Getting Started with NLTK

NLTK (Natural Language Toolkit) is a powerful Python library for working with human language data. It provides easy-to-use interfaces to multiple text corpora and lexical resources, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and more.

Task 1: Getting Started

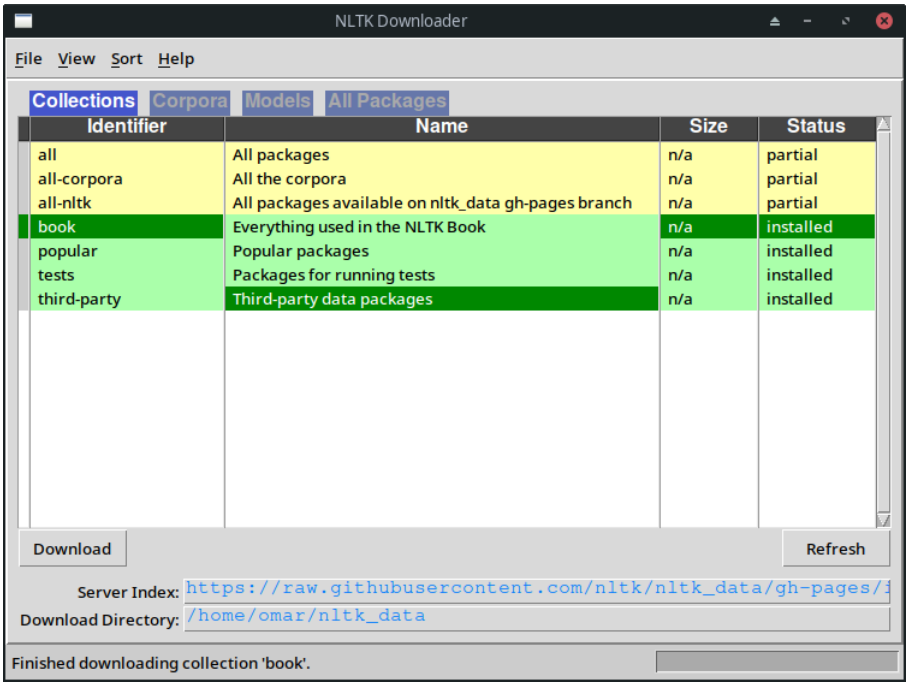
Firstly, ensure you have NLTK installed. If you haven't installed it yet, use pip (Python's package installer) in your command line:

```
pip install nltk
```

Browse the available packages using

```
nltk.download()
```

Here, you will see something similar to the following:



In the first tab “collection”, there is a list of standard download options. You can be selective in what packages and corpora you need right now or download all of them if space is not an issue. For this exercise, we would be needing the “book” collection so you can download them. You can also manually download without the UI interface as:

```
nltk.download('book')
```

Then, import the book.

```
from nltk.book import *
```

The book will be loaded as text1, text2, …. You can view these texts by giving text1 on the prompt to see their details.

```
text1
```

We will interact with these collections using the following:

1. **Set:** To see all unique vocabulary in a given book, you can use:

```
set(text1)
```

   Or to view in sorted by frequency, use:

```
sorted(set(text1))
```

2. **Length:** To view the size of the corpora, use:

```
len(text3)
```

3. **Lexical Richness:** We can determine the ratio of unique words / total words to determine the richness of a collection by using the previous two parameters.

```
def lexicalDiversity(text):
      return len(text) / len(set(text))

lexicalDiversity(text1)
```

4. **Concordance:** Search within a collection and return a word + some context

```
text1.concordance("monster")
```

5. **Similar:** The search above gives exact matching. To search for similar words, use:

```
text1.similar("monster")
```

6. **Common Context:** Each time the queries are executed, it returns some additional text (left and right of the word) as context. We can search for words in terms of common context as:

```
text1.common_contexts(["monster", "person"])
```

7. **Dispersion Plots:** To see the occurrence of a word in a text appears, how many times and where in a text, you can use dispersion plots (will not work in shell but in jupyter notebook)

```
text1.dispersion_plot(["monster", "evil", "devil"])
```

8. **Count:** To count the number of times a word appears in a text, use count.

```
text1.count("monster")
```

9. **Probability**: To determine the likelihood of a word appearing in a collection, we can define:

```
def P(search, text):
      return text.count(search) / len(text) * 100

P("monster", text1)
```

10. **FreqDist:** To extract the frequency distribution of words in diffeent collections, we use the FreqDist() method as follows:

```
fdist1 = FreqDist(text1)
```

    You can play around with the frequency and analyse it using the following:

```
fdist1.keys()
fdist1.plot(50, cumulative=True)
fdist1.hapaxes()
fdist1.most_common(10)
```

Q1: Now, fill the following table:

Corpus	Text1	Text2	Text3	Text4	Text5	Text6	Text7	Text8	Text9
Collection Name	Moby Dick	Sense & Sensibility	The Book of Genesis	Inaugural Address corpus	Chat Corpus	Monty Python	Wall Street Journal	Personals Corpus	Man who was Thursday

Collection Length	260819	141576	44764	156288	45010	16967	100676	4867	69213
Unique Words	19317	6833	2789	10200	6066	2166	12408	1108	6807
Lexical Richness	13.502	20.719	16.05	15.32	7.42	7.83	8.11	4.39	10.16

Task 2: Moving to Corpora

In task 1, we saw text collections in the form of **nltk.books**, which comprise of 9 different collections of text. Actual corpora (singular: corpus) are different. The book collection is useful for teaching purpose, but corpora are actual curated texts which are more important for language analysis. The corpora are present in **nltk.corpus**. Note that there is still a distinction between tools for corpora analysis and text analysis.

For the moment, we can load some of them as:

```
from nltk.corpus import gutenberg, brown
```

We can interact using the following commands:

1. **Fileids()**: To see the filenames of collections in the corpora, you can use file ID's fileids() method:

```
gutenberg.fileids()
brown.fileids()
```

   You can iterate through them as normally would using python, e.g.

```
for fileid in gutenberg.fileids():
      # do stuff here
```

   For the remaining few codes, we are going to select only Shakespeare's Macbeth:

```
macbeth = 'shakespeare-macbeth.txt'
```

2. **Categories()**: Some corpora have collections inside categories which represent genres of text. For instance, the brown corpus contains categories of adventure, news, religion, reviews, science_fiction, etc.

```
brown.categories()
```

3. **Raw()/Words()/Sents()**: List the character string, Words, and Sentences in the fileids of the corpora:

```
gutenberg.raw(fileids=macbeth)            # or gutenberg.raw(macbeth)
gutenberg.words(fileids=macbeth)          # or gutenberg.words(macbeth)
gutenberg.sents(fileids=macbeth)          # or gutenberg.sents(macbeth)
```

   You can tweak around in some corpora to get words of a particular genre, for instance:

```
brown.words(categories='news')
```

   To get their count, you will use len() function. For example, to get the total words in the entire gutenberg collection, you will use:

```
sum = 0
for fileid in gutenberg.fileids():
      sum = sum + len(gutenberg.words(fileid))
```

   To get the unique words, i.e. the vocabulary, you will use set() as previously in Task 1, or len(set()) as in Task 1 to get the count of the vocabulary.

4. **Plaintext Corpus Reader:** To move to your own text files as corpus, the PlaintextCorpusReader can take your plain text file(s) from disk and convert them into a corpus. You can specify multiple files to the same root, provided they have similar naming convention. For this you will need to provide a

regular expression to specify the naming convention being used.

```
from nltk.corpus import PlaintextCorpusReader

corpusRoot  = '/home/omar/work/codes/python/nlp/corpora'
myCorpus    = PlaintextCorpusReader(corpusRoot, '.*\.txt')
```

A sample text file in Urdu has been provided for you to try out (**file1.txt**). Then, you can proceed with corpora analysis using the techniques mentioned above in this section so far. For text analysis, you will still need to convert the corpus to nltk.Text mode as:

```
corpusWords = myCorpus.words()
corpusText  = nltk.Text(corpusWords)
```

5. **XML Corpus Reader:** Some corpora available online are in the form of XML documents. For this, XMLCorpusReader can take XML based file(s) from disk. For this exercise, we are choosing some Urdu based xml corpus provided at Makhzan.

```
git clone https://github.com/zeerakahmed/makhzan
```

Here, in the text folder are several xml files representing the corpora. We can load them as:

```
from nltk.corpus.reader import XMLCorpusReader

corpus_root = '/home/omar/work/codes/python/nlp/corpora/makhzan/text'
makhzan     = XMLCorpusReader(corpus_root, '.*\.xml')
```

Note that the conventions of regular expressions are similar over here also. The corpus "makhzan" is now loaded and you can do corpus analysis. Some analysis, for example sentences are not possible because NLTK doesn't know the boundaries of a sentence in an XML document. For this, and other text analysis, it is necessary to move the corpus over to nltk.Text mode for manual analysis. But since it is an xml document, some extra processing would be required to extract the text from the xml format. In our case, the xml format is:

```
<document>
  <meta>
                <title>پرویز مشرف میڈیا کو چکمہ دیکر پاکستان سے روانہ ہو گئے</title>
    <author>
      <name>سبو خسید</name>
      <gender>N/A</gender>
    </author>
    <publication>
          <name>آئی بی سی اردو</name>
      <year>2016</year>
      <city>Islamabad</city>
      <link>http://ibcurdu.com/news/19681/</link>
      <copyright-holder>آئی بی سی اردو</copyright-holder>
    </publication>
    <num-words>640</num-words>
    <contains-non-urdu-languages>
      No
    </contains-non-urdu-languages>
  </meta>
```

```
  <body>
    <section>
کراچی.......حکومتکیطرفسےایگزٹکنٹرولسٹسےنامنکالےجانےکےبعدپرویزمشرفایمرٹسائرلائنمینصبحتینبجکر55منٹپردبئیکیلئےروانہ<p>
</p>ہوگئے۔وہاپنےقریبیدوستطارقمحمودکےساتھبزنسکلاسکےکیبنمینسفرکےلیےسوارہوئے۔
</p>اصفہزردارینےکہاکہہمکبعدالتونکاسامناکرانےکےبجائےعدالتونسےبھاگتےرہینگے۔<p>
    </section>
  </body>
</document>
```

