Natural Language Processing in Python: Exploring Word Frequencies with NLTK

Natural Language Processing (NLP) is broadly defined as the manipulation of human language by software. It has its roots in linguistics but has evolved to encompass computer science and artificial intelligence, with NLP research largely devoted to programming computers to understand and process large amounts of natural language data, including speech and text. Today, NLP has wide-ranging applications in language translation, sentiment analysis, chatbots, voice assistants, and several more.

In the first of several upcoming tutorials in this series, we will explore one of the most basic tasks in NLP, word frequency analysis. While it is itself a comprehensive subject, we will be exploring a basic implementation using the Natural Language Toolkit or NLTK, a popular Python NLP library. The text we will be analyzing is the Great Gatsby, regarded as one of the greatest books ever written.

Importing the relevant libraries

To get started, we'll first need to import all the relevant libraries. If you haven't installed these already, feel free to do so using the pip install command.

```
#Import the relevant libraries
import nltk
from nltk import word_tokenize
from nltk.probability import FreqDist
import urllib.request
from matplotlib import pyplot as plt
from wordcloud import WordCloud
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
```

```
True
```

Loading the text

Next, we will need to source a copy of the text. We can easily grab a free copy from the Project Gutenberg website using the urllib.request library. Once we have it saved as a text file, we will need to read and decode it by using the read() and decode() functions respectively.

```
#Retrieve a copy of the text from the url
text_file = urllib.request.urlopen("https://www.gutenberg.org/cache/epub/64317/pg64317.txt")

#Read and decode the text
text = text_file.read().decode('utf-8')

#Preview a section of the text
print(text[1400:2000])
```

```
                          I

In my younger and more vulnerable years my father gave me some advice
that I've been turning over in my mind ever since.

"Whenever you feel like criticizing anyone," he told me, "just
remember that all the people in this world haven't had the advantages
that you've had."

He didn't say any more, but we've always been unusually communicative
in a reserved way, and I understood that he meant a great deal more
than that. In consequence, I'm inclined to reserve all judgements, a
habit that has opened up many curious natures to me and also made me
the vi
```

Tokenizing the words in the text

Our next task is to tokenize the words in the text. Word tokenization is the process of splitting the sentences in a given text into individual words. This is a requirement given that our goal is to determine the frequency of the words in the text. For this, we will use the word_tokenize() function.

```
#tokenize text by words
words = word_tokenize(text)

#check the number of words
print(f"The total number of words in the text is {len(words)}")
```

```
The total number of words in the text is 64006
```

The total number of words in the text is 64,006. We can now perform a count of each word occurrence using the FreqDist() function, and then select the top 10 words by using the most_common() method and passing in 10 as the argument.

```
#find the frequency of words
fdist = FreqDist(words)

#print the 10 most common words
fdist.most_common(10)
```

```
[(',', 3111),
 ('.', 2471),
 ('the', 2377),
 ('and', 1540),
 ('“', 1457),
 ('”', 1455),
 ('a', 1390),
 ('I', 1377),
 ('’', 1346),
 ('of', 1208)]
```

The output is a list of tuples with our top 10 most common words and their corresponding frequencies. But there is a problem.

