Natural Language Toolkit

Useful Functions & Cool Stuff

Eilidh Dunsmore

Table of Contents

Introduction (I)

What is NLTK?

Natural Language Toolkit (NLTK) is an open source python library used to build programs that work with human language data. Originally created by Steven Bird, Edward Loper, and Ewan Klein to be used in development and education, it contains more than fifty corpora and lexical sources for application by users such as Lin's Dependency Thesaurus, Penn Treebank Corpus, Open Multilingual Wordnet, and Problem Report Corpus. In addition, it includes text processing libraries for *tokenization*, *parsing, classification, stemming, tagging,* and *semantic reasoning.*

What is it used for?

The internet is a veritable cornucopia for those studying human language. Whether its researchers trying to determine the etymology of slang, lexicographers trying to determine the what new words to add to the dictionary, or even companies trying to determine product sentiment through social media, the internet provides a unique opportunity to map these things. Given the right tools, this data can be sorted through and analyzed. Raw information can be converted into useful intelligence that will be used to inform future advertising campaigns, product launches, and natural language processing projects.

How to install

These are the installation instructions for Mac OS X. I had a lot of difficulty, a) figuring out how to install this library and, b) trouble shooting the installation problems. With that in mind, what worked for me might not work for you and if that's the case you should do what I did and google it.

Step 1: *Know what version of Python you have.*

I have version 3.7 (for context) and in order for this installation to work you should have at least Python 3.5 which you can get by going to the *python.org* downloads page, double clicking on the .pkg file and following the prompts.

Step 2: *Install NLTK*

Open Terminal. Click around in your Applications folder until you find that if you don't already know where it is on your computer and then pin it to your toolbar (this is not a

necessary step, but I got tired of having to search for it all the time so I offer you the fruits of my mistakes). After you've opened Terminal, type "pip3 install -U nltk" and then click enter to execute.

Step 3: *Deal with the inevitable error messages.*

Okay, so you've downloaded NLTK. Congrats. That was the easy part. Now open whatever you're using to construct your programs (for me it's Sublime Text) and import NLTK (>>import nltk). It's unlikely that any programs you make with the NLTK library will work at this point and you should expect a certificate error upon trying to execute them (jump to *Useful Functions (II)* and try an exercise to be sure).

If you're like me, you'll receive a "certificate error" in which your computer becomes upset about how you've imported NLTK. Don't panic. I ended up installing certificates by finidng this: "/Applications/Python 3.7/Install Certificates.command" and then double clicking it to run and install. After that, no more certificate errors.

If that doesn't work for you, you can use this code to bypass the error:

```
######
try:
    _create_unverified_https_context = ssl._create_unverified_context
except AttributeError:
    pass
else:
    ssl._create_default_https_context = _create_unverified_https_context
######
```

The only warning about this is I don't understand what this code is doing or how it's working, but it does work.

Step 4: *One more installation.*

Now, when you try to utilize *stopwords*, you'll be greeted with another error as it requires a separate installation (however, it is relatively simple- thank God). This is what the error will look like:

```
**********************************************************************
  Resource stopwords not found.
  Please use the NLTK Downloader to obtain the resource:

  >>> import nltk
  >>> nltk.download('stopwords')

  Attempted to load corpora/stopwords

  Searched in:
    - '/Users/eilidh/nltk_data'
    - '/Library/Frameworks/Python.framework/Versions/3.7/nltk_data'
    - '/Library/Frameworks/Python.framework/Versions/3.7/share/nltk_data'
    - '/Library/Frameworks/Python.framework/Versions/3.7/lib/nltk_data'
    - '/usr/share/nltk_data'
    - '/usr/local/share/nltk_data'
    - '/usr/lib/nltk_data'
    - '/usr/local/lib/nltk_data'
**********************************************************************
```

And you should download “stopwords” to solve it:

```
atlantia:Computer Science eilidh$ python3
Python 3.7.0 (v3.7.0:1bf9cc5093, Jun 26 2018, 23:26:24)
[Clang 6.0 (clang-600.0.57)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import nltk
>>> nltk.download('stopwords')
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/eilidh/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
True
>>>
```

Step 5: *Now you’re done, except if you’ve gotta PC…*

If you have a Mac like me, you’re now finished with the installation process. However, if you have a PC… that’s rough. I will now refer those with non-Macs to the official NTLK installation page (which did not work for me, but it might work for you Window’s users): *https://www.nltk.org/install.html*

Useful Functions (II)

Tokenization

Tokenization is defined as "the act of breaking up a sequence of strings into pieces such as words, keywords, phrases, symbols, and other elements called tokens." Tokens can be anything, individual words, phrases, or even entire sentences.

