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.
  \[ \text{inf}(a) = \log_2 \left( \frac{1}{P(a)} \right) \]
- Examples:
  - \( P(a) = \frac{1023}{1024}, \text{inf}(a) = 0.000977 \)
  - \( P(a) = \frac{1}{2}, \text{inf}(a) = 1 \)
  - \( P(a) = \frac{1}{1024}, \text{inf}(a) = 10 \)

Entropy

- Entropy is the expected number of bit to code a symbol in the model with \( a_i \) having probability \( P(a_i) \).
  \[ H = \sum_{i=1}^{m} P(a_i) \log_2 \left( \frac{1}{P(a_i)} \right) \]
- 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
  
  \[
  \begin{array}{cccccccc}
  \text{context} & a & e & i & f & s & y \\
  \text{the} & 0 & 0 & 5 & 7 & 4 & 7 \\
  \text{he} & 1 & 1 & 7 & 10 & 9 & 7 \\
  \text{e} & 12 & 2 & 10 & 15 & 10 & 10 \\
  \langle \text{nil} \rangle & 50 & 70 & 30 & 35 & 40 & 13 \\
  \end{array}
  \]
- 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.
  - Uses adaptive arithmetic coding with context
JBIG Example

<table>
<thead>
<tr>
<th>next bit</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>frequency</td>
<td>100</td>
<td>10</td>
</tr>
<tr>
<td>H = -10 \log_{10} 100 + 10 \log_{10} 100 = -44</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

  Transform to 2 1 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
  \( x_1, x_2 - x_1, x_3 - x_2, \ldots, x_n - x_{n-1} \)
- 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 a b a b c b b c c c b d b c c
- 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 a b a b c b b c c c b d b c c
  0 1 2 3
  a b c d

Example

- Example: \textit{a b a a b c b c b c b c b c d b c c}
  0 1
  \begin{tabular}{cccc}
   0 & 1 & 2 & 3 \\
   a & b & c & d \\
  \end{tabular}
  ↓
  0 1 2 3
  b a c d

Example

- Example: \textit{a b a a b c b c b c b c b c}
  0 1 1
  \begin{tabular}{cccc}
   0 & 1 & 2 & 3 \\
   b & a & c & d \\
  \end{tabular}
  ↓
  0 1 2 3
  a b c d

Example

- Example: \textit{a b a a b c b c b c b c b c}
  0 1 1 1
  \begin{tabular}{cccc}
   0 & 1 & 2 & 3 \\
   a & b & c & d \\
  \end{tabular}
  ↓
  0 1 2 3
  b a c d

Example

- Example: \textit{a b a a b c b c b c b c b c}
  0 1 1 1 1
  \begin{tabular}{cccc}
   0 & 1 & 2 & 3 \\
   b & a & c & d \\
  \end{tabular}
  ↓
  0 1 2 3
  a b c d

Example

- Example: \textit{a b a a b c b c b c b c b c}
  0 1 1 1 1 0
  \begin{tabular}{cccc}
   0 & 1 & 2 & 3 \\
   a & b & c & d \\
  \end{tabular}

Example

- Example: \textit{a b a a b c b c b c b c b c}
  0 1 1 1 1 0 1
  \begin{tabular}{cccc}
   0 & 1 & 2 & 3 \\
   a & b & c & d \\
  \end{tabular}
  ↓
  0 1 2 3
  b a c d
Example

- Example: \[\text{a b a b a b c b c c b c b c c b c c b c c b c c b c c b c c b c c}}\]

\[
\begin{array}{c}
0 & 1 & 2 & 3 \\
\text{b a c d} \\
\downarrow \\
0 & 1 & 2 & 3 \\
\text{c b a d}
\end{array}
\]

Example

- Example: \[\text{a b a b a b c b c c b c b c c b c c b c c b c c b c c}}\]

\[
\begin{array}{c}
0 & 1 & 2 & 3 \\
\text{c b d a}
\end{array}
\]

Example

- Example: \[\text{a b a b a b c b c c b c c b c c b c c b c c b c c}}\]

\[
\begin{array}{c}
0 & 1 & 1 & 1 & 1 & 0 & 1 & 2 & 0 & 1 & 0 & 0 & 0 & 1 & 3 & 1 & 2 & 0 \\
\end{array}
\]

Frequencies of \{a, b, c, d\}

\[
\begin{array}{c}
a & b & c & d \\
4 & 7 & 8 & 1
\end{array}
\]

Frequencies of \{0, 1, 2, 3\}

\[
\begin{array}{c}
0 & 1 & 2 & 3 \\
8 & 9 & 2 & 1
\end{array}
\]

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.

Extreme Example

Input:

aaaaaaabbbbbbbbcoccccccccddddddddd

Output

000000000010000000000002000000003000000000000

Frequencies of \{a, b, c, d\}

\[
\begin{array}{c}
a & b & c & d \\
10 & 10 & 10 & 10
\end{array}
\]

Frequencies of \{0, 1, 2, 3\}

\[
\begin{array}{c}
0 & 1 & 2 & 3 \\
37 & 1 & 1 & 1
\end{array}
\]

Encoding Example

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

\[
\begin{array}{c}
0 & \text{abracadabra} \\
1 & \text{abra} & \text{cada} & \text{brab} & \text{rasa} & \text{a} & \text{bc} & \text{dabra} & \text{ c b a} & \text{d b}
\end{array}
\]
Encoding Example

