ELEN060-2 - Information and coding theory

Project 2 - Source coding, data compression and channel coding

March 2025

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.

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 9 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 5 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. Describe the decoding principle of the LZ77 algorithm.

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

In this part of the project, you will experiment with the compression algorithms implemented previously on two different texts. The first text (english_text.txt) is an excerpt from a well-known English book. For the sake of simplicity, there are no uppercase letters and no punctuation. Hence there are only 27 different characters: the 26 letters of the alphabet and the white space. The second text (encrypted_text.txt) is an encrypted version of the first one. The Vigenère cipher (https://en.wikipedia.org/wiki/Vigenère_cipher) has been used as an encryption algorithm. The key that has been chosen is “entropy”. The example below briefly illustrates how the algorithm works (you can also read the wikipedia page if you feel interested, but it is not mandatory). The message to be encrypted is “thisismymessage”. Since the first symbol of the key is “e” which is the 4th letter of the alphabet (counting from 0), the first letter of the message (“t”) is shifted from 4 positions and becomes a “x” in the encrypted message. Then, since the second symbol of the key is “n” which is the 13th letter of the alphabet, the second letter of the message (“h”) is shifted from 13 positions and becomes a “u” in the encrypted message, etc. When the last letter of the key has been used, the first letter is used again. The algorithm stops when the entire message has been encrypted. Note that the white space is considered as the last letter of the alphabet here.

Message:	t	h	i	s	i	s	m	y	m	e	s	s	a	g	e
Shift:	e	n	t	r	o	p	y	e	n	t	r	o	p	y	e
Encrypted message:	x	u	a	i	w	g	j	b	z	x	i	f	p	d	i

5. In the notebook, estimate the marginal probability distribution of all symbols (the 26 letters and the white space) from the given English text, and determine the corresponding binary Huffman code and the encoded English text. In the report, give the total length of the encoded English text and the compression rate.
6. Give the expected average length for your Huffman code. Compare this value with (a) the empirical average length, and (b) theoretical bound(s). Justify.
7. Plot the evolution of the empirical average length of the encoded English text using your Huffman code for increasing input text lengths.
8. Encode the English text using the *on-line Lempel-Ziv algorithm*. Give the total length of the encoded english text and the compression rate.
9. Encode the English text using the *LZ77 algorithm* with *window_size*=7. Give the total length of the encoded text and the compression rate.
10. 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.
11. Encode the English text using one of the combinations of LZ77 and Huffman algorithms you proposed in the previous question. Give the total length of the encoded English text and the compression rate.
12. Report the total lengths and compression rates using (a) LZ77 and (b) the combination of LZ77 and Huffman, to encode the English text 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.

13. It is typically assumed that repetitions occur at long distances in a text. Based on your results in the previous question(s), discuss what could be the best data compression algorithm(s) and/or how to adapt the algorithms used in this project.
14. Instead of encoding the English text, encode the encrypted text with a) the binary Huffman algorithm and b) the LZ77 algorithm. Report the compression rate for each case.
15. Compare your results with the results found in Question 5 and 9. Then try to describe intuitively the impact of the key (e.g., the impact of its size, the impact of the number of different characters inside it, etc.) on your results.
16. If Alice wants to send a private message to Bob, should Alice compress her message before or after its encryption? Ignore the security aspect.

Channel coding

Let us consider a grayscale image that is sent through a noisy channel. Let us take a .jpg file *image.jpg* as an image signal. Its quantisation is such that possible values are between 0 and 255, and its number of pixels is *width x height*. The channel is a binary symmetric channel with a probability of error equal to 0.01. In order to send the image signal through the channel, the signal is first encoded in a binary alphabet and then each binary symbol is sent through the channel. The pixel values are read from left to right and from top to bottom as for conventional text reading in English.

17. Implement a function to read the image and display the original image. Remember that the image should be read in grayscale (one channel per pixel).
18. Encode the image signal using a fixed-length binary code. What is the appropriate number of bits? Justify.
19. Simulate the channel effect on the binary image signal. Then decode the image signal and display the decoded image. What do you notice?
20. Instead of sending directly through the channel the binary image 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 image signal (from question 14).
21. Simulate the channel effect on the binary image signal with redundancy. Then decode the binary image signal. Display the decoded image signal. What do you notice? Explain your decoding procedure.
22. How would you proceed to reduce the loss of information and/or to improve the communication rate? Justify