Арифметическое кодирование

Иринёв Антон, Каширин Виктор

Оглавление

1. Введение 2

2. Описание алгоритма 2

3. Анализ алгоритма 4

4. Адаптивный алгоритм 7

Литература 9

1. Введение

В наши дни информация является главным инструментом в развитии фундаментальных наук, технологий, методов работы и обучения. Посредством передачи информации люди обмениваются опытом, знаниями и полезными сведениями, что, несомненно, ведет к прогрессу во всех областях человеческой деятельности.

Объёмы передаваемой электронной информации возрастают с огромной скоростью. Это связанно, прежде всего, с развитием Интернета и беспроводных технологий связи. Однако, несмотря на высокую пропускную способность современных каналов связи, передача больших объёмов информации всегда сопряжена со значительными временными затратами. Именно поэтому возникла такая область информационных технологий, как сжатие данных.

Теория сжатия данных объединяет в себе несколько различных направлений. В рамках данной теории методы принято разделять на методы кодирования информации без потерь (lossless) и методы кодирования информации с потерями (lossy). Как следует из названий, методы первой группы не ведут к информационным потерям (т.е. первоначальная информация может быть в точности восстановлена из сжатого состояния), тогда как использование методов второй группы сопряжено с такими потерями. Применение сжатия без потерь особенно важно для текстовой информации, записанной на естественных и на искусственных языках, поскольку в этом случае ошибки обычно недопустимы. В свою очередь сжатие с потерями применяется к информации, содержащей отдельные несущественные составляющие, которые при определенных условиях могут быть частично или полностью удалены. Методы сжатия с потерями часто используются для сжатия звука и изображений.

В этой статье мы рассмотрим один из методов сжатия информации без потерь, который носит название "Арифметическое кодирование". Основой арифметического кодирования является неопубликованный алгоритм П. Элайеса. Также существенный вклад в развитие данного метода в разное время внесли П. Говард, М. Гуаззо, Д. Клири, Г. Лэнгдон, Э. Моффат, Р. Нил, Р. Паско, Д. Риссанен, Ф. Рубин, Я. Уиттен и М. Шиндлер.

2. Описание алгоритма

Метод арифметического кодирования основан на идее преобразования входного потока в число с плавающей точкой из интервала [0, 1). Каждый поступающий на обработку символ уменьшает этот интервал пропорционально вероятности своего появления. По мере поступления очередного символа интервал уменьшается, а число бит, представляющие этот интервал - увеличивается. Заметим, что изначально длина интервала равна 1. Известно, что сумма всех вероятностей появления символов алфавита также равна 1. Пусть N - количество символов алфавита, p_i - вероятность появления в тексте символа с порядковым номером i. Тогда i-му символу поставим в соответствие интервал:

$$\left[\sum_{k=0}^{i-1} p_k, \sum_{k=0}^{i} p_k\right), \quad i = 1..N, \quad p_0 = 0 \quad (1)$$

Если вероятности появления символов в тексте изначально не известны, то можно воспользоваться какой-нибудь их оценкой (например, для текста на естественном языке можно взять среднестатистические вероятности появления символов для данного языка). Этот вопрос также будет затронут нами при описании адаптивного алгоритма. Пока же мы считаем, что перед началом работы алгоритма нам доступен весь текст, откуда мы получаем точные вероятности всех символов (p_i = кол-во вхождений i-го символа/общее количество символов).

Если первым на вход поступает символ с порядковым номером k в алфавите, то вместо исходного мы берём k-ый интервал и делим его на N частей в тех же пропорциях, что и исходный. Теперь мы готовы к приёму второго символа и т.д.

Затем для построенного интервала находится число, принадлежащее его внутренней части и равное целому числу, деленному на минимально возможную положительную целую степень двойки. Это число и будет кодом для рассматриваемой последовательности. Из него можно однозначно восстановить исходную последовательность символов. Т.к. все возможные конкретные коды — это числа из интервала [0, 1), то из соображений экономии можно отбрасывать лидирующий ноль и десятичную точку, но нужен еще один специальный код-маркер, сигнализирующий о конце сообщения.

Поясним все выше сказанное на примере. Рассмотрим текст "compressor" алфавита {c,e,m,o,p,r,s}.

