Арифметическое кодирование

5 января 2023 г. 1:23

Арифметическое кодирование является обобщением кода Гилберта-Мура для случая блокового кодирования.

Предположим, пришла какая-то последовательность из n символов.
Нужно вычислить:

1. Вероятность, с которой эта последовательность появится на выходе источника
2. Кумулятивную вероятность
3. Модифицированную кумулятивную вероятность
4. Какое количество бит мы должны потратить, т.е. длину кодового слова

Для $\boldsymbol{x} \in X^n$ вычислим:

1. $p(\boldsymbol{x}) = \prod_{i=1}^{n} p(x_i)$
2. $q(\boldsymbol{x})$?
3. $\sigma(\boldsymbol{x}) = q(\boldsymbol{x}) + p(\boldsymbol{x})/2$
4. $l(\boldsymbol{x}) = \lceil -\log p(\boldsymbol{x}) + 1 \rceil$

3 и 4 вычисляются с помощью 2, но как вычислить 2?
Как вычислить кумулятивную вероятность:
Для этого упорядочиваем все последовательности длины n в лексикографическом порядке

1. Пронумеруем все последовательности из X^n в алфавитном (лексикографическом) порядке.
2. Пусть i – наименьший индекс такой, что $x_i \neq y_j$, тогда $\boldsymbol{y}$ лексикографически предшествует $\boldsymbol{x}$ ($\boldsymbol{x} \prec \boldsymbol{y}$) если $x_i \prec y_i$.
3. apple ≺ energy ≺ entropy

Пример для двоичного случая:

0000
0001
0010
0011
0100
0101
0110
0111
1000
1001
…

Справка
Избыточность близка к 0 при устремлении количества символов к бесконечности, т.е. зная модель источника, мы можем приблизиться к его энтропии.
Недостаток - умножение на число от 0 до 1 (на вероятность), вследствие него очень быстро обнуляются регистры, хранящие произведение вероятностей и модифицированную кумулятивную вероятность, поэтому нужна ренормализация

Обозначим $q(\boldsymbol{x}) = \sum_{\boldsymbol{y} \prec \boldsymbol{x}} p(\boldsymbol{y})$. Эта куммулятивная вероятность может быть вычислена рекурсивно:

$$
\begin{aligned}
q(\boldsymbol{x}_1^n) &= \sum_{\boldsymbol{y}_1^n \prec \boldsymbol{x}_1^n} p(\boldsymbol{y}_1^n) = \\
&= \sum_{\boldsymbol{y}_1^{n-1} \prec \boldsymbol{x}_1^{n-1}} \sum_{y_n} p(\boldsymbol{y}_1^{n-1} y_n) + \sum_{\boldsymbol{y}_1^{n-1} = \boldsymbol{x}_1^{n-1}} \sum_{y_n \prec x_n} p(\boldsymbol{y}_1^{n-1} y_n) = \\
&= \sum_{\boldsymbol{y}_1^{n-1} \prec \boldsymbol{x}_1^{n-1}} p(\boldsymbol{y}_1^{n-1}) + \sum_{\boldsymbol{y}_1^{n-1} = \boldsymbol{x}_1^{n-1}} p(\boldsymbol{y}_1^{n-1}) \sum_{y_n \prec x_n} p(y_n) = \\
&= q(\boldsymbol{x}_1^{n-1}) + p(\boldsymbol{x}_1^{n-1}) q(x_n)
\end{aligned}
$$

Т.е. мы упорядочили все последовательности длины n лексикографически, и теперь вычислим кумулятивную вероятность:
q(**x**) - сумма всех вероятностей y, предшествующих нашему **x**, где **x** - рассматриваемая нами последовательность длины n, а **y** - предшествующие ей последовательности длины n

Таким образом, в арифметическом кодировании на каждом шаге рекурсивно перевычисляются вероятности и кумулятивные вероятности по формулам:

$$q(\boldsymbol{x}_{[1,n]}) = q(\boldsymbol{x}_{[1,n-1]}) + p(\boldsymbol{x}_{[1,n-1]}) q(x_n),$$

$$p(\boldsymbol{x}_{[1,n]}) = p(\boldsymbol{x}_{[1,n-1]}) p(x_n).$$

Алгоритм
Кодер
На вход: длина алфавита M, количество символов n, которое надо закодировать, и сами эти символы {x1,x2,…,xn}
На старте кумулятивная вероятность (т.е. для первого символа) = 0
Далее первым делом вычисляем кумулятивные вероятности для входных символов
F - кумулятивная вероятность, которую вычисляем на каждом шаге (на старте 0)
G - произведение вероятностей (на старте 1, чтобы не обнулялось)
Проходим по данным n символам в цикле и вычисляем F и G
На выход передаем кодовое слово, которое является первыми $\lceil -log(G) \rceil + 1$ битами двоичной записи F + G/2

Input: $M, \{p_1, \ldots, p_M\}$, n, $\{x_1, \ldots, x_n\}$

$q_1 \leftarrow 0$
for $i = 2, .., M$ **do**
  $q_i \leftarrow q_{i-1} + p_{i-1}$
end for
$F \leftarrow 0, G \leftarrow 1$
for $i = 1, .., n$ **do**
  $F \leftarrow F + q(x_i)G$
  $G \leftarrow p(x_i)G$
end for

Output: $\boldsymbol{c} \leftarrow$ первые $\lceil -\log G \rceil + 1$ бит двоичной записи $F + G/2$.

