Chapter 25

Compression, Information and Entropy – Huffman’s coding

NEW CS 473: Theory II, Fall 2015
December 1, 2015

25.1 Huffman coding

25.2 start

25.2.0.1 Codes...

(A) Σ: alphabet.
(B) binary code: assigns a string of 0s and 1s to each character in the alphabet.
(C) each symbol in input = a codeword over some other alphabet.
(D) Useful for transmitting messages over a wire: only 0/1.
(E) receiver gets a binary stream of bits...
(F) ... decode the message sent.
(G) prefix code: reading a prefix of the input binary string uniquely match it to a code word.
(H) ... continuing to decipher the rest of the stream.
(I) binary/prefix code is prefix-free if no code is a prefix of any other.
(J) ASCII and Unicode’s UTF-8 are both prefix-free binary codes.

25.2.0.2 Codes...

(A) Morse code is binary+prefix code but not prefix-free.
(B) ... code for S (· · ·) includes the code for E (·) as a prefix.
(C) Prefix codes are binary trees...

(D) ...characters in leafs, code word is path from root.
(E) prefix treestree! prefix tree or code trees.
(F) Decoding/encoding is easy.

25.2.0.3 Codes...

(A) Encoding: given frequency table:
\[ f[1 \ldots n]. \]
(B) \( f[i] \): frequency of \( i \)th character.
(C) code(i): binary string for \( i \)th character.
len(s): length (in bits) of binary string \( s \).
(D) Compute tree \( \mathcal{T} \) that minimizes
\[
\text{cost}(\mathcal{T}) = \sum_{i=1}^{n} f[i] \times \text{len(code}(i)\), \tag{25.1}
\]

25.2.1 Frequency table for...

25.2.1.1 “A tale of two cities” by Dickens

\[
\begin{array}{|c|c|c|c|c|c|}
\hline
\text{char} & n & \text{freq} & \text{code} & \text{freq} & \text{code} \\
\hline
\text{'A'} & 48165 & 1110 & \text{'N'} & 42380 & 1100 \\
\text{'B'} & 8414 & 101000 & \text{'O'} & 46499 & 1101 \\
\text{'C'} & 13896 & 00100 & \text{'P'} & 9957 & 101001 \\
\text{'D'} & 28041 & 0011 & \text{'Q'} & 667 & 1111011001 \\
\text{'E'} & 74809 & 011 & \text{'R'} & 37187 & 00101 \\
\text{'F'} & 13559 & 111111 & \text{'S'} & 37575 & 1000 \\
\text{'G'} & 12530 & 111110 & \text{'T'} & 54024 & 000 \\
\text{'H'} & 38961 & 1001 & \text{'U'} & 16726 & 01001 \\
\text{'I'} & 41005 & 1011 & \text{'V'} & 5199 & 1111010 \\
\text{'J'} & 710 & 1111011010 & \text{'W'} & 14113 & 00101 \\
\text{'K'} & 4782 & 11110111 & \text{'X'} & 724 & 1111011011 \\
\text{'L'} & 22030 & 10101 & \text{'Y'} & 12177 & 111100 \\
\text{'M'} & 15298 & 01000 & \text{'Z'} & 215 & 1111011000 \\
\hline
\end{array}
\]
25.2.2 The Huffman tree generating the code

25.2.2.1 Build only on A-Z for clarity.

25.2.2.2 Mergeability of code trees

(A) two trees for some disjoint parts of the alphabet...
(B) Merge into larger tree by creating a new node and hanging the trees from this common node.

(C) \[ \begin{array}{c} M \ U \\ \Rightarrow \end{array} \begin{array}{c} M \\ U \end{array} \]

(D) ...put together two subtrees.

\[ \begin{array}{c} A \\ \Rightarrow \end{array} \begin{array}{c} A \\ B \end{array} \]

25.2.3 The algorithm to build Hoffman’s code

25.2.3.1 Building optimal prefix code trees

(A) take two least frequent characters in frequency table...
(B) ... merge them into a tree, and put the root of merged tree back into table.
(C) ...instead of the two old trees.
(D) Algorithm stops when there is a single tree.
(E) Intuition: infrequent characters participate in a large number of merges. Long code words.
(F) Algorithm is due to David Huffman (1952).
(G) Resulting code is best one can do.
(H) **Huffman coding**: building block used by numerous other compression algorithms.

