Homework 1: Getting our Feet Wet

Most NLP tasks begin with the same overall pattern:

1. Get data from somewhere
2. Figure out how to read/parse it, and find the part we're interested in
3. Basic pre-processing (tokenization, stopword removal, unicode normalization, etc.)
4. Counting stuff

In this assignment, you will demonstrate your ability to work with some of the basic file formats and Python libraries we will be using this term. You will be reading data from a standard corpus of newswire text, and computing a simple measure of word association to extract collocations.

This is not intended to be a difficult assignment, but *is* meant to help you get used to the sorts of things we'll be doing more of later in the term.

A note on turning in code: Please set up a Github or Gitlab repository to store your code, and provide me with a link to that repository in Sakai. CSLU has a Gitlab instance that is available for student use, located at repo.cslu.ohsu.edu. If you do not have an account and would like one, ask Ethan Van Matre.

Part 0: Getting set up

You'll need to have a working Python installation (preferably Python 3), and the following libraries installed:

1. `nltk`: A useful suite of tools for natural language processing, optimized for educational use (rather than industrial/production use)
2. `lxml`: An XML parsing library

Each of these should be installable via either `pip` or Anaconda, depending on your preference.

Once you've installed NLTK, you'll need to download some language resources. NLTK can download a variety of resource such as text corpora, pre-trained models, etc. using the `downloader` module:

```
python -m nltk.downloader
```

This will open a flaky but usable GUI with which you can browse available resources. Install the `stopwords` and `punkt` packages.

You can also run that command in a headless manner, by specifying the corpus/resource you wish to download:

```
python -m nltk.downloader stopwords punkt
```

If you have difficulty with these steps, ask for help!

Part 1: Reading some data

This directory contains news documents from the Central News Agency (Taiwan) drawn from the 5th edition of the Gigaword corpus (LDC2011T07). Each file is a compressed XML file containing multiple documents (roughly corresponding to news articles). Each document is wrapped with a `<DOC>` tag, which also have a `type` attribute. Immediately below `<DOC>` tags in the hierarchy, story text is separated from the headline and dateline by the `<TEXT>` tag. Finally, paragraphs of story text are marked by `<P>` tags. Your first assignment is download, decompress, and serialize these files, extracting the text of all paragraphs (`<P>`) whcih are part of the `<TEXT>` of all `<DOC>`s of `type="story"`. Your script should operate on a list of gzipped XML files given as command-line arguments, e.g.:

```
python your_deserialization_script.py cna_eng/*.xml.gz > deserialized.txt
```

Note: You may *not* use regular expressions for XML parsing and deserialization. There is a good theoretical reason for this: XML is not a regular language. You must use a proper XML parser.

What to turn in:

1. Your program
2. Sample terminal output, showing perhaps the first 100 lines of its output
3. A sentence or two describing your approach and any bugs you encountered.

Tips:

- Don't download the data files by hand; use `wget` or some other such tool
- Make sure to watch out for empty paragraphs- the data is realistically noisy
- Use XPath selectors to identify the right stories and elements to process!
- If you're not sure which libraries to use for this, my reference implementation uses Python's `gzip` library to decompress the files in memory, and `lxml.etree` for XML parsing. Both are rather fast.

Part 2: Structuring the data

Once you've gotten the data out of the files, you'll notice that it is a bit disjointed- each sentence is split on to multiple lines. The goal for this part of the assignment is write a program to transform Part 1's output from its current "hard-wrapped" format to a format where each line of the file contains exactly one sentence.

Use the NLTK `tokenize` module's `sent_tokenize` function to split sentences. By default, this uses the Punkt sentence tokenization algorithm (which we will be discussing further in future weeks). Note that, since in its current form, many sentences span multiple lines, you will need to do some cleanup before you can run `sent_tokenize`!

Once you've reconstituted the data into a one-line-per-sentence format, you're ready to perform word-level tokenization. The `tokenize` NLTK package also includes a `word_tokenize` function. Your final output from this section should be a file where each line consists of a single sentence, and the line's contents have been tokenized using `word_tokenize`.

Newswire text has a lot of punctuation, which for this assignment we don't care about. Remove any tokens that are solely punctuation. Finally, for this assignment, we do not care about capitalization, so let's go ahead and turn everything into upper-case letters.

What to turn in:

1. How many sentences are there in the CNA-GW corpus?

Tips:

- You may choose whether to write several separate scripts and chain them together using UNIX pipes, or write one larger monolithic script. Often, smaller, more modular scripts are useful for this sort of data-prep process.

- Removing punctuation can be done in many ways, but what I would recommend in this case would be to use the `string` package:

  ```
  import string
  punct_set = set(string.punctuation)
  ...
  no_punct_toks = [t for t in tokens if t not in punct_set]
  ```

  Obviously, for a more realistic task, you'd need to do something more sophisticated than this!

Part 3: Counting and comparing

Now that you've got your corpus prepared, you can start doing some analysis. Collocations are pairs of words that frequently occur together, such as "New York". For this part of the assignment, treat a "word" as a token that comes out of your script from Part 2.

Word counting & distribution

Compute unigram and bigram frequency counts for each word. For this assignment, do not worry about padding your bigrams with start/end-of-sentence markers or anything like that.

1. How many unique *types* are present in this corpus?
2. How about unigram *tokens*?

3. Produce a rank-frequency plot (similar to those seen on the Wikipedia page for Zipf's Law) for this corpus.

4. What are the twenty *most common* words?

5. You may notice that the most common are words that occur very frequently in the English language (stopwords). What happens to your type/token counts if you remove stopwords using `nltk.corpora`'s `stopwords` list?

6. After removing stopwords, what are the 20 most common words?

Word association metrics

There are many ways to identify collocated words, but one common one is to use *Pointwise Mutual Information*. This measure captures how much more likely it is that two events occur together than would be the case if the events were statistically independent.

$$PMI(w_1, w_2) = \frac{P(w_1, w_2)}{P(w_1)P(w_2)}$$

For this part of the assignment, compute unigram and bigram *probabilities* for all unigrams and bigrams in the corpus (ignore issues of smoothing your probability estimates, for now), and compute PMI values for each bigram.

- Recalling Emily Bender's sage advice- "Look at your data!"- examine the 30 highest-PMI word pairs, along with their unigram and bigram frequencies. What do you notice?

One drawback of using PMI in this way is that it is unstable when word frequencies are low. There are a variety of ways to solve this problem; one common way is to simply set a threshold, and only consider bigrams that occur with frequency above that threshold.

- Experiment with a few different threshold values, and report on what you observe.
- With a threshold of 100, what are the 10 highest-PMI word pairs?
- Examine the PMI for "New York". Explain in your own words why it is not higher.

What to turn in:

Your answers to the bullet-pointed questions above.

Back to index