NLTK supports sentence and word tokenization and I'm going to show you how to do both (tokenization is the basis of a lot of more complex programs, so really take the time to understand what it does and how it works).

Sentence Tokenization

```
######
## Begin by importing (I've shortened 'sent_tokenize' to 'sent', feel free to call it
## whatever you'd like
from nltk.tokenize import sent_tokenize as sent
## I established some example text so that I had something to tokenize
exampleText = "Last weekend I delivered a letter to the city council. I complained
about my terrible neighbors and their terrible children. I signed it: Sincerely,
Brenda."
## I then tokenized it
thisTextHasBeenTokenized = sent(exampleText)
## Then I printed it
print(thisTextHasBeenTokenized)
######
```

```
This is the example text tokenized:
['Last weekend I delivered a letter to the city council.', 'I complained about my terrible
neighbors and their terrible children.', 'I signed it: Sincerely, Brenda.']
```

The example text has been broken up into individual sentences and each sentence has become a 'token'. The variable thisTextHasBeenTokenized now essential works as a list in which you can call on various tokens.

```
######
print(thisTextHasBeenTokenized[0])
######
```

```
This is the example text tokenized:
['Last weekend I delivered a letter to the city council.', 'I complained about my terrible
neighbors and their terrible children.', 'I signed it: Sincerely, Brenda.']

Last weekend I delivered a letter to the city council.
```

You have now mastered sentence tokenization.

Word Tokenization

```
      ######
## Begin by importing (I've shortened 'word_tokenize' to 'word, feel free to call it
## whatever you'd like
from nltk.tokenize import word_tokenize as word
## Utilizing the example text from the last example…
exampleText = "Last weekend I delivered a letter to the city council. I complained
about my terrible neighbors and their terrible children. I signed it: Sincerely,
Brenda."
## I then tokenized it
imTokenized = word(exampleText)
## Then I printed it
print(imTokenized)
      ######
```

```
This is the example text tokenized:
['Last', 'weekend', 'I', 'delivered', 'a', 'letter', 'to', 'the', 'city', 'council', '.', 'I',
 'complained', 'about', 'my', 'terrible', 'neighbors', 'and', 'their', 'terrible', 'children',
 '.', 'I', 'signed', 'it', ':', 'Sincerely', ',', 'Brenda', '.']
```

Same as in sentence tokenization, the example text has been broken up into individual words with each word now a 'token'. You can continue to call upon each of the tokens using the list method we discussed above.

Frequency Distribution

Frequency distribution is a way of demonstrating how *frequencies* are *distributed* over values. Generally illustrated through *histograms* (a form of graph that is used to visualize frequencies for *intervals* of values rather than every distinct value).

Tl;dr: Frequency distribution is used to measure the amount of times a data point occurs rather than the data itself.

Using NLTK, you can map the frequency distribution of certain words and phrases utilizing *matplotlib* (which you don't need to be familiar with for this example, as I explain it, however, you do need to have downloaded the library & these instructions assume that you have done so).

Frequency Distribution & Printing

```
######
## Begin by importing
from nltk.probability import FreqDist
## You need to utilize text that has been tokenized by word (or phrase, but it's
## generally more useful to use tokenized words). We're going to use the previously
## established 'imTokenized' from the Word Tokenization example.
fdist = FreqDist(imTokenized)
print(fdist)
######
```

Now, this information contained within the print out is borderline useless:

```
The frequency distribution:
<FreqDist with 25 samples and 30 outcomes>
```

To remedy that, there is a way to show the most common of the distributed frequencies.

