目次

★3　パターン情報の表現 (2) — データ圧縮と情報量（承前）

「承前」って？ ⇒ 辞書引きましょう

★3.7　エントロピー符号化

前節の Q の条件で，ほげおくんが書いた文字の並び（‘H’,‘O’,‘G’,‘E’ の 4 種のみでできている）を単純に 1 文字あたり 2bit で符号化するなら，この文字の並びのデータ量は当然 1 文字あたり 2bit となる．しかし，各文字の出現頻度に偏りがあるせいで，彼の書いた文字の並びの平均情報量（エントロピー）は 2bit よりも小さくなっていた．何か工夫することで，1 文字あたり 2bit より少ないデータ量で符号化できる方法があるんではないだろうか？

実は，データ圧縮の手法として，エントロピーの概念に基づく**エントロピー符号化**という符号化法が知られている．

★3.7.1　エントロピー符号化の考え方

表に示す 6 つの記号が，**それらを並べた記号列中の並び方や位置に無関係に**(☆ 1)，表に示す確率で出現するとする．この記号列を 0,1 で符号化したい．

☆ 1) H のあとに O が出てきやすいとか，最初の方は H が出てきやすい，というようなことがない，という意味．

記号	D	E	F	G	H	O
確率	$\frac{10}{100}$	$\frac{2}{100}$	$\frac{15}{100}$	$\frac{13}{100}$	$\frac{20}{100}$	$\frac{40}{100}$

最も単純な方法は，各記号を 3bit の符号で表現する（記号 D に 000，E に 001，等），というものである．この場合，たとえば `HOGEHOHOOO` という 10 文字の並びは，$10 \times 3 = 30$bit のビット列となる．この例では，どの記号にも 3bit を割り当てることになるから,「1 つの記号を平均何 bit で表せるか」を表す**平均符号語長**を考えると，その値は 3bit となる．

しかし，これらの記号の出現頻度には偏りがあるので，平均情報量はもっと小さいと考えられる (☆ 2)．この例の場合に実際に計算してみると，

$$H = -\frac{10}{100}\log_2\frac{10}{100} - \frac{2}{100}\log_2\frac{2}{100} - \cdots - \frac{40}{100}\log_2\frac{40}{100} = \text{約 } 2.23[\text{bit}] \quad (1)$$

となる．このことは，符号の割り当て方を工夫すれば，平均符号語長をここまで小さくできる可能性があることを意味している．たとえば，右表のように出現確率の高いものに短い符号を割り当てるようにしてやると，平均符号語長を

$$\frac{10}{100}\times 4 + \frac{2}{100}\times 4 + \frac{15}{100}\times 3 + \frac{13}{100}\times 3 + \frac{20}{100}\times 3 + \frac{40}{100}\times 1 = 2.32[\text{bit}] \quad (2)$$

にすることができる．実際に上記の 10 文字の並びの例でやってみると，

$$11101011000011101111000 \quad (3)$$

となり，30bit だったものを 21bit に減らせている．

☆ 2) ちなみに，6 通りの記号が等確率で出現する場合でも平均情報量は $-6\times\frac{1}{6}\log_2\frac{1}{6} = \log_2 6 = $ 約 2.58[bit] である．

記号	符号
D	1001
E	1000
F	110
G	101
H	111
O	0

このように，エントロピー符号化とは，記号の出現頻度（確率）に偏りがある，すなわち平均情報量が小さい場合に，その偏りを利用して効率的な（平均符号語長の短い）符号化を実現しようとするものである．上記の例では平均情報量が約 2.23bit であるから，理想的なエントロピー符号化ができれば平均符号長をそこまで減らせることを意味している．

Q1. 式 (3) の符号を表に従って先頭から順に復号していくとちゃんと元通りになることを確認してみよう．

★ 3.7.2 ハフマン符号化

エントロピー符号化の一種である**ハフマン符号化**は，記号毎に算出した出現頻度を用いて，次のような手順で木（ハフマン木）を作ることで，それぞれの記号に割り当てる符号を求める（授業中にもう少し詳しく説明します）．

ハフマン符号化: Huffman coding. 1950 年代前半に D. A. Huffman が提案した符号化法. 画像圧縮方式 JPEG でも，離散コサイン変換（そのうち紹介するかも）したデータの符号化に用いられている．

1. 個々の記号に対応した葉節点をつくる．出現確率をそれぞれの値とする．
2. 親のいない節点の中で最も小さい値をもつもの 2 つを選び，それらの親となる節点をつくる．親節点の値は，2 つの子節点の値の和とする．
3. 親を持たない節点が一つになるまで 2. を繰り返す．
4. 「左」を 0,「右」を 1 として，できた木の各節点から左右に接続した枝のそれぞれに 0,1 を割り当てる（「左」を 1 としてもよい）．
5. 根から葉までたどる際に現れる 0,1 のならびをそれぞれの葉の記号に対応した符号とする．

ハフマン符号化は純粋なエントロピー符号化法であり，記号の出現頻度のみを考慮して符号を決めるアルゴリズムである．したがって，出現頻度に偏りがないと有効な圧縮ができない (☆ 3)．また，特定の記号が何度も繰り返すとか，特定の並び（HOGE みたいな）が何度も出てくるとか，ある記号のあとに特定の記号が続くことが多い（G のあとには E がよく出てくるみたいな）というような記号の並び方は，データ圧縮の性能に影響しない（それを活かした圧縮はできない）．

