HW-1

Name: Ren, Jing

NetID: jrb1992

Problem 1

For this problem, I mainly compared two libraries: NLTK and Spacy. I applied word tokenization, stemming, and POS tagging on full text corpus, but Spacy does not contain any functions for stemming so I used lemmatization instead.

This table shows running time taken to complete each task:

	Tokenization	Stemming/Lemmatization	POS tagging	Parallelization
NLTK	3.33	8.78	18.84	171.13
Spacy	58.68	56.59	58.74	500+

In the most cases, Spacy returns results much faster than NLTK but costs much more memory. However, in my case, NLTK had better performance in terms of running time, which could due to my laptop's limited memory. Comparing the ease of parallelization, NLTK also had faster running time. Since it took too long to run parallelization for Spacy, I commented out this block of code to save time testing code.

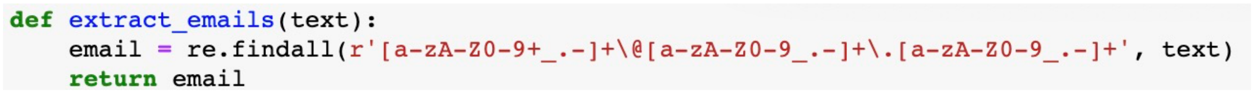
Despite the difference between their running time, I prefer the formatting of results returned by Spacy. When applying tokenization, Spacy has the support for word vectors while NLTK does not. In addition, when it comes to POS tagging, Spacy has much more elegant output where I can clearly tell if a word is a noun or a verb, while NLTK uses a code that is hard to interpret without checking documentation.

- Data Source: http://qwone.com/~jason/20Newsgroups/
- Relevant Code: https://github.com/MSIA/jrb1992_msia_text_analytics_2020/blob/homework1/HW1.py

Problem 2

Write and test regular expressions for the following tasks:

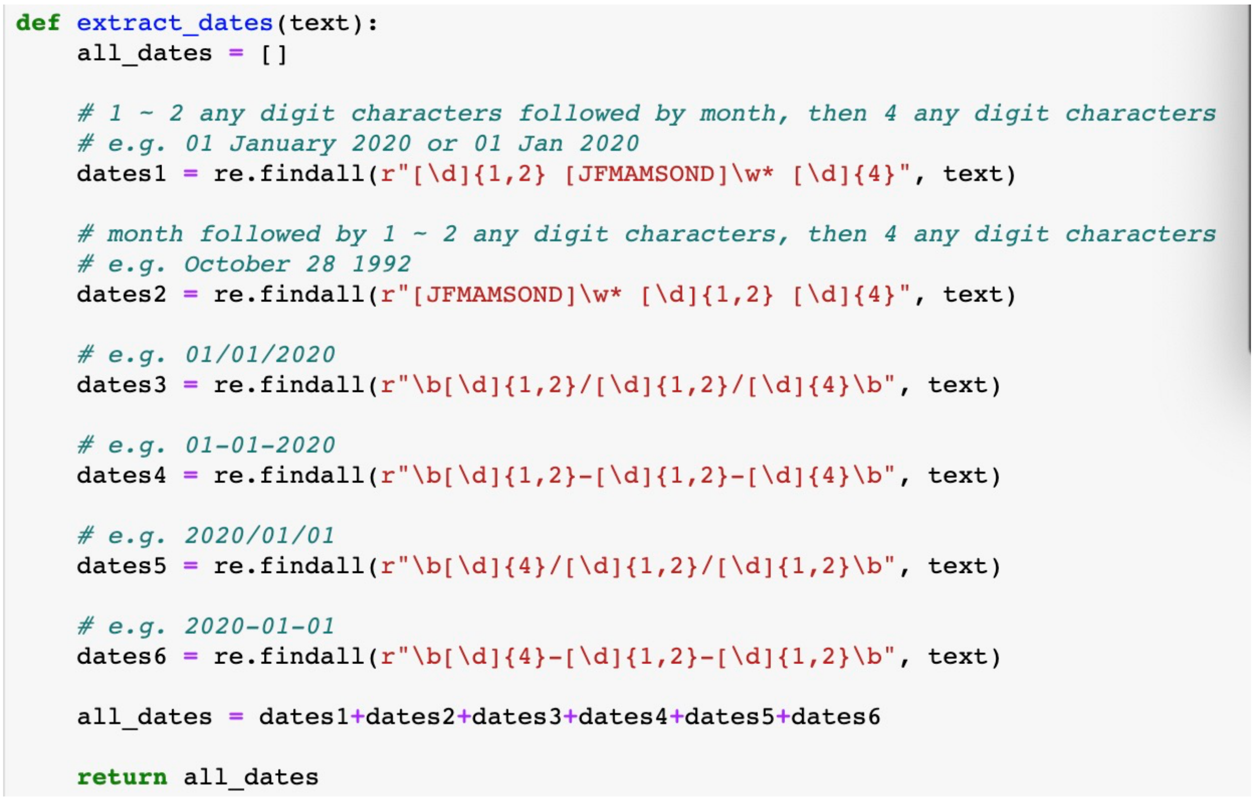
2.1 Match all emails in text and compile a set of all found email addresses.



```
def extract_emails(text):
    email = re.findall(r'[a-zA-Z0-9+_.-]+\@[a-zA-Z0-9_.-]+\.[a-zA-Z0-9_.-]+', text)
    return email
```

