Lesson Natural Language Processing

computer, understand my words

Much of what we have been doing in this classroom is within the domain of **N**atural **L**anguage **P**rocessing (i.e. NLP). It's a field that is very popular and important since a majority of what people create in terms of historical records is in the form of written text. NLP is about writing algorithms to help organize, process and understand text.

Two of the more popular Python libraries for NLP are NLTK (Natural Language ToolKit) and SpaCy. We will explore SpaCy in the advanced class. Another good tool that we will not explore is the Pattern library. You should look into it once your time frees up.

Building a better NLP Processor

We will attempt to find characters in a novel by coding a heuristic (lesson on finding characters). However, most NLP libraries have models (a system of rules built by an algorithm using lots of data) that were created using machine learning techniques for processing text -- for example, finding entities like people, places, and organizations is essentially a classification problem. The "models" (i.e. heuristics) you built were small and very fast. The models inside libraries like NLTK and SpaCy are large and were computationally expensive to build (and sometimes to run). However, they should perform better.

The one issue is that if a model was trained using data that is not similar to the data on which it will be fed or tested, the accuracy can be very poor. For example, a language model built on Project Gutenberg text would perform very poorly on Twitter streams. That's also an issue with NLTK -- some of its models are outdated.

> ***Learner's Log:*** *There is a rigor to typing it all by yourself, line by line that makes you understand these lessons better. When you copy&paste, you almost always take a macro view and conclude with 'ah, I get the primary logic'. When you type it in line by line, you are seeing each line and are forced to inspect it more. For the love of Python (and your brain), and a grade for this lesson, type in the code; line by line.*

NLP with NLTK

The NLTK (natural language toolkit) library is one of the most popular natural language processing libraries for Python. The project started in the early 2000's and supports multiple languages. You can visit the How To page to see all of its capabilities. We will do some quick demonstrations

Tokenizing

One of the basic functionalities of NLP is to tokenize text into sentences and words. In our examples, we used a regular expression to decide how to split the text into tokens. For NLTK, it's based on a more elaborate set of rules.

Run the following: (if an error is generated, read the next section for the steps to resolve it)

```
import nltk
def nltk_tokenize_demo(text):
  for sentence in nltk.sent_tokenize(text):
    tokens = nltk.word_tokenize(sentence)
    print(tokens)

demo = "This is a simple sentence. Followed by another!"
nltk_tokenize_demo(demo)
```

You will get an error about a model name in the Punkt corpus. So you need to download and install that model as well. Add the following:

```
import nltk
nltk.download('punkt') #sentence tokenizer
```

Re-run the demo. You should now see the tokens for each of the sentences. Note how the tokens include punctuation. You should think about how you would use a regular expression to build a sentence parser -- it's actually very difficult. You can do it, but as we will see later, it's easier to use a 2-pass algorithm.

Part of Speech Tagging

NLP also includes the ability for determining the parts of speech within a sentence. This includes finding nouns, pronouns, adjectives, verbs, adverbs, prepositions, conjunctions and interjections within a sentence. With NLTK, it's very simple to get these tags. Put the following in a new code cell:

```
def nltk_pos_demo(text):
  for sent in nltk.sent_tokenize(text):
    tokens = nltk.word_tokenize(sent)
    tagged = nltk.pos_tag(tokens)
    for t in tagged:
      print(t)
demo = "This is a simple sentence. Followed by another!"
nltk_pos_demo(demo)
```

```
# type&run the above example/exercise in this cell
```

When you run that cell, you get the following error:

Resource averaged_perceptron_tagger not found

You need to add another download:

```
import nltk
nltk.download('punkt') # sentence tokenizer
nltk.download('averaged_perceptron_tagger') # pos tagger
```

Note: It's best to gather all your imports in the first cell, even though they are shown in-line. It will make it much easier when you get ready to submit the notebook for grading. When you add a new import to that code cell, you will have to re-run the cell.

Run the above code cell again, and then run the block that uses nltk_pos_demo. The output should look like the following:

```
('This', 'DT')
('is', 'VBZ')
('a', 'DT')
('simple', 'JJ')
('sentence', 'NN')
('.', '.')
('Followed', 'VBN')
('by', 'IN')
('another', 'DT')
('!', '.')
```

In this example, the word 'simple' is an adjective (marked by the JJ tag). You can look up the meaning of each of the tags (e.g. JJ,VB, etc) here.

