lecture 22

data compression

http://www.cim.mcgill.ca/~langer/423.html

- optimal prefix codes
- Huffman coding
- run length coding

Information Theory

http://en.wikipedia.org/wiki/Information_theory

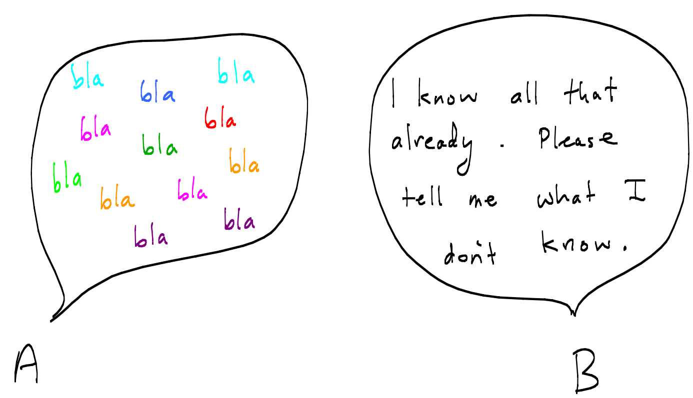
"A Mathematical Theory of Communication"
[C. Shannon, 1948]

very readable, available online

Information Theory - basic idea

- When A communicates a message to B, A sends a bit string that encodes the message.
- The amount of information in the message depends not on the number of bits sent, but rather on the probability of that message being sent.

How much information does a message transmit?



Data Compression

- when A communicates to B, they first need to agree on a code.
- They choose a code such that messages that are more likely to be sent are encoded using fewer bits. This yields shorter messages on average.
- the length of the message should be ~ equal to the amount of information communicated. (Shannon clarified what that means)

Codes and Codewords

Suppose you have a sample space S. (S is often called an "alphabet".)

Define a code to be a mapping

$$C : S \longrightarrow \{\text{bit strings}\}$$

For any $s \in S$, $C(s)$ is the codeword of s.

The length of a codeword is the number of bits in that codeword.

Extension of a Code

For any code C on an alphabet S, we have a naturally defined code on sequences of elements from S.

e.g. $C(a_1\, a_4\, a_3\, a_3)$

$= c(a_1)\, c(a_4)\, c(a_3)\, c(a_3)$

i.e. concatenate the codewords of the elements of the sequence

Fixed length code:

all code words have same length

eg. $S = \{a_1, a_2, a_3\}$

$c(a_1) = 00,\quad c(a_2) = 10,\quad c(a_3) = 11$

e.g. ASCII (8 bits), unicode (16 bits)

Variable length code:

Code words can have different lengths

e.g. Morse Code

A •−
B −•••
C −•−•
D −••
E •
F ••−•
G −−•
H ••••
I ••
J •−−−
K −•−
L •−••
M −−
N −•
O −−−
P •−−•
Q −−•−
R •−•
S •••
T −
U ••−
V •••−
W •−−
X −••−
Y −•−−
Z −−••

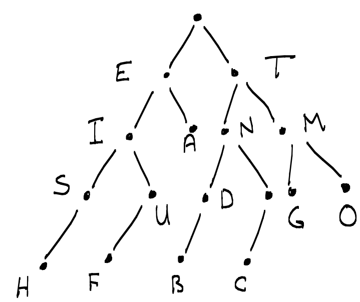
• dot (0)
− dash (1)

Note: More common letters have shorter codewords.

Any code can represented using a binary tree. Each codeword is a path from root to a node.
(0 for left child, 1 for right child)

Example:

A •−
B −•••
C −•−•
D −••
E •
F ••−•
G −−•
H ••••
I ••
J •−−−
K −•−
L •−••
M −−
N −•
O −−−
P •−−•
Q −−•−
R •−•
S •••
T −
U ••−
V •••−
W •−−
X −••−
Y −•−−
Z −−••



etc.

One big problem with Morse Code is that messages are ambiguous.

e.g. $c(A) = \bullet -$

$c(E) = \bullet$

$c(T) = -$

Q: How to distinguish $c(A)$ from $c(E)c(T)$?

