CSC128_Lesson07_TextDataAnalytics_NLTK

October 22, 2023

```
[ ]: # First Steps with Python's NLTK Library
     # Uses Libraries:  NLTK
     # Runtime:  Google CoLab (cpu)
     # Traditional Machine Learning Approach
     #
     # Owner:  Lorrie Tomek
     #
     # Data:
     # variety of corpora downloaded at runtime from NLTK
     #
     # Reference:  Real Python
     # URL: https://realpython.com/python-nltk-sentiment-analysis/
      ↪#customizing-nltks-sentiment-analysis
     # The tutorial is freely available on the internet.  The content of this Google␣
      ↪Colab notebook has been modified for teaching purposes.
     # (Verified December 2022)
```

1 Text Data Analytics using NLTK

1.1 Learning Objectives

By the end of this tutorial, you'll be ready to:

- Split and filter text data in preparation for analysis
- Analyze word frequency
- Find concordance and collocations using different methods

1.2 What is NLTK?

The NLTK library has many utilities for manipulation and analysis of linguistic (natural language) data. These include text classifiers that can be used for many different types of classification, including sentiment analysis. .

To use NLTK, we need to install it using pip, and import the libraries we need.

```
[1]: # use pip to install the nltk library
     ! pip install nltk
```

```
Requirement already satisfied: nltk in /usr/local/lib/python3.10/dist-packages
(3.8.1)
```

```
Requirement already satisfied: click in /usr/local/lib/python3.10/dist-packages
(from nltk) (8.1.7)
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages
(from nltk) (1.3.2)
Requirement already satisfied: regex>=2021.8.3 in
/usr/local/lib/python3.10/dist-packages (from nltk) (2023.6.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages
(from nltk) (4.66.1)
```

[2]:
```
# import libraries we will use
import nltk
from nltk.stem import PorterStemmer
from pprint import pprint
```

1.3 Downloading NLTK Resources

NLTK has a large collection of resources that can be freely downloaded. Here is a list of some of them:

- names: A list of common English names compiled by Mark Kantrowitz
- stopwords: A list of really common words, like articles, pronouns, prepositions, and conjunctions
- state_union: A sample of transcribed State of the Union addresses by different US presidents, compiled by Kathleen Ahrens
- movie_reviews: Two thousand movie reviews categorized by Bo Pang and Lillian Lee
- twitter samples: A sample of twitter data
- averaged_perceptron_tagger: A data model that NLTK uses to categorize words into their part of speech
- vader_lexicon: A scored list of words and jargon that NLTK references when performing sentiment analysis, created by C.J. Hutto and Eric Gilbert
- punkt: A data model created by Jan Strunk that NLTK uses to split full texts into word lists
- shakespeare: A sample of text from Shakespeare

[3]:
```
# download some of the many nltk resources
nltk.download([
  "names",
  "stopwords",
  "state_union",
  "twitter_samples",
  "movie_reviews",
  "averaged_perceptron_tagger",
  "vader_lexicon",
  "punkt",
  "wordnet",
  "shakespeare"
])
```

```
[nltk_data] Downloading package names to /root/nltk_data…
[nltk_data]   Unzipping corpora/names.zip.
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data…
[nltk_data]   Unzipping corpora/stopwords.zip.
[nltk_data] Downloading package state_union to /root/nltk_data…
[nltk_data]   Unzipping corpora/state_union.zip.
[nltk_data] Downloading package twitter_samples to /root/nltk_data…
[nltk_data]   Unzipping corpora/twitter_samples.zip.
[nltk_data] Downloading package movie_reviews to /root/nltk_data…
[nltk_data]   Unzipping corpora/movie_reviews.zip.
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     /root/nltk_data…
[nltk_data]   Unzipping taggers/averaged_perceptron_tagger.zip.
[nltk_data] Downloading package vader_lexicon to /root/nltk_data…
[nltk_data] Downloading package punkt to /root/nltk_data…
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package wordnet to /root/nltk_data…
[nltk_data] Downloading package shakespeare to /root/nltk_data…
[nltk_data]   Unzipping corpora/shakespeare.zip.
```

[3]: `True`

2 What is a corpus?

A corpus is a large collection of related text samples. In the list above, we downloaded several corpra. Corpra is the plural form of corpus. We downloaded: state of the union, twitter samples, movie reviews as well as shakespeare. In the context of NLTK, corpora are compiled with features for natural language processing (NLP), such as categories and numerical scores for particular features.

