Tutorial: Computing Word Frequencies in Python

Statistics for Social Data

1. Install the necessary software. If you don't already have it on your system, install Python 2 from `http://www.python.org`. In class, I use version 2.7.5, but probably any version of the form 2.7.x will work. Do *not* install version 3.0 or higher.

Once you have installed Python, install the Natural Language Tool Kit (NLTK) by following the directions at `http://www.nltk.org`.

When you are done with the installation, start a python shell, and then run the following commands:

```
>>> import csv
>>> import json
>>> import re
>>> import nltk
```

None of these commands should report any output. If a command raises an `ImportError`, then there is a problem with your installation.

2. Read the raw data into Python. The file `federalist.json` contains the text of the federalist papers, along with their associated metadata. This file was obtained by processing Project Gutenberg's version of the documents (`http://www.gutenberg.org/ebooks/18`). Each line of the file contains a federalist paper, with metadata including `paper_id`, `title`, `author`, `text`, `footnotes`, and some other information. The data is encoded using the JSON is a data-interchange format.

Read the contents of the file into Python by running the following command from the interpreter:

```
>>> f = open('federalist.json', 'r')
>>> lines = f.readlines()
>>> f.close()
```

At this point, `lines` is an array of JSON strings. To parse the string data into Python's native representation, run the command

```
>>> papers = [json.loads(l) for l in lines]
```

Now, the `papers` array contains the data for the papers. We index the elements of the array using zero-based indexing. The first element is `papers[0]`, the second is `papers[1]`, and so on.

See which metadata is available for the first paper by typing

```
>>> papers[0].keys()
```

Find the id, the author, and the text of the first paper by typing

```
>>> papers[0]['paper_id']
>>> papers[0]['author']
>>> papers[0]['text']
```

3. Tokenize the text into words.. The text for each paper is stored in one long string. Before processing the raw string data, we need to remove the footnote annotations. We can do this by defining a function `remove_footnotes` as follows:

```
>>> def remove_footnotes(s):
...     return re.sub(r'\[\d+\]', '', s)
...
```

This function uses a regular expression to find patterns of the form `[ddd]` and remove them (more precisely, it substitutes empty strings where these patterns occur). See the documentation of the `re` package for more details, or read up on "Regular Expressions."

To "tokenize" the text of the first document into sentences, we can use the `nltk.sent_tokenize` funciton. This function won't work if there are newlines (carriage returns) in the text, so before calling the function, we replace all whitespace with ordinary spaces. We also need to remove footnotes. The full sequence of commands is as follows:

```
>>> p = papers[0]
>>> raw_text = p['text']
>>> text = remove_footnotes(raw_text)
>>> text = re.sub(r'\s+', ' ', text) # normalize whitespace
>>> sents = nltk.sent_tokenize(text)
```

At this point, `sents` contains a list of the sentences in the first paper. The first sentence of the first paper is stored in `sents[0]`. We can find the number of sentences in the paper by typing `len(sents)`.

Tokenize the sentences into words by running the command

```
>>> raw_words = []
>>> for s in sents:
...     for w in nltk.word_tokenize(s):
...         raw_words.append(w)
...
```

4. Normaize the words. We don't want to distinguish between different capitalizations of the same word. To this end, define `lwords`, an array of words with capital letters replaced by lowercase:

```
>>> words0 = [w.lower() for w in raw_words]
```

5. Remove punctuation. We don't want to include punctuation in our analyses. To this end, we need to filter out these tokens from our word list. First, we define a function to indicate whether or not a particular word is a punctuation marker:

```
>>> def is_punct(w):
...     return re.match(r'^\W+$', w) != None
...
```

Next, we use this function to remove punctuation from the word list:

```
>>> words = [w for w in words0 if not is_punct(w)]
```

We can check to see if we were successful in filtering out all punctuation by running `words.sort()` and then ensuring that `words[0]` is not a punctuation marker.

6. Compute the word frequencies. At this point, we have a list of all of the words in the document. We want to count how many times each word appears in the document. We can write a function to do this computation:

```
>>> def histogram(words):
...     freq = {}
...     for w in words:
...          if freq.has_key(w):
...              freq[w] = freq[w] + 1
...          else:
...              freq[w] = 1
...     return freq
...
```

Next, we apply the function to the word list from the first document:

```
>>> freq = histogram(words)
```

We can find the frequencies of the words `enough` and `upon` by the values `freq['enough']` and `freq['upon']`. If we try to get the frequency of the word `while` by typing `freq['while']`, then we get a `KeyError`; this is because that word never appeared in the document.

7. Save the word frequencies to a CSV file. We will want to analyze the frequencies in R. The easiest way to get the data into R is to save the word frequencies in a Comma-Separated Value (CSV) file, and to then read that file into R.

```
>>> f = open(”words.csv”, ”w”)
>>> writer = csv.writer(f)
>>> writer.writerow([”word”, ”freq”])
>>> for w in sorted(freq.keys()):
...     writer.writerow([w, freq[w]])
...
>>> f.close()
```

This will save the word frequencies from the first Federalist paper to the file `fed1.csv`.