Теория информации

Лекция 6.

Символьные коды (продолжение)

Рассмотрим какова цена использования кода с неоптимальной длиной кодового слова. Пусть $\{p_i\}$ это вероятности в A_x и наш код завершенный с длинами $\{l_i\}$. Поскольку код завершенный, то $z=1$ и вероятности, задаваемые множеством длин кодовых слов определяются как $q_i=2^{-l_i}$. Тогда можно записать следующие выкладки для стоимости кодирования:

$$L(C,X) = \sum_i p_i l_i = \sum_i p_i \log_2 \frac{1}{q_i} = \sum_i p_i \log_2 \frac{1}{p_i} + \sum_i p_i \log_2 \frac{1}{q_i} - \sum_i p_i \log_2 \frac{1}{p_i} = \\ = H(X) + \sum_i p_i \log_2 \frac{p_i}{q_i} = H(X) + D_{KL}(\boldsymbol{p},\boldsymbol{q}) \qquad .(6.1)$$

Таким образом, стоимость кодирования $L(C,X)$ превышает энтропию ансамбля $H(X)$ как раз на расстояние Кульбака-Лейблера между распределениями $\boldsymbol{p}$ и $\boldsymbol{q}$. Перейдем теперь с практическому построению символьного кода.

Кодирование Хаффмана (Haffman)

Идея алгоритма Хаффмана заключается в том, что можно построить двоичное дерево префиксного кода в обратном порядке, начиная с его листьев. Для этого нужно сделать следующее:

1. Взять два наименее вероятных символа в алфавите. Эти два символа дадут самые длинные кодовые слова равной длины, отличающиеся только в последнем разряде.
2. Объединить ветви от этих слов в одну, просуммировав вероятности, и начать сначала, пока не закончим в корне дерева.

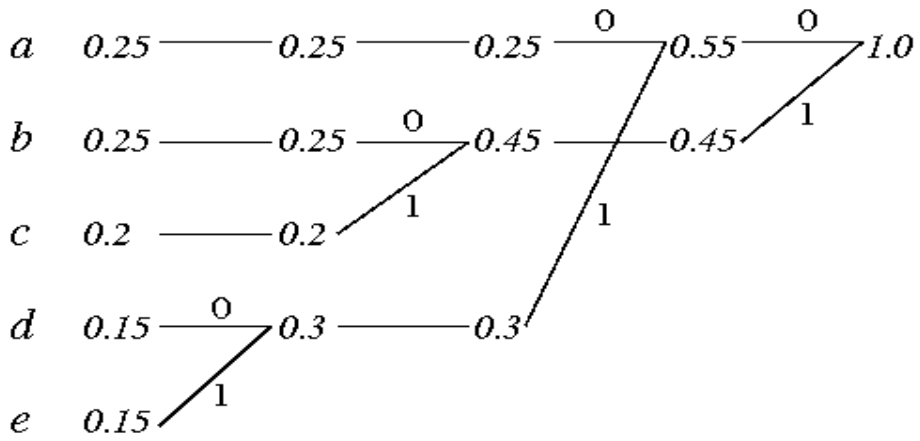
Поскольку каждый шаг алгоритма уменьшает алфавит на единицу, то через $|A_x|-1$ шагов все символы получат свои кодовые слова.

Пример:

$$A_x=\{a,b,c,d,e\}$$

$$P_x=\{0.25,0.25,0.2,0.15,0.15\}$$

Ниже приведено построение дерева префиксного кода:



Кодовые слова из этого дерева можно получить конкатенацией нулей и единиц, начиная с корня дерева. Множество кодовых слов в данном примере:

$$C = \{00,\ 10,\ 11,\ 010,\ 111\} \quad .$$

Все величины, относящиеся к построенному коду, представлены в таблице:

a_i	p_i	$h(p_i)$	l_i	$c(a_i)$
a	0.25	2.00	2	00
b	0.25	2.00	2	10
c	0.2	2.32	2	11
d	0.15	2.74	3	010
e	0.15	2.74	3	111

Из таблицы видно, что длина кодового слова иногда длиннее оптимальной, а иногда короче. Стоимость кодирования в данном примере $L(C,X)=2.3$, в то время как $H(X)=2.2855$.