25.2.4 Analysis

25.2.4.1 Lemma: lowest leaves are siblings...

**Lemma 25.2.1.** (A) \( \mathcal{T} \): optimal code tree (prefix free!).
(B) Then \( \mathcal{T} \) is a full binary tree.
(C) ... every node of \( T \) has either 0 or 2 children.
(D) If height of \( T \) is \( d \), then there are leafs nodes of height \( d \) that are sibling.

25.2.4.2 Proof...
(A) If \( \exists \) internal node \( v \in V(T) \) with single child...
...remove it.
(B) New code tree is better compressor: \( \text{cost}(T) = \sum_{i=1}^{n} f[i] \times \text{len(code}(i)) \).
(C) \( u \): leaf \( u \) with maximum depth \( d \) in \( T \). Consider parent \( v = p(u) \).
(D) \( v \): has two children, both leaves

25.2.4.3 Infrequent characters are stuck together...

Lemma 25.2.2. \( x, y \): two least frequent characters (breaking ties arbitrarily).
\( \exists \) optimal code tree in which \( x \) and \( y \) are siblings.

25.2.4.4 Proof...
(A) Claim: \( \exists \) optimal code s.t. \( x \) and \( y \) are siblings + deepest.
(B) \( T \): optimal code tree with depth \( d \).
(C) By lemma... \( T \) has two leafs at depth \( d \) that are siblings,
(D) If not \( x \) and \( y \), but some other characters \( \alpha \) and \( \beta \).
(E) \( T' \): swap \( x \) and \( \alpha \).
(F) \( x \) depth inc by \( \Delta \), and depth of \( \alpha \) decreases by \( \Delta \).
(G) \( \text{cost}(T') = \text{cost}(T) - (f[\alpha] - f[x])\Delta \).
(H) \( x \): one of the two least frequent characters.
...but \( \alpha \) is not.
(I) \( f[\alpha] \leq f[x] \).
(J) Swapping \( x \) and \( \alpha \) does not increase cost.
(K) \( T \): optimal code tree, swapping \( x \) and \( \alpha \) does not decrease cost.
(L) \( T' \) is also an optimal code tree
(M) Must be that \( f[\alpha] = f[x] \).

25.2.4.5 Proof continued...
(A) \( y \): second least frequent character.
(B) \( \beta \): lowest leaf in tree. Sibling to \( x \).
(C) Swapping \( y \) and \( \beta \) must give yet another optimal code tree.
(D) Final opt code tree, \( x, y \) are max-depth siblings.

25.2.4.6 Huffman’s codes are optimal

Theorem 25.2.3. Huffman codes are optimal prefix-free binary codes.

25.2.4.7 Proof...
(A) If message has 1 or 2 diff characters, then theorem easy.
(B) \( f[1...n] \) be original input frequencies.
(C) Assume \( f[1] \) and \( f[2] \) are the two smallest.
(E) lemma $\implies \exists$ opt. code tree $T_{\text{opt}}$ for $f[1..n]$
(F) $T_{\text{opt}}$ has 1 and 2 as siblings.
(G) Remove 1 and 2 from $T_{\text{opt}}$.
(H) $T_{\text{opt}}$: Remaining tree has 3, …, $n$ as leafs and “special” character $n + 1$ (i.e., parent 1, 2 in $T_{\text{opt}}$).

25.2.4.8 La proof continued...

(A) character $n + 1$: has frequency $f[n + 1]$.

