Information Theory — Assessed Assignment

Iain Murray

Due: 4pm Thursday 20 November, 2014

This assignment is out of 20 marks and forms 20% of your final grade.

Assessed work is subject to University regulations on academic conduct:
`http://tinyurl.com/UoEconduct`
Do not show your code, answers or write-up to anyone else.
Never copy-and-paste material into your assignment and edit it.

You should submit your answers on paper to the ITO office in Appleton Tower by the deadline. Handwritten submissions are acceptable if neat and legible.

Policy on computer code: You may write code in the programming language of your choice. You must hand in a print-out of your code, which will help assure me that you answered the questions yourself. However, your answers must be clear without the code: marks will only be awarded to the description of what you did and the numerical answers that you report along with that description. Rationale: that's how research, including your project, is assessed. See also point 2.3 of Alan Bundy's "Researcher's Bible":
`http://homepages.inf.ed.ac.uk/bundy/how-tos/resbible.html`

Late submissions: The School of Informatics policy is that late coursework normally gets a mark of zero. See `http://tinyurl.com/edinflate` for exceptions to this rule, e.g. in case of serious medical illness or serious personal problems. Any communications regarding late work should be taken up with the ITO, or via your Personal Tutor.

Part 1: Source coding

Obtain `thesis.txt` from the course website. This ASCII file contains $N = 344026$ characters from an alphabet of 27 characters: `a`–`z` and space. Assume that the length and alphabet are known. For arithmetic coding, the end-of-file character has zero probability, until after character $N = 344026$, when end-of-file has probability one.

Programming note: To answer this question you will need to write computer programs. However, you will not need to implement or even run an actual compression system to compute how the systems discussed will perform.

Checking: This part is meant to be relatively straight-forward (meet with me if it is not). But are you sure that your numerical answers are correct? If your numbers don't coincide exactly with mine, I will want to know why. Does your code give correct answers on trivial files with one or two characters? Do your probability distributions sum to one? A wrong answer with no evidence of checking is liable to obtain zero marks for that part.

1. **Character statistics:** write a computer program to read in the file and count how many times each letter in the alphabet appears. Normalizing these counts will give the probability distribution $p(x_n)$ of a character chosen randomly from the file. Compute the entropy $H(X_n)$ of this distribution and report it in bits to 3 significant figures.

   [1 mark]

2. **Bigram statistics:** compute the distribution over pairs of adjacent characters $P(x_n, x_{n+1})$ corresponding to selecting a character x_n uniformly at random from the first $(N-1)$ characters, and also reading in the next character x_{n+1}. While explaining your computations:

(a) Report the joint entropy of this distribution $H(X_n, X_{n+1})$ in bits to 3 sig. figures.
(b) Explain why your answer is (or should be!) less than $2H(X_n)$.
(c) Report the conditional entropy $H(X_{n+1} \mid X_n)$ in bits to 3 significant figures.

(Conditional entropy will be covered in the 'week 6' notes, or read MacKay p138.)

[2 marks]

Some real arithmetic encoder implementations perform less well than was described in the lectures and in the MacKay chapter due to precision limitations in their implementation. For the following questions assume ideal behaviour, with no limitations on precision.

3. **Compression with known distributions:** assume that both the sender and receiver of a compressed file (somehow) know the distribution you computed in question 1. By using the (wrong) model that the characters are generated i.i.d. with $p(x_n)$, what is the maximum number of bits an arithmetic coder might use to represent `thesis.txt`?

   Hoping to improve performance, the sender and receiver now use a more sophisticated model. The first character is generated from $p(x_n)$, all subsequent characters are generated from $p(x_{n+1} \mid x_n)$ computed from the joint probability in question 2. Again report the maximum number of bits that an arithmetic coder might need to encode `thesis.txt` using this known, fixed model.

   Explain the steps of the calculations you performed.

   (An integer is expected. Pay careful attention to whether you should round up or down.)

   [2 marks]

4. **Compression with limited precision header:** in a real system the compressed file must contain a description of the model in use. The sender and receiver decide to use a simple (albeit suboptimal) scheme: each probability is encoded to the next largest multiple of 2^{-8} using 8 bits, $q_i^* = \lceil 2^8 \, p_i \rceil / 2^8$. These 'probabilities' will no longer sum to one, so they are renormalized before use: $q_i = q_i^* / \sum_j q_j^*$.

   How many bits might be required to encode `thesis.txt` using the models in question 3, but using rounded and renormalized distributions? Explaining the steps you took, report the size of the header, the compressed data and their total: the compressed file size. (There is more than one answer: any consistent, well-explained scheme will be given full credit.)

   [2 marks]

5. **Compression with adaptation:** an alternative to including a header is for both the sender and receiver to infer the distributions as they read and decode the file. The Laplace prediction rule for the i.i.d. model is:

$$P(x_{n+1} = a_i \mid \mathbf{x}_{\leq n}) = \frac{k_i + 1}{n + |\mathcal{A}|},$$

   where k_i is the number of times character a_i has occurred so far in $\mathbf{x}_{\leq n}$, and $|\mathcal{A}| = 27$ is the size of the alphabet. Although better models are available, a possible prediction rule for the bigram model is:

$$P(x_{n+1} = a_i \mid x_n = a_j, \mathbf{x}_{<n}) = \frac{k_{i|j} + 1}{n_j + |\mathcal{A}|},$$

   where $k_{i|j}$ is the number of times character a_i has previously appeared after character a_j, and n_j is the number of times we have previously made predictions conditioned on a_j being the previous character.