We can see that the actual content for analysis is in the xml tags /document/body/section/p. In XML documents, you can use Xpath query to reach to these locations. Let's see how it is done:

```
corpus_words = [ word
 for fileid in makhzan.fileids()
 for p in makhzan.xml(fileid).findall('.//body/section/p')
 if p.text
 for word in nltk.word_tokenize(p.text)
]

makhzanText = nltk.Text(corpus_words)
```

For the sentences (delineated in english using the dot symbol), we can use:

```
from nltk import sent_tokenize, word_tokenize

raw_text = ' '.join(makhzan.words())
sentences = [word_tokenize(sent) for sent in sent_tokenize(raw_text)]
```

Whereas for Urdu (delineated using the symbol ۔), we can use:

```
from nltk import sent_tokenize, word_tokenize

raw_text  = ' '.join(makhzanText)
sentences = [s.strip() for s in raw_text.split('۔') if s.strip()]
sentences = [word_tokenize(sent) for sent in sentences]
```

Q2: Now, fill the following table:

Corpus	gutenberg	brown	webtext	nps_chat	Custom Plain Text	Custom Makhzan
Total Characters	11793318	9964284	1726164	2568552		72
Total Words	2621613	1161192	396733	45010		255
Total Sentences	98552	57340	25733	10567		3
Total Unique Words	41487	40234	16453	4671		63
Average Word Length (Characters / Words)	4.17	4.683	4.262	3.813		3.54
Average Sentence Length (Words / Sentences)	26.60	20.25	15.417	4.259		24
Lexical Richness (Words / Vocabulary)	51.47	24,40	18.601	7.34		1.142

Task 3: Term Frequency

We have already seen the createion of functions lexicalDiversity() and P() in Task 1. Similar to these, define now a function TF(), IDF(), and TFIDF() which can compute the equation:

$$TF_{t,d} = \log_{10}(count_{t,d} + 1)$$

$$IDF_t = \log_{10}(N / DF_t)$$

$$TFIDF_{t,d} = TF_{t,d} * IDF_t$$

Some functions to help you are:

```
text1.count('monster')
math.log(12345, 10)
```

Some of the most probable + lengthy words from our corpora can be identified by running:

```
selected = [w for w in fdist1.keys() if len(w) > 5 and fdist1[w]/len(text1)*100 > .1]
```

Acquire the unique sorted words (without punctuations) from text1 and store them in a list called **words**. Example of text1 is given below (but you can use stop words stop punctuations also)

```
words = sorted(set(text1))[280:]
```

You can browse through this list with any set of variations. For example:

```
longwords = [w for w in words if len(w) > 16]

high_freq = [w for w in words if fdist1[w] > 500]

high_freq_idf = [w for w in words if IDF(text1, w) > 1]

eign_words = [w for w in words if w.endswith('eign')
```

You can also use functions such as w.startswith(a), w.endswith(z), substring in w, w.islower(), w.isupper(), w.istitle() for camel case, w.isalpha(), w.isdigit(), etc. with the conditions. Note you can also use regular control structures for these activities:

```
for w in words:
    if w.endswith('eign'):
        print(w)
```

Now, attempt the following:

Q3: Fill the following table:

	Text1	Text2	Text3
Number of words with length > 16	13	13	13
Number of words ending in “ed”	2196	2196	2196
Number of words with frequency > 500	58	37	10
Number of words with probability between 0 and 0.05	19031	19031	19031
Number of words with probability between 0.05 and 0.1	1	1	1
Number of words with probability between 0.1 and 100.	0	0	0
List of 3 words having length > 7 and having largest probability in dataset.	Through, captain, himself	Through, himself, without	Without, through, himself

Q4: Prepare a table where rows contain the identified 9 words from Q3, and columns represent TF(), IDF(), and TFIDF(). Fill the table with your values.

