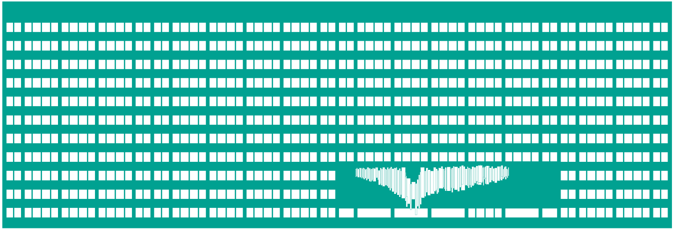
VŠB TECHNICKÁ UNIVERZITA OSTRAVA

VSB TECHNICAL UNIVERSITY OF OSTRAVA



www.vsb.cz

Statistical Compression

Part II - Arithmetic coding

Michal Vasinek

VSB – Technical University of Ostrava

name.surname@vsb.cz

May 23, 2019

VSB TECHNICAL UNIVERSITY OF OSTRAVA

Overview

- Encoding messages.
- Arithmetic Coding.
- Adaptive coding.
- Higher-order coding.
- Prediction by Partial Matching.

Encoding messages - motivation

- Consider following simple example: let length of the message be $|m| = 3$, $\Sigma = \{a, b, c\}$ and $f(a) = f(b) = f(c) = 1$.
- How many messages with frequency distribution f exists?
- There are 6 different messages: $M = \{abc, acb, bac, bca, cab, cba\}$.
- To distinguish between 6 objects we need $\log 6 = 2.58$ bits.
- To obtain number of bits per symbol => divide by the message length: $\frac{1}{3} \log 6 = 0.86$ bits.
- Empirical entropy in per symbol interpretation: $H(X) = -3\frac{1}{3} \log \frac{1}{3} = 1.58$ bits.

Question

Does it mean that we can compress below entropy if we encode messages instead of symbols?

Encoding messages - motivation

Question

Does it mean that we can compress below entropy if we encode messages instead of symbols?

- No, we cannot.
- Entropy is based on probabilities, so there are non-zero probabilities for messages: $aaa, aab, aac, \ldots$. $p(X_1X_2X_3) = p(x_1)p(x_2)p(x_3)$.
- Knowledge of frequencies means, that once we encode one symbol, the rest is encoded by adjusted frequencies:
  1. Start with $f_0(a) = f_0(b) = f_0(c) = 1$. Say the first symbol was a. Encode it by $\log 3$ bits.
  2. Adjust frequencies $f_1(a) = 0$ and $f_1(b) = f_1(c) = 1$. Encode the next symbol, say b, by $\log 2$ bits.
  3. Adjust frequencies $f_2(a) = f_2(b) = 0$ and $f_2(c) = 1$. Only one symbol left, encode using $\log 1 = 0$ bits.
  4. Together: $\log 2 + \log 3 = \log 6$ bits.

Entropy - revisited

Entropy - Definition

Number of bits needed to distinguish one message from other messages having the knowledge of symbol frequencies.

- We have a very long message of N symbols over alphabet $\Sigma = \{0, 1\}$.
- It is simple task to compute frequencies $f(0)$ and $f(1)$.

Question

How many distinct messages can we create using $f(0)$ zeros and $f(1)$ ones?

Distinct messages

Question

How many distinct messages can we create using $f(0)$ zeros and $f(1)$ ones?

Permutation with repetition:

$$\frac{N!}{f(0)!f(1)!}$$

To describe uniquely k different objects we need $\log k$ bits. The same for messages.

$$\log \frac{N!}{f(0)!f(1)!} = \log N! - log f(0)! - log f(1)!$$

Stirling approximation

Problem

How to handle logarithm of factorial?

