Кодирование недвоичных данных при помощи бинаризации и двоичного арифметического кодирования

5 января 2023 г. 1:24

Допустим, надо закодировать с помощью двоичного арифметического кодера недвоичные данные.
Чтобы была возможность закодировать недвоичные данные с помощью двоичного кодера, необходимо произвести бинаризацию этих данных. Возникает закономерный вопрос: как бинаризовать, чтобы не было потери по сжатию?

Рассмотрим ансамбль $X = \{A, B, C, D\}$, с распределениями вероятностей p_A, p_B, p_C, p_D. Энтропия ансамбля:

$$H = -p_A \log p_A - p_B \log p_B - p_C \log p_C - p_D \log p_D.$$

Бинаризация равномерным кодом с независимым кодированием:

$$A \to 00$$
$$B \to 01$$
$$C \to 10$$
$$D \to 11$$

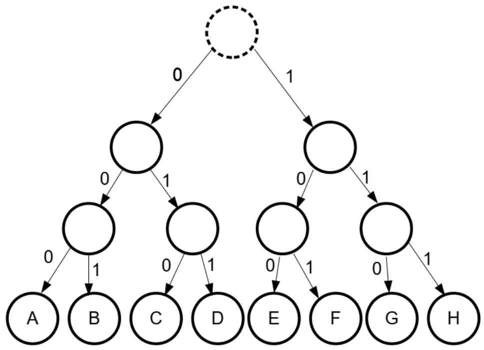
т.е. возьмем эти коды и будем передавать в кодер по 1 биту

в этом случае мы потеряем в сжатии, получим результат сильно хуже начальной энтропии

При независимом кодировании первого и второго символов:

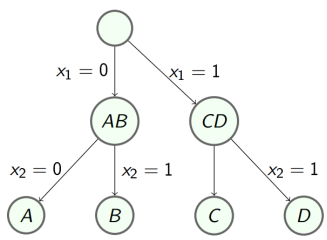
$$H_1 = H(X_1) + H(X_2) \geq H(X_1) + H(X_2|X_1) = H.$$

Древовидная бинаризация работает лучше, т.к. позволяет кодировать каждый бит с учетом предыдущего

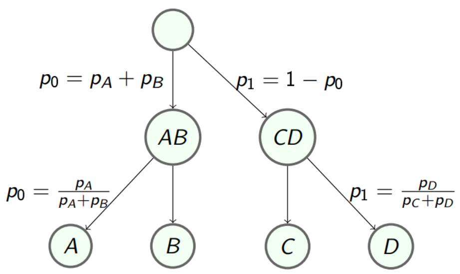


Здесь, чтобы закодировать А, нужно передать в кодер три нуля по очереди
Но кодирование будет условное: то есть второй 0 будет кодироваться, учитывая условие, что первым тоже был 0

Докажем, что при таком кодировании энтропия будет такая же, как и исходная, для примера выше
Рассмотрим ансамбль $X = \{A, B, C, D\}$, с распределениями вероятностей p_A, p_B, p_C, p_D.



Для буквы А сначала кодируем 0, показывая, что это АВ, а не CD, и затем кодируем 0, показывая, что из А и В нам нужно А



$$h_1 = -(p_A + p_B)\log(p_A + p_B) - (p_C + p_D)\log(p_C + p_D)$$
$$h_2 = (p_A + p_B)\left(-\frac{p_A}{p_A+p_B}\log\frac{p_A}{p_A+p_B} - \frac{p_B}{p_A+p_B}\log\frac{p_B}{p_A+p_B}\right) +$$
$$+(p_C + p_D)\left(-\frac{p_C}{p_C+p_D}\log\frac{p_C}{p_C+p_D} - \frac{p_D}{p_C+p_D}\log\frac{p_D}{p_C+p_D}\right) =$$
$$H - h1.$$
$$\Rightarrow h_1 + h_2 = H.$$

h1 - энтропия первого символа
h2 - второго

Вывод: при древовидной бинаризации мы не теряем в сжатии

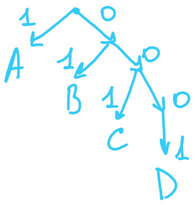
Унарная бинаризация

n	$unar(n)$
1	1
2	01
3	001

4	0001
5	00001
n	$\underbrace{00\ldots0}_{n-1}1$

Унарная бинаризация:

$$A \to 1$$
$$B \to 01$$
$$C \to 001$$
$$D \to 0001$$



убыточность!

$$h_1 = -p_A \log p_A - (1-p_A)\log(1-p_A).$$
$$h_2 = -\frac{p_B}{p_B+p_C+p_D}\log\frac{p_B}{p_B+p_C+p_D} - \frac{p_C+p_D}{p_B+p_C+p_D}\log\frac{p_C+p_D}{p_B+p_C+p_D}.$$
$$h_3 = -\frac{p_C}{p_C+p_D}\log\frac{p_C}{p_C+p_D} - \frac{p_D}{p_C+p_D}\log\frac{p_D}{p_C+p_D}.$$
$$h_4 = 0.$$

$$h_1 + (1-p_A)h_2 + (1-p_A-p_B)h_3 =$$
$$-p_A\log p_A - (1-p_A)\log(1-p_A) -$$
$$-p_B\log\frac{p_B}{p_B+p_C+p_D} - (p_C+p_D)\log\frac{p_C+p_D}{p_B+p_C+p_D} -$$
$$-p_C\log\frac{p_C}{p_C+p_D} - p_D\log\frac{p_D}{p_C+p_D} = H.$$

Что касается ренормализации, существует байтовая ренормализация для недвоичного алфавита, про которую препод в лекции толком ничего не сказал, поэтому я на экзамене тоже не скажу.
Здесь нет промежуточных вычислений, буфера и т.д. Здесь есть PUTBYTE, который сразу выдает байт на выход в файл

Байтовая ренормализация (range coder) для недвоичного[3] и двоичного[4] алфавита

два вида байтовой ренормализации

while $(L \oplus (L+R)) < 2^{24}$ *or* $R < 2^{16}$ **do**
  if $R < 2^{16} \wedge (L \oplus (L+R)) \geq 2^{24}$ **then**
    $R \leftarrow (!L+1) \wedge (2^{16}-1)$
  end if
  PUTBYTE $(L \gg 24)$
  $R \leftarrow R \ll 8$
  $L \leftarrow L \ll 8$
end while

if $(L \oplus (L+R)) < 2^{24}$ **then**
  PUTBYTE $(L \gg 24)$
  $R \leftarrow R \ll 8$
  $L \leftarrow L \ll 8$
else if $R < 2^{16}$ **then**
  $R \leftarrow (!L+1) \wedge (2^{16}-1)$
  PUTBYTE $(L \gg 24)$
  $R \leftarrow R \ll 8$
  $L \leftarrow L \ll 8$
end if