Most Common Distributed Frequencies

```
######
## Jumping right back in with the code we used/explained in the previous example
from nltk.probability import FreqDist
fdist = FreqDist(imTokenized)
## Now instead of printing 'fdist' we're going to ask to print the two most common
## distributions
print(fdist.most_common(2))
######
```

This printout demonstrates that within the set of tokenized words from the previous example ("Last weekend I delivered a letter to the city council. I complained about my terrible neighbors and their terrible

```
The top two most common distributions:
[('I', 3), ('.', 3)]
```

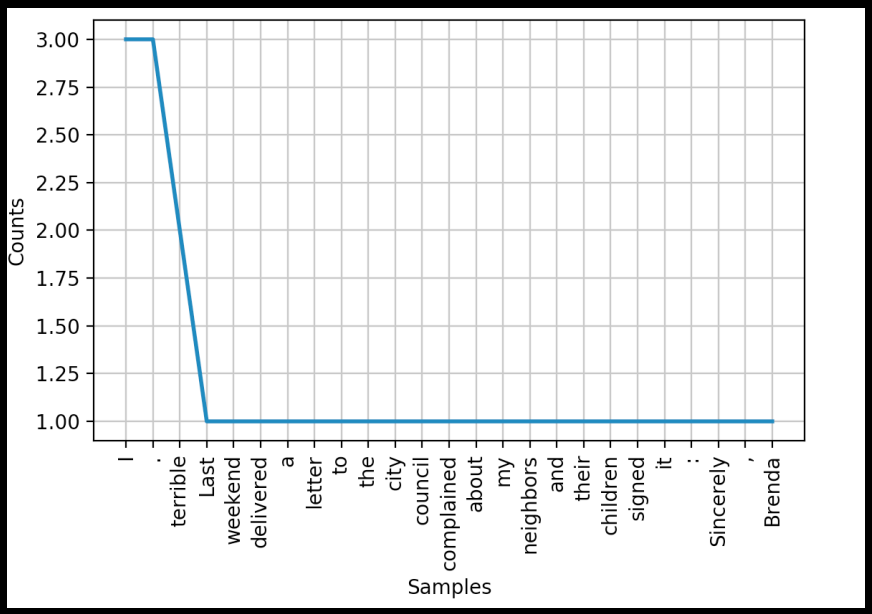
children. I signed it: Sincerely, Brenda.”), the top two most common tokens were ‘I’ occurring with a frequency of three and ‘.’ occurring, also, with a frequency of three.

There is a better, more readable way to present this information however: A histogram.

Frequency Distribution & Histograms

```
######
## Jumping right back in again…
from nltk.probability import FreqDist
fdist = FreqDist(imTokenized)
## Now instead of printing ‘fdist’ and its most common distributions, we’re going to
## plot it on a histogram using matplotlib
## Import matplotlib as plt
import matplotlib as plt
## Utilize the frequency distribution information to plot a histogram
fdist.plot(30, cumulative = False)
## Show the graph
plt.show()
######
```

This is the output of that graph:



If your goal is creating graphs/visual representations of frequency distribution, then I encourage you to look further into matplotlib. Matplotlib has tons of graph options/variants for those interested in sexier graphs.

Otherwise, this should be sufficient for your basic frequency distribution mapping needs.

Stopwords

In language science, “stop words” are words that can be filtered out. Sometimes described as “text noise”, stop words include words such as is, am, this, a, an, the, as well as others. You can use NLTK to remove stop words from the data you’re combing through which can be extremely helpful for when you are trying to map frequency distribution as it is unlikely that you are interested in knowing that the most used token in your data set was “the”.

