do we compute $F$ and $T$ from $L$ and $X$?

F is just $L$ sorted

$$F = \begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ a & a & a & a & b & b & c & d & r & r \\ \end{bmatrix}$$

L = r d a r c a a a a b b

Note that $L$ is the first column of $A^*$ and $A^*$ is in the same order as $A$.

If $i$ is the $k$-th $x$ in $F$ then $T[i]$ is the $k$-th $x$ in $L$. 

Decoding Example

• Let $F$ be the first column of $A$, it is just $L$, sorted.

$$F = \begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ a & a & a & a & b & b & b & b & c & c & c \\ \end{bmatrix}$$

$$T = \begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 2 & 5 & 6 & 7 & 8 & 9 & 10 & 4 & 1 & 0 & 3 \\ \end{bmatrix}$$

• Follow the pointers in $T$ in $F$ to recover the input starting with $X$. 

Decoding Example

$$F = \begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ a & a & a & a & b & b & c & d & r & r \\ \end{bmatrix}$$

$$T = \begin{bmatrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\ 2 & 5 & 6 & 7 & 8 & 9 & 10 & 4 & 1 & 0 & 3 \\ \end{bmatrix}$$

ab
Decoding Example

\[
\begin{align*}
F &= \text{a a a a b b c d r r} \\
L &= \text{r d a r c a a a a b b} \\
T &= \text{0 1 2 3 4 5 6 7 8 9 10} \\
&= \text{2 5 6 7 8 9 10}
\end{align*}
\]

Notes on BW

- Alphabetic sorting does not need the entire cyclic shifted inputs.
  - Sort the indices of the string
  - Most significant symbols first radix sort works
- There are high quality practical implementations
  - Bzip
  - Bzip2 (seems to be w/o patents)
Encoding Exercise

Encode the string \texttt{abababababababab} = (ab)^9
1. Find L and X
2. Do move-to-front coding of L.
3. Estimate the length of the code using first order entropy.

Decoding Exercise

Decode L = baaaba, X = 6
1. First Compute F and T
2. Use those to decode.