2. Sort the strings alphabetically into array A

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>A₀</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>abracadabra</td>
<td>0 aabracadabr</td>
</tr>
<tr>
<td>1</td>
<td>bracadabra</td>
<td>1 aabraabracd</td>
</tr>
<tr>
<td>2</td>
<td>raacadabra</td>
<td>2 abracadabra</td>
</tr>
<tr>
<td>3</td>
<td>acadabraabr</td>
<td>3 acadabraabr</td>
</tr>
<tr>
<td>4</td>
<td>cadabraabr</td>
<td>4 adabraabr</td>
</tr>
<tr>
<td>5</td>
<td>adabraabr</td>
<td>5 abracaadba</td>
</tr>
<tr>
<td>6</td>
<td>dabraabra</td>
<td>6 braacadabra</td>
</tr>
<tr>
<td>7</td>
<td>abracadad</td>
<td>7 cadabraabr</td>
</tr>
<tr>
<td>8</td>
<td>braacadada</td>
<td>8 dabracabra</td>
</tr>
<tr>
<td>9</td>
<td>raabracada</td>
<td>9 raabracada</td>
</tr>
<tr>
<td>10</td>
<td>aabracadab</td>
<td>10 racadabraa</td>
</tr>
</tbody>
</table>

Encoding Example

3. L = the last column

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>L</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>abracadabra</td>
<td>aabracadab</td>
</tr>
<tr>
<td>1</td>
<td>abracadabra</td>
<td>abracadabra</td>
</tr>
<tr>
<td>2</td>
<td>aabracadab</td>
<td>aabracadab</td>
</tr>
<tr>
<td>3</td>
<td>acadabraabr</td>
<td>acadabraabr</td>
</tr>
<tr>
<td>4</td>
<td>dabraabra</td>
<td>dabraabra</td>
</tr>
<tr>
<td>5</td>
<td>raabracada</td>
<td>raabracada</td>
</tr>
<tr>
<td>10</td>
<td>racadabraa</td>
<td>racadabraa</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>A*</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>abracadabra</td>
<td>0 aabracadab</td>
</tr>
<tr>
<td>1</td>
<td>abracadabra</td>
<td>1 dabracadraw</td>
</tr>
<tr>
<td>2</td>
<td>abracadabra</td>
<td>2 abracadabra</td>
</tr>
<tr>
<td>3</td>
<td>acabraabra</td>
<td>3 acabraabra</td>
</tr>
<tr>
<td>4</td>
<td>dabraabra</td>
<td>4 dabracabra</td>
</tr>
<tr>
<td>5</td>
<td>braacadada</td>
<td>5 braacadada</td>
</tr>
<tr>
<td>6</td>
<td>braacadada</td>
<td>6 braacadada</td>
</tr>
<tr>
<td>7</td>
<td>cadabrabra</td>
<td>7 cadabrabra</td>
</tr>
<tr>
<td>8</td>
<td>dabraabra</td>
<td>8 dabraabra</td>
</tr>
<tr>
<td>9</td>
<td>raabracada</td>
<td>9 raabracada</td>
</tr>
<tr>
<td>10</td>
<td>racadabraa</td>
<td>10 racadabraa</td>
</tr>
</tbody>
</table>

Decoding Example

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

- Let $F$ be the first column of $A$, it is just $L$, sorted.
  
  $$F = \begin{bmatrix}
0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 \\
\text{a} & \text{a} & \text{a} & \text{a} & \text{b} & \text{b} & \text{c} & \text{d} & \text{r} & \text{r}
\end{bmatrix}$$

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

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

Decoding Example

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

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

Decoding Example

- Why does this work?

Decoding Example

- How do we compute $F$ and $T$ from $L$ and $X$? $F$ is just $L$ sorted

$$A = \begin{bmatrix}
0 & \text{abra}\text{bra}\text{dab} & 2 \\
1 & \text{abra}\text{bra}\text{cac} & 5 \\
2 & \text{abra}\text{bra}\text{dab} & 6 \\
3 & \text{ac}\text{ada}\text{bra}\text{abr} & 7 \\
4 & \text{ad}\text{abra}\text{bra}\text{rac} & 8 \\
5 & \text{bra}\text{bra}\text{rad} & 9 \\
6 & \text{bra}\text{bra}\text{dabra} & 10 \\
7 & \text{cab}\text{bra}\text{bra}\text{ra} & 4 \\
8 & \text{dabra}\text{bra}\text{rac} & 1 \\
9 & \text{ra}\text{bra}\text{bra}\text{dab} & 0 \\
10 & \text{rac}\text{a}\text{bra}\text{bra}\text{ra} & 3
\end{bmatrix}$$

$$A^v = \begin{bmatrix}
0 & \text{ra}\text{bra}\text{bra}\text{dab} & 0 \\
1 & \text{dabra}\text{bra}\text{rac} & 1 \\
2 & \text{bra}\text{bra}\text{rad} & 2 \\
3 & \text{ac}\text{ada}\text{bra}\text{abr} & 3 \\
4 & \text{ad}\text{abra}\text{bra}\text{rac} & 4 \\
5 & \text{babra}\text{rad} & 5 \\
6 & \text{abra}\text{bra}\text{dabra} & 6 \\
7 & \text{ac}\text{ada}\text{bra}\text{abr} & 7 \\
8 & \text{bra}\text{abra}\text{dabra} & 8 \\
9 & \text{dabra}\text{bra}\text{rac} & 9
\end{bmatrix}$$

Note that $L$ is the first column of $A^v$ and $A^v$ 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

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

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 abababababababab = (ab)^8
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 = baaaaaba, X = 6
1. First Compute F and T
2. Use those to decode.