Stop Words… What Are They?

```
######
## Begin by importing
from nltk.corpus import stopwords
## Define which stop words
stop_words = set(stopwords.words(“english"))
## Print stop words
print(stop_words)
######
```

Here’s a print out of all of the stop words included in the NLTK’s English library:

```
Stop words in English:
{'me', 'more', 'it', 'their', 'a', 'those', 'few', 'having', 'and', 'isn', 'your
selves', "that'll", 'd', "doesn't", "shan't", "you're", 'before', 'on', "mightn'
t", 'off', 'here', 'doing', "don't", 'hasn', 'both', 'does', "didn't", 'll', 'ai
n', 'some', 'be', 'shouldn', "hasn't", 'was', 'mightn', 'only', 'which', 'have',
 'should', 'themselves', 'so', 'had', 'she', 'he', "it's", 'who', 'each', "mustn
't", 'its', "needn't", 'this', 'in', 'over', 'until', "weren't", "wasn't", 'o',
'very', 'other', 'why', 'above', 'between', "aren't", 'while', 'needn', 'weren',
 'wouldn', 'below', 'then', 'doesn', 'ma', 'mustn', 'shan', 'has', 'whom', 's',
'haven', 'as', 'during', 'our', 'no', "shouldn't", 'out', 'most', 'own', 'hersel
f', 'am', 'to', 'itself', 'we', 'because', "couldn't", "wouldn't", 'into', 'if',
 'them', 'couldn', 'down', 'hadn', 'her', 'm', "haven't", 'your', 'being', 'you'
, 'once', 't', 'his', 'ourselves', 'my', 'can', 'is', 'at', 'any', 'that', 'an',
 'what', 'did', 've', 'were', 'against', 'but', 'hers', 'ours', 'yours', 'all',
'y', 'him', 'myself', 'the', 'up', 'himself', 'than', 'they', 'been', 'further',
 "won't", "you'd", 'about', "should've", 'now', 'not', 'do', 'of', 'will', 'didn
', "isn't", 'same', 'just', 'or', 'from', 'by', 'there', "hadn't", 'are', 'don',
 'where', 'yourself', 'under', 'again', 'how', 'for', "she's", 'with', "you'll",
 "you've", 'these', 'nor', 're', 'aren', 'after', 'theirs', 'such', 'wasn', 'won
', 'through', 'when', 'too', 'i'}
```

Now that you know (roughly) which words are considered “stop words” and which aren’t then we can move on to eliminating stop words from data sets where they are deemed nonessential.

Removing Stop Words

```
######
## Jumping back in…
from nltk.corpus import stopwords
stop_words = set(stopwords.words(“english"))
## In order to utilize ‘stopwords’ you must use text that has been tokenized by word
## (refer to above tutorials for answers on how to do that), so for that we’re going
## to use the text we’ve already tokenied: ‘imTokenized’.
## Create a new list for the filtered sentences (sentences w/out stop words)
filtered_words = []
## Use a ‘for’ loop to remove the stop words
for x in imTokenized:
    if x not in stop_words:
```

```
            filtered_words.append(x)
## Then print
print("Original:", imTokenized)
print("Filtered:", filtered_words)
      ######
```

```
Original: ['Last', 'weekend', 'I', 'delivered', 'a', 'letter', 'to', 'the', 'city'
, 'council', '.', 'I', 'complained', 'about', 'my', 'terrible', 'neighbors', 'and'
, 'their', 'terrible', 'children', '.', 'I', 'signed', 'it', ':', 'Sincerely', ','
, 'Brenda', '.']
Filtered: ['Last', 'weekend', 'I', 'delivered', 'letter', 'city', 'council', '.',
'I', 'complained', 'terrible', 'neighbors', 'terrible', 'children', '.', 'I', 'sig
ned', ':', 'Sincerely', ',', 'Brenda', '.']
```

The stop words 'a', 'to', 'the', 'about', 'my', 'and', as well as 'it' have all been removed. If you were to plot this new, filtered set of tokens onto a histogram, you would have essentially eliminated all of the "noise" present in your list of tokens.

Lexicon Normalization

Where “tokenization” is the process of separating a sequence into tokens, “normalization” is the steps taken to condense those terms into lexical units.