Код Хаффмана является наилучшим символьным кодом, то есть не существует кода с меньшей стоимостью кодирования $L(C,X)$. Чтобы доказать это, достаточно показать, что основная идея присвоения кодовых слов максимальной длины двум символам (а не одному наименее вероятному) не ухудшает код. Примем это утверждение без доказательства в рамках данного курса. (Желающие могут сделать это в качестве упражнения или посмотреть в книге

David J.C. MacKey Information Theory, Inference and Learning Algorithms)

Хотя код Хаффмана назван здесь оптимальным, он все еще недостаточно хорош для практического использования. Рассмотрим основные недостатки этого кода.

Если мы работаем с ансамблем с неизменными вероятностями, то код Хаффмана достаточно хорош. Но если ансамбль меняется (в разных источниках-текстах символы встречаются с разной вероятностью), то возникает необходимость вычислять код для каждого источника (вычислять вероятности, просматривая весь источник, строить снова дерево). Таким образом этот код не позволяет работать с источником потоковым образом, не просматривая его до конца.

Имеются лишние биты, связанные с неравенством $H(X) \le L(C,X) < H(X)+1$, то есть имеются накладные расходы битов в интервале от 0 до 1 бита на символ. При большой энтропии ансамбля $H(X)$ эти битовые расходы невелики, однако в случае ансамблей с малой энтропией (она может быть даже <<1) они существенно возрастают.

Потоковые коды

Рассмотрим две схемы упаковки данных: арифметическое кодирование и кодирование Лемпеля-Зива.

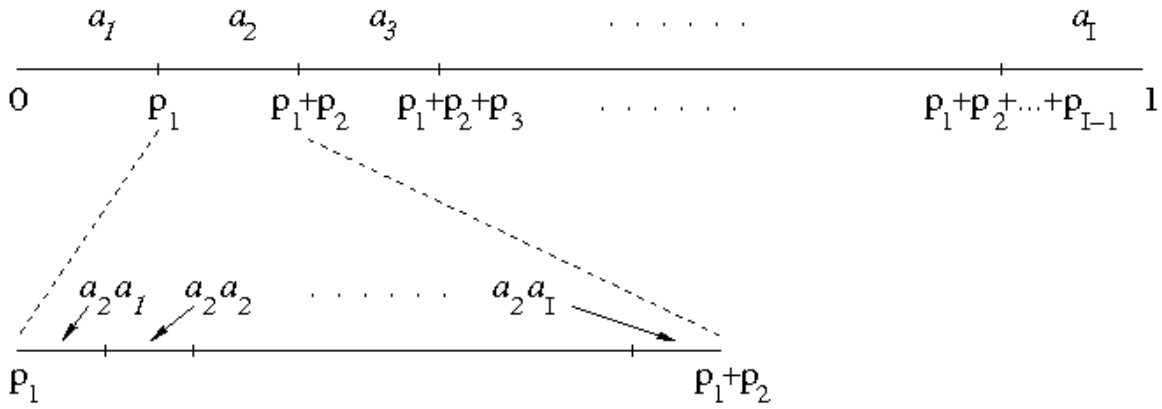
Арифметическое кодирование

Пусть алфавит источника $A_x = \{a_1, a_2, \dots, a_I\}$ и последний символ в нем a_I имеет специальное значение конца передачи. Источник поставляет последовательность символов $x_1 x_2 \dots x_n \dots$, причем необязательно все символы независимые и идентично распределенные случайные переменные. Предположим, что программа, которая занимается кодированием, строит вероятностное распределение на основе уже полученных символов, т.е. вычисляет $P(x_n = a_i \mid x_1 x_2 \dots x_{n-1})$. Кроме того, предположим, что программа-декодер также может вычислять эти вероятности.

Определим отображение двоичной последовательности на интервал [0,1) следующим образом. Если есть двоичная последовательность, скажем 01, то она отображается на интервал [0.01,0.10), где все числа двоичные (в десятичном виде это интервал [0.25,0.5)). Если последовательность длиннее, например 01101, то она соответствует меньшему интервалу [0.01101,0.01110), который меньше и находится внутри интервала, соответствующего подстроке в начале 01. При таком отображении файл размером 1 Мб (2^{23} бит) отобразится в некоторое число в интервале [0,1), которое будет представлено в десятичном виде с

точностью около 2 миллионов цифр после десятичной точки.

С другой стороны, мы можем разделить интервал [0,1) на I интервалов длиной, равной вероятности $P(x_1=a_i)$. Интервал будет разбит точками с координатами $P(x_1=a_1),\ P(x_1=a_1)+P(x_1=a_2),\ldots,P(x_1=a_1)+\ldots+P(x_1=a_{I-1})$. Ниже приведен рисунок иллюстрирующий такое разбиение.