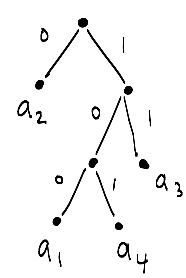
A: Morse Code inserts "space" between the codewords. So, its not really a binary code.

Prefix Code

C is a prefix code if no codeword is a prefix of any other codeword.

$\Rightarrow$ codewords are leaves in binary tree.

e.g.



$c(a_2) = 0$

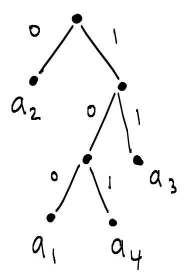
$c(a_1) = 100$

$c(a_4) = 101$

$c(a_3) = 11$

Prefix codes avoid the sort of ambiguities that we saw with Morse code. How? Suppose B is sent a sequence of bits and B wants to decode this sequence. B wants to know the sequence of symbols that was encoded. If the code is a prefix code, then there is a simple method for decoding. Repeatedly traverse the binary tree from root to leaf. Each time B reaches a leaf, it reads off the symbol at the leaf, then returns to the root.

e.g



Q: how to decode

0 0 1 0 1 1 1 0 ?

A: $a_2 \, a_2 \, a_4 \, a_3 \, a_2$

Average Code Length

[really, it should be called "expected codeword length"]

Given S, c, p, define:

$$\underbrace{\bar{\lambda}}_{\text{expected value of codeword length}} \equiv \sum_{s \in S} p(s) \underbrace{\lambda(s)}_{\text{length of codeword } c(s)}$$

Optimal Prefix Codes

Given S, $p()$, how to choose a prefix code that minimizes the average code length?

Such a code is called an "optimal prefix code."

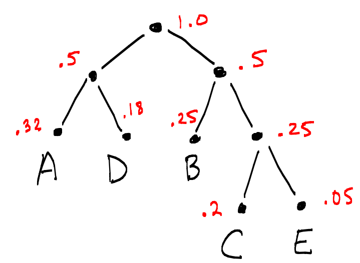
Soon we will see a simple greedy algorithm, for building an optimal prefix code, called Huffman coding.

Possible Approach (which surprisingly does not always yield optimal prefix code):

Recursively partition S into two subsets such that each split divides the probability in half as closely as possible.

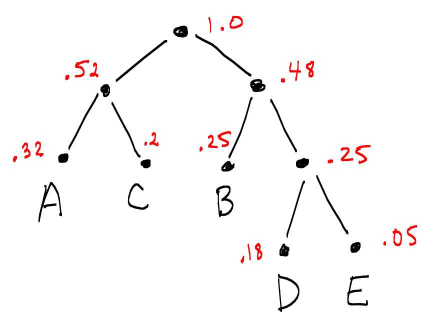
Example:

eg. $p(A) = .32$
$p(B) = .25$
$p(C) = .2$
$p(D) = .18$
$p(E) = .05$



Q: How could you improve the code i.e. reduce the average code length?

A: Swap the codewords of C and D. This gives:

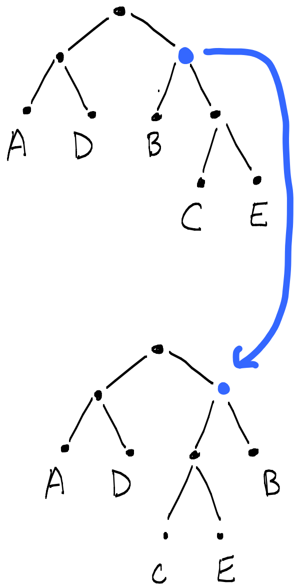


I will next sketch out some basic properties of optimal prefix codes that motivate Huffman's algorithm for constructing an optimal prefix code.

Exercises will fill in some of the details of proofs.

