Information Theory and Networks

Lecture 9: Compression and Coding

Matthew Roughan
<matthew.roughan@adelaide.edu.au>
http://www.maths.adelaide.edu.au/matthew.roughan/Lecture_notes/InformationTheory/

School of Mathematical Sciences,
University of Adelaide

September 18, 2013

Part I

Compression and Coding

> He can compress the most words into the smallest ideas of any man I ever met.
>
> *Abraham Lincoln*

He can compress the most words into the smallest ideas of any man I ever met.
Abraham Lincoln

Optimal codes

- Efficiency of a code is measured by its expected length, i.e.,

$$L = E\left[\ell(X)\right] = \sum_{k=1}^{m} \ell_k p_k,$$

  where p_k is the probability of the kth symbol of Ω, and ℓ_k is the length of the kth codeword.
- Optimal codes minimise the expected length
  - more probable symbols get shorter codewords
  - can we be a bit more formal about this?

Optimal codes

Problem

Minimise

$$L = \sum p_k \ell_k$$

over all integers $\ell_1, \ell_2, \ldots, \ell_m$ satisfying the Kraft inequality

$$\sum D^{-\ell_k} \leq 1.$$

Ignoring integer constraints on ℓ_i, and assuming the bound is satisfied, then we can solve using a Lagrange multiplier, i.e., minimise

$$J = \sum p_k \ell_k + \lambda \left[\sum D^{-\ell_k} - 1 \right].$$

Optimal codes

Minimise

$$J = \sum p_k \ell_k + \lambda \left[\sum D^{-\ell_k} - 1 \right].$$

Take the derivative with respect to ℓ_k:

$$\frac{\partial J}{\partial \ell_k} = p_k - \lambda D^{-\ell_k} \log_e D = 0.$$

Hence

$$D^{-\ell_k} = \frac{p_k}{\lambda \log_e D}$$

The (equality) constraint gives

$$1 = \sum D^{-\ell_k} = \frac{\sum p_k}{\lambda \log_e D} = \frac{1}{\lambda \log_e D}$$

so $\lambda = 1/\log_e D$, and hence $D^{-\ell_k} = p_k$, and so the optimal code lengths are

$$\ell_k = -\log_D p_k$$

Optimal codes

So if we don't mind non-integer code lengths, the best we can do is to take:

$$\ell_k = -\log_D p_k$$

In this case

$$L = \sum_k p_k \ell_k = -\sum_k p_k \log_D p_k = H_D(X)$$

- In reality, the codeword lengths must be integers, so this is a lower-bound on the minimum expected codeword length.
- We will achieve the bound iff $p_k = D^{-\ell_k}$ for some integers ℓ_k

Optimal codes

Theorem

The expected length L of any prefix-free code for a random variable X is bounded below by the entropy of X, i.e.,

$$L \geq H_D(X)$$

with equality iff $D^{-\ell_i} = p_i$ for some integers ℓ_i.

Proof.

See above, or [CT91, p.86] for a more technical proof. □

Optimal codes
Theorem
The expected length L of any prefix-free code for a random variable X is bounded below by the entropy of X, i.e.,
$L \geq H_D(X)$
with equality iff $D^{-\ell_i} = p_i$ for some integers ℓ_i.
Proof.
See above, or [CT91, p.86] for a more technical proof. □

Note that $H_D(X)$ refers to Shannon entropy calculated with *log* base D for a D-ary code.

Upper Bound on Optimal Codes

Theorem

The expected length L of the optimal code for a random variable X is bounded below by the entropy of X, i.e.,

$$H_D(X) \leq L < H_D(X) + 1.$$

Proof.

Take $\ell_k^* = \lceil \log_D (1/p_k) \rceil$. These satisfy the Kraft inequality:

$$\sum D^{-\left\lceil \log_D\left(\frac{1}{p_k}\right)\right\rceil} \leq \sum D^{-\log_D\left(\frac{1}{p_k}\right)} = \sum p_k = 1.$$

And $-\log_D p_k \leq \ell_k^* < -\log_D p_k + 1$ so multiplying by p_k and summing

$$H_D(X) \leq L^* < H_D(X) + 1.$$

The optimal code can only be better than L^*, but not better than $H_D(X)$. □

Upper Bound on Optimal Codes
Theorem
The expected length L of the optimal code for a random variable X is bounded below by the entropy of X, i.e.,
$H_D(X) \leq L < H_D(X) + 1.$
Proof.
Take $\ell_k^* = \lceil \log_D (1/p_k) \rceil$. These satisfy the Kraft inequality:
$\sum D^{-\left\lceil \log_D\left(\frac{1}{p_k}\right)\right\rceil} \leq \sum D^{-\log_D\left(\frac{1}{p_k}\right)} = \sum p_k = 1.$
And $-\log_D p_k \leq \ell_k^* < -\log_D p_k + 1$ so multiplying by p_k and summing
$H_D(X) \leq L^* < H_D(X) + 1.$
The optimal code can only be better than L^*, but not better than $H_D(X)$. □

Remember the ceiling $\lceil x \rceil$ is the smallest integer $\geq x$.