This expression finds strings start with lower, upper case letters, dash, dot, underscores, followed by @, then repeat the same collection of characters right before and after the dot in email addresses.

2.2 Find all dates in text (e.g. 04/12/2019, April 20th 2019, etc).



```
def extract_dates(text):
    all_dates = []

    # 1 ~ 2 any digit characters followed by month, then 4 any digit characters
    # e.g. 01 January 2020 or 01 Jan 2020
    dates1 = re.findall(r"[\d]{1,2} [JFMAMSOND]\w* [\d]{4}", text)

    # month followed by 1 ~ 2 any digit characters, then 4 any digit characters
    # e.g. October 28 1992
    dates2 = re.findall(r"[JFMAMSOND]\w* [\d]{1,2} [\d]{4}", text)

    # e.g. 01/01/2020
    dates3 = re.findall(r"\b[\d]{1,2}/[\d]{1,2}/[\d]{4}\b", text)

    # e.g. 01-01-2020
    dates4 = re.findall(r"\b[\d]{1,2}-[\d]{1,2}-[\d]{4}\b", text)

    # e.g. 2020/01/01
    dates5 = re.findall(r"\b[\d]{4}/[\d]{1,2}/[\d]{1,2}\b", text)

    # e.g. 2020-01-01
    dates6 = re.findall(r"\b[\d]{4}-[\d]{1,2}-[\d]{1,2}\b", text)

    all_dates  = dates1+dates2+dates3+dates4+dates5+dates6

    return all_dates
```

There are 6 cases that I considered.

1. Find all dates with 1 ~ 2 any digit characters followed by month, then 4 any digit characters, e.g. 01 January 2020 or 01 Jan 2020.
2. Find all dates with month followed by 1 ~ 2 any digit characters, then 4 any digit characters, e.g. October 28 1992.

3. Dates in form: 01/01/2020
4. Dates in form: 01-01-2020
5. Dates in form: 2020/01/01
6. Dates in form: 2020-01-01

- Relevant Code:
  https://github.com/MSIA/jrb1992_msia_text_analytics_2020/blob/homework1/HW1.py
- Collection of emails:
  https://github.com/MSIA/jrb1992_msia_text_analytics_2020/blob/homework1/emails.txt
- Collection of dates:
  https://github.com/MSIA/jrb1992_msia_text_analytics_2020/blob/homework1/dates.txt