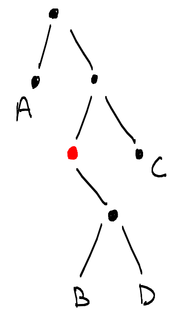
Claim: For any optimal prefix code, you can generate many other optimal prefix codes by swapping left and right children.

i.e. This doesn't change the average code length.



note: code word of B changes from 10 to 11.
Codewords of C and E change also.

Claim: for an optimal prefix code, every internal node of the tree has two children.

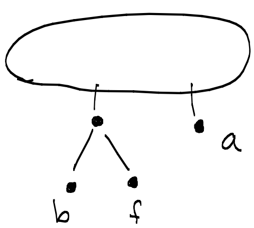


e.g. This tree cannot be optimal because we could get a code with a smaller average code length by decreasing the lengths of B's & D's codewords by 1.

Claim: For any optimal prefix code C, and any two $a, b \in S$, if $p(a) < p(b)$ then $\lambda(a) \geq \lambda(b)$

Proof: by contradiction: [See Exercises Q5.]

Suppose $\lambda(a) < \lambda(b)$.



Then, swapping $c(a)$ and $c(b)$ reduces $\bar{\lambda}$.

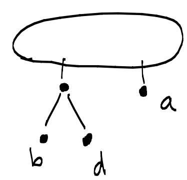
Hence C was not optimal.

(We saw an example earlier)

Claim: for any optimal prefix code, the two least probable elements have the same codeword length.

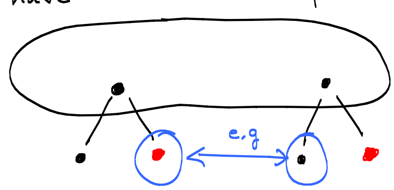
Proof: Otherwise, we would contradict the result on the previous two slides.

Why? Let a, b have smallest probability. and let $\lambda(a) < \lambda(b)$.



Then b has a sibling d (why?) and we could reduce the average code length by swapping a with d.

Claim: there exists an optimal prefix code in which the two least probable elements of S have the same parent in the tree.



Proof:

The two least probable elements have the same codeword length, so they are at the same level in the tree. If we swap codewords of the same length, we don't change the average code length.

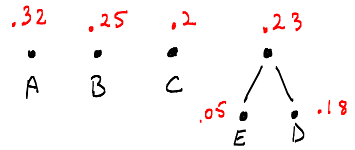
The last claim gives an algorithm for finding an optimal prefix code, called Huffman coding (1954).

I will illustrate the algorithm on an example.

e.g. $p(A) = .32$
$p(B) = .25$
$p(C) = .2$
$p(D) = .18$
$p(E) = .05$

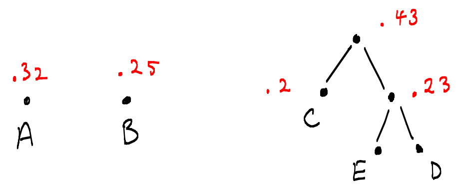
From the above claims, D and E must have the same code word length. We can make them siblings in the tree.

Now we have a problem instance with $n=4$ events.



$p(A) = .32$
$p(B) = .25$
$p(C) = .2$
$p(\{D,E\}) = .23$

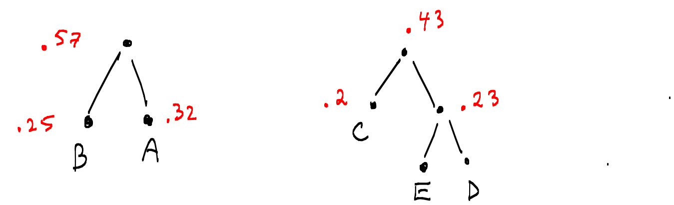
The two least probable events are C, $\{D,E\}$
So we make these events siblings.



$p(A) = .32$
$p(B) = .25$
$p(\{C,D,E\}) = .43$

Now we have $n=3$. The two least probable events are A, B.

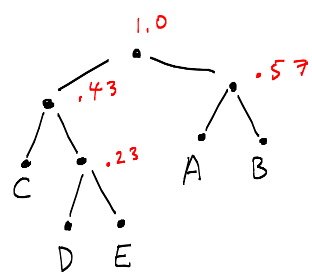
So we make them siblings, giving:



Now we have $n=2$.