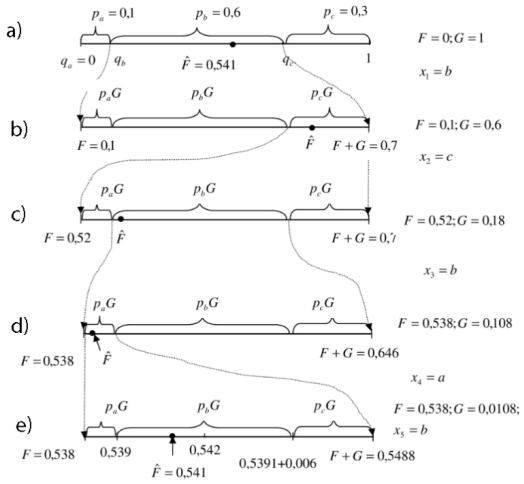
Пример:

Шаг i	x_i	$p(x_i)$	$q(x_i)$	F	G
0	-	-	-	0,0000	1,0000
1	b	0,6	0,1	0,1000	0,6000
2	c	0,3	0,7	0,5200	0,1800
3	b	0,6	0,1	0,5380	0,1080

4	*a*	0,1	0,0	0,5380	0,0108
5	*b*	0,6	0,1	0,5391	0,0065
6	Длина кодового слова $\lceil -\log G + 1 \rceil = 9$ Кодовое слово $F + G/2 = 0,5423... \rightarrow$ $\rightarrow \hat{F} = 0,541 \rightarrow 100010101$				

вероятность

Графическая интерпретация



Декодер

На вход: размер исходного алфавита M, вероятности M символов, количество символов n декодированной последовательности, округленная модифицированная кумулятивная вероятность, полученная в результате кодирования

Как и кодер, декодер сначала считает кумулятивные вероятности, принимая первую за 0, а дальше вычисляя путем прибавления вероятностей в цикле

S и G устанавливаются соответственно в 0 и 1

S - кумулятивная вероятность, которую вычисляем на каждом шаге (на старте 0)

G - произведение вероятностей (на старте 1, чтобы не обнулялось)

Дальше работает аналогично кодеру, но есть отличие: еще один цикл while

В этом цикле мы рассматриваем отрезки, на которые отрезок от 0 до 1 делится кумулятивными вероятностями q, и смотрим, если округленная модифицированная кумулятивная вероятность будет находиться между двумя какими-то значениями. Как только находим эти два значения, возвращаем индекс этого отрезка

Input: $M, \{p_1, ..., p_M\}, n, \hat{F}$

1: $q_1 \leftarrow 0$
2: **for** $i = 2, .., M$ **do**
3: $\quad q_i \leftarrow q_{i-1} + p_{i-1}$
4: **end for**
5: $q_{M+1} \leftarrow 1, S \leftarrow 0, G \leftarrow 1$
6: **for** $i = 1, .., n$ **do**
7: $\quad j \leftarrow 1$
8: $\quad$ **while** $S + q_{j+1}G < \hat{F}$ **do**
9: $\quad\quad j \leftarrow j + 1$
10: $\quad$ **end while**
11: $\quad S \leftarrow S + q_jG$
12: $\quad G \leftarrow p_jG$
13: $\quad x_i \leftarrow j$
14: **end for**

Output: $\{x_1, ..., x_n\}$

$X = \{a, b, c\}$. $p_a = 0,1$, $p_b = 0,6$, $p_c = 0,3$. 0100010101

Шаг	S	G	Гипотеза x	$q(x)$	$S + qG$	Решение x_i	$p(x)$
0	100010101$\rightarrow \hat{F} = 0,541$						
1	0,0000	1,0000	*a*	0,0	$0,0000 < \hat{F}$	*b*	0,6
			b	0,1	$\mathbf{0,1000} < \hat{F}$		
			c	0,7	$0,7000 > \hat{F}$		
2	0,1000	0,6000	*a*	0,0	$0,1000 < \hat{F}$	*c*	0,3
			b	0,1	$0,1600 < \hat{F}$		
			c	0,7	$\mathbf{0,5200} < \hat{F}$		
3	0,5200	0,1800	*a*	0,0	$0,5200 < \hat{F}$	*b*	0,6
			b	0,1	$\mathbf{0,5380} < \hat{F}$		
			c	0,7	$0,6460 > \hat{F}$		
4	0,5380	0,1080	*a*	0,0	$\mathbf{0,5380} < \hat{F}$	*a*	0,1
			b	0,1	$0,5488 > \hat{F}$		
5	0,5380	0,0108	*a*	0,0	$0,5380 < \hat{F}$	*b*	0,6
			b	0,1	$\mathbf{0,5391} < \hat{F}$		
			c	0,7	$0,5456 > \hat{F}$		

Реализация

- Проблемы:
  1. Разрядность регистров: Количество бит, необходимое для F и G растёт с каждым умножением на вероятность.
  2. Задержка: кодовое слово формируется после кодирования последнего символа.
- Решение:
  1. Представить входные вероятности при помощи целых чисел.
  2. Выдавать старшие разряды кодового слова, которые не будут меняться и сокращать разряды (ренормализовать), необходимые для F and G

3. Использовать специальный счётчик "btf" (bits to follow), если старшие разряды кодового слова не определены на данном шаге. (...011111... или ...100000...)