If you were interested in doing sentiment analysis (discussed later) using only adjectives and adverbs, getting the POS tags would be good start.

```
# type&run the above example/exercise in this cell
```

Named Entity Recognition (NER)

Getting labeled entities like People, Location, Organizations, Money, Dates from text is also a common use for NLP. Basically it's a classifier (a machine learning technique for labeling data). In the NLTK we just extend the pos demo:

```
def nltk_ne_demo(text):
  for sent in nltk.sent_tokenize(text):
    tokens = nltk.word_tokenize(sent)
    tagged = nltk.pos_tag(tokens)
    for chunk in nltk.ne_chunk(tagged):
      print(chunk)
demo = 'San Francisco considers banning sidewalk delivery robots'
nltk_ne_demo(demo)
```

```
# type&run the above example/exercise in this cell
```

When we run that we get another error: Resource maxent_ne_chunker not found. So we need to download that as well:

```
import nltk
nltk.download('punkt')                             # sentence tokenizer
nltk.download('averaged_perceptron_tagger')        # pos tagger
nltk.download('maxent_ne_chunker')                 # NE tagger
nltk.download('words')                             # needed for tagging
```

After you run that cell again, and the cell that contains nltk_ne_demo, you should now see:

```
(GPE San/NNP)
(PERSON Francisco/NNP) (
'considers', 'NNS')
('banning', 'VBG')
('sidewalk', 'NN')
('delivery', 'NN') ('robots', 'NNS')
```

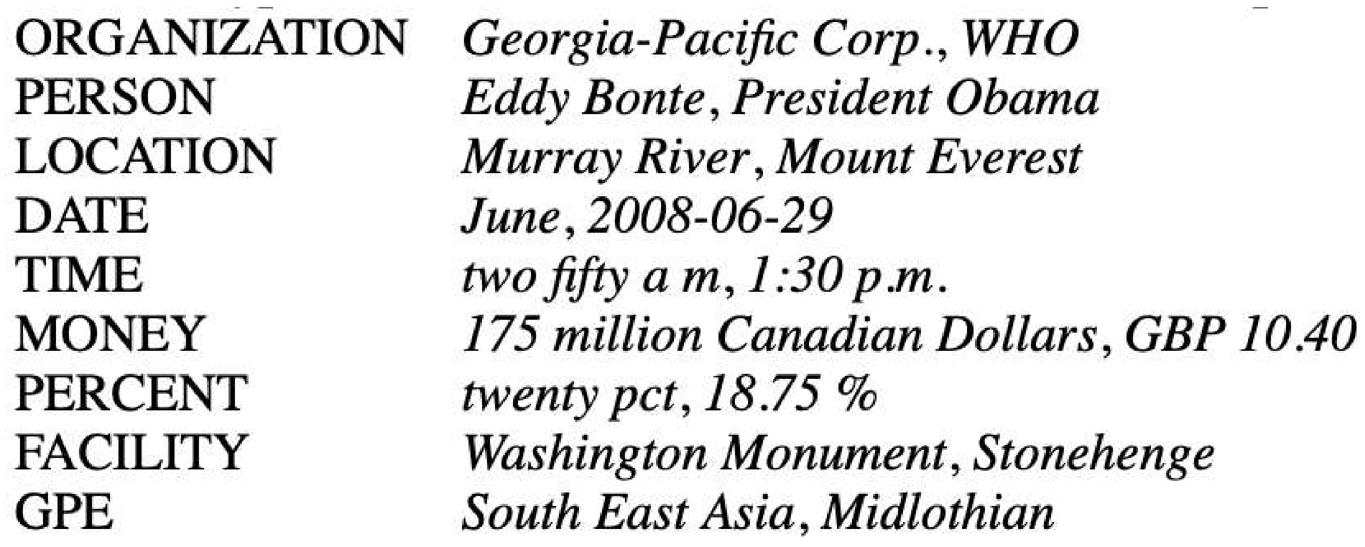
As a shortcut, you could just add nltk.download() to download everything. But sometimes it is good to be minimalist and know what the basic requirements are for each NLTK task. Also, downloading everything, takes a lot of time and space.

You should experiment with different sentences to get a feel for the accuracy of the model that finds named entities:

```
s1 = 'San Francisco considers banning sidewalk delivery robots'
s2 = 'In San Francisco, Aunt Polly considers paying sidewalk delivery robots $20.00.'
nltk_ne_demo(s2)
```

Note that in the s1, it thinks Francisco is a PERSON.

