# Information Theory — Assessed Assignment


### Iain Murray and Yichuan Zhang


### Due: 4pm Thursday 25 November, 2010

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

**Remember that plagiarism is a university offence.** Please read the policy at:
`http://www.inf.ed.ac.uk/teaching/plagiarism.html`
In addition, do not show your code, answers or write-up to anyone else.

You should submit this assignment **manually** to the ITO office in Appleton Tower by the deadline. Handwritten paper 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 us 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.

**Late submissions:** The policy stated in the School of Informatics MSc Degree Guide is that you will not normally be allowed to submit coursework late. See
`http://www.inf.ed.ac.uk/teaching/years/msc/courseguide10.html#exam`
for exceptions to this, e.g. in case of serious medical illness or serious personal problems. Any communications regarding late work should be taken up with the ITO.

## Part 1: Source coding

Obtain `thesis.txt` from the course website. This ASCII file contains $N = 344026$ characters from the alphabet `a–z` and space.

**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.

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 file 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.

   [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.

   [3 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.

   [3 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 seen character $a_j$.

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

   [3 marks]

## Part 2: Digital Fountain Codes

*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 this exercise we will explore the LT code reviewed in Chapter 50 of the MacKay text.*

6. **The erasure channel and feedback:** Recall the definition of the binary erasure channel from Mackay, p148. Assume we want to send a large $K$-bit source file (large enough to approach close to the limits derived by Shannon). The maximum average rate of errorless transmission, which does not require feedback, is the capacity $C$. Write down $K/C$ an estimated number of uses of the channel needed by such an ideal scheme. Report your answer in terms of the erasure rate $f$.

Does feedback improve this bound? Assume that the $K$ bits are passed unencoded into the channel. On average, how many bits will not be received? If the receiver uses a noiseless feedback channel to ask for retransmission of those bits, how many bits will still need retransmitting? If the receiver keeps asking for any unknown bits to be retransmitted, how many uses of the channel are required on average to receive the whole file?

[2 marks]

7. **Repetition code:** Assume that we have no feedback, so at no time do we know which bits the receiver has managed to receive so far. The simplest, but sub-optimal, transmission scheme is to repeatedly send the whole raw file over the channel. In terms of the erasure rate $f$, how many times would we need to send each bit so that the probability that the sender receives that bit is at least $(1 - 10^{-15})$? Evaluate your answer for $f = 0.1$ and explain your working.

[1 mark]

8. **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]

9. **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 overleaf) give, in order, 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 packet 8, you should be able to deduce that the 10th 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.

[4 marks]

A copy of the contents of `packets.txt` follows. The rows give, in order, the source packets used in each received packet in Question 9:

```
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
```