

Programming for LT Applications

Artur Kulmizev

February 24, 2021

This assignment package will be rolled out in three parts, each corresponding to a lab section: **Web Scraping**, **Pre-processing and Visualization**, and **Modular Programming**. This document will be updated with each part accordingly, which will have separate exercises associated with it. The goal of this assignment is to get to you to write practical applications in Python, using the concepts we covered in the lectures. You should hand in a single tarball¹, named according to the following format: `assignment4.tar.gz`. The directory structure of the tarball should be the following:

```
A3_0_assignment_3
├── twitter_query
│   ├── README.txt
│   └── twitter_query.py
├── wiki_query
│   ├── README.txt
│   └── wiki_query.py
├── term_plotter
│   ├── README.txt
│   └── term_plotter.py
├── tagger
│   ├── tagger
│   │   ├── __init__.py
│   │   ├── pre_process.py
│   │   └── model.py
│   ├── README.txt
│   ├── run.py
│   └── config.yaml
```

In other words, the examiner should easily be able to navigate to a particular exercise folder, and call the exercise script directly via e.g.

```
$ python script.py ARG1 ARG2
```

This code should run out of the box² and produce the desired results. You can approach the problems in any way that you want (using classes, functions, etc.), but the code has to be executable from the corresponding exercise script file and return the desired output. You should be also mindful in making your code clean and interpretable, commenting where necessary. The PEP 8 style guide³ can give you some tips in this regard, but you are by no means required to use it. Each exercise folder should also include a `README.txt` file that explains how to run your code, as well as how you went about writing it (e.g. did you make a class?).

¹<https://www.howtogeek.com/362203/what-is-a-tar.gz-file-and-how-do-i-open-it/>

²We shouldn't get errors when running a simple query.

³<https://www.python.org/dev/peps/pep-0008/>

Lab 1: Web Scraping

Finding good data is a crucial first step in many NLP applications. When ready-made corpora are not available, we often revert to scraping relevant texts from the internet. In the web scraping lecture, we learned a bit about web scraping: how websites are built upon HTML files, how we can parse these files to extract information, among other things. If you recall, we discussed two distinct scenarios that might arise when scraping the web. In the first scenario, you might have access to a website's API, which you can use to interact with the website directly. Otherwise, you might have to do the hard work yourself: understand the structure of the website data and parse it accordingly. We'll explore these situations in this lab, where you'll build two tools: one for extracting Tweets for a specified user on Twitter, and another for extracting the descriptions of concepts from Wikipedia.

Extracting Tweets

Your task: Write a program that takes a Twitter user's handle as an argument and returns the user's most recent Tweet(s), as well as the time of their posting. Your script should also accept an optional integer N , which, if specified, returns the user's N -most recent Tweets. If not specified, it should return only the single most recent Tweet. For example:

```
$ python3 twitter_query.py jack 3
11:28 PM, Feb 21, 2020: I've discovered the way to make slack grind
to a crawl and crash repeatedly
3:29 AM, Feb 21, 2020: Talked #bitcoin Africa with #NiiQuaynor,
@PindarWong, and @moneyball! Earth globe europe-africa
11:25 PM, Feb 20, 2020: I just talked with @leslieberland's mom
on the phone AMA
```

You should work with the Tweepy⁴ library for navigating the Twitter API. In order to do so, you will need a set of credentials: a consumer key and secret, and an access token and access token secret. These can be accessed via a JSON file found at `/local/kurs/advprog/module3/credentials.json`. You can decide how to handle replies, retweets, images, emoji, etc., but make sure to document your decision in doing so. Your script should be called `twitter_query.py`.

Mining Wikipedia

Your task: Write a program that takes a search term as an argument and then prints the content of the Wikipedia page for that term. For example:

```
$ python3 wiki_query.py ankylosaurus
Ankylosaurus is a genus of armored dinosaur. Its fossils have been
found in geological formations dating to the very end of the Cretaceous
Period, about 68-66 million years ago, in western North America,
making it among the last of the non-avian dinosaurs. It was named by
Barnum Brown in 1908, ...}
```

Make sure that your program uses at least one try-except clause (you can choose which exceptions to try to catch). Also ensure that you can search for terms with non-ASCII characters. For example, check that your program works with the search term