Removing punctuation

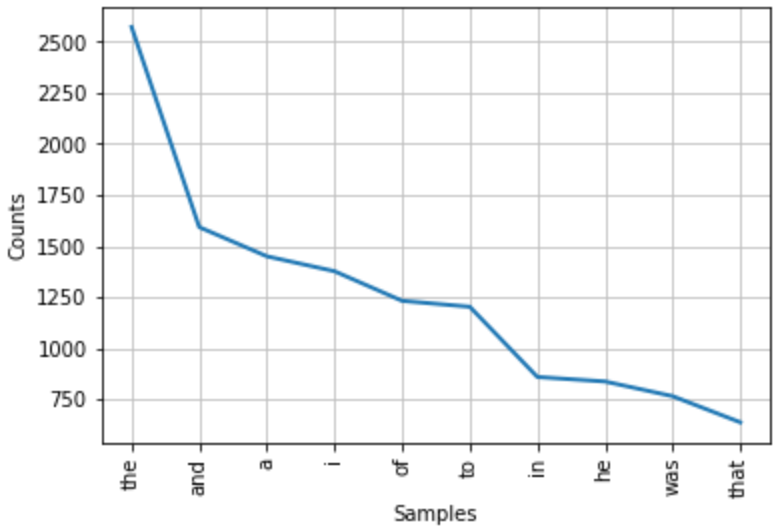
The list includes punctuations like commas, periods, and apostrophes. We will exclude these characters from our analysis.

```
#create an empty list to store words
words_no_punc = []

#iterate through the words list to remove punctuations
for word in words:
    if word.isalpha():
        words_no_punc.append(word.lower())

#print number of words without punctuation
print(f"The total number of words without punctuation is {len(words_no_punc)}")
```

```
The total number of words without punctuation is 50927
```

The total number of words is down to 50,927 with the exclusion of punctuations. Now we can go ahead and check out our list of top 10 words without punctuations, except this time, we will plot it on a graph using the plot() method.

```
#find the frequency of words
fdist = FreqDist(words_no_punc)

#Plot the 10 most common words
fdist.plot(10)
plt.show()
```



Removing stopwords

Now our top 10 list includes actual words without any punctuations. But we still have a problem. The list is populated entirely by stopwords. Stopwords are words in a language that occur frequently. They carry little semantic weight and don’t provide any information of value to the reader. Words like 'the', 'an', 'they', 'and' etc. are usually considered as stop words.

While no universal list of stopwords exists, the NLTK library does come with its own list of stopwords that we can use. We will only need to download and import it.

```
#Download and import list of stopwords
nltk.download("stopwords")
from nltk.corpus import stopwords

#list of stopwords
stopwords_list = stopwords.words("english")
print(stopwords_list)
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll",
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

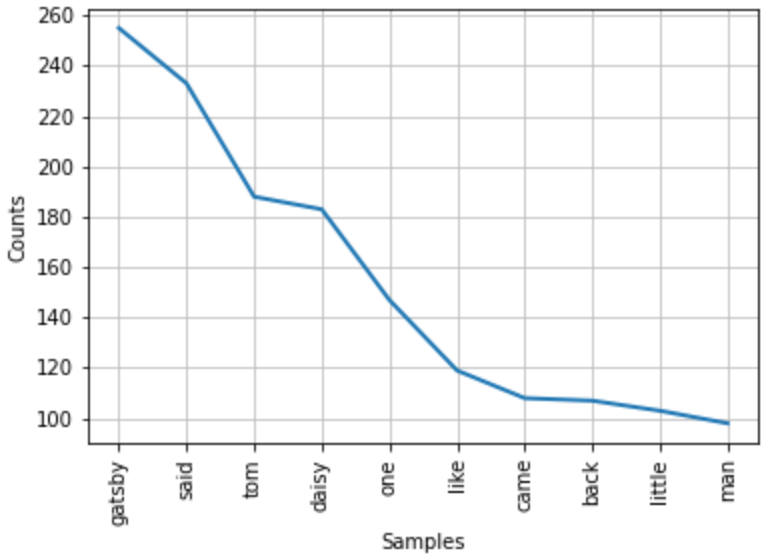
As you can see from the output, our list of stopwords is an actual python list object. To filter out stopwords from our words_no_punc list, we can easily write a for loop to iterate through every word on that list and append any word that is not in the list of stopwords to a new list, which we can call clean_words.

```
#create an empty list to store clean words
clean_words = []

#Iterate through the words_no_punc list and add non stopwords to the new clean_words list
for word in words_no_punc:
    if word not in stopwords_list:
        clean_words.append(word)

print(f"The total number of words without punctuation and stopwords is {len(clean_words)}")
```

```
The total number of words without punctuation and stopwords is 23783
```

After removing all words from NLTK's list of stopwords, we now have a total word count of 23,783. Let's plot our 10 most common words on a graph.