   How many bits might be required to compress `thesis.txt` using each of these predictions rules? Explain how you obtained your answers.

   [2 marks]

Part 2: Noisy Channel Coding

We won't have covered everything relevant to these questions until around November 7th. However you should have no trouble reading ahead and starting earlier than then if you wish. Questions 6 and 7 contain every detail for you to work out how to do them from scratch (and points you to pseudo-code). The point of question 8 is to be creative, outside what we cover in class, although working through the Hamming code material that we'll cover will help.

Digital Fountain Codes are codes for the erasure channel. In the binary erasure channel some bits are replaced with question marks. In general it may be whole packets that either arrive intact or not at all. In the next two questions we will implement a decoder for the LT code reviewed in Chapter 50 of the MacKay text. This is a necessary first step if you wish to empirically investigate this code further.

6. **XOR-ing packets:** Many codes use the `XOR` operator, addition modulo 2 applied to corresponding bits in two packets. Work out how to do bit-wise `XOR` in your programming environment. The answer is `bitxor` in Matlab, and the '`^`' operator in C or Python. Also work out how, if necessary, to convert between ASCII characters and numbers that your language will let you `XOR`. For example to bitwise `XOR` '`Z`' and '`a`' and output the result, '`;`', as a character you would do:

```
chr(ord('Z') ^ ord('a')) # in Python
char(bitxor(uint8('Z'), uint8('a'))) % in Matlab
printf("%c\n", 'Z' ^ 'a'); /* in C */
```

To display the answer as its ASCII integer value, 59, you could leave off the outer character conversion in Python/Matlab or change the C printf statement to use "`%d`".

As a warm-up exercise, `XOR` the ASCII string:

```
nutritious snacks
```

with bytes with numerical values:

```
59 6 17 0 83 84 26 90 64 70 25 66 86 82 90 95 75
```

and report the ASCII string that results. [1 mark]

7. **Decoding packets from a digital fountain:** In this question you obtain packets of length 8 bits, i.e., bytes. Usually the packets would be longer. The packets you managed to receive had the following values (in decimal):

```
79 68 96 55 112 114 65 75 65 103 94 68 27 59 43 115 103 55 20 84 54 69 100
```

These 23 numbers are also in `received.txt` on the website, or as a series of bytes in the 23-byte binary file `received.dat`.

The packets resulted from adding together particular source bytes of the original message, modulo 2. The 23 rows of `packets.txt` (also listed at the end of the assignment) give, in order, the identities of the source bytes that were used for each of the 23 received packets.

As the last received packet only used the 8th source packet, the 8th character of the original message must be ASCII value 100: '`d`'. The second received packet, `68`, was the `XOR` of source packets 8 and 10. As you now know source packet 8, you should be able to deduce that the 10th source packet was a space character.

Write a computer program to continue propagating your knowledge about the source message and to decode the message completely. The pseudo-code on MacKay p591 might help. Give a high-level overview of how your code works. Don't worry if it isn't optimized/efficient, as long as it works. Report the decoded string and which of the received packets you actually used.

[3 marks]

The next question asks you to consider coding for a channel with additive noise, rather than the erasure channel.

8. **Creating a code:** Invent an error correcting code that could be used with the binary symmetric channel (BSC). It is acceptable if 'your' code already exists, but the code

must not be a repetition code or a Hamming code. (If you wish, combinations of these codes would be ok.) I require a good description, in your own words, of how your code works. You should include both encoding and decoding algorithms.

What is the rate and bit error probability under your code? Consider flip probabilities $f = 0.4$, 0.1, and 0.001. You may compute, estimate or even just bound the properties of your code, either analytically or numerically. You may not claim that the bit error probability of a code is approximately zero however. I'd like a meaningful approximation that can be placed on a log scale.

Bit error: Be sure to estimate the *bit* error probability of your code, rather than a *block* error probability. See Exercise 1.6 (MacKay, p13 with solution p17) for an example of the distinction, which assumes you know about the Hamming code (p8).

Scope: You do *not* need to create a good code to obtain full credit. Coming up with good codes is *hard*. The intention is to get you to think about what encoding and decoding involve, and what makes noisy channel coding challenging. You may wish to refer to MacKay's exercises 1.9–1.11 on p14, although the 'solutions' to those exercises may lead you into considering options more complicated than I require.

Ground rules: using an existing implementation of an error correcting code, or a library designed for constructing codes, is not acceptable. I want you to fully implement and understand something yourself.

[7 marks]

A copy of the contents of `packets.txt` follows. The rows give, in order, the identities of the source packets that were used in each received packet in Question 7:

```
6 15 16
8 10
2 10 12
1 2 4 8 11 13 18
3 9 11 15
1 2 4 7 10 14
1 10 17
4 10 14 18
1 2 12 13 14
5 7 14
1 5 14 15 18
4 6 11
3 7 9 15 16
1 2 11 13 15
1 7 9 13
3
7 8 10 13 14
8 9 11 17
1 5 13
7 8 10 14
7 8 10
4 7 8 10
8
```