Entropy Codings

Stream codes

October 2018

Content

- Arithmetic coding
- Other approaches
  - LZW coding
  - Ex-Golomb
  - Run length coding

Stream code - Arithmetic code

- Arithmetic code defines a way to efficiently encode a sequence of symbols into binary string
- Each sequence of symbols $\Longleftrightarrow$ a number in unit interval $[0, 1)$ (tag value)

Given:

- Source $X = a_i | \forall i$,
- Probability (pdf) of each symbol: $Pr(X = a_i) = P(a_i) = p_i$
- Cdf of each symbol: $F_X(i) = \sum_{k=1}^{i} P(X = k) = \sum_{k=1}^{i} p_k$

Assign each symbols onto unit interval $[0, 1)$:

- Symbol i is mapped to $[F_X(i-1), F_X(i))$ (*)
- Denote $I(i) = [F_X(i-1), F_X(i))$

Arithmetic coding

Coding Algorithm: *find tag value for a sequence*

1. Start with interval $I = [0, 1)$
2. Map symbols on I according to (*)
3. Read a symbol x from source.
4. Let $I = I(x)$. Repeat to step 1

Arithmetic coding - Interval contains tag value

$$interval = [l^{(k)}, u^{(k)})$$

$$
\begin{array}{rl}
sequence & interval \\
NULL: & \left\{ \begin{array}{lcl} l^{(0)} & = & 0 \\ u^{(0)} & = & 1 \end{array} \right. \\
a_i: & \left\{ \begin{array}{lcl} l^{(1)} & = & F_X(i-1) \\ u^{(1)} & = & F_X(i) \end{array} \right. \\
a_i a_j: & \left\{ \begin{array}{lcl} l^{(2)} & = & F_X(i-1) + F_X(j-1)(F_X(i) - F_X(i-1)) \\ & = & l^{(1)} + F_X(j-1)(u^{(1)} - l^{(1)}) \\ u^{(2)} & = & F_X(i-1) + F_X(j)(F_X(i) - F_X(i-1)) \\ & = & l^{(1)} + F_X(j)(u^{(1)} - l^{(1)}) \end{array} \right.
\end{array}
$$

- Recursively identify intervals as receiving symbols
- The more symbols received, the narrower intervals become
- The mid-point of the final interval can be used as tag value for a sequence.

Arithmetic Coding - From tag value to Sequence

Decoding Algorithm I: *find a sequence for a tag value*

1. Start with interval $I = [0, 1)$
2. Map symbols on I according to (*)
3. Find an interval that the tag value lies in. Output corresponding symbol x of that interval.
4. Let $I = I(x)$. Repeat to step 1

($\#$) Note: We should know the length of sequence in advance

Arithmetic Coding - From tag value to code word

Convert tag value (real number) into binary representation.

- How?
- We should truncate the binary string -> How long should the binary string is?

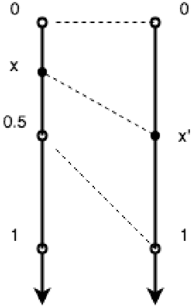
Convert decimal fractions to binary

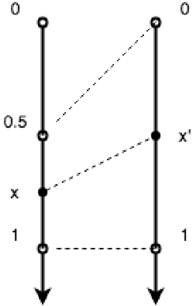
0 - 1/2	0	0 - 1/4	00	0 - 1/8	000
				1/8-1/4	001
		1/4 - 1/2	01	1/4 - 3/8	010
				3/8 - 1/2	011
1/2 - 1	1	1/2 - 3/4	10	1/2 - 5/8	100
				5/8 - 3/4	101
		3/4 - 1	11	3/4 - 7/8	110
				7/8 - 1	111

Algorithm: *find binary representation of a fraction*

1. Test if the fraction is in lower half of upper half.
2. Emit "0" if lower half and "1" if upper half.
3. Rescale the half to $[0, 1)$
4. Repeat step 1, until ...

Rescale upper half and lower half





$$x' = 2 \times x$$

$$x' = 2 \times (x - 0.5)$$

Arithmetic coding: Encoding with rescaling

Algorithm: *Output binary string for a sequence of symbols*

1. Let $I \leftarrow [0, 1)$. Map symbols on I
2. Read a symbol (x) from sequence. Find the corresponding sub-interval $s(x)$.
3. While $0.5 \notin s(x)$ do
   - If the sub-interval $s(x) \subset lowerhalf(I)$, emit "0". Let $I \leftarrow rescale\_lowerhalf(I))$
   - ElseIf the sub-interval $s(x) \subset upperhalf(I)$, emit "1". Let $I \leftarrow rescale\_upperhalf(I))$
   - Else $I \leftarrow s(x)$
4. Repeat step 2, until the last symbol
5. At the last symbol, emit binary representation of a point in $s(x)$
6. End and output the whole bin string.

