582650 Information-Theoretic Modeling (Fall 2014)

Homework 1 (due September 11)

1. Without studying compression techniques (unless you already have), come up with a compression method of your own invention. The method should take as input a sequence of lower case English letters *a*–*z* and spaces ‘ ’, and output a sequence of bits (you can output ASCII symbols `0` and `1` so you don't need to worry about bit-level operations). It is probably easiest to let each input symbol be mapped onto a fixed codeword (binary sequence). For example $a \mapsto 00000$, $b \mapsto 00001$, ... Each symbol should be mapped onto a distinct codeword so that you have a chance to recover the input from the output. Note that you need not let all the codewords be of equal length. The codewords are concatenated into a long binary sequence without any delimiters such as spaces.

   Try to come up with an encoding (mapping) such that typical English text is compressed as much as possible, i.e., that the number of `0`s and `1`s in the output is as small as possible. Is your encoding decodable, i.e., could you actually recover the input from the output sequence? In particular, can you figure out where one codeword ends and another one begins? If not, try to think about ways to solve the problem without introducing any delimiters.

2. Continuing with the previous exercise: Test your encoding by evaluating the length of the output on Chapter I of *Alice in Wonderland* (starting “CHAPTER I.” and ending at “off the cake.”) which you can download from `www.gutenberg.org/ebooks/11.txt.utf-8`. Replace uppercase letters by the corresponding lowercase letters and skip all non-alphabetic symbols except spaces.

3. Find the shortest computer program (or as short as you can) in a programming language of your choice to output an arbitrary file, `random.file`, of length 1kB (kilobyte) so that `random.file` cannot be compressed significantly by programs such as `gzip` or `WinZip`. For example,

   ```
   > gzip random.file
   ```

   should create a file `random.file.gz` whose size is about 1kB.

   However, you are *not* allowed to use pseudo-random number generators (such as python's `random` module) or to implement any of the well-known algorithms for producing pseudo-random numbers.

4. Binary lottery. For next week's exercise session, we will generate a binary sequence x of length $n = 100$ at random from a source where each bit takes value 1 with probability 0.1 independently of the other bits. In other words, we will flip a biased coin 100 times.

   Suppose you could buy lottery tickets labelled by each of the 2^{100} possible binary sequences of length $n = 100$ for a fixed price per ticket. If you would like to make sure that you will win with probability at least 93 %, what is the fewest number of tickets you'd need to buy? Which tickets?

   Hint: You will probably want to use the binomial distribution. In `R`, for example, `dbinom` can be used to evaluate the probability mass function of a binomial distribution.

5. Attend the Distinguished Lecture by Prof Wojciech Szpankowski on Monday, Sep 8 (for more information, see `www.hiit.fi/HelsinkiITLectures`). If you can't attend the lecture, you may be able to find some information online. However, attending the lecture is a good idea in any case (and there is a cocktail session after it). Please register — even if it's after the recommended registration deadline — by going to the above url and following links there.

   (a) Broadly speaking, what are the key open questions in information theory that the Center for Science of Information (`soihub.org`) is trying to solve?

   (b) What is the basic difference between the information content in a sequence and the information content in a structure?

   (c) What does the Structural ZIP algorithm do? (You don't need to explain how it does what it does, just the task it is meant for.)