Data Pre-processing Results and Binary Classifier

Anindya Dutta and Aarushi Goel

October 9 2017

1 JSON

1.1 Work done so far

We started by analyzing every text file to find structural similarities. The following are the results.

- The title of the book is on a separate line preceded with the text **Title:**.
- The year of the book is on a line preceded with the text **Release date:**.
- The author of the book is on a line preceded with the text **Author:**.
- Every book has an **Ebook #** that can be used as the primary key.

Using these features, it is easy to extract the features from the text files and create JSON objects out of them. Our script `json_create.py` creates a JSON file using text files saved in a directory, by extracting all the above features and creating an array of JSON objects.

Attached in the e-mail is an example JSON file (`data.json`) with two books. We store the extracted data in a `meta` object along with the actual text in the file. However, this file is currently not preprocessed and has all the extra jargon that Gutenberg adds to each text file.

1.2 Next Steps

Since scraping Gutenberg's site allows only 100 books in a script, we are looking for a way to do this automatically. The site PyPi Gutenberg module has inbuilt APIs to cleanup the files and we can use it to keep only the text in the file. The open GitHub project OpenZim Gutenberg has a scraper for downloading the entire ebooks repository of project Gutenberg, but there are limitations in both. This will take some time, as noted in the PyPi link, for example, such that the creation of the cache may take about 18 hours to complete.

2 Binary sentiment classification

We implemented binary sentiment classification on the text, to describe the entire text as either positive or negative based on the resource links provided in the earlier document. This results in files being tagged as `pos` or `neg` based on the frequency of positive and negative words in the text. For example, for the sample JSON file attached, the output is stored as in `answers.txt`. We wrote the script `simplesent.py` to achieve this.

2.1 Next Steps

The next step is to make the binary classifier more robust. We are looking into resources to find dictionaries that may have phrases stored as positive and negative words, so that we can do bigram and trigram sentiment analysis, and augment it to the unigram dictionary, to increase the precision of the algorithm. Once we have a good dictionary, we can move forward with a multi-class sentiment classification.