Shannon Code

Call the code with

$$\ell_k^* = \left\lceil \log_D \left(\frac{1}{p_k} \right) \right\rceil$$

a Shannon Code.

- Its seems (from previous slide) like this might be a good choice
- Counter-example: $D = 2$ (binary code)

$$\begin{aligned} p_0 &= 0.999 \quad \leftrightarrow \quad \ell_0 = 1 \\ p_1 &= 0.001 \quad \leftrightarrow \quad \ell_1 = 10 \end{aligned}$$

So we might choose the binary code 0 and 1000000000, but we know that 0 and 10 would be better.

Is it true that the codeword lengths for an optimal code are always less than the Shannon codeword lengths? (see [CT91, p.96]).

Practical optimal coding

- So now we have some idea of the best we can theoretically do
- How close can we get, in practise?
- Is there a reasonable procedure for getting there?
- Huffman coding!

Binary Huffman Coding

1. We are building a tree
2. Start with each symbol in Ω as a leaf of the tree.
3. Repeat rule
   1. merge the two current nodes with the lowest probability masses to get a new node of the tree
4. The root is when we get a probability 1.

Binary Huffman Coding
1. We are building a tree
2. Start with each symbol in Ω as a leaf of the tree.
3. Repeat rule
   1. merge the two current nodes with the lowest probability masses to get a new node of the tree
4. The root is when we get a probability 1.

For simplicity look at binary case, but can be generalised.

In general there may be more than one optimal code for a source, and some codes can't be constructed with the Huffman procedure.

Huffman coding example 1 [CT91, p.93]

X	Probability
a	0.25
b	0.25
c	0.2
d	0.15
e	0.15

Huffman coding example 1 [CT91, p.93]

X	Probability
a	0.25 → 0.25
b	0.25 → 0.25
c	0.2 → 0.2
d	0.15 → 0.3
e	0.15

Huffman coding example 1 [CT91, p.93]

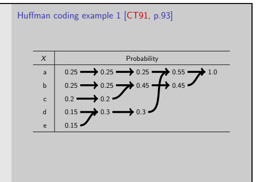
X	Probability		
a	0.25	0.25	0.25
b	0.25	0.25	0.45
c	0.2	0.2	
d	0.15	0.3	0.3
e	0.15		

Huffman coding example 1 [CT91, p.93]

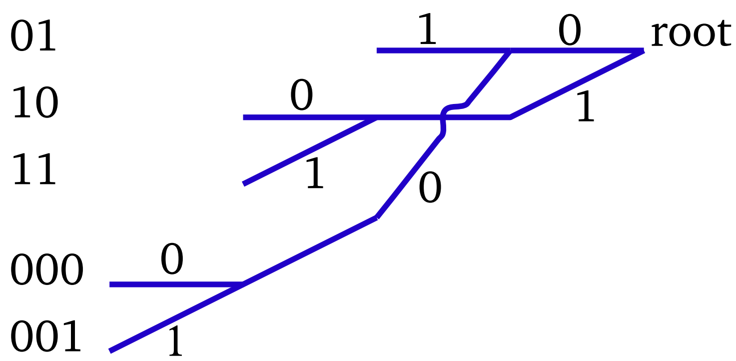
X	Probability			
a	0.25	0.25	0.25	0.55
b	0.25	0.25	0.45	0.45
c	0.2	0.2		
d	0.15	0.3	0.3	
e	0.15			

Huffman coding example 1 [CT91, p.93]

X	Probability				
a	0.25	0.25	0.25	0.55	1.0
b	0.25	0.25	0.45	0.45	
c	0.2	0.2			
d	0.15	0.3	0.3		
e	0.15				



Huffman coding example 1

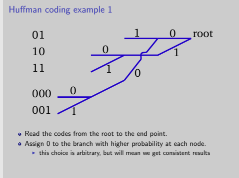


- Read the codes from the root to the end point.
- Assign 0 to the branch with higher probability at each node.
  - this choice is arbitrary, but will mean we get consistent results

