UNIVERSITY OF CAMBRIDGE

ENGINEERING PART IIA

3F7 FULL TECHNICAL REPORT

CONTEXT-ADAPTIVE ARITHMETIC CODING

Name: Mihai Varsandan
College: Girton College
CRSiD: mv436
Date: 5/12/2018

ABSTRACT:

The aim of this report is studying the implementation and performance of adaptive and context-adaptive arithmetic coding compared to other coding algorithms such as Huffman, Shannon-Fano and normal arithmetic coding. The conclusion of the report is that the performance of the adaptive arithmetic coding is worse than the non-adaptive counterpart but still much better than the Huffman and Shannon-Fano coding., while the performance of the context-adaptive method is proved to be the best method as it performs better than the all other compression methods.

1. Introduction

In the short report, the other 3 methods of compression were studied. The report showed that the best method, which resulted in the best compression ratio, is the arithmetic algorithm. However, the main problem with these algorithms is that it requires you to know all the data and therefore knowing the probability distribution of the whole data. This could be useful if may want to send a text to someone or a file containing the text, but for situations where you don't know the data such as live streaming the adaptive algorithm allows to compress the data. This is because the algorithm can calculate the frequencies 'on the go' while encoding and therefore it does not require extra knowledge. The encoding stops when a special symbol the means end of transmission is seen. The way to reduce further the compression ratio is to look at conditional probability. This way a new method called context-adaptive method is introduced.

2. Method

For this section please see the files attached together with the report. The adaptive arithmetic encoder and decoder can be seen in the file **arithmetic_ftr_adaptive.py**, while the context_adaptive method can be seen in **arithmetic_ftr.py**. The modified camzip and camunzip files to take into account the context-adaptive method can be seen in **camzip.py** and **camunzip.py**.

ADAPTIVE METHOD

ENCODING

First, two dictionaries are initialised containing all the 256 characters along with the frequency count of 1, and the other dictionary containing again the 256 characters with a uniform probability of 1/256. (lines 16-18 in **arithmetic_ftr_adaptive.py)**

The symbol for the end of transmission is: ◇ which is the character 4 in the ASCII code. This is added and the end of the text that is needed to be compressed.(line 23 in the **arithmetic_ftr_adaptive.py.)**

Next, the characters are compressed one by one. Each time the CDF is recalculated based on the updated model which is done after calculating the *hi* and *lo* values so that they would not be biased unnecessarily. Calculating the CDF can be seen in lines 30-33 while the model update where the in the probability and frequency dictionary are updated with the new character can be seen in lines 57-59 in the **arithmetic_ftr_adaptive.py**.

The encoding is stopped when the end of transmission character is seen.

DECODING

For the decoding, the same principles and functions used for the encoder are also used here. Therefore the definition of the initial probability and updating the model is the same as in the encoder part. This can be seen in lines 131-133, 155-159 and 184-186 in **arithmetic_ftr_adaptive.py**.

As the decoder does not know the length of the file it has to decode, the decoding stops when the character decoded is the End of Transmission symbol.

CONTEXT-ADAPTIVE METHOD

So the main strategy for this was to use a nested dictionary in which every 256 characters are the keys to another dictionary containing again the 256 characters but this is with each one having the frequency count. From this, a nested probability dictionary was created. So the main idea for this approach was the if ‘a’ was the previous letter and now were are encoding letter ‘e’ we look at the dictionary that has the key a and update the frequency of letter ‘e’ than if the next letter is ‘r’ but this time we look at the dictionary that has the key ‘e’.

ENCODING

First, the nested dictionary is initialised and a uniform cdf is created based on the 256 characters(lines 13-22 in **arithmetic_ftr.py**)

Next, the characters are compressed one by one. For the first character, the uniform cdf is used while for the rest of the characters the cdf is calculated each time based on the dictionary of frequencies of the previous character. (lines 57-70 in **arithmetic_ftr.py**)

DECODING

For the decoder, the same principles used for the encoder also applies to the decoder. (lines 146-155,175-188 in **arithmetic_ftr.py**)

CAMZIP AND CAMUNZIP

In the camzip algorithm, the two new methods were added that included the context-adaptive and adaptive arithmetic encoder. This can be seen in the lines 32-36 and therefore the new file contains the ending is ‘czad’ and ‘czca’. Also now the camzip function will also return the entropy, size, and compression ratio of the respective method for a file that is compressed (lines 65-69).

