ELEN060-2 - Information and coding theory

Project 2 - Source coding, data compression and channel coding

April 2023

The goal of this second project is to apply some of the principles seen in the lectures about source coding, data compression and channel coding. We ask you to write a brief report (pdf format) collecting your answers to the different questions. All codes must be written in Python inside the Jupyter Notebook provided with this assignment, no other code file will be accepted. Note that you can not change the content of locked cells or import any extra Python library than the ones provided (except for plotting the results or when explicitly written).

The assignment must be carried out by groups of two students and the report and the notebook should be submitted on Gradescope (https://www.gradescope.com/) before May 10 23:59. Note that attention will be paid to how you present your results and your analyses. By submitting the project, each member of a group shares the responsibility for what has been submitted (e.g., in case of plagiarism). From a practical point of view, every student should have registered on the platform before the deadline. Group, archive and report should be named by the concatenation of your student ID (sXXXXXX) (e.g., s000007s123456.pdf and s000007s123456.ipynb).

Implementation

1. Implement a function that returns a *binary* Huffman code for a given probability distribution. Give the main steps of your implementation. Verify your code on Exercise 7 of the second exercise session (TP2), and report the output of your code for this example. Explain how to extend your function to generate a Huffman code of *any (output)* alphabet size.
2. Given a sequence of symbols, implement a function that returns a dictionary and the encoded sequence using the *on-line Lempel-Ziv algorithm* (see *State of the art in data compression*, slide 50/53). Reproduce and report the example given in the course.
3. Compare (without implementing the basic version) the two versions of the Lempel-Ziv algorithm seen in the theoretical course (i.e., the basic and the on-line versions). Discuss what are the (practical) advantages and drawbacks of each version.
4. The *LZ77* algorithm is another dictionary algorithm. It consists in replacing repeated sequences of symbols with references to a previous occurrence of the same sequence of symbols. Typically, the encoder considers a sliding window search buffer of size *window_size* and a look-ahead buffer (you can constrain the size of the look-ahead buffer if you wish, but pay attention to choose a relevant size and to

specify it in your report), and searches for past occurrences of the beginning of the look-ahead buffer (i.e, the prefix) within the search buffer. Each codeword is made of three elements: the offset (i.e, distance) of the longest prefix in the search buffer, the length of the prefix, and the symbol following the prefix. Note that the offset and the length are in practice rewritten in binary.

Implement a function that returns the encoded sequence using the *LZ77* algorithm as described by the algorithm below given an input string and a sliding window size. Reproduce the example given in Figure 2 with *window_size*=7.

Algorithm 1: LZ77 compression algorithm sliding window

```
Result: Write here the result
A sliding window size l;
An input string;
while input is not empty do
    prefix := longest prefix of input that begins in window;
    if prefix exists in window then
        d := distance to the start of the prefix;
        p := length of prefix;
        c := char following the prefix in input;
    else
        d := 0;
        p := 0;
        c := first symbol of input;
    end
    append (d, p, c) to encoded input;
    shift the sliding window by p + 1 symbols (i.e., discard p + 1 symbols from the beginning of
      window and add the p + 1 first symbols of the input at the end of the window).
end
```

7	6	5	4	3	2	1							output
							a	b	r	a	c	ada...	(0,0,a)
						a	b	r	a	c	a	dab...	(0,0,b)
					a	b	r	a	c	a	d	abr...	(0,0,r)
				a	b	r	a	c	a	d	a	bra...	(3,1,c)
		a	b	r	a	c	a	d	a	b	r	ad...	(2,1,d)
a	b	r	a	c	a	d	a	b	r	a	d		(7,4,d)
a	d	a	b	r	a	d							
sliding window							look-ahead buffer						

Figure 2 - Example of the encoding of the string *abracadabrad* using the LZ77 algorithm. Taken from this source: http://jens.jm-s.de/comp/LZ77-JensMueller.pdf

Source coding and reversible (lossless) data compression

An image is a visual representation of an object or scene that can be captured and stored in a digital format. For grayscale images, each pixel is assigned a brightness value between 0 and 255 to represent the different shades of gray. One popular format for storing and transmitting grayscale images is the Portable Network Graphics (PNG) format, which is a lossless compression format. This allows PNG images to be smaller in size than raw image formats without sacrificing image quality. In this assignment, you are given a pixel.txt containing the original raw grayscale image pixels of size 512x512 encoded in bytes, where the pixels are ordered from left to right and top to bottom. Additionally, you are provided with the same image that has been encoded in the PNG format and transformed in bytes. For simplicity each byte is represented in the files with its corresponding hexadecimal representation.