Stemming

For our purposes though, that’s a pretty complex and abstract definition. What I’m teaching you to do with lexicon normalization has to do with reducing another kind of linguistic “noise”. “Stemming” is a process that reduces derivationally related forms of a word to a common root word and chops off derivational affixes. For example: “help” and “helped” have the same root and stemming would allow you to reflect that in whatever visual model you’re building.

```
      ######
## Begin by importing (I’ve shortened it to be ‘ps’, call it whatever you’d like)
from nltk.stem import PorterStemmer as ps
## To demonstrate NLTK’s stemming capabilities, we’re going to use different
## example text & then go ahead and tokenize that
someText = “She wanted to cancel the trip. She canceled by calling on the phone and
telling the man on the other end about wanting to do so.”
## We’re going to tokenize the words
tokenizedWords = word(someText)
## Create a new list for the stemmed words to go into
stemmed_words = []
## Use a ‘for’ loop to normalize the words
For x in tokenizedWords:
      stemmed_words.append(ps.stem(x))
## Print
print(“Tokenized Sentence:”, tokenizedWords)
print(“Stemmed Sentence:”, stemmed_words)
      ######
```

```
Tokenized Sentence: ['She', 'wanted', 'to', 'cancel', 'the', 'trip', '.', 'She
', 'canceled', 'by', 'calling', 'on', 'the', 'phone', 'and', 'telling', 'the',
 'man', 'on', 'the', 'other', 'end', 'about', 'wanting', 'to', 'do', 'so', '.'
]
Stemmed Sentence: ['she', 'want', 'to', 'cancel', 'the', 'trip', '.', 'she', '
cancel', 'by', 'call', 'on', 'the', 'phone', 'and', 'tell', 'the', 'man', 'on'
, 'the', 'other', 'end', 'about', 'want', 'to', 'do', 'so', '.']
```

As you can see, words are de-capitalized, “wanted” and “wanting” and converted to their root “want”. If you wanted to figure out the distribution frequency of this example text, it would make sense to lump the words “wanted” and “wanting” together in the same bucket rather than treating them as their own, separate entities. Stemming allows you to do that.

Cool Stuff (III)

Importing text sources

One of the things I most wanted to do with this project was learn to apply the NLTK library to my own written work. So, I've learned to how to access text files on my computer and use them in Python.

Reading Local Files

It's actually pretty simple to import your local files into whatever program you're working on. As long as you make sure that your word file is saved as a ".txt" file in the folder you're already accessing to run your program (for me, I have to drag drop my .txt file into my "Computer Science" file for it to work), you should be good.

```
      ######
## I have sacrificed one of my own poems for your enjoyment. It is called "Flower
## Head" and I've saved it as 'flowerhead.txt' in my 'Computer Science' folder
flowerHead = open('Flower Head.txt')
## Now we're going to tokenize 'flowerHead' by word so we can do stuff with it
tokenizedFlowerHead = word(flowerHead.read())
## Now we're gonna take out all of the stop words, just for funsies
filtered_words = []
for x in tokenizedFlowerHead:
      if x not in stop_words:
            filtered_words.append(x)
## Add an asterisk before 'filtered_words' to clean up the output
print("Filtered Sentences:", *filtered_words)
      ######
```

Here's the original poem:

And underneath it is the poem after it's been tokenized by word and then had all of its stop words filtered out. You can utilize all of your new NLTK knowledge on whatever text files you do so wish.

```
There is a flower, sprouting out through my empty skull
I am long dead & currently a small plant is residing
inside my empty head
I can feel it grow & push
It wants to get big & grow-up & make its way as an adult
The sand beneath us is black and fine, and only the moon
shines in the sky
The flower grows without sun and without good soil
It forces its way through me until it finally reaches
open air
Its growth splits a crack right down my cranium
It asks for no permission and it feels no sorry for how
it desecrates my bones
It is life & the responsibility of life is to crunch the
dead underfoot as they go

Flower, make your way! I want to say
Do what I failed and make yourself strong
Get big & get fierce
And grow roots that wrap around your resting place and
keep me safe
```

```
Filtered Sentence: There flower , sprouting empty skull I long dead & currently small plant
 residing inside empty head I feel grow & push It wants get big & grow-up & make way adult
The sand beneath us black fine , moon shines sky The flower grows without sun without good
soil It forces way finally reaches open air Its growth splits crack right cranium It asks p
ermission feels sorry desecrates bones It life & responsibility life crunch dead underfoot
go Flower , make way ! I want say Do I failed make strong Get big & get fierce And grow roo
ts wrap around resting place keep safe
```

Sentiment gathering

Sentiment analysis is the automated process used to understand an opinion about something from written or spoken language. A real-world application of the NLTK is sentiment analysis. It can be incredibly useful for companies or political interest groups to understand their costumers or base. Sentiment analysis generally includes three components: *polarity* (is the opinion being expressed positive or negative), *subject* (what is being talked about), and *opinion holder* (the person/entity that is expressing the opinion in question).