In the camunzip function, it was added so that the function would recognise that ‘czad’ or ‘czca’ compression file was done via adaptive or context-adaptive arithmetic algorithm and therefore the correspondent decoder is used(lines 17-20,47-54).

3. Results

Compression Ratios using different methods for multiple data sets

			Compression Ratios (Bits per byte)				
File	**Size(KB)**	**Entropy(bits)**	**Shannon - Fano**	**Huffman**	**Arithmetic Coding**	**Arithmetic Adaptive**	**Context-Adaptive**
Hamlet.txt	207	4.44986	4.81818	4.47266	4.44990	4.46169	3.62791
Alice29.txt	148	4.51287	5.05357	4.55535	4.51295	4.52868	3.82416
Asyoulik.txt	125	4.80811	5.31840	4.84471	4.80815	4.82636	3.81706
Cp.html	11.1	5.00769	5.53040	5.04179	5.00807	5.13578	Error
Grammar.lsp	3.72	4.63226	5.19430	4.66541	4.63316	4.94275	5.25020
lcet10.txt	419	4.62271	5.18347	4.65375	4.62272	4.62894	3.72989
plrabn12.txt	471	4.47713	4.98975	4.519617	4.47715	4.48289	3.58229
Xargs.1	4.23	4.89843	5.427963	4.92453	4.89992	5.18003	5.61533

Table 1

4. Discussion

ADAPTIVE METHOD

The results show that the adaptive arithmetic algorithm performs worse than the non-adaptive arithmetic algorithm. This is only because the adaptive algorithm does not have access to all the information and therefore its probability distribution is not ideal. However, if the file size is big such as the lcet10.txt, then it can be seen the adaptive one gets closer to the non-adaptive. This is because as the data is getting bigger the probability distribution will get close to the true distribution.

However, it can be observed that the adaptive arithmetic coding is better than Shannon-Fano method of compression 100% of the time. By comparing the adaptive method to the Huffman method it can be observed that for small file size the Huffman performs better while for large file size the adaptive method has a better compression ratio. The reason for this is again the fact for a small data the adaptive method does not have time to improve its probability distribution as the Huffman has already pre-calculated it.

CONTEXT-ADAPTIVE METHOD

Looking at the context-adaptive method now, it can be observed that its entropy is much better than all other cases almost all the time. This is because it is known that H(X) > H(X/Y) and also it also known that English text non-i.i.d. The only time when the context-

adaptive is worse than the all other methods is when the data sets are small. This is because as the data sets are small the uniform distributions are used throughout the encoding as the algorithm does not have enough time to update its frequency count and thus the respective cdf so it performs worse. It can be seen that is even worse than the adaptive method. This is because a nested dictionary is used, so, updating the frequency count to reasonable probability distribution would take longer.

One disadvantage of the context-adaptive method is that the data needs to be non- i.i.d.. So the method would perform much worse if a data containing non-text data would have to be compressed.

5. Conclusion

- The adaptive method was seen to perform very close to the non-adaptive method provided that the data is large enough. The difference in the compression ratio is due to the fact that the adaptive method does not have knowledge of the complete probability distribution of the data

- The adaptive method is better than Shannon-Fano all the time and it is better than the Huffman only when the size of the data is big enough to for the adaptive method to be able to get close to the true probability distribution

- The main advantage of the adaptive method is that it can compress data even when the length of the data is unknown. This is very useful for streaming data.

- The context-adaptive method was seen to perform much better than all the other methods provided that again the data is large enough. The problem with small data compression is the fact that the uniform distribution is used multiple times and the algorithm does not have enough time to improve its probability distribution.

- The main advantage of the context-adaptive method is that its compression ratio is much better and again it can compress the data even when the length of the data is unknown. The main disadvantage is that non-i.i.d. datasets would result in worse compression ratios.