$p(\{A,B\}) = .57$
$p(\{C,D,E\}) = .43$

We make these two events siblings giving an optimal prefix code.



Huffman Coding Algorithm (Greedy)

make a forest of n trees, each with one element

while number of trees is greater than 1 {

- find two trees with lowest probability (at the root)
- merge them into a new tree whose probability is the sum

// to define a unique solution, put lower probability tree in left child

}

Lower bound on average code length

Given S and p, what can we say about the average code length of an optimal prefix code, i.e. without building the code?

Entropy

$$H \equiv \sum_{s \in S} p(s) \log \frac{1}{p(s)}$$

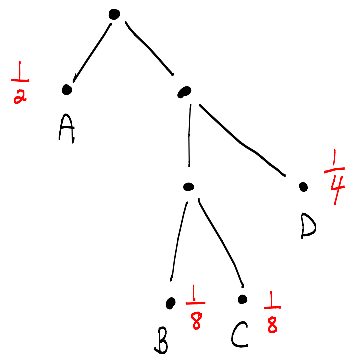
One can show (beyond this course) that, for any prefix code,

average code length $\bar{\lambda} \geq H$

i.e. fundamental lower bound

Example:

$\bar{\lambda} = H$



$\bar{\lambda} = \frac{1}{2} \cdot 1 + \frac{1}{4} \cdot 2 + \frac{1}{8} \cdot 3 + \frac{1}{8} \cdot 3$

$= \frac{14}{8}$

One can show:

$$H \equiv \sum_{s \in S} p(s) \log \frac{1}{p(s)}$$

$$\bar{\lambda} \equiv \sum_{s \in S} p(s)\, \lambda(s)$$

$H = \bar{\lambda}$ if and only if $\lambda(s) = \log \frac{1}{p(s)}$ for all $s \in S$.

That is, $p(s) = 2^{-\lambda(s)}$ for all s.

"Run length" Coding

Flip an unfair coin repeatedly.
i.e. "Bernoulli trials."

Let X be number of coin flips until we get heads.

Recall $\Pr\{X = i\} = p_0^{\,i-1} \cdot (1 - p_0)$

Q: What would be a good code for X?

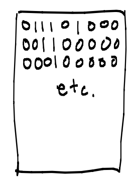
Note: $X \in \{1, 2, 3, 4, 5, 6, \ldots\}$.
so Huffman coding cannot be used.

Applications of Run Length Coding

http://www.cim.mcgill.ca/~langer/423/lecture10.pdf

eg. Encoding binary images:

document scanned by "fax machine"



You tend to get clusters of 0's or 1's.
Lots of technical details ... basic idea is that fax machines encode "run lengths".

Example:

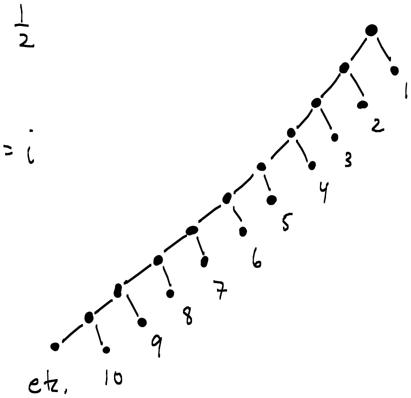
$$X \to \underbrace{0001}_{4}\ \underbrace{01}_{2}\ \underbrace{00001}_{5}\ \underbrace{01}_{2}\ \underbrace{01}_{2}\ \underbrace{1}_{1}\ \ldots$$

We want a code for positive integers such that $\lambda(i) \approx \log \frac{1}{p(i)}$ for all i.

Example 1

$p_0 = \frac{1}{2}$

$\lambda(i) = \log \frac{1}{(\frac{1}{2})^i} = i$



If $p_0 = \frac{1}{2}$, then $\lambda(i) = i$ so we would just use the original sequence.