Encoding with rescaling: Example

Sequence "$a_1 a_3 a_2 a_1$";
$P(a_1) = 0.8$
$P(a_2) = 0.02$
$P(a_3) = 0.18$

- ...
- ...

Arithmetic coding: Decoding with rescaling

Algorithm: *Output sequence of symbols for a binary string*

1. Let $I \leftarrow [0, 1)$. Map symbols on I.
2. Convert bit string to decimal fraction v. Find the corresponding sub-interval $s(x) : v \in s(x)$. Emit x
3. Shift left bit string (padding 0 to the right) and rescale $s(x)$ until $s(x) \ni 0.5$
4. Let $I \leftarrow s(x)$.
5. Repeat step 2, until the bit string becomes all-zero
6. End and output the whole symbol sequence.

Decoding algorithm

Algorithm: *Output sequence of symbols for a binary string*

1. Let $I \leftarrow [0, 1)$. Map symbols on I.
2. Let $J \leftarrow [0, 1)$
3. Read bit from bitstream gradually and calculate $J = intervalOf(readBits)$ until $J \in s(x)$ Emit x
4. Shift left bit string (padding 0 to the right) and rescale $s(x)$ until $s(x) \ni 0.5$
5. Let $I \leftarrow s(x)$.
6. Repeat step 2, until the bit string becomes all-zero
7. End and output the whole symbol sequence.

Decoding with rescaling: Example

Bit string "11000110";
$P(a_1) = 0.8$
$P(a_2) = 0.02$
$P(a_3) = 0.18$

- ...
- ...

Arithmetic coding: Characteristics

AC can be encoded and decoded incrementally $\rightarrow$ "stream code"

If we consider a source $S = mX$ where $X = \{x_i|_i\}$, then

- arithmetic coding AC is a prefix symbol code
- Code word length of AC:

$$H(mX) \leq L(AC, mX) < H(mX) + 2$$

- Thus, bit length per symbol:

$$H(X) \leq L(AC, X) = \frac{L(AC, mX)}{m} < H(X) + \frac{2}{m}$$

Dictionary-based compression - LZW

Encoding Algorithm: *Produce bit stream for stream of ASCII string*

1. Load ASCII table into dictionary $dict$. Let $w \leftarrow NULL$ // *empty string*
2. Read a character k
3. If $wk \in dict$, then $w \leftarrow wk$
   Else
   - $dict.push(wk)$
   - emit bit string (code) c where $c \leftarrow dict.getCode(w)$
   - Let $w \leftarrow k$
4. repeat step 2 until no more symbol to read

LZW - Example

Input string “AHAHA”

step	w	k	wk	output	dict
0					ASCIITab
1		A	A		ASCIITab
2	A	H	AH	<A>	ASCIITab + <AH>
3	H	A	HA	<H>	.. + <AH> + <HA>
4	A	H	AH		.. + <AH> + <HA>
5	AH	A	AHA	<AH>	.. + <AH> + <HA> + <AHA>
6	A		A	<A>	

Resulting bit stream: “$< A >< H >< AH >< A >$”

- Length of bit stream : $4 \times l(entry)$, $l(entry) > 8$
- Typical length of entry in real implementation is 12

LZW - Decoding algorithm

Decoding Algorithm: *Produce stream of ASCII symbols from input bit stream*

1. Load ASCII table into dictionary $dict$.
2. Read in a code $< k >$ (say 12 bit).
3. Emit $k \leftarrow dict.getString(< k >)$
4. Let $w \leftarrow k$
5. Read in another code $< k >$.
6. $k \leftarrow dict.getString(< k >)$
7. $dict.push(w + k[0])$
8. Let $w \leftarrow k$
9. Repeat step 5 until no more code exists

Decoding LZW - Example

Input bit stream: “$<A><H><AH><A>$”

step	w	k	output	wk	dict
0					ASCIITab
1		A	A		ASCIITab
2	A	H	H	AH	ASCIITab + $<AH>$
3	H	AH	AH	HA	.. + $<\text{AH}>$ + $<HA>$
4	AH	A	A	AHA	.. + $<\text{AH}>$ + $<\text{HA}>$ + $<AHA>$

Resulting ASCII stream: “AHAHA”

Ex-Golomb code

Input: Integer x
Output: Binary string $code(x)$
Algorithm:

$$code(x) = 0\{length(binary(x+1)) - 1\} \oplus binary(x+1)$$

Examples:

x	$binary(x+1)$	$code(x)$
0	1	1
1	10	010
2	11	011
3	100	00100
...	...	...
23	11000	000011000

Can you prove that Ex-Golomb is prefix code?

Run length Encoding

- There exists a run of symbols sometime
- Replace a run of symbols with a pair $(symbol, count)$
- We can easily design a way to encode a pair into binary form

Example: Coding a grayscale image