Information Theory

2013-09-18

└─Huffman coding example 1



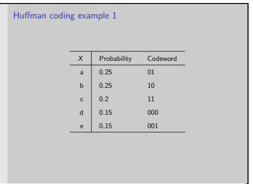
Huffman coding example 1

X	Probability	Codeword
a	0.25	01
b	0.25	10
c	0.2	11
d	0.15	000
e	0.15	001

Information Theory

2013-09-18

└─Huffman coding example 1



Huffman coding example 2 [Yeu10, p.89]

X	Probability	Code
a	0.35	
b	0.1	
c	0.15	
d	0.2	
e	0.2	

Huffman coding example 2 [Yeu10, p.89]

X	Probability	Code
a	0.35 $\rightarrow$ 0.35	
b	0.1 $\xrightarrow{0}$ 0.25	
c	0.15 1	
d	0.2 $\rightarrow$ 0.2	
e	0.2 $\rightarrow$ 0.2	

Huffman coding example 2 [Yeu10, p.89]

X	Probability	Code
a	0.35 $\rightarrow$ 0.35 $\rightarrow$ 0.35	
b	0.1 $\xrightarrow{0}$ 0.25 $\rightarrow$ 0.25	
c	0.15 1	
d	0.2 $\rightarrow$ 0.2 $\xrightarrow{0}$ 0.4	
e	0.2 $\rightarrow$ 0.2 1	

Huffman coding example 2 [Yeu10, p.89]

X	Probability	Code
a	0.35 $\rightarrow$ 0.35 $\rightarrow$ 0.35 $\xrightarrow{0}$ 0.6	
b	0.1 $\xrightarrow{0}$ 0.25 $\rightarrow$ 0.25 1	
c	0.15 1	
d	0.2 $\rightarrow$ 0.2 $\xrightarrow{0}$ 0.4 $\rightarrow$ 0.4	
e	0.2 $\rightarrow$ 0.2 1	

Huffman coding example 2 [Yeu10, p.89]

X	Probability	Code
a	0.35 → 0.35 → 0.35 → 0.6 → 1.0	
b	0.1 → 0.25 → 0.25	
c	0.15	
d	0.2 → 0.2 → 0.4 → 0.4	
e	0.2 → 0.2	

Huffman coding example 2 [Yeu10, p.89]

X	Probability	Code
a	0.35 → 0.35 → 0.35 → 0.6 → 1.0	00
b	0.1 → 0.25 → 0.25	010
c	0.15	011
d	0.2 → 0.2 → 0.4 → 0.4	10
e	0.2 → 0.2	11

Huffman coding example 3 [CT91, p.93]

X	Probability	Code
a	0.25	
b	0.25	
c	0.2	
d	0.15	
e	0.15	

Huffman coding example 3 [CT91, p.93]

X	Probability	Code
a	0.25 → 0.25	
b	0.25 → 0.25	
c	0.2 → 0.5	
d	0.15	
e	0.15	

Huffman coding example 3 [CT91, p.93]

X	Probability			Code
a	0.25	0.25	1.0	
b	0.25	0.25		
c	0.2	0.5		
d	0.15			
e	0.15			

Huffman coding example 3 [CT91, p.93]

X	Probability			Code
a	0.25	0.25	1.0	1
b	0.25	0.25		2
c	0.2	0.5		00
d	0.15			01
e	0.15			02

Examples

Example	D	$H(X)$	$H_D(X)$	Average Code length
1	2	2.285	2.285	2.300
2	2	2.202	2.202	2.250
3	3	2.285	1.442	1.500

We can see that Huffman coding is doing a pretty good job of finding a code that is close to the optimal code (the entropy). Is that a general rule?

Huffman coding optimality

Theorem

Huffman coding is optimal (in the sense that the expected length of its codewords is at least as good as any other code).

To do the proof, we need a couple of lemmas: based on the proof in [Yeu10, pp.90-92].

We'll do the proof for binary codes, but it is obviously extendable to D-ary codes.

Huffman coding optimality

Lemma

In an optimal code, shorter codewords are assigned to larger probabilities.

Proof.

Consider $1 \leq i < j \leq m$ such that $p_i > p_j$. Assume that in a code, the codewords c_i and c_j are such that $\ell_i > \ell_j$, i.e., a shorter codeword is assigned to a smaller probability. then by exchanging c_i and c_j, the expected length of the code is changed by

$$(p_i\ell_j + p_j\ell_i) - (p_i\ell_i + p_j\ell_j) = (p_i - p_j)(\ell_j - \ell_i) < 0,$$

since $p_i > p_j$ and $\ell_i > \ell_j$. In other words, the code can be improved and therefore is not optimal. □

Note that WLOG we order the probabilities so that $p_1 \geq p_2 \geq \cdots \geq p_m$.