Символ	Вероятность	Интервал
c	0.1	[0.0, 0.1)
e	0.1	[0.1, 0.2)
m	0.1	[0.2, 0.3)
o	0.2	[0.3, 0.5)
p	0.1	[0.5, 0.6)
r	0.2	[0.6, 0.8)
s	0.2	[0.8, 1.0)

И кодировщику, и декодировщику известно, что в самом начале интервал [0, 1). После просмотра первого символа "c", кодировщик сужает интервал до [0.0, 0.1), который модель выделяет этому символу. Второй символ "o" сузит этот новый интервал до промежутка [0.3, 0.5). Продолжая в том же духе, имеем:

В начале	[0.0,	1.0	)
*После просмотра "**c**"*	[0.0,	0.1	)
*После просмотра "**o**"*	[0.03,	0.05	)
*После просмотра "**m**"*	[0.034,	0.036	)
*После просмотра "**p**"*	[0.0350,	0.0352	)
*После просмотра "**r**"*	[0.03512,	0.03516	)
*После просмотра "**e**"*	[0.035124,	0.035128	)
*После просмотра "**s**"*	[0.0351272,	0.0351280	)
*После просмотра "**s**"*	[0.03512764,	0.03512800	)
*После просмотра "**o**"*	[0.035123748,	0.035127820	)
*После просмотра "**r**"*	[0.0351239912,	0.0351239056	)

Итак, текст закодирован интервалом [0.0351239912, 0.0351239056)
Число, характеризующее этот интервал – 2357387 / 67108864.
$67108864 = 2^{26}$, т.е. мы можем закодировать исходный текст с помощью 26 бит.

Предположим, что все, что декодировщик знает о тексте, это конечный интервал [0.0351239912, 0.0351239056). Он сразу же понимает, что первый закодированный символ есть "c", т.к. итоговый интервал целиком лежит в интервале, выделенном моделью этому символу согласно таблице вероятностей. Теперь повторим действия кодировщика:

В начале	[0.0, 1.0)
*После просмотра "**c**"*	[0.0, 0.1)

Отсюда ясно, что второй символ - это "o", поскольку это приведет к интервалу [0.03, 0.05), который полностью вмещает итоговый интервал [0.0351239912, 0.0351239056). Продолжая работать таким же образом, декодировщик извлечет весь текст.

Однако, чтобы завершить процесс, декодировщику нужно вовремя распознать конец текста. Для устранения неясности мы должны обозначить завершение каждого текста специальным символом EOF, известным и кодировщику, и декодировщику. Когда декодировщик встречает этот символ, он прекращает свой процесс.

Напишем процедуру кодирования текста.

Алгоритм кодирования текста. На вход подаётся массив интервалов frequency[0..n] (n - число символов в нашем алфавите), делящих отрезок [0, 1] соответственно частотам вхождения символов в текст.

```
 1  procedure encode_text(frequency);
 2  begin
 3       low = 0.0;
 4       high = 1.0;
 5       repeat
 6            symbol = next_symbol();
 7            range = high - low;
 8            high = low + range * frequency[symbol];
 9            low  = low + range * frequency[symbol - 1];
10       until (symbol == EOF);
11  end;
```

В строках 3 - 4 происходит инициализация стартового отрезка. В строках 5 - 10 алгоритм последовательно обрабатывает каждый символ текста. В строках 5 и 10 вызывается процедура *next_symbol*, возвращающая последующий символ декодируемого текста. Переменная symbol хранит в себе номер символа в алфавите. С учетом частоты вхождения конкретного символа изменяется рабочий интервал. По завершении обработки всех символов текста получаем искомый интервал - [low, high).

Алгоритм декодирование текста. На вход подается число value, полученное в результате кодирования, и массив frequency.

```
 1  procedure decode_text (value, frequency);
 2  begin
 3       low = 0.0;
 4       high = 1.0;
 5       repeat
 6            repeat
 7                 symbol++;
 8            until (frequency[symbol -1] <= (value - low)/(high - low)) and
                   ((value - low)/(high - low) < frequency[symbol]);
 9            range = high – low;
10            high = low + range * cum_freq [symbol - 1];
11            low  = low + range * cum_freq [symbol ];
12            print(symbol);
13       until (symbol == EOF);
14  end;
```

В строках 3 - 4 происходит инициализация стартового отрезка. В цикле 5 – 13 мы декодируем символ за символом, пока не приходим к метке конца текста. В цикле 6 – 8 алгоритм, с помощью перебора алфавита, находит символ, попадающий в конкретный интервал. В строках 9 – 12 сужается интервал для следующего шага, выводится декодированный символ.