The following shows the common NLTK entity types (GPE: Geo-Political Entity):



ORGANIZATION	*Georgia-Pacific Corp., WHO*
PERSON	*Eddy Bonte, President Obama*
LOCATION	*Murray River, Mount Everest*
DATE	*June, 2008-06-29*
TIME	*two fifty a m, 1:30 p.m.*
MONEY	*175 million Canadian Dollars, GBP 10.40*
PERCENT	*twenty pct, 18.75 %*
FACILITY	*Washington Monument, Stonehenge*
GPE	*South East Asia, Midlothian*

To get a specific named entity, you have to work with the chunk entity:

```
def nltk_find_people_demo(text):
  for sent in nltk.sent_tokenize(text):
    tagged = nltk.pos_tag(nltk.word_tokenize(sent))
    for chunk in nltk.ne_chunk(tagged):
      if hasattr(chunk, 'label') and chunk.label() == 'PERSON':
        name = ' '.join(c[0] for c in chunk)
        print(name)
s3 = 'In San Francisco, Aunt Polly considers paying sidewalk delivery robots $20.00.'
nltk_find_people_demo(s3)
```

```
# type&run the above example/exercise in this cell
```

Fair Warning Reminder

If you leave your browser window open and it remains inactive, the browser may detach from the VM. If this happens you need to reconnect and re-run any code cells that download resources for doing NLP.

Stopwords

NLTK provides a list of stopwords to use as well:

```
 nltk.download('stopwords')
 from nltk.corpus import stopwords

 def nltk_stop_word_demo():
   stop_words = stopwords.words('english')
   print("nltk", stop_words)
```

```
# type&run the above example/exercise in this cell
```

Ngrams

The ngrams function from nltk allows you to create any sized n-grams:

```
 import collections
 from nltk import ngrams

 def nltk_ngram_demo(text):
   tokens = text.lower().split()
   grams = ngrams(tokens, 2)

   c = collections.Counter(grams)
   print(c.most_common(10))
```

```
text = "We went to a clump of bushes, and Tom made everybody swear to keep the secret, and th
nltk_ngram_demo(text)
```

Sentiment Analysis

Sentiment analysis is about extracting an opinion (positive, negative) from text. It can be done at different levels of granularity (i.e. document vs sentence). Although the topic of building our own sentiment analyzer would be fun, the details of doing so would be an advanced topic (i.e. involves machine learning). However, in the meantime, we can use NLTK's implementation:

```
nltk.download('vader_lexicon')
from nltk.sentiment.vader import SentimentIntensityAnalyzer

def nltk_sentiment_demo():
  sentiment_analyzer = SentimentIntensityAnalyzer()

  # helper
  def polarity_scores(doc):
      return sentiment_analyzer.polarity_scores(doc)
  doc1 = "INFO 490 is so fun."
  doc2 = "INFO 490 is so awful."
  doc3 = "INFO 490 is so fun that I can't wait to take the follow on course!"
  doc4 = "INFO 490 is so awful that I am glad there's not a follow on course!"

  print(polarity_scores(doc1)) # most positive
  print(polarity_scores(doc2)) # most negative
  print(polarity_scores(doc3)) # mostly positive, neutral
  print(polarity_scores(doc4)) # mostly negative, a little positive too

nltk_sentiment_demo()
```

Stemming and Lemmatization.

One of the underlying issues with the exercises that involved word frequency counters or dictionaries is that a word is separately counted even though it may exist in the dictionary but in a different form. For example, argue, arguing, argues, argued would all be distinct keys in our counter even though they essentially are the 'same' word. Both word stemming and word lemmatization are attempts to solve this issue.

Stemming

Stemming uses an algorithm (and/or a lookup table) to remove the suffix of tokens so that words with the same base but different inflections are reduced to the same form. For example: ‘argued’ and ‘arguing’ are both stemmed to the form: ‘argu’. The goal is to arrive at a common root form of the word. The two common algorithms are porter and lancaster.

The lancaster algorithm is more aggressive than porter. A good discussion on the complete Porter algorithm is given here and its implementation.

Lemmatization

Lemmatization reduces tokens to their lemmas, which is the canonical dictionary form. For example, ‘argued’ and ‘arguing’ are both lemmatized to ‘argue’. The process uses the part of speech of the word and then applies different rules based on its usage (e.g. The house is our home vs We will house them) in order to figure out the base form.

So the major difference between the two is that a lemma is a canonical form of the word, while a stem may not be a real word -- stemming can often create non-existent words, whereas lemmas are actual words.

