Processing Data using Gutenberg API

Anindya Dutta and Aarushi Goel

October 24 2017

1 Using the Gutenberg API

1.1 Understanding the Gutenberg API

Instead of manually retrieving documents from the Project Gutenberg website, we decided to leverage available APIs to conduct our experiment. We are using the Gutenberg 0.5.0 PyPi package. The functionality provided by this package includes:

- Downloading texts from Project Gutenberg.
- Cleaning the texts: removing all the crud, leaving just the text behind.
- Making meta-data about the texts easily accessible.

We picked a random list of 12 authors and used the API to download a total of 486 e-books for our analysis. The list of authors is listed in `author.txt`.

1.2 How we used the API

Before using one of the `gutenberg.query` functions we needed to populate the local metadata cache. This one-off process took quite a while to complete (11 hours on my machine) but once it was done, any subsequent calls to `get_etexts` or `get_metadata` will be very fast. If we fail to populate the cache, the calls will raise an exception.

Once the cache is populated, we started querying the metadata database. Our script `gutenberg_preprocess.py` calls the APIs and creates a set of `Book` objects (from `Book.py`). The following is a pseudocode of what we implemented.

Algorithm 1 Retrieve books

```
1:  procedure InitBooks
2:      authors ← Get all author names from the text file.
3:      book_list ← empty list
4:      for author in authors do
5:          ids ← Get the IDs of all books written by author.
6:          for id in ids do
7:              Retrieve title of the book from metadata using get_metadata('title', id).
8:              Retrieve and strip extra headers from text using strip_headers(load_etext(id)).strip().
9:              book ← Create a new Book with the above data.
10:             Add book to book_list
```

1.3 Complexity Analysis

The above algorithm is of the complexity of $O(mn)$ where m = number of authors and n = max (number of books by an author). We have attached the code for `gutenberg_preprocess.py` and `Book.py`.

2 Storing the data

Since the book-list takes up some time to be created by picking out data for 486 books, we have saved the required data using `pickle` into a binary file. This file is essentially a `list` of `Book`s, which can be used throughout the project by loading the necessary file. The output is stored in a file called `data`. The size of this file currently is 163 MB and therefore has not been attached in the e-mail.

3 Next Steps

Our next step would be to perform document clustering on these documents before we can classify their emotions. We are considering using tf-idf vectorizers and k-means clustering for this step. We think an unsupervised learning as the first step to document classification would make our process easier, as that would help us in grouping similar novels, which can then be categorized based on emotions.