☆ 3) 復号の際にハフマン木を使うので，データにハフマン木の情報も加える必要がある．そのため，極端な場合には元よりデータ量が増えてしまうこともありえる．

Q2. 上の（授業中に説明した）Huffman 木を用いて，次のデータを復号しなさい．1011000111010101110

Q3. HOGEHOGEHOGEHOGE... とひたすら繰り返すだけのテキストデータがあったとする．このデータにハフマン符号化を適用しても，有効なデータ圧縮はできない．その理由を述べなさい．

★ 3.7.3　よだんだよん

- エントロピー符号化の方法としては，Huffman 符号化よりも高効率な算術符号化という方法もある．
- 符号化は何もデータ圧縮のためだけに行なうものではない．雑音等のせいで送信データに誤りが生じてしまう状況で誤りを見つける（誤り検出）／誤りを訂正する（誤り訂正）ためにも用いられる (☆ 4)．最も原始的な誤り検出の符号化法は，ビット列の最後に，ビット列中の 1 の数が常に偶数になるように 0 か 1 を付け加える，というものである．興味のある人はいろいろ調べてみるとおもしろいかも．
- 符号化のさらに別の目的として，暗号化がある．

☆ 4) 情報通信機器では当たり前に使われている．図書の ISBN，様々な商品の JAN コード（バーコードの元になってるコード）などにも使われている．

情報理論の参考書: 高橋の手元には「情報理論」甘利俊一，ダイヤモンド社，ISBN4-478-82000-7，といういい本があるのですが，残念ながら現在は手に入らないようです (☆ 5)．他にいい本ないか探索中．
データ圧縮の参考書: 「圧縮アルゴリズム 符号化の原理と C 言語による実装」昌達 K'z，ソフトバンクパブリッシング，ISBN4-7973-2552-6

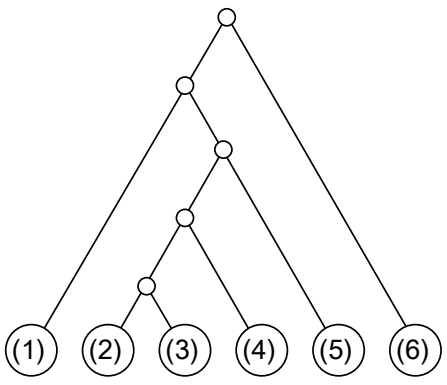
☆ 5) 最近，ちくま学芸文庫で復活しました．

Q4.

ある記号列を調べたところ，6 通りの記号が次表のような頻度で出現することがわかった．このような記号列をハフマン符号化法でビット列に符号化するとして，次の問に答えなさい（答えのみでよい）．

記号	あ	か	0	1	2	3
頻度	0.07	0.14	0.2	0.03	0.55	0.01

(1)–(6) 上記の表にもとづいてハフマン木を求めると，右図のようになった．(1) から (6) までの各葉節点に割り当てられる記号を答えなさい．ただし，このハフマン木は，授業中に説明したものと同様に，子をもつ全ての節点について，(左の子の値) < (右の子の値) となるように描かれているものとする．



(7) 右図のハフマン木の子をもつ全ての節点について，**左の子にむかってのびた枝に符号** 1 を，右の子にむかってのびた枝に符号 0 このハフマン木を用いて，次のビット列を記号列に復号しなさい．

110100010111

★4　パターン情報の成分分析 (1) — 直交展開

今回からしばらくは，パターン情報の成分を分析する方法を考える．今回と次回は特に，ベクトルで表されるパターンの成分分析を扱う．

★4.1　成分を分析したい

「成分を分析する」とはどういうことか，スープの成分を分析するという例（強引な例やけど）で考えてみよう．

スープの味が，塩味や甘味などのいくつかの味の成分の量で決まっており，それらの量は，それぞれの味に対応した調味料の含有量ではかれるものとする．このとき，あるスープに各成分がどれ位ずつ含まれているかがわかれば，右図のようなグラフを描ける．

このような情報が得られることには，次のような利点がある．

- そのスープの特徴（どんな成分を多く含んでいるか）がよくわかる
- ある成分の量を増減させることで味を調節できる
- 成分の量が分かっているので，別のところでそのスープの味を再現できる

音響信号や画像のようなパターン情報でも，これと同じようなことをできると便利そうである (☆ 6)．というわけで，パターンを

$$(\text{あるパターン}) = \alpha_1 \times (\text{成分 } 1) + \alpha_2 \times (\text{成分 } 2) + \alpha_3 \times (\text{成分 } 3) + \cdots \quad (4)$$

のように成分に分解して表す方法を考えてみよう．その際には，次のようなことに気をつけないといけないだろう．

- 何を「成分」とすればよいのか,「成分」はいくつあればよいのか
- $\alpha_1, \alpha_2, \ldots$ はどうやって求めるのか

☆ 6) パターンの特徴をつかめる，特定の成分を増減させてパターンを加工できる，各成分の含有量の情報を伝送するだけでパターンを再構成できる．