Huffman coding optimality

Lemma

There exists an optimal code in which the codewords assigned to the two smallest probabilities are siblings (in the code tree), i.e., the two codewords have the same length and they differ only in the last symbol.

Proof.

From the last lemma, the codeword c_m assigned to p_m is the longest. Note also that the sibling of c_m cannot be a prefix for any other code (as then that code would be longer).

We claim that the sibling of c_m must be a codeword. To see this, assume it is not. Then replace c_m by its parent to improve the code, because the length of codeword would be reduced by 1, while all the other codewords remain unchanged. Hence the sibling must be a codeword.

If the sibling of c_m is assigned to p_{m-1} then the code has the desired property. If it isn't, then we can perform a swap (as above) that won't increase the average code lengths so that it is. $\square$

Note that WLOG we order the probabilities so that $p_1 \geq p_2 \geq \cdots \geq p_m$.

Huffman coding optimality

Lemma

If we merge two siblings in a code tree, i.e., we replace the codes c_i and c_j of the siblings by a common parent (call it c_{ij}), then we obtain a reduced code tree, and the probability of c_{ij} is $p_i + p_j$, and the expected length of the reduced code L' is related to that of the original code L by

$$L' = L - (p_i + p_j)$$

Proof.

Everything remains the same, except the codes c_i and c_j are replaced by one code c_{ij}, which has length 1 less than the two original codes, so the difference in expected lengths is

$$\begin{aligned} L - L' &= (p_i\ell_i + p_j\ell_j) - (p_i + p_j)(\ell_i - 1) \\ &= (p_i\ell_i + p_j\ell_i) - (p_i + p_j)(\ell_i - 1) = p_i + p_j \end{aligned}$$

(as $\ell_i = \ell_j$ because c_i and c_j are siblings). $\square$

Huffman coding optimality

Theorem

Huffman coding is optimal (in the sense that the expected length of its codewords is at least as good as any other code).

Proof.

The lemma above states that optimal code in which c_m and c_{m-1} are siblings exists. Let p_i' be the new PMF we get by merging p_{m-1} and p_m.

The previous lemma shows that there is a fixed relationship between the expected lengths of the two codes, and so minimising the length of L is equivalent to minimising the length of L'. So if one is minimal so too is the other.

We can then repeat the merging procedure until we reach the root of the tree, for which the optimal code is obvious. □

Slice coding

Definition (Slice Code)

A slice code or an alphabetic code is one where the lexicographic (alphabetic) ordering of the codes corresponds to the ordering of the probabilities (in descending order).

- Huffman codes may not be slice codes
- If we take the lengths of the Huffman codes, we can generate an equivalent slice code.

For example: the code generate in Example 1 is not a slice code, but the following table gives one.

X	Probability	Huffman	Slice
a	0.25	01	00
b	0.25	10	01
c	0.2	11	10
d	0.15	000	110
e	0.15	001	111

Block encoding

- We can see that there is at least a small loss of efficiency for codes, when we don't have natural integer length codes.
- This can actually be quite a big cost, in terms of optimality
  - in binary codes its up to one bit per symbol
- We can spread the overhead out by coding blocks of symbols at a time
  - next week we'll look at this

Block encoding
- We can see that there is at least a small loss of efficiency for codes, when we don't have natural integer length codes.
- This can actually be quite a big cost, in terms of optimality
  - in binary codes its up to one bit per symbol
- We can spread the overhead out by coding blocks of symbols at a time
  - next week we'll look at this

Assignment

Consider JPEG encoding of an image

1. Discrete Cosine Transform (DCT) of blocks (usually 8×8)
2. Quantise (round off)
3. Huffman coding

Take a particular (black and white) image. Compute the DCT of each 8×8 block of the image, and then quantise the coefficients to be $16, 12, 8, 4, 2$, and 1 bit integers, and calculate the entropy of the resulting coefficients:

- as a function of the quantisation level; and
- for 4 bit quantisation, as a function of the position in the block.

Compare these to quantisation of an equivalent set of uniformly chosen random numbers.
Consider the implications for the way JPEG might perform quantisation and coding, and write up 1-2 pages on your results.

Further reading I

Thomas M. Cover and Joy A. Thomas, *Elements of information theory*, John Wiley and Sons, 1991.

Raymond W. Yeung, *Information theory and network coding*, Springer, 2010.