3 Data Exploration

NLTK includes a number of functions that will help you meaningfully analyze text, even before doing anything more complex like machine learning. Many of NLTK's utilities are helpful in preparing your data for more advanced analysis.

Let's explore data using from our downloaded corpra to learn about frequency distribution, cordordance, and collections.

[6]:
```
# Let's start with the State of the Union corpus
# Let's look at the words in the state_union corpus
# We use the .words() method to get the words, and build a list of words that
# are alphabetic (so we don't include "words" that are punctuation symbols)
words = [w for w in nltk.corpus.state_union.words() if w.isalpha()]
# You can just type:  nltk.corpus.state_union_words() to see what you
# would get if you did not filter for stopwords

# Here we show the first 100 words:
pprint(words[:100], width=79, compact=True)
```

```
['PRESIDENT', 'HARRY', 'S', 'TRUMAN', 'S', 'ADDRESS', 'BEFORE', 'A', 'JOINT',
 'SESSION', 'OF', 'THE', 'CONGRESS', 'April', 'Mr', 'Speaker', 'Mr',
 'President', 'Members', 'of', 'the', 'Congress', 'It', 'is', 'with', 'a',
 'heavy', 'heart', 'that', 'I', 'stand', 'before', 'you', 'my', 'friends',
 'and', 'colleagues', 'in', 'the', 'Congress', 'of', 'the', 'United', 'States',
 'Only', 'yesterday', 'we', 'laid', 'to', 'rest', 'the', 'mortal', 'remains',
 'of', 'our', 'beloved', 'President', 'Franklin', 'Delano', 'Roosevelt', 'At',
 'a', 'time', 'like', 'this', 'words', 'are', 'inadequate', 'The', 'most',
 'eloquent', 'tribute', 'would', 'be', 'a', 'reverent', 'silence', 'Yet', 'in',
 'this', 'decisive', 'hour', 'when', 'world', 'events', 'are', 'moving', 'so',
 'rapidly', 'our', 'silence', 'might', 'be', 'misunderstood', 'and', 'might',
 'give', 'comfort', 'to', 'our']
```

3.0.1 Stopwords

In the list of words shown above, you will see many common words like "a", "the", "and". In NLP processing, it is often useful to filter out these common words called stopwords, because they do not add a lot to the meaning of the text.

We use the NLTK stopwords corpus to remove the stopwords. You will notice that the stopwords in that corpus are all lower case.

When we want to filter out words from our state_union word list we use str.lower() so that we compare lower case words to lower case words. If we don't do that, we end up with upper case or mixed case words like "The" and "THE" not being recognized and filtered as stop words.

[5]:
```
# Get a list of stopwords from the stopwords corpus
stopwords = nltk.corpus.stopwords.words("english")
pprint(stopwords[:100], width=79, compact=True)
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're",
 "you've", "you'll", "you'd", 'your', 'yours', 'yourself', 'yourselves', 'he',
 'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'it',
 "it's", 'its', 'itself', 'they', 'them', 'their', 'theirs', 'themselves',
 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those',
 'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had',
 'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if',
 'or', 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with',
 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after',
 'above', 'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off',
 'over', 'under', 'again', 'further', 'then', 'once']
```

[7]:
```
# Remove the stopwords
words = [w for w in words if w.lower() not in stopwords]
# notice in the filtered list below, there are no stopwords
pprint(words[:100], width=79, compact=True)
```

```
['PRESIDENT', 'HARRY', 'TRUMAN', 'ADDRESS', 'JOINT', 'SESSION', 'CONGRESS',
 'April', 'Mr', 'Speaker', 'Mr', 'President', 'Members', 'Congress', 'heavy',
 'heart', 'stand', 'friends', 'colleagues', 'Congress', 'United', 'States',
```

```
'yesterday', 'laid', 'rest', 'mortal', 'remains', 'beloved', 'President',
'Franklin', 'Delano', 'Roosevelt', 'time', 'like', 'words', 'inadequate',
'eloquent', 'tribute', 'would', 'reverent', 'silence', 'Yet', 'decisive',
'hour', 'world', 'events', 'moving', 'rapidly', 'silence', 'might',
'misunderstood', 'might', 'give', 'comfort', 'enemies', 'infinite', 'wisdom',
'Almighty', 'God', 'seen', 'fit', 'take', 'us', 'great', 'man', 'loved',
'beloved', 'humanity', 'man', 'could', 'possibly', 'fill', 'tremendous',
'void', 'left', 'passing', 'noble', 'soul', 'words', 'ease', 'aching',
'hearts', 'untold', 'millions', 'every', 'race', 'creed', 'color', 'world',
'knows', 'lost', 'heroic', 'champion', 'justice', 'freedom', 'Tragic', 'fate',
'thrust', 'upon', 'us']
```

3.1 Tokenization

For the purpose of this example, Python's string methods are sufficient to allow us to analyze the words in the corpus.