We're going to do some basic (*very* basic… this is a very complex topic and I could have done a whole separate project on it it's so large) sentiment analysis on a product review I posted on Amazon about some headphones I like.

```
      ######
## Open the review
review = open('Amazon Headphones Review.txt')
## Tokenize by word
tokenizedReview = word(review.read())
## Create a new list for filtering stop words
filtered words = []
for x in tokenizedReview:
      if x not in stop_words:
            filtered_words.append(x)
## Create a new list for stemming
stemmed_words = []
for x in filtered_words:
      stemmed_words.append(ps.stem(x))
print("Filtered Review:", *stemmed_words)
## Search for sentiment-geared words, albeit, very crudely
goodTally = 0
badTally = 0
unknownTally = 0
for x in range(len(stemmed_words)):
      if (stemmed_words[x] == "good") or (stemmed_words[x] == "nice") or
(stemmed_words[x] == "best") or (stemmed_words[x] == "reliabl"):
            goodTally+= 1
      elif (stemmed_words[x] == "bad") or (stemmed_words[x] == "cheap") or
(stemmed_words[x] == "terribl"):
            badTally+= 1
      else:
            unknownTally+= 1
## Print results
print("\nGood-Sentiment words:", goodTally)
print("Bad-Sentiment words:", badTally)
```

```
print("Unknown-Sentiment words:", unknownTally)
      ######
```

This is the original review:

```
I buy exclusively these earbuds after first purchasing them on a whim. I will probably
continue to buy them until either they stop production of this particular line or I die.
Would strongly recommend, good sound, nice bass, and they stay in your ears nicely. By far
the best earbuds I've ever purchased, reliable, and cheap.
```

And this is the filtered and sentiment-gathered review:

```
Filtered Review:
 I buy exclus earbud first purchas whim . I probabl continu buy either stop product particular
 line I die . would strongli recommend , good sound , nice bass , stay ear nice . By far best
earbud I 've ever purchas , reliabl , cheap .

Good-Sentiment words: 5
Bad-Sentiment words: 1
Unknown-Sentiment words: 41
```

Obviously, as this is a *very* crude sentiment gathering system (I manually entered key words that I deemed bad) a result of which is even the incorrect filing of “cheap” under “Bad-Sentiment words” (something that would be corrected by integrating the ability to suss the meaning of a word through context into the program). However, generally, this program gets it right. The majority of the review contains good-sentiment words which generally means it is a positive review.

This is just a quick and easy example of basic sentiment gathering to give you an idea of what it means and how to do it. I’m sure you could put together a much cleverer and efficient sentiment mapping program than I have—which you should do. Mine’s a bit rubbish.

Sources (IV)

Krame, Aaron. "Introduction to Natural Language Processing, Part 1: Lexical Units." Data Science. Last modified January 24, 2017. Accessed January 30, 2019. https://www.datascience.com/blog/natural-language-processing-lexical-units.

Monkey Learn. "Sentiment Analysis: nearly everything you need to know." Monkey Learn. Accessed January 30, 2019. https://monkeylearn.com/sentiment-analysis/.

NLTK Project. "Natural Language Toolkit." Natural Language Toolkit -- NLTK 3.4 documentation. Last modified November 17, 2018. Accessed January 28, 2019. https://www.nltk.org.

SPSS Toutorials. "What Is A 'Frequency Distribution.'" SPSS Tutorials. Last modified 2018. Accessed January 29, 2019. https://www.spss-tutorials.com/frequency-distribution-what-is-it/.

Techopedia. "What is the Natural Language Toolkit (NLTK)?" Techopedia. Accessed January 28, 2019. https://www.techopedia.com/definition/30343/natural-language-toolkit-nltk.