Интервалы можно обозначить в соответствии с символами алфавита, то есть интервал [*0, P(x₁ = a₁)*) обозначить соответствующим символом a_1, интервал [$P(x_1=a_1),P(x_1=a_1)+P(x_1=a_2)$) обозначить символом a_2 и т.д. Аналогично каждый интервал a_i можно разбить на меньшие интервалы обозначенные $a_ia_1,a_ia_2,\ldots,a_ia_I$, причем длина каждого такого интервала будет равна совместной вероятности появления этих сочетаний:

$$P(x_1=a_i,x_2=a_j)=P(x_1=a_i)P(x_2=a_j\mid x_1=a_i) \quad . \tag{6.1}$$

Если и дальше продолжать эту процедуру, то интервал [0,1) можно разбить на последовательность интервалов соответствующим всем возможным строкам $x_1x_2\ldots x_N$ конечной длины, так, что длина соответствующего интервала будет равна вероятности появления такой строки. Процесс разбиения может быть явно сформулирован в терминах нижней и верхней накопленной вероятности:

$$Q_n(a_i\mid x_1,\ldots,x_{n-1}) \;=\; \sum_{i'=1}^{i-1} P(x_n=a_{i'}\mid x_1,\ldots,x_{n-1}) \quad , \tag{6.2}$$

$$R_n(a_i\mid x_1,\ldots,x_{n-1}) \;=\; \sum_{i'=1}^{i} P(x_n=a_{i'}\mid x_1,\ldots,x_{n-1}) \quad . \tag{6.3}$$

По мере получения из источника n-го символа, $(n-1)$ -ый интервал разбивается точками определяемыми Q_n и R_n . Алгоритм такого деления можно записать

в виде такого псевдокода:

```
u := 0.0
v := 1.0
p := v-u
for n = 1 to N {
     Вычислить Qn и Rn по формулам (6.2) и (6.3)
     u := p*Qn
     v := p*Rn
     p := v-u
}
```

Как только определен интервал, соответствующий строке $x_1 x_2 \ldots x_N$ она кодируется с помощью двоичной строки, чей интервал лежит внутри этого интервала. Такое кодирование может быть осуществлено на ходу.

Декодер использует ту же самую вероятностную модель, что и программа-кодер. Поэтому, читая двоичные символы по одному он определяет внутри какого интервала соответствующего последовательности символов лежит интервал соответствующий полученной двоичной последовательности. После получения символа конца передачи, процедура заканчивается. Символ конца передачи обнаруживается, когда последний интервал оказывается последним в предыдущем интервале.

Кодирование Лемпеля-Зива.

В отличие от арифметического кодирования, этот алгоритм не использует вероятностную модель. Имеется две версии этого алгоритма: теоретическая и практическая, которые выполняют примерно одно и тоже, однако первая допускает доказательство асимптотической оптимальности этого алгоритма, а вторая легче в программной реализации. В данном курсе мы кратко рассмотрим практическую реализацию.

Идея кодирования заключается в замене подстроки указателем на ее раннее вхождение в кодируемую последовательность. Рассмотрим следующий пример (см. http://www.data-compression.com/lossless.shtml#lz, здесь также имеется анимация, иллюстрирующая алгоритм). Пусть $A_x = \{a, b\}$. Алгоритм кодирования можно представить в виде следующей последовательности действий:

1. инициализировать словарь всеми односимвольными строками;
2. найти самый длинный блок w , встречавшийся в словаре;
3. кодировать его индексом в словаре;
4. добавить к w следующий символ и добавить получившуюся строку в словарь;
5. перейти к 2.

Пример строки:

data: *a b b a a b b a a b a b b a a a a b a a b b a*

0 1 1 0 2 4 2 6 5 5 7 3 0

Этому примеру соответствует словарь:

Индекс	0	1	2	3	4	5	6	7	8	9	10	11	12	13
строка	*a*	*b*	*ab*	*bb*	*ba*	*aa*	*abb*	*baa*	*aba*	*abba*	*aaa*	*aab*	*baab*	*bba*

Это самое простое изложение кодирования Лемпеля-Зива. Детали реализации вне рамок данного курса. Можно отметить лишь, что данный алгоритм используется программах архивирования (например в gzip).