```
#find the frequency of words
fdist = FreqDist(clean_words)

#Plot the 10 most common words
fdist.plot(10)
plt.show()
```



And now we have a more meaningful list that includes the names of some characters in the book. There are still some words in the list, like 'said', 'like', 'came' and 'back' that could count as stopwords. Bear in mind that designating a word as a stopword is discretional and depends largely on the objective of the analysis and the text in question. For our purposes, the aforementioned words can be excluded from the text as they contribute little to our understanding of the subject matter of the novel.

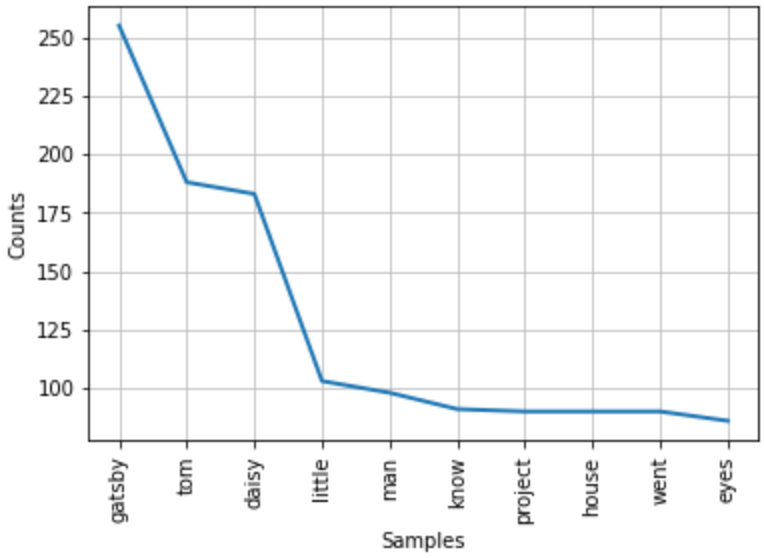
Updating the stopwords

Luckily, we don't need to create a completely new list. Remember the NLTK stopwords list is a regular Python list object? Well, that means we can simply add new words to it by using the extend() method.

```
#Update the stopwords list
stopwords_list.extend(["said","one","like","came","back"])

#create an empty list to store clean words
clean_words = []

#Iterate through the words_no_punc list and add non stopwords to the new clean_words list
for word in words_no_punc:
    if word not in stopwords_list:
        clean_words.append(word)
```

With our list of stopwords updated and applied, we can go ahead and visualize the final iteration of our top 10 most common words.

```
#find the frequency of words
fdist = FreqDist(clean_words)

#Plot the 10 most common words
fdist.plot(10)
plt.show()
```



And there we have it. A semantically rich list that not only tells us who the main characters in the novel are but also points to what things or places feature heavily in the novel.

Building the word cloud

Our final task is to create a word cloud. A word cloud is an image that is composed of the words in a text, where the size of each word varies depending on its frequency.

The wordcloud library in Python makes it easy to build a word cloud. We already imported it at the beginning of this tutorial, so we can go ahead and create a WordCloud object, and then call the generate() method on it. Note that the generate() method accepts a string object as its argument, so we will first convert our clean_words list into a string of words with a space between them. This can be implemented by using the join() method.

```
#Convert word list to a single string
clean_words_string = " ".join(clean_words)

#generating the wordcloud
wordcloud = WordCloud(background_color="white").generate(clean_words_string)

#plot the wordcloud
plt.figure(figsize = (12, 12))
plt.imshow(wordcloud)

#to remove the axis value
plt.axis("off")
plt.show()
```

As you can see, a word cloud is a very intuitive way of visualizing word occurrences in a text. With just a glance, we can easily tell which words occur most frequently.

Conclusion

In this code-along tutorial, we have covered the basics of word frequency analysis, including acquiring a text, performing tokenization, removing punctuations and stopwords, visualizing the frequencies on a chart, and building a word cloud. I hope that this was useful to you and that you learned something new. Thank you for reading, and all the best in your data journey!