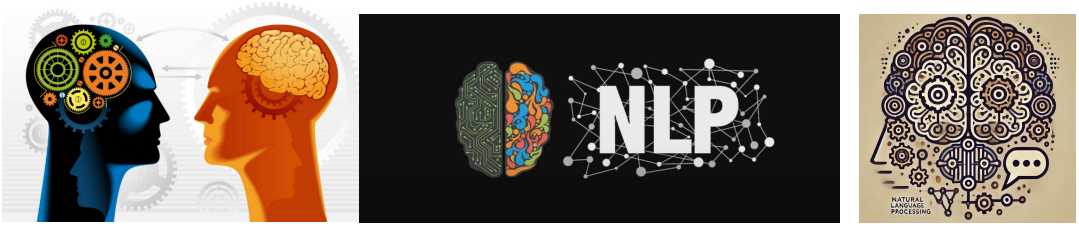
An extensible framework for natural language processing



Prof. Rachlin

DS3500: Advanced Programming with Data

Overview

As we progress into advanced programming and begin to embrace attitudes of professional software developers and data scientists, our focus turns more to reusability, code maintainability, and extensibility. In prior courses you may have carried out one-off analysis of some dataset. But what if what we want to do is build a framework that can handle a variety of datasets? In this assignment, you'll build a reusable framework for comparative text analysis, and you will demonstrate its reusability by comparing a set of documents of your own choosing.

Framework Requirements

1. The library should be implemented as the object instance of a class. You are free to import other libraries to facilitate the implementation of your library methods.

2. You should load or register 8 or more documents. You may choose the documents and the theme, but they should be related in some way (see below for some suggestions.)

3. Each text file that you load should be pre-processed so that the pre-processor cleans the data, removing unnecessary whitespace, punctuation, and capitalization. You may want the pre-processor to gather some statistics such as average sentence or word length, readability scores, sentiment, and so on, but at minimum you should include a word-count as this will support one of the required visualizations described below.

4. Implement a generic parser and pre-processor for handling simple unstructured text files. Also implement the ability to specify a custom domain-specific parser. Now,

when registering the file, you can specify custom parsing function that will carry out the parsing and pre-processing of your unique files! You should implement support for this even if you might not need it for your documents.

5. The library must support at least *insightful* three visualizations, one of which is very specific, the other two are left to your discretion.

    a. (Specific) Text-to-Word Sankey diagram. Given the loaded texts and either a set of user-defined words OR the set of words drawn from the k most common words of each text file, generate a Sankey diagram from text name to word, where the thickness of the connection represents the wordcount of that word in the specified text.

    b. (Flexible) Any type of visualization containing sub-plots, one sub-plot for each text file. For example, an array of word clouds, one for each text file would satisfy this requirement, although I'm not a huge fan of word clouds.

    c. (Flexible) Any type of comparative visualization that overlays information from each text file onto a single plot.

Create a framework class with, at minimum, the following methods:

load_stop_words(stopfile)
A list of common or stop words. These get filtered from each file automatically

load_text(self, filename, label=None, parser=None)
Register a text file with the library. The label is an optional label you'll use in your
visualizations to identify the text

wordcount_sankey(self, word_list=None, k=5)
Map each text to words using a Sankey diagram, where the thickness of the line
is the number of times that word occurs in the text. Users can specify a particular
set of words, or the words can be the union of the k most common words across
each text file (excluding stop words).

your_second_visualization(self, misc_parameters)
A visualization array of subplots with one subplot for each text file.
Rendering subplots is a good, advanced skill to know!

your_third_visualization(self, misc_parameters)
A single visualization that overlays data from each of the text files. Make sure your
visualization distinguishes the data from each text file using labels or a legend

As you pre-process each text file, you'll want to store intermediate results: word counts, statistics, maybe the clean version of text, etc. A standard way to do this might be to have a state variable: self.data that is a dictionary. Presented below is a generic solution using dictionaries within dictionaries. The layout and content are ultimately up to you but try to come up with an approach that makes it easy to add custom parsers that store other kinds of information and custom visualizations that can access this information in standard ways.

```
{
      word_count: {text1: wordcount1, text2: wordcount2, …
      word_length: {text1: wordlength1, text2: wordlength2, … }
      sentiment: {text1: sentiment1, text2: sentiment2, … }
      .
      etc.
}
```

Data Sources

Identify 8 or more related documents. The type of file you choose is entirely up to you, but I recommend something different than presidential inaugural speeches or song lyrics. (Been there, done that.) Some possible ideas or data sources:

- Project Gutenberg (www.gutenberg.org)
- Political speeches
- National constitutions
- Tweet compilations
- Corporate filings
- Letters/Journals/Diaries
- News articles or blog posts
- Religious texts
- Philosophical tracts
- AI-generated text

What to submit:

Submit all your code and a one-slide poster using the poster template provided.