Answer: Use Stirling approximation.

$$\log k! = k \log k - k$$

Task

Use Stirling formula to simplify the expression:

$$\log \frac{N!}{f(0)!f(1)!} = \log N! - log f(0)! - log f(1)!$$

Entropy by long messages

- $N = f(0) + f(1)$
- $p(x) = \frac{f(x)}{N}$ => $f(x) = p(x)N$

$$
\begin{aligned}
\log \frac{N!}{f(0)!f(1)!} &= \log N! - log f(0)! - log f(1)! \\
&= N \log N - N - f(0) \log f(0) + f(0) \\
&- f(1) \log f(1) + f(1) \\
&= N \log N - f(0) \log f(0) - f(1) \log f(1) \\
&= N \log N - Np(0) \log p(0)N - Np(1) \log p(1)N \\
&= N(\log N - p(0) \log p(0)N - p(1) \log p(1)N)
\end{aligned}
$$

Entropy by long messages

$$p(x)\log Np(x) = p(x)\log p(x) + p(x)\log N$$

$$-p(0)\log N - p(1)\log N = -(p(0)+p(1))\log N = -\log N$$

Taking it together:

$$\begin{aligned}
&= N(\log N - p(0)\log p(0)N - p(1)\log p(1)N) \\
&= N(-p(0)\log p(0) - p(1)\log p(1) \\
&= NH(X)
\end{aligned}$$

Since there are N symbols in the message dividing by N gives the entropy related to one symbol, i.e. Shannon's entropy.

Encoding message - Summary

- For sufficiently long messages, there is no difference between encoding per symbol or per message.
- To encode one message we have to establish mapping between messages and binary code of $c = NH(X)$ bits.

$$m_0 \to 0^c$$
$$m_1 \to 0^{c-1}1$$
$$\ldots \to \ldots$$

- The instance of such mapping: let M and B be two alphabetically ordered set of messages, where M is a set of possible input messages (knowing frequency distribution) and B is a set of output binary messages. The size of both sets is $NH(X)$. i-th message $M[i]$ is then assigned a $B[i]$ binary code.

- Principle proposed by Peter Elias in early 60s.
- Encodes message instead of symbols.
- Represents a message as a real number in $[0;1)$ interval.
- Achieves bits per symbol very close to entropy.

Arithmetic coding - example

Let $m = abac$ then $p(a) = 0.5$ and $p(b) = p(c) = 0.25$.

L	$L = 0$	$L = 0$	$L = 0.25$
H	$H = 1$	$H = 0.5$	$H = 0.375$
Symbol	a	b	a
a	[0;0.5)	[0;0.25)	[0.25;0.3125)
b	[0.5;0.75)	[0.25;0.375)	[0.3125;0.34375)
c	[0.75;1)	[0.375;0.5)	[0.34375;0.375)

L	$L = 0.25$	$L = 0.296875$	
H	$H = 0.3125$	$H = 0.3125$	
Symbol	c		
a	[0.25;0.28125)		
b	[0.28125;0.296875)		
c	[0.296875;0.3125)		

Any number from interval [0.296875;0.3125) can be used for representation of m.

Do you know how to do?

Task

Decode binary number 0.101 into decimals.

Task

Encode decimal number 0.375 into binary represenation.

Do you know how to do?

Task

Decode binary number 0.101 into decimals.

$$2^{-1} + 2^{-3} = 0.625$$

Task

Encode decimal number 0.375 into binary represenation.

$$0.011 = (0)2^{-1} + (1)2^{-2} + (1)2^{-3}$$

Example - coding floating point numbers

Objective

We want to find a shortest binary number within interval [L;H).

- The interval be [0.296875;0.3125).
- Representation:

$$b = b_0 2^{-1} + b_1 2^{-2} + b_2 2^{-3} \ldots$$

binary	interval	decimal
.0...	[0;0.5)	0
.01...	[0.25;0.5)	0.25
.010...	[0.25;0.375]	0.25
.0100...	[0.25;0.3125]	0.25
.01001...	[0.28125;0.3125)	0.28125
.010011...	[0.296875;0.3125)	0.296875

Example - summary

- Message $m = abac$ will be encoded by binary code $b = 010011$.
- $H(X) = 1.5$ bits per symbol, message will be encoded optimally by $mH(X) = 6$ bits.
- We have an optimal coding of m!