3. Анализ алгоритма

В теории сжатия данных без потерь существуют два основных способа кодирования информации: статистический и словарный. Лучшие статистические методы применяют арифметическое кодирование, лучшие словарные - метод Лемпеля-Зива. В статистическом сжатии

каждому символу присваивается код, основанный на вероятности его появления в тексте. Символы, вероятность появления которых высока, получают короткие коды, и наоборот.

В основе словарных методов лежит идея, совершенно отличная от идеи статистического сжатия. Словарный кодировщик добивается сжатия с помощью замены групп последовательных символов индексами некоторого словаря. Словарь есть список таких фраз, которые, как ожидается, будут часто использоваться. Индексы устроены таким образом, что в среднем занимают меньше места, чем кодируемые ими фразы, за счет чего и достигается сжатие.

Любая практическая схема словарного сжатия может быть сведена к соответствующей статистической схеме, и найден общий алгоритм преобразования словарного метода в статистический. Поэтому при поиске лучшего сжатия статистическое кодирование обещает быть наиболее плодотворным, хотя словарные методы и привлекательны своей быстротой.

Генерация систем кодов для статистических методов в свою очередь получается либо с использованием префиксного кодирования, либо с использованием арифметического кодирования.

Префиксное кодирование является простейшим методом генерации кодов переменной длины. Коды, получаемые с использованием префиксного кодирования, имеют целую длину в единицах представления информации. Преимущество префиксного кодирования состоит в его простоте и высокой производительности, а недостаток – в невозможности создавать коды нецелой длины, что, естественно, отрицательно сказывается на эффективности кодирования.

В отличие от префиксного кодирования, арифметическое кодирование позволяет генерировать коды как целой, так и нецелой длины. Являясь теоретически оптимальным методом, арифметическое кодирование превосходит в эффективности префиксное кодирование. Оно также опережает префиксное кодирование в скорости построения системы кодов, однако из-за повышенной сложности нередко заметно уступает в скорости самого кодирования (имеется в виду процесс генерации кодовой последовательности).

Несмотря на некоторые существенные преимущества арифметического кодирования, данный метод до последнего времени был недостаточно распространен на практике. На сегодняшний день в большинстве коммерческих приложений для построения системы кодов переменной длины используется кодирование Хаффмана, являющееся лучшей с точки зрения эффективности реализацией префиксного кодирования. Арифметическое кодирование применяется лишь в тех случаях, когда требуется добиться максимально возможного качества информационного представления и когда скорость работы не имеет решающего значения.

Рассмотрим данный вопрос подробнее. Для этого сравним два наиболее популярных метода статистического кодирования информации – алгоритм Хаффмана и Арифметическое кодирование.

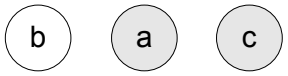
Вкратце поясним, в чем заключается алгоритм Хаффмана. На начальном этапе работы алгоритма каждому символу информационного алфавита ставится в соответствие вес, равный вероятности появления данного символа в тексте. Символы помещаются в список, который сортируется по убыванию весов. На каждом шаге алгоритма два последних элемента списка объединяются в новый элемент, который затем помещается в список вместо двух объединенных. Новому элементу списка ставится в соответствие вес, равный сумме весов замещаемых элементов. Каждая итерация заканчивается упорядочиванием полученного нового списка, который всегда содержит на один элемент меньше, чем старый список.

Параллельно с работой указанной процедуры осуществляется последовательное построение кодового дерева. На каждом шаге алгоритма любому элементу списка соответствует корневой узел бинарного дерева, состоящего из вершин, соответствующих элементам, объединением которых был получен данный элемент. При объединении двух элементов списка происходит объединение соответствующих деревьев в одно новое бинарное дерево, в котором корневой узел соответствует новому элементу, помещаемому в список, а замещаемым элементам списка соответствуют дочерние узлы этого корневого узла. Алгоритм завершает работу, когда в списке остается один элемент, соответствующий корневому узлу построенного бинарного дерева. Это дерево называется деревом

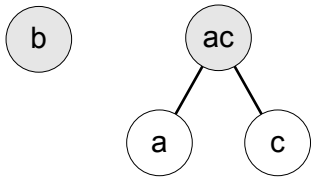
Хаффмана. Система префиксных кодов может быть получена путем присваивания конкретных двоичных значений ребрам этого дерева.

