Programming for Data Science – Class Exercise 1/4
Text processing using Python

MSc in Data Science

The Exercise

Text-processing systems often perform a number of pre-processing steps to improve performance and functionality. Some of the most common tasks are

- Removal of non-alphanumeric symbols, e.g. !, @, #, $, %, etc.
- Removal of very small words, say up to 3 characters.
- Stemming: the reduction of words into their "stems", e.g. the words "initialize" and "initial" are both reduced into "initi". This makes makes indexing more efficient and matching between text queries and documents easier.
- The removal of stop-words: Frequently-occurring terms, such as "a", "the", etc. convey little meaning and are often removed.
- The creation of an index or an inverted file – similar to the index of a book, the creation of a data structure from terms to document IDs.

In this exercise you will need to implement the above with as few loops as possible (some loops may still be required). Instead try to implement a sequence of list comprehensions, `map`, `filter` and `reduce` calls as appropriate. Many of these calls will be instantly parallelisable in a map-reduce fashion. However, the goal is to familiarise yourselves with commonly used Python constructs and not to write a parallel program. **Your program will be slower than a conventional single-core Python program, this is OK!** You are expected to make use of appropriate Python modules and the Cranfield experimental text collection.

Deliverable: A single Python source file with a `main` function, as well as any number of meaningful auxiliary functions, constants, etc. You are encouraged to use the following template:

```
#!/usr/bin/env python
import sys
from nltk.stem.porter import PorterStemmer
from nltk.corpus import stopwords

SYMBOLS = '!@#$%^&*()[]{};\':",.<>/?`~-_=+'

def extr_docs(cran_file):
    """Extract and return the Cranfield documents into a [(id,term)] form"""
    pass

def main():
    # Get the cranfield collection filename as the 1st argument:
    cran_file = ''

    # Extract the documents in a convenient format:
    lst = extr_docs(cran_file)

    # Get list of stop-words and instantiate a stemmer:
    stop_words = set(stopwords.words('english'))
    stemmer = PorterStemmer()

    # Remove non-alphanumeric symbols:
    # Remove words <= 3 characters:
    # Remove stopwords:
    # Stem terms:
    # Sort according to term:
```

```
    # Create the index:

if __name__ == '__main__':
    main()
```

Input: A filename pointing to the Cranfield collection as the first command line argument.

Output: A list of terms (words), each followed by a list of document IDs corresponding to the documents the terms are found in. Each *term*, *IDs* list should appear on its own line, on the standard output. For each term, the corresponding document IDs should be presented in increasing order. The terms must not contain stop-words, must have been stemmed and they must appear in alphabetical order. For example, a single line of the output could be:

```
result 63 84 89 101 108 209
```

Background

The Cranfield Collection

The Cranfield experimental collection is already downloaded in your VM at `$HOME/Datasets/cran/`. If you need to, you can download the collection at `http://ir.dcs.gla.ac.uk/resources/test_collections/cran/` and extract it somewhere in your home folder. For this exercise you will only need to use the text collection file `cran.all.1400` – ignore the rest. This file is structured as follows:

```
.I 21
.T
on heat transfer in slip flow .
.A
stephen h. maslen
.B
lewis flight propulsion laboratory, naca, cleveland, ohio
.W
on heat transfer in slip flow .
a number of authors have considered the effect of slip on the heat
transfer and skin friction in a laminar boundary layer over a flat plate .
reference 1 considers this by a perturbation on the usual laminar
boundary-layer analysis while some other studies.dash e.g., reference
the impulsive motion of an infinite plate .
```

Lines beginning with a `.` are used to delimit sections of interest. For each document in the file, `.I` is followed by the document ID, while `.W` is followed by the text body. The body of each document is delimited by the next line starting with `.I`. You do not need to be concerned with the other delimiters.

In this exercise you first need to extract the text indicated by `.W` and associate it with its document ID in a way that makes it convenient to work with list comprehensions, `map`, `reduce`, etc.

The Porter Stemmer

The Porter stemmer is one of the most widely used stemmers for the English language. An efficient stemmer is included in the `nltk` module and you can use it as follows:

```
>>> from nltk.stem.porter import PorterStemmer
>>> stemmer = PorterStemmer()
>>> stemmer.stem('initialize')
u'initi'
```

Stop-words

You can obtain a set of stop-words in Python also via the `nltk` module as follows:

```
>>> from nltk.corpus import stopwords
>>> stop = set(stopwords.words('english'))
```