Assume message m, alphabet $\Sigma = \{x_1, x_2, \ldots, x_\sigma\}$ with frequency distribution F and let $Pr[X < x]$ be a cumulative probability.

1. Count probabilities of symbols in m. Set $L = 0$, $H = 1$ and $i = 0$.
2. Divide interval $[L; H)$ proportionaly to probabilities. To encode symbol $x = m[i]$: set

$$L_1 = L_0 + (H_0 - L_0)Pr[X < x]$$

   and

$$H_1 = L_0 + (H_0 - L_0)Pr[X \leq x]$$

3. Increment i, if there are no more symbols output number from interval $[L; H)$ so that its binary representation is the shortest one, otherwise continue with Step 2.

Example - decoding

Decoding 010011:

decoding	0	1	0
interval	[0;0.5)	[0.25;0.5)	[0.25;0.375]
output	a		b
a	[0;0.5)	[0;0.25)	[0;0.25)
b	[0.5;0.75)	[0.25;0.375)	[0.25;0.375)
c	[0.75;1)	[0.375;0.5)	[0.375;0.5)

decoded	0	1	1
interval	[0.25;0.3125]	[0.28125;0.3125)	[0.296875;0.3125)
output	a		c
a	[0.25;0.3125)	[0.25;0.28125)	[0.25;0.28125)
b	[0.3125;0.34375)	[0.28125;0.296875)	[0.28125;0.296875)
c	[0.34375;0.375)	[0.296875;0.3125)	[0.296875;0.3125)

Arithmetic coding - decoding

1. Reconstruct the initial intervals using probabilities.
2. Read bit and adjust $[L; H)$ interval.
3. If the interval is a subinterval of some initial interval then output corresponding symbol and adjust intervals for symbols.
4. Repeat step 2

Arithmetic coding - notes

- Achieves better compression rate than Huffman coding.
- Generally slower compression and decompression $O(n \log \sigma)$.
- Historically restricted by patents, even though almost all expired, not used in many comercial applications.
- JPEG supports arithmetic coding from 1990, but very few manufactures used ac coding as Huffman based one was free.
- Used in the state of the art compressors like PPMd and PAQ familly as entropy coders.

Arithmetic coding - implementation details

- The procedure we have described sofar is impractical => using standard **double** data type we can represent precisely only 14(15) decimal places.
- Any practical implementation of arithmetic coding should use integers.
- Low and High are integers, but we still dont want to let them grow too much.

Important note

Once the leftmost digits of Low and High become identical, they never change.

The leftmost identical digits of Low and High are sent to output and bit-wise shifted. Using this procedure we adjust intervals corresponding to symbols and they never grow above certain predefined level.

Arithmetic coding - integer based encoding

symbol	p(x)	interval
a	0.5	[0,0.5)
b	0.25	[0.5,0.75)
c	0.25	[0.75,1)

- Encoded using 4-bit buffer.
- Value range: $< 0, 16)$.
- Encode 'abac'.

$L = 0$	0000
$H = 15$	1111
read 'a'	
$L = 0$	0000
$H = 7$	0111
write 0, shift 1 bit	
$L = 0$	0000
$H = 15$	1111
read 'b'	
$L = 8$	1000
$H = 11$	1011
write 10, shift 2 bits	

Arithmetic coding - integer based encoding

symbol	p(x)	interval
a	0.5	[0,0.5)
b	0.25	[0.5,0.75)
c	0.25	[0.75,1)

- Encoded using 4-bit buffer.
- Value range: $< 0, 16)$.
- Encode 'abac'.
- Encoded to: 010011.