⁴<http://docs.tweepy.org/en/latest/index.html>

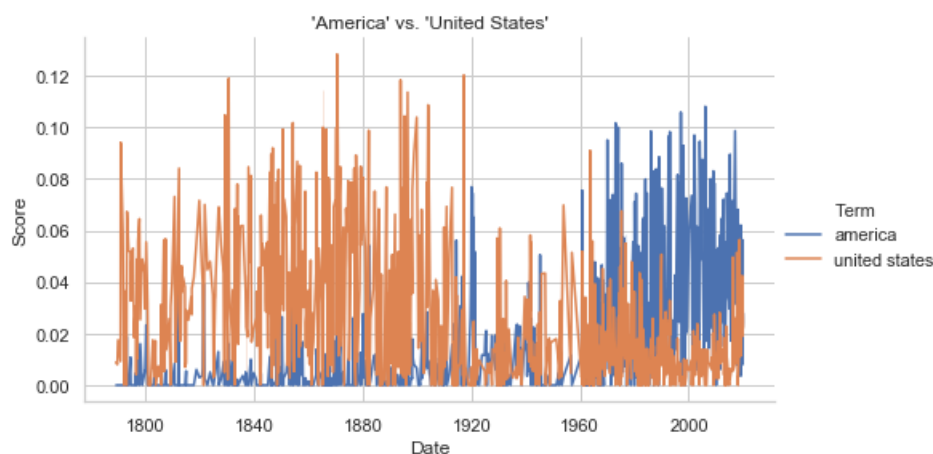
"*Miloš Zeman*". You might want to use the libraries `urllib` and `beautifulsoup` (see the lecture notebook and online documentation), but feel free to approach the problem however you want. Your script should be called `wiki_query.py`.

Lab 2: Pre-processing and Visualization

After collecting raw text data, it is important to know how to process it. This can involve anything from tokenization to dependency parsing. We learned how to do this in the lecture, using a power library called `SpaCy`. Once your data has been processed, you can begin to gather some insights about how it's structured and what it represents. Visualization is a key component of this process, and we covered how to generate basic scatter, line, and bar plots in Python. In this lab, you will be asked to generate a line plot that takes all documents in our presidential speech corpus as input and plots the tf-idf scores of specified terms, sorted by year. This should give a sense of how particular terms/concepts came to prominence throughout the history of the US (as told by its presidents).

Diachronic Term Relevance

Your task: Write a program that calculates the tf-idf scores of a set of specified terms as they occur in each of our scraped presidential speeches and plots each term's score as a function of time (i.e. the speech's date) Your program should output an image that looks something like this:



You should use `argparse` for this assignment, where you set the arguments as follows:

- **terms:** a **list** of up to five terms, in quotations, separated by white space. Each “term” can consist of three tokens (e.g a trigram), with English stopwords assumed to be filtered out.
- **path:** an optional **string** argument that specifies where the speech directory is. If not provided, the program should assume to look at the folder in the current directory, e.g.: `./us_presidential_speeches`.
- **title:** an optional **string** argument that adds a title to the plot. If not provided, the program outputs the plot without a title.
- **output:** an optional **string** argument that saves the plot under the specified name. If not provided, saves the plot as the concatenation of all terms, separated by an underscore token. e.g.: `america_united_states.png`.

Ultimately, in order to produce the output shown here, the examiner should be able to call the script as follows:

```
$ python3 term_plotter.py --terms "america" "united states" \  
                        --title "'America' vs. 'United States'"
```

This will look in the current directory for a folder called `us_presidential_speeches/` and generate a file called `america_united_states.png`.

To complete this assignment, you will make use of several libraries, including `numpy`, `pandas`, `seaborn`, and `scikit-learn`. To obtain the `tf-idf` score of a given word, as it occurs in a speech, you will need to fit `scikit-learn`'s `TfidfVectorizer` on a list of speeches, using `fit_transform()`⁵. Using the fitted vectorizer, you will need to extract the `tf-idf` score of each specified term as it occurs in each speech. Ultimately, you should aim to populate a `pandas DataFrame` that looks like the following:

	Date	Score	Term
0	1841-09-09	0.000000	america
1	1945-08-06	0.000000	america
2	1991-02-27	0.026612	america
..
992	1885-12-08	0.068363	united states
993	1963-11-27	0.012194	united states
994	1869-05-19	0.069120	united states

[1990 rows x 3 columns]

You can then pass this `DataFrame` to `seaborn` in order to generate the desired plot.

When working on the assignment, be mindful of the possible exceptions you can catch. For example, what happens if the directory passed to the `path` argument does not exist? Also, what if one or more of the terms passed to the `terms` argument are not in the speech vocabulary? What if none of them are? Instead of returning errors, you should instead aim to deal with these exceptions accordingly, letting the user know what the (potential) problem is and how it can be fixed. In your final submission, these cases need to be addressed, but it cannot hurt to include other types of exceptions as well.

⁵note that, when fitting the vectorizer, you'll need to 1) account for unigrams, bigrams, and trigrams occurring in your vocabulary, and 2) filter out English stopwords.