Now, $f[n + 1] = f[1] + f[2]$, we have

$$\text{cost}(T_{\text{opt}}) = \sum_{i=1}^{n} f[i] \text{depth}_{T_{\text{opt}}}(i)$$

$$= \sum_{i=3}^{n+1} f[i] \text{depth}_{T_{\text{opt}}}(i) + f[1] \text{depth}_{T_{\text{opt}}}(1)$$
$$+ f[2] \text{depth}_{T_{\text{opt}}}(2) - f[n + 1] \text{depth}_{T_{\text{opt}}}(n + 1)$$

$$= \text{cost}(T'_{\text{opt}}) + (f[1] + f[2]) \text{depth}(T_{\text{opt}})$$
$$- (f[1] + f[2]) (\text{depth}(T_{\text{opt}}) - 1)$$

$$= \text{cost}(T'_{\text{opt}}) + f[1] + f[2].$$

25.2.4.9 La proof continued...

(A) implies min cost of $T_{\text{opt}} \equiv \min \text{cost } T'_{\text{opt}}$.
(B) $T'_{\text{opt}}$: must be optimal coding tree for $f[3..n + 1]$.
(C) $T'_{H}$: Huffman tree for $f[3, \ldots, n + 1]$

$T_{H}$: overall Huffman tree constructed for $f[1, \ldots, n]$.
(D) By construction:

$T'_{H}$ formed by removing leafs 1 and 2 from $T_{H}$.
(E) By induction:

Huffman tree generated for $f[3, \ldots, n + 1]$ is optimal.
(F) $\text{cost}(T'_{\text{opt}}) = \text{cost}(T'_{H})$.
(H) $\implies$ Huffman tree has the same cost as the optimal tree.

25.2.5 What do we get

25.2.5.1 What we get...

(A) A tale of two cities: 779,940 bytes.
(B) using above Huffman compression results in a compression to a file of size 439,688 bytes.
(C) Ignoring space to store tree.
(D) gzip: 301,295 bytes

bzip2: 220,156 bytes!
(E) Huffman encoder can be easily written in a few hours of work!
(F) All later compressors use it as a black box...
25.2.6 A formula for the average size of a code word

25.2.6.1 Average size of code word

(A) input is made out of \( n \) characters.
(B) \( p_i \): fraction of input that is \( i \)th char (probability).
(C) use probabilities to build Huffman tree.
(D) Q: What is the length of the codewords assigned to characters as function of probabilities?
(E) special case...

25.2.7 Average length of codewords...

25.2.7.1 Special case

Lemma 25.2.4. \( 1, \ldots, n \): symbols.
Assume, for \( i = 1, \ldots, n \):
(A) \( p_i = 1/2^i \): probability for the \( i \)th symbol
(B) \( l_i \geq 0 \): integer.
Then, in Huffman coding for this input, the code for \( i \) is of length \( l_i \).

25.2.7.2 Proof

(A) induction of the Huffman algorithm.
(B) \( n = 2 \): claim holds since there are only two characters with probability 1/2.
(C) Let \( i \) and \( j \) be the two characters with lowest probability.
(D) Must be \( p_i = p_j \) (otherwise, \( \sum_p p_k \neq 1 \)).
(E) Huffman’s tree merges this two letters, into a single “character” that have probability \( 2p_i \).
(F) New “character” has encoding of length \( l_i - 1 \), by induction
( on remaining \( n - 1 \) symbols).
(G) resulting tree encodes \( i \) and \( j \) by code words of length \( (l_i - 1) + 1 = l_i \).  

25.2.7.3 Translating lemma...

(A) \( p_i = 1/2^i \)
(B) \( l_i = \lg 1/p_i \).
(C) Average length of a code word is

\[
\sum_i p_i \lg \frac{1}{p_i}.
\]

(D) \( X \) is a random variable that takes a value \( i \) with probability \( p_i \), then this formula is

\[
\mathbb{H}(X) = \sum_i \Pr[X = i] \lg \frac{1}{\Pr[X = i]},
\]

which is the entropy of \( X \).