- Результат: 16-битная реализация.

Реализация

- Обозначим через L (*low*) регистр, в котором хранится F, через R (*range*) регистр, в котором хранится G и введем регистр $H = L + R$ (*high*).
- В начале работы алгоритма $L = 0, H = 2^b - 1$, где b разрядность алгоритма.
- В результате операции $R = p_j R$ регистр R может обнулиться.
- Задача ренормализации держать регистры $H \geq \frac{3}{4}2^b$, $L \leq \frac{1}{4}2^b$, т.е. $R \geq \frac{1}{2}2^b$.

В алгоритме присутствует умножение R на вероятность p, то есть на число от 0 до 1, это приводит к тому, что R становится дробным, и если из раза в раз выполнять это умножение, в какой-то момент регистр R обнулится. Чтобы этого избежать, выполняется ренормализация. Ее суть в том, чтобы держать регистры в положении, из которого они не могут переполниться. Если они выходят из этого положения, то мы выдаем соответствующее количество бит на выход выполняем сдвиг в регистрах.

См. двоичное арифметическое кодирование в "Реализация двоичного арифметического кодирования без умножений"

Целочисленная реализация на MATLAB

```
function y=int_arithm_encoder(x,q);
% x is input data sequence,
% q is cumulative distribution (model)
% y is binary output sequence

% Constants
k=16;
R4=2^(k-2); R2=R4*2; R34=R2+R4;% half,quarter,ei
R=2*R2;                        % Precision

% Initialization
Low=0;      % Low
High=R-1;   % High
btf=0;      % Bits to Follow
y=[ ];      % code sequence

% Encoding
for i=1:length(x);
   Range=High-Low+1;
   High=Low+fix(Range*q(x(i)+1)/q(m))-1;
   Low=Low+fix(Range*q(x(i))/q(m));

   % Normalization
   while 1
      if High<R2
         y=[y 0 ones(1,btf)]; btf=0;
         High=High*2+1; Low=Low*2;
      else
         if Low>=R2
            y=[y 1 zeros(1,btf)]; btf=0;
            High=High*2-R+1; Low=Low*2-R;
         else
            if Low>=R4 & High<R34
               High=2*High-R2+1; Low=2*Low-R2;
               btf=btf+1;
            else
               break;
            end;
         end;
      end;
   end; % while
end; % for

% Completing
if Low<R4
   y=[y 0 ones(1,btf+1)];
else
   y=[y 1 zeros(1,btf+1)];
end;
```

```
function x=int_arithm_decoder(y,q,n);
% y is binary encoded data sequence,
% q is cumulative distribution (model)
% x is output sequence
% n is number of messages to decode

% Constants
k=16; R4=2^(k-2); R2=R4*2; R34=R2+R4; R=2*R2;
m=length(q);

% Start decoding. Reading first k bits
Value=0; y=[y zeros(1,k)];
for ib=1:k
   Value=2*Value+y(ib);
end;

% Initialization
Low=0; High=R-1;

% Decoding
for j=1:n
   Range=High-Low+1;
   aux=fix(( (Value-Low+1)*q(m)-1)/Range);
   i=1; % message index
   while q(i+1)<=aux, i=i+1; end;
   x(j)=i;
   High=Low+fix(Range*q(i+1)/q(m))-1;
   Low=Low+fix(Range*q(i)/q(m));

   % Normalization
   while 1
      if High<R2
         High=High*2+1; Low=Low*2;
         ib=ib+1;
         Value = 2*Value+y(ib);
      else
         if Low>=R2
            High=High*2-R+1; Low=Low*2-R;
            ib=ib+1;
            Value = 2*Value-R+y(ib);
         else
            if Low>=R4 & High<R34
               High=2*High-R2+1; Low=2*Low-R2;
               ib=ib+1;
               Value = 2*Value-R2+y(ib);
            else
               break;
            end;
         end;
      end
   end; % while
end; % for
```

Кратко поясним работу кодера. Числу F в формальном описании алгоритма в программе соответствует переменная Low. Числу G – переменная Range, приблизительно равная разности High−Low. Если High меньше половины диапазона или Low больше половины (это соответствует значениям $F+G < 1/2$ и $F > 1/2$), очередной кодовый бит может быть выдан на выход кодера и переменные High, Low соответствующим образом модифицируются. Случай, когда Low>=R4 и одновременно High<R34, соответствует описанной выше неопределенной ситуации, возникающей при F, близких к $1/2$, но меньших $1/2$. В переменной btf накапливается длина последовательности вида $0111\ldots1$ либо $1000\ldots0$, формируемой кодером после разрешения неопределенности.