$L = 0$	0000
$H = 15$	1111
read 'a'	
$L = 0$	0000
$H = 7$	0111
write 0, shift 1 bit	
$L = 0$	0000
$H = 15$	1111
read 'c'	
$L = 12$	1100
$H = 15$	1111
write 11, shift 2 bits	

Arithmetic coding - integer based decoding

symbol	p(x)	interval
a	0.5	[0,8)
b	0.25	[8,12)
c	0.25	[12,15)

- Encoded using 4-bit buffer.
- Value range: $< 0, 16)$.
- Encode 'abac'.
- Encoded to: 010011.

$L = 0$	0000
$H = 15$	1111
read (0100)11	C=4
write 'a'	
$L = 0$	0000
$H = 7$	0111
shift 1 bit	
read 0(1001)1	C=9
write 'b'	
$L = 8$	1000
$H = 11$	1011
shit 2 bits	
$L = 0$	0000
$H = 15$	1111

Arithmetic coding - integer based decoding

symbol	p(x)	interval
a	0.5	[0,8)
b	0.25	[8,12)
c	0.25	[12,15)

- Encoded using 4-bit buffer.
- Value range: $< 0, 16)$.
- Encode 'abac'.
- Encoded to: 010011.

$L = 0$	0000
$H = 15$	1111
read 010(0110)	C=6
write 'a'	
$L = 0$	0000
$H = 7$	0111
shift 1 bit	
read 0100(1100)	C=12
write 'c'	

Arithmetic coding - implementation details

- For more discussion see Arithmetic Coding chapter in Solomon, Data Compression Handbook.
- Further implementation details: Witten, Neal, Cleary(89) `https://dl.acm.org/doi/pdf/10.1145/214762.214771`
- Real implementation of arithmetic coding based on Witten: `http://homel.vsb.cz/~vas218/source/acs/Arithmetic.h`

Arithmetic coding - adaptive version

- We don't have to use the first pass through message to count symbols.
- We can read symbol after symbol and adjust frequencies => adapt the model to the current data.
  - The model is based on all preceeding symbols.
  - The model is based on k preceeding symbols.
- Initialization
  - We know the alphabet => set the initial frequency of each symbol to 1.
  - We don't know the alphabet => define a special escape symbol η, once we process yet unseen symbol we escape it by η and encode in binary.
- No need to store the frequency model => zero order-model are usually small but higher order...

Arithmetic coding - higher order coding

- We can use any estimate of probabilities(k-th order context or nerual network) to adjust intervals and L and H.
- We only have to ensure that compressor and decompressor are synchronized, i.e. they deduce the same probability when adjusting the model.
- We can easily build k-th order adaptive arithmetic model. `http://homel.vsb.cz/~vas218/source/acs/AdaptiveArithmetic.h`
- There is no problem if k is fixed, but what if we let the k to vary?

Prediction by partial matching

- Prediction by partial matching, in short PPM.
- Uses context-based (N preceeding symbols) estimate of probability wich is feeded to arithmetic coder.
- PPM encoder is usually adaptive.
- PPM can switch to a shorter context when a longer one has resulted in 0 probability.
- PPM searches for symbol S in the context C, if it finds no occurence C, it switches to $N-1$ length context and so on.

PPM - short example

- Suppose the current order-3 context is the string "the".
- Its current frequency is 27 and it was followed by r(11 times), s(9 times), n (6 times), m (once).
- The encoder assigns these cases probabilities 11/27, 9/27, 6/27 and 1/27.
- If the next symbol is "r", then we sent "r" to adaptive arithmetic coder with probability 11/27.
- If the next symbol is "a", then PPM switches to order-2 context and try it again.
- Each context switch is represented by special escape symbol.
- If we encounter symbol for the first time, we switch context till we reach context = -1 and we will encode the symbol with probability 1/size of the alphabet.

Thank you for your attention

Michal Vasinek

VSB – Technical University of Ostrava

name.surname@vsb.cz

May 23, 2019

VSB TECHNICAL UNIVERSITY OF OSTRAVA