Implementation of a Simple PPMC Text Compression Algorithm

Chaoping Deng

Electrical and Computer Engineering

Cornell University

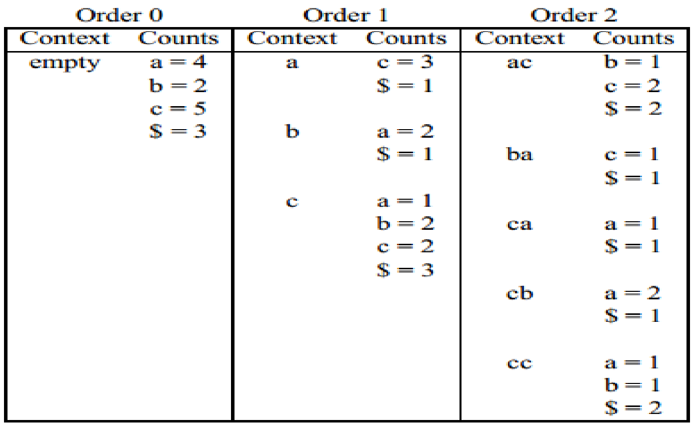
Introduction

Prediction by partial matching (PPM) is an adaptive statistical data compression technique introduced by John Cleary and Ian Witten in 1984. It is based on prediction and context modeling. PPM predicts the next symbol in stream by using previous uncompressed symbols in the stream. The main idea of PPM is to use the previous m characters to generate a conditional probability of the current character. The conditional probability of character x following string s is given by

$$p(x|s) = N(x|s) * N(s)$$

Where N(x|s) represents number of times that x appears after string s and N(s) represents the number of times that string s appears. Each character will have its own probability constructed based on the corresponding context. The probability distribution is then encoded by an arithmetic encoder. For decoding, the process is reversed. The arithmetic decoder will first retrieve the probability for each character and reconstruct each character in order based on the probability assigned by each character's context.

PPMC is a derived from PPM and the only difference is PPMC uses different rules for escape character. PPMC uses the number of different characters seen following the given context as the escape number. (An escape character is the identifier for the decoder to know where to move down a size for the context.) A simple demonstration of PPMC is as following:



Order 0		Order 1		Order 2	
Context	Counts	Context	Counts	Context	Counts
empty	a = 4	a	c = 3	ac	b = 1
	b = 2		$ = 1		c = 2
	c = 5				$ = 2
	$ = 3	b	a = 2		
			$ = 1	ba	c = 1
					$ = 1
		c	a = 1		
			b = 2	ca	a = 1
			c = 2		$ = 1
			$ = 3		
				cb	a = 2
					$ = 1
				cc	a = 1
					b = 1
					$ = 2

Figure 1: PPMC for string "accbaccacba"

As shown in the graph, $ represents the escape character. The probability for context ac can therefore be computed as, P(b following an ac) = 1/5, P(no match following ac) = 2/. Both encoder and decoder interpret the text as of order 2, which means two symbols as context for the following character. Therefore, whenever there is a need for a decreased size of context, the encoder need to put an escape character to notify the decoder where to find the corresponding character based on the probability. For example, in order to decode the first a, the encoder needs encode two $ before an “a”.

Implementation

There are two parts of the program, a 30-bit arithmetic encoder/decoder and PPMC encoder/decoder. The PPMC scheme refers to the combination of the two. The program implements an order-4 PPMC algorithm.

PPMC encode/decode

The function of PPMC encode/decode is to generate/recover from the probability distribution according to the basic scheme discussed in introduction.

String encoding

String (character)	Encoded
‘a’ to ‘z’,’A’ to ‘Z’	“%s” % c
‘ ‘	““‘\\s’’’
‘\r’	“’\\r’’’
‘\n’	“\\n’’’
‘\t’	“\\\t’’’
“\’’’	““\\’’’’
‘\’’’	‘\\\’’’’
other	‘\\x%02X’%ord(other)

Data Structure

Dict: an object that represents a collection of symbols.

Buddle: a list of matching strings named predictors

Exclusions: it contains the possibilities of certain characters from a shorter context when switching down happens. In the example given above, when ‘a’ is encoded, two escapes will be sent but ‘c’ will be excluded from the counts in the zero length context The algorithm should be able to readjust the probability and give ‘a’ a probability of 2/3 instead of 4/11

Encoder Structure

The following general scheme is followed:

```
  While (not end of file)
          readSymbol;
          shorten context
          while (context not found and context length valid)
                  output(escape sequence)
                  shorten context
          output(character)
          while (context length valid)
                  increase count of character
                  shorten context
```

The implementation is given as:

```
While(not end of file){
  While (String not found){
        reduce predictor string length (abandon the first symbol)
}
  Set bundle containing the predictor string
  While (bundle is not empty){
     Decide if bundles have all symbols excluded
     If yes, shorten the bundle and restart the loop
   If(symbol found in bundle)
   For(each symbol) comptue probability,upper bound and lower bound  of a symbol and
compress using arithmetic encoder
```

```
        else output(escape character) into arithmetic encoder, append the exclusion list and shorten the bundle length
    }
    If (bundle empty) {Directly encode the current character }
    Update the bundles
    Ensure the predictor string’s length is no more than the order, which is 4 in this project
}
```

The decoding process follows the idea of encoding and is further explained in the code.

Arithmetic Encode/Decode

The function of arithmetic encode/decode is to compress/decompress the probability distribution generated by the PPMC encoder/decoder. This is a bitwise arithmetic encoder/decoder implementation and 8-byte hexadecimal representation are used for all the numbers. It follows the general rule of arithmetic data compression algorithm except for the following unique features of this project:

The encoded data are rescaled to ensure that the mappings from bundle space are unique.

The rescaling follows the formula:

$$upbound = lowbound + (range * upbound + bundle\_extent - 1)/ bundle\_extent - 1$$

The hexadecimal data was checked in the following way:

If the most significant bit of upper bound and lower bound are not equal, proceed with the check. If the upper bound has 1 in the most significant bit and the lower bound has 1 in the second most significant bit (0 in more significant bit), upper bound is no less than 0xc0000000

lower bound is no more than 0x3fffffff.

Performance

The program can be run by calling python eyre.cd (compressed file) using python compiler 2.7.6. It compresses the file eyre.txt from 1MB to 291KB.

Acknowledgements

1. Bellochm, Guy., *Introduction to Data Compression*, Retrieved from http://www.cs.duke.edu/courses/spring11/cps296.3/compression.pdf

2.Drinic,Kirovski & Potkonjak, *PPM Model Cleaning,* Retrieved from

http://www.cs.ucla.edu/~miodrag/papers/Drinic_DCC_03.pdf

3.PPM, *CAP5015*, University of Central Florida, Retrieved from

http://www.cs.ucf.edu/courses/cap5015/ppm.pdf

4.Mackay, David. *Compression Algorithms in Python*, Retrieved from http://www.inference.phy.cam.ac.uk/mackay/python/compress/

5.PPMC, Retrieved from

http://www.stringology.org/DataCompression/ppmc/index_en.html

6.DIZ: PPMC compressor, Retrieved from

http://encode.ru/threads/1583-diz-py-release

7.Mahoney,Matt. Data Compression Programs. Retrieved from http://cs.fit.edu/~mmahoney/compression/

8.Crook:A new Binary PPM compressor. Retrieved from http://encode.ru/threads/1504-Crook-a-new-binary-PPM-compressor