NLTK provides an implementation of the different stemming algorithms:

```
def nltk_stem_and_lemm_demo():

  words = ["game","gaming","gamed","games","gamer","grows","fairly","nonsensical"]

  ps  = nltk.stem.PorterStemmer()
  sno = nltk.stem.SnowballStemmer('english')
  lan = nltk.stem.lancaster.LancasterStemmer()

  for word in words:
    base  = ps.stem(word)
    sbase = sno.stem(word)
    lbase = lan.stem(word)

    s = ''
    if (sbase != base):
      s += "(or {})".format(sbase)
    if (lbase != base and lbase != sbase):
      s += "(or {})".format(lbase)

    print("{:11s} stems to {:s} {}".format(word, base, s))
```

NLTK's Lemmatizer is based on WordNet which is an open source dictionary that closely resembles a thesaurus. It is a database of English words that are linked together by their semantic relationships.

```
 nltk.download('wordnet')
 def nltk_wordnet_demo():
   lemma = nltk.stem.WordNetLemmatizer()
   print(lemma.lemmatize('dogs'))
```

```
# type&run the above example/exercise in this cell
```

Accuracy

All of these models (both NLTK and SpaCy) will not give you perfect results. However, given a large enough sample and assuming the text you are analyzing is somewhat similar to the text that was used to train and build the underlying models, accuracy can be in the 80% - 95% range (see https://spacy.io/usage/facts-figures).

If accuracy is an issue for your particular analysis, you can even build your own models (either for a new language or from a new source of text) and integrate them with both spaCy and NLTK. Model building is an advanced topic that will be addressed in the sequel class (Data, Machines & the Python).

Before you go, you should know:

- the main use cases of nlp
- how to use nltk for tokenizing, ngrams, entity, part of speech tagging

Lesson Assignment

Please make sure you appreciate all the power you learned and re-read if you are unsure.

NLP and Finding Characters

You will use the power of nltk to find characters in a novel. Write the function find_characters_nlp. It will use nltk.

```
def find_characters_nlp(text, topn):
  # returns top n characters found in text
  return []
```

```
    []
```

Please make sure you appreciate all the power you learned and re-read if you are unsure.

```
def huck_test():
    with open('huck.txt') as fp:
        text = fp.read()
    text = 'huck.txt' #write code to read
    topn = find_characters_nlp(text, 6)
    print(topn)
huck_test()
```

You should see these numbers: ('Jim', 336) ('Tom', 150) ('Huck', 41) ('Tom Sawyer', 37) ('Aunt Sally', 37) ('Buck', 32)

Notes:

- note that with nltk, there's no need for stopwords, ngrams
- most of the code is already done for you

- what did you notice about the running time for how long it takes to find the characters using nlp?

Submission

After implementing all the functions and testing them please download the notebook as "solution.py" and submit to gradescope under "Week13:DS:NLP" assignment tab and Moodle.

NOTES

- Be sure to use the function names and parameter names as given.
- DONOT use your own function or parameter names.
- Your file MUST be named "solution.py".
- Comment out any lines of code and/or function calls to those functions that produce errors. If your solution has errors, then you have to work on them but if there were any errors in the examples/exercies then comment them before submitting to Gradescope.
- Grading cannot be performed if any of these are violated.

References and Additional Readings

- https://www.nltk.org/book/

POS

From https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html

1.	CC	Coordinating conjunction
2.	CD	Cardinal number
3.	DT	Determiner
4.	EX	Existential there
5.	FW	Foreign word
6.	IN	Preposition or subordinating conjunction
7.	JJ	Adjective
8.	JJR	Adjective, comparative
9.	JJS	Adjective, superlative
10.	LS	List item marker
11.	MD	Modal
12.	NN	Noun, singular or mass
13.	NNS	Noun, plural
14.	NNP	Proper noun, singular

```
15.    NNPS    Proper noun, plural
16.    PDT    Predeterminer
17.    POS    Possessive ending
18.    PRP    Personal pronoun
19.    PRP$    Possessive pronoun
20.    RB    Adverb
21.    RBR    Adverb, comparative
22.    RBS    Adverb, superlative
23.    RP    Particle
24.    SYM    Symbol
25.    TO    to
26.    UH    Interjection
27.    VB    Verb, base form
28.    VBD    Verb, past tense
29.    VBG    Verb, gerund or present participle
30.    VBN    Verb, past participle
31.    VBP    Verb, non-3rd person singular present
32.    VBZ    Verb, 3rd person singular present
33.    WDT    Wh-determiner
34.    WP    Wh-pronoun
35.    WP$    Possessive wh-pronoun
36.    WRB    Wh-adverb
```