На следующем рисунке приведен пример работы алгоритма Хаффмана для текста “abcbb” с алфавитом {a, b, c}. Для элементов a, b и c их вероятности появления соответственно равны 0.2, 0.6 и 0.2.

1 шаг: b (0.6) a (0.2) c (0.2)

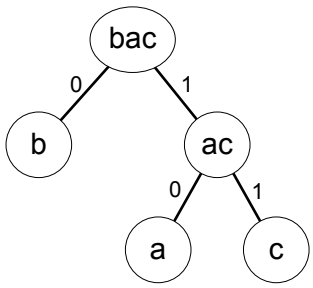


2 шаг: b (0.6) ac (0.4)



3 шаг: bac (1.0)

(построено дерево Хаффмана)



Согласно построенному дереву Хаффмана для кодирования исходного сообщения нам потребуется последовательность из 7 битов: “1001100” (a – “10”, b – “0”, c – “11”).

Для сравнения введем следующую величину: будем характеризовать метод кодирования числом ML(S), где S – исходное сообщение. ML(S) равняется среднему числу бит, затраченных на кодирование каждого символа сообщения выбранным методом сжатия, т.е. выражается следующей формулой:

$$ML(S) = L(S') / L(S) \qquad (2)$$

, где L(S’) – количество бит, требуемое для представления закодированного сообщения; L(S) – количество бит, требуемое для представления исходного сообщения.

Из теории, основы которой были заложены К. Шенноном, следует, что в системе представления информации с основанием m символу a_k, вероятность появления которого равна $p(a_k)$, оптимально и, что особенно важно, теоретически допустимо ставить в соответствие код длины:

$$-\log_m p(a_k) \qquad (3)$$

Из этого утверждения в частности следует, что символ с высокой вероятностью должен кодироваться несколькими битами, тогда как маловероятный требует многих битов. Также данная формула показывает, почему арифметическое кодирование в общем случае превосходит по эффективности префиксные методы, которые, в отличие от первого, кодируют символы с помощью

целого количества бит, что снижает степень сжатия и сводит на нет точные предсказания вероятностей. В свою очередь, метод арифметического кодирования обеспечивает теоретически неограниченную точность.

Рассмотрим следующий пример. Закодируем слово из алфавита {0, 1} с использованием метода арифметического кодирования, и сравним показатель ML с аналогичным показателем для метода Хаффмана.

Пусть дан текст S = “1110”.

Символ	Вероятность	Интервал
1	3/4	[0.0, 3/4)
0	1/4	[3/4, 1.0)

Применим метод арифметического кодирования:

В начале	[0.0,	1.0	)
*После просмотра **"1"***	[0.0,	3/4	)
*После просмотра **"1"***	[0.0,	9/16	)
*После просмотра **"1"***	[0.0,	27/64	)
*После просмотра **"0"***	[81/256,	27/64	)

Тогда для кодирования нам подойдет значение 3/8. В двоичном коде – 0.011.
Итого величина ML(S) = 3 бита/ 4 символа = 0.75.

Замечание. Для кодирования мы вполне могли выбрать и другое число, например, 81/256. Тогда длина кода была бы равна не трем, а восьми битам, так как 81/256 в двоичной форме представляется как 0.01010001. Ясно, что в этом случае мы получили бы крайне неэффективное информационное представление.

При кодировании методом Хаффмана, и на 0 и на 1 придется тратить не менее одного бита: ML(S) = 4 бита/ 4 символа = 1.

Итак, мы видим, что среднее количество бит на единицу информации для арифметического кодирования получилось существенно меньшим, чем для кодирования Хаффмана. Показанный пример демонстрирует преимущества арифметического кодирования. Так же его замечательным свойством является то, что эффективность кодирования с ростом объема данных лишь увеличивается, в отличие от остальных методов, эффективность которых неизменна при изменении входных параметров.