Looking ahead, there is a rich topic called **tokenization** which splits text into individual items called **tokens**. We may have **word tokenization**, which splits text into words which are seperated by spaces. We may want to remove punctuation during tokenization, or we may want punctuation symbols to be tokens. We may want the word "don't" to be one token: "don't" or multiple tokens: "do" "n't" or multiple tokens: "don" "'" "t".

Using nltk, you can build your own custom corpus. To do so, you would need to learn more about tokenization.

You can learn more about NLTK corpus at this URL: https://www.nltk.org/howto/corpus.html

For now, let's take a multi-line string that we hard-code in a variable called text, and use nltk's word_tokenize() method to get list of words.

[8]:
```
# here's some arbitrary multi-line text
example_text = """
For some quick analysis, creating a corpus could be overkill.
If all you need is a word list,
there are simpler ways to achieve that goal.
"""

# use nltk's word_tokenize() method get the tokens
example_text_tokens = nltk.word_tokenize(example_text)

# display the tokens for our example text
pprint(example_text_tokens, width=79, compact=True)

# OOPS.... see that there is a token that is a "."
# Let's filter out the non-alphabetic tokens
example_text_word_tokens = [w for w in nltk.word_tokenize(example_text) if w.
 ↪isalpha()]

pprint(example_text_word_tokens, width=79, compact=True)
```

```
['For', 'some', 'quick', 'analysis', ',', 'creating', 'a', 'corpus', 'could',
 'be', 'overkill', '.', 'If', 'all', 'you', 'need', 'is', 'a', 'word', 'list',
 ',', 'there', 'are', 'simpler', 'ways', 'to', 'achieve', 'that', 'goal', '.']
['For', 'some', 'quick', 'analysis', 'creating', 'a', 'corpus', 'could', 'be',
 'overkill', 'If', 'all', 'you', 'need', 'is', 'a', 'word', 'list', 'there',
 'are', 'simpler', 'ways', 'to', 'achieve', 'that', 'goal']
```

3.2 Stemming

[30]:
```python
from nltk.stem import PorterStemmer

# Initialize the Porter Stemmer
stemmer = PorterStemmer()

# here's some arbitrary multi-line text
example_text = """
For some quick analysis, creating a corpus could be overkill.
If all you need is a word list,
there are simpler ways to achieve that goal.
"""

# use nltk's word_tokenize() method get the tokens
example_text_tokens = nltk.word_tokenize(example_text)

# Stem each word in the list
stemmed_words = [stemmer.stem(word) for word in example_text_tokens]

# Print the original words and their stemmed forms
for i in range(len(stemmed_words)):
    print(f"Original: {example_text_tokens[i]}\tStemmed: {stemmed_words[i]}")
```

```
Original: For   Stemmed: for
Original: some  Stemmed: some
Original: quick Stemmed: quick
Original: analysis      Stemmed: analysi
Original: ,     Stemmed: ,
Original: creating      Stemmed: creat
Original: a     Stemmed: a
Original: corpus        Stemmed: corpu
Original: could Stemmed: could
Original: be    Stemmed: be
Original: overkill      Stemmed: overkil
Original: .     Stemmed: .
Original: If    Stemmed: if
Original: all   Stemmed: all
Original: you   Stemmed: you
Original: need  Stemmed: need
Original: is    Stemmed: is
```

```
Original: a     Stemmed: a
Original: word  Stemmed: word
Original: list  Stemmed: list
Original: ,     Stemmed: ,
Original: there Stemmed: there
Original: are   Stemmed: are
Original: simpler       Stemmed: simpler
Original: ways  Stemmed: way
Original: to    Stemmed: to
Original: achieve       Stemmed: achiev
Original: that  Stemmed: that
Original: goal  Stemmed: goal
Original: .     Stemmed: .
```

3.3 Frequency Distributions

Now let's look at frequency distributions. A frequency distribution is a table showing how many times each word appears within a given text.

In NLTK, frequency distributions are a specific object type implemented as a distinct class called FreqDist. This class provides useful operations for word frequency analysis.

