582650 Information-Theoretic Modeling (Fall 2014)
Homework 6 (due 16 October)

1. Multinomial NML (1 point). Implement the multinomial NML formula using the recursion

$$C_n^m = C_n^{m-1} + \frac{n}{m-2} C_n^{m-2}.$$

Check that for a sequence $X_1, ..., X_{262}$ where outcome `0` appears 115 times, outcome `1` appears 57 times, and outcome `2` appears 90 times, you get code-length about 408.93 bits.

2. Bayesian networks (2 points). There is a data set, `survey.txt`, that goes with this exercise on the course web page. Your task is to use NML – or to be more precise, *factorized* NML – for learning the structure of a Bayesian network based on the data.

The data consists of a sample of size $n = 500$ of six discrete variables, labelled A, R, E, O, S, and T. Variables A and T have three possible values each and the others have two.

Now the fNML criterion is based on a very similar approach as we used in last week's exercises on Markov chains. You should encode one variable at a time and sum up the code-lengths. When encoding a variable $X \in \{A, R, E, O, S, T\}$ whose parents are $\mathrm{Pa}_i \subseteq \{A, R, E, O, S, T\} \setminus \{X_i\}$, you should split the data from variable X_i into blocks according to the parent variables' values (called their *configuration*). If, for instance, you are encoding node A and it happens to have as its parents nodes R and E, then you should consider separately the $2 \times 2 = 4$ subsets of the data where R and E appear in distinct configurations. In each block or subset, you can use the multinomial NML for encoding the values of variable X_i (in the example, A).

To make your life easy, we provide a nifty program, `splitcfg.py`, which takes a Bayesian network structure (parent lists) as command line argument and prints out counters for each variable's values broken down by parent configuration.

For example, when variables R and E take their first configuration (which happens to be $R =$ `big`, $E =$ `high`), variable A takes each of its three values $115, 57$, and 90 times, respectively. In the next parent configuration ($R =$ `big`, $E =$ `uni`), variable A takes each of its three values $63, 11$, and 43 times, and so on.

These counters are all you need to evaluate the fNML criterion, which is defined as

$$\sum_{i=1}^{p} \sum_{j=1}^{q_i} \ell_{\mathrm{NML}}(X_i[\mathrm{Pa}_i = j]),$$

where p is the number of variables, q_i is the number of parent configurations for variable i (which of course depends on the parent set Pa_i), and $X_i[\mathrm{Pa}_i = j]$ denotes the sequence of observations of variable X_i for which the parent variables take their jth configuration.

(a) (1 point) Run the command

```
> python splitcfg.py survey.txt "1 2" "" "" "" "" ""
```

to produce the counts from the data set `survey.txt` when the parents of variable A are R and E, and all other variables have no parents at all. Apply the multinomial NML universal code to each of the resulting set of counts just like you did for the first set in exercise 1. This gives you the terms $\ell_{\mathrm{NML}}(X_i[\mathrm{Pa}_i = j])$ needed in the fNML criterion. Check that you get total fNML code-length about 2899.86 bits.

(b) (1 point) Implement some kind of a search algorithm for a structure that minimizes the fNML code-length. Note that there shouldn't be any cycles in the network (and in particular, no variable should be a parent of itself!). If it helps, you can limit the number of parents of each variable to three. *Hint:* The search becomes significantly easier if you fix a total ordering '$\prec$' of the variables and only allow the edge $X_i \to X_j$ if $X_i \prec X_j$.

3. Elementary programming (1 point). Find the shortest program, in a programming language of your choice, to print the first section, *Definitions*, of Book I of Euclid's *Elements*, following the on-line version by David E. Joyce available at
`aleph0.clarku.edu/~djoyce/java/elements/bookI/bookI.html#defs`.

   The text begins as follows.

```
    Definitions

Definition 1.
    A point is that which has no part.

Definition 2.
    A line is breadthless length.

Definition 3.
    The ends of a line are points.

Definition 4.
    A straight line is a line which lies evenly with the points on itself.

Definition 5.
    A surface is that which has length and breadth only.

Definition 6.
    The edges of a surface are lines.
```

   The last item to include is Definition 23 (parallel lines).

   Include all spacing, punctuation, and other symbols, including capitalization (but not text formatting such as fonts or underlining, etc).

   Rule #1: You are not allowed use compression libraries to do the job for you. In other words, it is not allowed to compress the data using, for instance, `gzip` and then call `zlib.decompress` to decompress the compressed data. That wouldn't be much fun, would it?

   Hints:

   (a) You can, for instance, define variables as string literals such as `d="Definition"`, `i=" is that which "`, `s=" surface"`, and use them as

```
print d+" 5.\n    A"+s+i+"has length and breadth only."
```

   Compared to simply printing the whole text as a single string literal, this would reduce the number of bytes (characters) spent for Definition 5 from 70 to 54 (ignoring the code for defining the string literals.)

   (b) Even though you are not allowed to use existing compression libraries, you are allowed to use compression (and decompression) techniques such as Huffman coding to obtain a concise encoding if you really like. However, we rather recommend that you try to think about other approaches, such as string literals, etc. To support compression methods, it is okay to provide a separate data file in addition to the actual source code of your program. (You can also provide an executable file but that is likely to be bigger than the source code.) The total size of the data file and the program file are counted as your score.

   (c) Some programming languages are more concise than others. We don't expect you to learn a new programming language just for this purpose – although it would of course probably be a good idea in general. The point is to think about language-independent strategies.

4. Google distance (1 point). Define the *normalized Google distance* (NGD) as

$$\mathrm{NGD}(\texttt{word1},\texttt{word2}) = \frac{\max\{\log f(\texttt{word1}), \log(\texttt{word2})\} - \log f(\texttt{word1},\texttt{word2})}{\log M - \min\{\log f(\texttt{word1}), \log f(\texttt{word2})\}} \ ,$$

where $f(\texttt{word})$ denotes the number of pages indexed by Google that include the word `word`. You can retrieve this number by typing the word in Google search. The page count appears before the actual results. For example, typing `Kolmogorov` in the search field, you should get something like "`About 2,730,000 results (0.32 seconds)`", which gives you $f(\texttt{Kolmogorov}) = 2\,730\,000$. Likewise, you should get something like $f(\texttt{Kolmogorov},\texttt{complexity}) = 625\,000$ by typing both words in Google search. The term M is the total number of web documents indexed by Google. You can use the approximation $M \approx 5 \cdot 10^{10}$ (fifty billion).

(a) Evaluate the pairwise distances $\mathrm{NGD}(\texttt{word1},\texttt{word2})$ for words in the set

$$\{\texttt{Andrey},\texttt{Kolmogorov},\texttt{complexity},\texttt{stochastic},\texttt{porridge},\\ \texttt{Rissanen},\texttt{breakfast},\texttt{omelette},\texttt{broccoli}\} \ .$$

For the first one, should get a result close to 0.526. Don't be alarmed if some of the results appear weird. This is probably due to Google's distorted way of reporting the page counts. If you find a better way (using perhaps another API to Google search, or even another search engine), go for it.

(b) Record the values obtained in the previous item in a 9×9 distance matrix. You can assume that $\mathrm{NGD}(\texttt{word1},\texttt{word1}) = 0$ and that the matrix is symmetric.

Visualize the distances using your favourite technique. Good alternatives might include, for instance, a minimum spanning tree or a heatmap. What do you learn about the words? The Google distance is supposed to correlate with the (lack of) semantic relatedness of the words in question. Does this hold? Feel free to try other word pairs to test the claim.