5. Write a function to read and display both images (you may use external libraries). What is the number of symbols required to represent all possible images in both image representations? What is the length (in bytes) of both files?
6. Estimate the marginal probability distribution of all symbols (in hexadecimal representation) from the given PNG representation sequence of the image, and determine the corresponding binary Huffman code and the encoded PNG sequence. Give the total length of the encoded PNG sequence and the compression rate.
7. Give the expected average length for your Huffman code. Compare this value with (a) the empirical average length, and (b) theoretical bound(s). Justify.
8. Plot the evolution of the empirical average length of the encoded PNG using your Huffman code for increasing input sequence lengths.
9. Encode the PNG sequence using the *on-line Lempel-Ziv algorithm*. Give the total length of the encoded sequence and the compression rate.
10. Encode the PNG sequence using the *LZ77 algorithm* with *window_size*=7. Give the total length of the encoded sequence and the compression rate.
11. Famous data compression algorithms combine the LZ77 algorithm and the Huffman algorithm. Explain two ways of combining those algorithms and discuss the interest of the possible combinations.
12. Encode the PNG using one of the combinations of LZ77 and Huffman algorithms you proposed in the previous question. Give the total length of the encoded PNG sequence and the compression rate.
13. Report the total lengths and compression rates using (a) LZ77 and (b) the combination of LZ77 and Huffman, to encode the PNG sequence for different values of the sliding window size (use sliding window sizes from 1 to 11000 with a step of 1000). Compare your result with the total length and compression rate obtained using the on-line Lempel-Ziv algorithm. Discuss your results.
14. Instead of encoding the PNG sequence, encode directly the pixel values with the binary Huffman algorithm. Give the average expected length, the experimental length of the encoded text and the compression rate.
15. Compare the values found at the previous question with the ones found in Question 5. In particular, is it better to first encode the text with PNG code before the Huffman encoding or to directly encode the pixel image with Huffman? Discuss.

Channel coding

Consider a text document that is sent through a noisy communication channel. This document may contain important information such as business reports, academic papers, or personal messages. The text file may be encoded using ASCII, which uses 8 bits per character. This means that each character in the text document can be represented by a sequence of 8 binary digits, or bits. The channel used for transmission is a binary symmetric channel, which means that the transmitted bits can be flipped with a certain probability of error. The probability of error for this channel is 0.02, which means that 2% of the transmitted bits are expected to be flipped during transmission. This can result in errors in the received text, which can lead to misunderstandings or loss of important information. Therefore, it is important to develop techniques to ensure the accurate and reliable transmission of the text document over the noisy communication channel.

16. Implement a function to read the text document and encode the text signal using the binary ASCII fixed-length binary code. Count the number of bits required for the text provided with this assignment (text.txt).
17. Simulate the channel effect on the binary text signal. Then decode the text signal and display the decoded text. What do you notice?
18. Instead of sending directly through the channel the binary text signal, you will first introduce some redundancy. To do that, implement a function that returns the Hamming (7,4) code for a given sequence of binary symbols. Then, using your function, encode the binary text signal (from question 16).
19. Simulate the channel effect on the binary text signal with redundancy. Then decode the binary text signal. Display the decoded text. What do you notice? Explain your decoding procedure.
20. Given the text document encoded in binary ASCII, implement a Python program to simulate transmission over a binary symmetric channel with a noise probability ranging from 0 to 0.5, with a step of 0.01. Compute the per character error rate for each noise probability, both with and without Hamming (7,4) code. Plot the per character error rate as a function of the noise probability, and compare the performance with and without Hamming (7,4) code.
21. How would you proceed to reduce the probability of errors and/or to improve the communication efficiency? Justify.