If $p_0 \neq \frac{1}{2}$, we would like

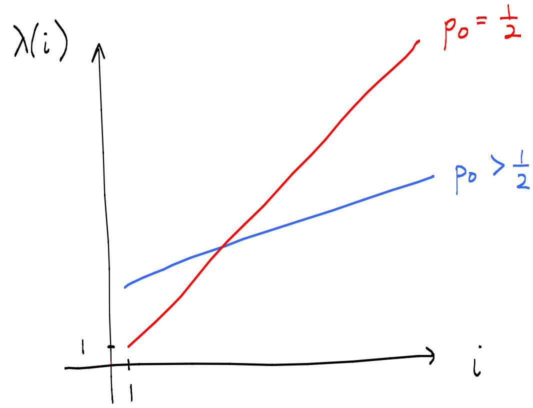
$$\lambda(i) \approx \log \frac{1}{p(i)}$$

Since $p(i) = p_0^{\,i-1}(1-p_0)$,

we would like

$$\boxed{\lambda(i) \approx \log \frac{1}{(1-p_0)} + (i-1)\log\frac{1}{p_0}}\,.$$

$$\lambda(i) \approx \log \frac{1}{(1-p_0)} + (i-1)\log\frac{1}{p_0}$$



Exercise: we ignore case $p_0 < \frac{1}{2}$. Why?

Example 2

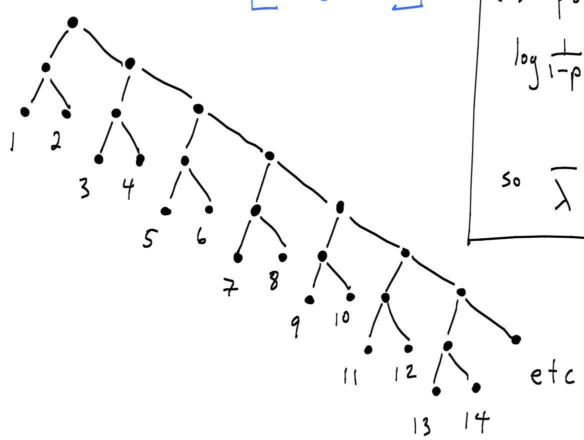
$$\lambda(i) = 2 + \left\lfloor \frac{i-1}{2} \right\rfloor$$

$\log \frac{1}{p_0} = \frac{1}{2}$

$\Leftrightarrow \; p_0 = \frac{1}{\sqrt{2}} \approx .7$

$\log \frac{1}{1-p_0} = 1.77$
$\neq 2$

so $\bar{\lambda} \neq H$



Example 3

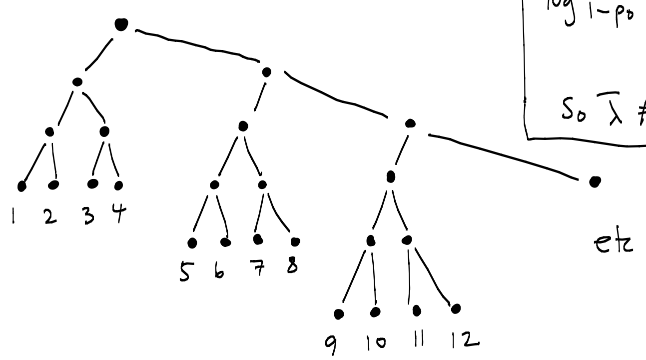
$$\lambda(i) = 3 + \left\lfloor \frac{i-1}{4} \right\rfloor$$

$\log \frac{1}{p_0} = \frac{1}{4}$

$\Leftrightarrow \; p_0 = \frac{1}{\sqrt{\sqrt{2}}} \approx .84$

$\log \frac{1}{1-p_0} = 2.65$
$\neq 3$

So $\bar{\lambda} \neq H$



Announcements

- no more lectures
- next week 9-5 office hours (with a few exceptions that I'll post on mycourses announcements)
- See end of lecture 20 for discussion of final exam
- Course evaluations (if at least 2/3 respond then I will post comments in mycourses)