Pre-workshop

- Revise the following terms, as they are used in a text processing context: “corpus”; “document”; “term”; “token”.
- Revise the problem of “tokenisation” of a text; revise “stemming” of tokens.
- Revise “inverted indices”, what they are used for in an information retrieval context, and their structure.
- Revise “stop words”, and why they are often removed from a text in an information retrieval context. Use the Web to find a list of stop words for English — are there any words in the list that you might consider not to be a stop word? Are there any words that you consider to be stop words that are missing from the list?
- (Re-)familiarise yourself with Python, if you haven’t used it recently. In particular, focus on string and array processing, including regular expressions. Also revise functions and mapping mathematical formulae to Python syntax (including the numpy package). Familiarise yourself with running Python in the CIS labs.
- Familiarise yourself with the Natural Language Toolkit (NLTK). You might like to use the e-book http://nltk.org/book as a resource; it also covers some of the basics of Python, if you’ve never used the language before.

Workshop

1. Name some text processing problems and applications. Are there any text processing systems that you use on a daily basis? (Hint: the answer is “yes.”)
2. What are the main differences between “natural language processing” and “information retrieval”?
3. What are “stemming” and “lemmatisation”, and how are they different? Give examples.
4. A simple tokenisation strategy for English documents was given in the lectures, as follows:
   - Strip formatting
   - Strip punctuation
   - Break at whitespace
   - Fold case
   - Stem
   - Remove stop words

(a) Write a program in Python (optionally using NLTK) to tokenise a plain text document written in English using the above strategy.
(b) Identify some theoretical tokenisation issues in English. Tokenise a document (e.g. Herman Melville’s *Moby Dick*) and identify some instances where you might wish to have tokenised the text differently.

5. Following on from the previous question:

(a) Extend your program from 4(a) to produce a “bag of words” representation of the document.

(b) Write a Python program (based on your bag-of-words representation) that builds a term-document matrix over a collection of documents. How would you use it to evaluate Boolean queries?

(c) Alter the above program to use a sparse representation of the term-document matrix. Observe that you now have an inverted index. Consider how you might include other useful information, like “document frequency”, and positional information in your index.

(d) Revise “skip pointers”. Identify the changes you would need to make to your algorithm above to incorporate this information.

**Post-workshop**

- (Extension) Identify some tokenisation issues in a language (other than English) of your choice. How much alteration would need to be made to the tokenisation strategy from (4) to account for these issues?

- How does the size of the vocabulary of a document relate to the length of that document? Take different slices (e.g. [:1000], [:10000], etc.) of a single document and observe how the size of the vocabulary changes. (You can use your program from 5(a), or the `set` function.)

- (Extension) Choose a selection of documents (e.g. from Project Gutenberg\(^1\)) and calculate the size of the vocabulary it uses (after suitable tokenisation).

- Use your program from 5(a) to plot the distribution of term frequencies in one or more documents. How closely does it follow a Zipfian distribution?

- (Extension) What is the median frequency of the terms in a document? Does it depend upon the document under consideration? What implications does this have for natural language processing?

- (Project preparation) Sketch out how you would evaluate Boolean queries using an inverted index. Write a Python program to evaluate simple (conjunctive) Boolean queries, based on your representation from 5(c).

- (Extension) Alter your program from 5(d) to incorporate skip pointers. Construct a suitably large inverted index, and observe how the cost of query evaluation changes with skips of differing width.

\(^1\)http://www.gutenberg.org