Task 4: Bigrams & Trigrams

Moving towards language models, we would first convert a given text corpus to its bigram model using:

```
list(bigrams(text1))
```

You can view the first 10 by appending [:10] to the given command. To view the most frequent bigrams in the text, you can use the collocations feature of NLTK as:

```
text1.collocations()
```

The difference between collocation and bigram is that collocations are two words that occur unusually often. As an example, “red apple” is a collocation as well as bigram. Whereas “the apple” is a bigram but not a collocation.

For any model higher than a bigram, you can use ngrams from NLTK as:

```
from nltk.util import ngrams

list(ngrams(text1, 3))
```

For usage of trigram with collocations, you can use:

```
from nltk.collocations import *

TrigramCollocationFinder.from_words(text1).nbest(TrigramAssocMeasures().pmi,  10)
```

Q4: Now, fill the following table for the most common bigrams and trigrams in the given texts:

	Text1	Text2	Text3	Text4	Text5
10 frequently occuring Bigrams	[((',', 'and'), 2607), (('of', 'the'), 1847), (("'", 's'), 1737), (('in', 'the'), 1120), ((',', 'the'), 908), ((';', 'and'), 853), (('to', 'the'), 712), (('.', 'But'), 596), ((',', 'that'), 584), (('.', '"'), 557)]	[((',', 'and'), 1598), (("'", 's'), 700), ((';', 'and'), 605), (('Mrs', '.'), 529), (('of', 'the'), 430), (('to', 'be'), 428), (('."', '"'), 428), ((',', '"'), 392), (('.', '"'), 369), (('in', 'the'), 348)]	: [((',', 'and'), 1491), (('.', 'And'), 1038), (('of', 'the'), 372), (('in', 'the'), 287), ((';', 'and'), 262), (('said', ','), 259), (("'", 's'), 255), (('And', 'he'), 192), (('And', 'the'), 185), (('said', 'unto'), 178)]	[(('of', 'the'), 1784), ((',', 'and'), 1404), (('in', 'the'), 765), (('to', 'the'), 716), (('of', 'our'), 635), (('.', 'The'), 606), (('.', 'We'), 562), (('and', 'the'), 470), ((',', 'the'), 388), (('.', 'It'), 354)]	[(('.', 'ACTION'), 346), (('PART', 'JOIN'), 191), (('JOIN', 'PART'), 136), (('PART', 'PART'), 115), (('JOIN', 'JOIN'), 107), (('I', "'m"), 89), (('.', 'JOIN'), 79), (('pm', 'me'), 76), (('in', 'the'), 75), (('are', 'you'), 65)]
10 frequently occuring Collocations	[(',', 'and'), ('of', 'the'), ("'", 's'), ('in', 'the'), (',', 'the'), (';', 'and'), ('to', 'the'), ('.', 'But'), (',', 'that'), ('.', '"')]	[(',', 'and'), ("'", 's'), (';', 'and'), ('Mrs', '.'), ('of', 'the'), ('."', '"'), ('to', 'be'), (',', '"'), ('.', '"'), ('in', 'the')]	s: [(',', 'and'), ('.', 'And'), ('of', 'the'), ('in', 'the'), (';', 'and'), ('said', ','), ("'", 's'), ('And', 'he'), ('And', 'the'), ('said', 'unto')]	[('of', 'the'), (',', 'and'), ('in', 'the'), ('to', 'the'), ('of', 'our'), ('.', 'The'), ('.', 'We'), ('and', 'the'), (',', 'the'), ('.', 'It')]	[('.', 'ACTION'), ('PART', 'JOIN'), ('JOIN', 'PART'), ('PART', 'PART'), ('JOIN', 'JOIN'), ('I', "'m"), ('.', 'JOIN'), ('pm',

					'me'), ('in', 'the'), ('are', 'you')]
5 frequently occuring Trigrams	[((',', 'and', 'the'), 187), (('don', "'", 't'), 103), (('of', 'the', 'whale'), 101), ((',', 'in', 'the'), 93), ((',', 'then', ','), 87)]	[(('Mrs', '.', 'Jennings'), 230), (('Mrs', '.', 'Dashwood'), 121), ((',', 'however', ','), 88), ((',', 'and', 'the'), 87), (('.', 'Mrs', '.'), 80)]	[(('.', 'And', 'he'), 162), (('.', 'And', 'the'), 158), (('the', 'land', 'of'), 101), (('he', 'said', ','), 86), (('And', 'he', 'said'), 84)]	[(('.', 'It', 'is'), 155), (('the', 'United', 'States'), 154), ((',', 'and', 'the'), 145), (('.', 'We', 'have'), 102), (('of', 'the', 'United'), 101)]	[(('PART', 'JOIN', 'PART'), 35), (('JOIN', '.', 'ACTION'), 34), (('.', 'ACTION', 'is'), 33), (('JOIN', 'PART', 'JOIN'), 33), (('PART', 'JOIN', 'JOIN'), 25)]