To build a frequency distribution with NLTK, construct the nltk.FreqDist class with a word list:

[15]:
```
# Let's go back to looking at our state_union corpus
words = [w for w in nltk.corpus.state_union.words() if w.isalpha()]

# Get the Frequency Disbution using NLTK's FreqDist() method
fd = nltk.FreqDist(words)

# NLTK's FreqDist() method creates a python object of type nltk.probability.
 ↪FreqDist
# which is similar to a python dictionary, but has additional methods
print(type(fd))
```

```
<class 'nltk.probability.FreqDist'>
```

[16]:
```
# Find the most common words (in this case the top 5); notice what is returned␣
 ↪is a list of tuples
# showing that 'the' is the most common word, and that 'the' appars in the␣
 ↪corpus is 19,191 times
fd.most_common(5)
```

[16]:
```
[('the', 19191), ('of', 12854), ('to', 11868), ('and', 11748), ('in', 6936)]
```

[17]:
```
# Use the .tabulate() method to find the same information in a more readable␣
 ↪representation
fd.tabulate(5)
```

```
  the    of    to   and    in
19191 12854 11868 11748  6936
```

[18]:
```
# We can use the FreqDist to find the frequency of any word
fd["America"]
```

[18]:
```
1076
```

[19]:
```
# Each case of the letters is a different word token.
# Here we see that we have 3 cases of AMERICA, all in capitals
fd["AMERICA"]
```

[19]:
```
3
```

[20]:
```
# Here we can also see that if a word is not the corpus
# as "america" all lower case is not in the corpus, we do not
# get an exception as we would if we looked for the value of
# an item in a Python dictionary where the key does not exist
fd["america"]
```

[20]:
```
0
```

[21]:
```
# We can build a Frequency Distribution for words irregardless of
# their case (capitals/lower case)
words_lower = [w.lower() for w in nltk.corpus.state_union.words() if w.
 ↪isalpha()]
lower_fd = nltk.FreqDist(words_lower)

print(lower_fd["america"])
```

```
1079
```

3.4 Concordance

In NLP, a concordance is a collection of word locations along with their context.

- Number of times a word appears
- Position where each occurrence appears
- What words surround each occurrence

In NLTK, there is a Text object with a method called concordance(). As you will notice below, the case of the word is ignored, and just the words that surround each occurrence are shown.

[22]:
```
text = nltk.Text(nltk.corpus.state_union.words())
text.concordance("america", lines=5)
```

```
Displaying 5 of 1079 matches:
 would want us to do . That is what America will do . So much blood has already
ay , the entire world is looking to America for enlightened leadership to peace
beyond any shadow of a doubt , that America will continue the fight for freedom
```

```
 to make complete victory certain , America will never become a party to any pl
nly in law and in justice . Here in America , we have labored long and hard to
```

[23]:
```
concordance_list = text.concordance_list("america", lines=2)
for entry in concordance_list:
   print(entry.line)
```

```
 would want us to do . That is what America will do . So much blood has already
ay , the entire world is looking to America for enlightened leadership to peace
```

[24]:
```
example_words = nltk.word_tokenize(
"""Beautiful is better than ugly.
Explicit is better than implicit.
Simple is better than complex."""
)
example_text= nltk.Text(example_words)
# The line below is equivalent to: example_fd = nltk.FreqDist(example_words)
example_fd = example_text.vocab()
example_fd.tabulate(3)
```

```
   is better   than
    3      3      3
```

3.5 Collocations

Collocations are series of words that frequently appear together in a given text. In the State of the Union corpus, for example, you'd expect to find the words United and States appearing next to each other often. Those two words appearing together is called a collocation.

NLTK can quickly find collocations. Collocations can be made up of two or more words, typically 2-4 words, which are called:

- Bigrams: Frequent two-word combinations
- Trigrams: Frequent three-word combinations
- Quadgrams: Frequent four-word combinations

The general term is n-grams where n is the number of words collocated.

[25]:
```
# The NLTK collocations object has BigramCollocationFinder,␣
 ↪TrigramCollocationFinder
# and QuadgramCollocationFinder
words = [w for w in nltk.corpus.state_union.words() if w.isalpha()]
bifinder = nltk.collocations.BigramCollocationFinder.from_words(words)
trifinder = nltk.collocations.TrigramCollocationFinder.from_words(words)
quadfinder = nltk.collocations.QuadgramCollocationFinder.from_words(words)
```

[ ]:
```
# We can use any of the *gramFinder objects to find the frequency distribution
# of the n-grams.  We can therefore find the top-k most common, using the
# most_common() with parameter k.
print(f"Most Common 2:  {trifinder.ngram_fd.most_common(2)}")
```

```
Most Common 2:  [(('the', 'United', 'States'), 294), (('the', 'American',
'people'), 185)]
```