4. Адаптивный алгоритм

Для арифметического кодирования существуют также адаптивный алгоритм, т.е. алгоритм, который при каждом сопоставлении символу кода, кроме того, изменяет внутренний ход вычислений так, что в следующий раз этому же символу может быть сопоставлен другой код, т.е. происходит “адаптация” алгоритма к поступающим для кодирования символам. При декодировании происходит аналогичный процесс. Необходимость применения адаптивного алгоритма возникает в том случае, если вероятностные оценки для символов сообщения неизвестны до начала работы алгоритма.

Построение арифметического кода для последовательности символов из заданного множества можно реализовать следующим алгоритмом. Каждому символу сопоставляется его вес, вначале вес для всех равен 1. Все символы располагаются в естественном порядке, например по возрастанию. Вероятность каждого символа устанавливается равной его весу, деленному на суммарный вес всех символов. После получения очередного символа и постройки интервала для него, вес этого символа увеличивается на 1. Для того чтобы обеспечить остановку алгоритма распаковки вначале сжимаемого сообщения, надо поставить его длину или ввести дополнительный символ-маркер.

Пример: произведем кодирование слова “ABBACD”.

На первом шаге веса всех символов равны единице. Текущий интервал [0,1]. Первым кодируется символ A. Так как символов 4, и их веса одинаковые, то мы берем четверть от исходного отрезка. Получается интервал [0, 1/4]. Символ A обработан, увеличиваем его вес на единицу. Общий вес так же увеличился на 1. Следующим кодируется символ B. Разбиваем текущий интервал соответственно отношениям весов символов к весу всех символов. Выбираем интервал, соответствующий символу B - [1/10, 3/20]. Увеличиваем вес символа B на единицу, и так далее. Все шаги подробно описаны в таблице 1.

Веса символов				Общий вес	Кодируемая буква	Текущая длина промежутка	Получившийся интервал
A	B	C	D				
1	1	1	1	4	A	1	[0, 1/4]
2	1	1	1	5	B	1/4	[1/10, 3/20]
2	2	1	1	6	B	1/20	[7/60, 8/60]
2	3	1	1	7	A	1/60	[7/60, 51/420]
3	3	1	1	8	C	1/210	[202/1680, 203/1680]
3	3	2	1	9	D	1/1680	[1826/15120, 1827/15120]

Табл. 1

1979/16384 = $1979/2^{14}$, отсюда получаем, что исходное сообщение можно представить двоичным числом 0.00011110111011_2 є [1826/15120, 1827/15120]. Таким образом, мы закодировали сообщение с помощью 14 битов. ML = 2.3 бит/сим.

Декодирование происходит следующим образом: на каждом шаге определяется интервал, содержащий данный код – по этому интервалу однозначно задается исходный символ сообщения. Затем из текущего кода вычитается нижняя граница содержащего интервала, полученная разность делится на длину этого же интервала. Полученное число считается новым текущим значением кода. Получение маркера конца или заданного перед началом работы алгоритма числа символов означает окончание работы.

Пример: распакуем код 00011110111011, зная множество символов, из которых состояло исходное сообщение. Итак, 00011110111011_2 = 1979/16384.
Результаты расчетов приведены в таблице 2.

Веса символов				Число-код и его интервал	Декодированный символ	Длина интервала
A	B	C	D			
1	1	1	1	1979/16384 є [0, 1/4]	A	1/4
2	1	1	1	1979/4096 є [2/5, 3/5]	B	1/5
2	2	1	1	1703/4096 є [1/3, 2/3]	B	1/3
2	3	1	1	1013/4096 є [0, 2/7]	A	2/7
3	3	1	1	7091/8192 є [6/8, 7/8]	C	1/8
3	3	2	1	7576/8192 є [8/9, 1]	D	1/9

Табл. 2

Таким образом, из кода 00011110111011_2 мы восстановили исходное сообщение (ABBACD).

Литература

1. Лидовский В. В. Теория информации: Учебное пособие. – М.: Компания Спутник+, 2004.

2. Семенюк В. В. Экономное кодирование дискретной информации. – СПб.: СПбГИТМО (ТУ), 2001 (доступна на http://www.compression.ru).

3. Witten I. H., Neal R. M., Cleary J. G. Arithmetic Coding for Data Compression, CACM, 1987 (доступна на http://www.compression.ru).