NLTK Basics

Introduction

The Python's Natural Language Toolkit (NLTK) (http://www.nltk.org) is a suite of libraries and programs for Natural Language Processing for English. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries.

Download and Install nltk

In [ ]:
```
!pip install nltk
```

You can install nltk by using `pip` (See http://www.nltk.org/install.html (http://www.nltk.org/install.html)). NLTK is included in Anaconda (https://www.continuum.io/downloads). I recommend you to use Anaconda to ease python package management tasks.

After downloading nltk, you should install nltk.data (http://www.nltk.org/data.html). NLTK comes with many corpora, toy grammars, trained models, etc. A complete list is posted at: http://nltk.org/nltk_data/ (http://nltk.org/nltk_data/).

In [ ]:
```
import nltk
nltk.download()
```

A new window should open, showing the NLTK Downloader. Click on the File menu and select Change Download Directory. For central installation, set this to C:\nltk_data (Windows), /usr/local/share/nltk_data (Mac), or /usr/share/nltk_data (Unix). Next, select the packages or collections you want to download.

In [1]:
```
import nltk
nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\Dileep\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

Out[1]:
```
True
```

`punkt` is a Sentence Tokenizer. This tokenizer divides a text into a list of sentences.

Basic Text Processing Tasks

Natural Language Content Analysis is a fundamental step in every text mining project. Depending on the text mining task, you may want to generate representations of text data in different levels. For instance, sentences in a text data may be simply represented as a **bag of words** or may be annotated with semantic word classes.

- Sentence segmentation
- Word Tokenization
- Word Lemmatization
- Word Stemming
- Filtering stop words.
- Part-of-speech tagging

1. Sentence segmentation

In many cases, we want to split a document or paragraph into a list of sentences. For instance, we want to identify sentiment of a sentence in the sentiment analysis task, or we may want to analyze structures of sentences.

To illustrate how to do it with `nltk` , we first create a paragraph.

In [3]:
```
para = "Hello World. It's good to see you. Thanks for taking this course."
```

Now, we want to split `para` into sentences. We will use module `nltk.tokenize` to do that.

In [4]:
```
from nltk.tokenize import sent_tokenize
sent_tokenize(para)
```

Out[4]:
```
['Hello World.', "It's good to see you.", 'Thanks for taking this course.']
```

Now, we have a list of sentences for further processing.

If you're going to be tokenizing a lot of sentences, it's more efficient to load the `PunktSentenceTokenizer` once, and call its `tokenize()` method instead.

In [5]:
```
import nltk.data
tokenizer = nltk.data.load('tokenizers/punkt/english.pickle')
tokenizer.tokenize(para)
```

Out[5]:
```
['Hello World.', "It's good to see you.", 'Thanks for taking this course.']
```

2. Word Tokenization

Tokenization is process of splitting a text object into smaller units. Smaller units can be words, numbers, symbols, ngrams, characters.

We can do the task with the basic word tokenization by using the function `word_tokenization` .

In [6]:
```
from nltk.tokenize import word_tokenize
sent = 'The history of NLP generally starts in the 1950s, although work can be found from earlier periods.'
print(word_tokenize(sent))
```

```
['The', 'history', 'of', 'NLP', 'generally', 'starts', 'in', 'the', '1950s', ',', 'although', 'work', 'can', 'be', 'found', 'from', 'earlier', 'periods', '.']
```

We obtain a list of tokens as above.

In [7]:
```
word_tokenize("I can't swim.")
```

Out[7]:
```
['I', 'ca', "n't", 'swim', '.']
```

We can have many alternatives for word tokenization. The above task can also be done using `TreebankWordTokenizer` :

In [8]:
```
from nltk.tokenize import TreebankWordTokenizer
tokenizer = TreebankWordTokenizer()
print(tokenizer.tokenize(sent))
```

```
['The', 'history', 'of', 'NLP', 'generally', 'starts', 'in', 'the', '1950s', ',', 'although', 'work', 'can', 'be', 'found', 'from', 'earlier', 'periods', '.']
```

`TreebankWordTokenizer` use conventions in Penn Treebank corpus.

WordPunctTokenizer

`WordPunctTokenizer` splits all punctuations into separate tokens.

In [9]:
```
from nltk.tokenize import WordPunctTokenizer
tokenizer = WordPunctTokenizer()
tokenizer.tokenize("I can't swim.")
```

Out[9]:
```
['I', 'can', "'", 't', 'swim', '.']
```

RegexpTokenizer

We can use regular expression to tokenize words in a sentence.

In [10]:
```
from nltk.tokenize import RegexpTokenizer
tokenizer = RegexpTokenizer("[\w']+")
tokenizer.tokenize("I can't swim.")
```

Out[10]:
```
['I', "can't", 'swim']
```

Or we can do in a simpler way.

In [11]:
```
from nltk.tokenize import regexp_tokenize
regexp_tokenize("I can't swim.", "[\w']+")
```

Out[11]:
```
['I', "can't", 'swim']
```

We can use simple whitespaces as delimiters for word tokenization.

In [12]:
```
tokenizer = nltk.RegexpTokenizer('\s+', gaps=True)
tokenizer.tokenize("I can't swim.")
```

Out[12]:
```
['I', "can't", 'swim.']
```

In [13]:
```
nltk.RegexpTokenizer?
```

3. Word Lemmatization

As a definition, lemmatization is to "**remove inflectional endings only and to return the base or dictionary form of a word, which is known as the lemma.**"

First, we create a raw text and tokenize it into tokens using the function `word_tokenize`.

We use WordNet lemmatizer for word lemmatization. The WordNet lemmatizer removes affixes only if the resulting word is in its dictionary.

In [14]:
```
nltk.download('wordnet')
```

```
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\Dileep\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

Out[14]:
```
True
```

In [15]:
```
wnl = nltk.WordNetLemmatizer()
text = "studies studying cries cry"
tokenization = nltk.word_tokenize(text)
for w in tokenization:
    print("Lemma for {} is {}".format(w, wnl.lemmatize(w)))
```

```
Lemma for studies is study
Lemma for studying is studying
Lemma for cries is cry
Lemma for cry is cry
```

4. Stemming

Stemming is to chop off ends of words using some rules. In NLTK, we have several Stemmer to do the job. The following code will try two stemmers in nltk.

In [16]:
```
from nltk.stem.porter import PorterStemmer
ps  = PorterStemmer()
text = "studies studying cries cry"
tokenization = nltk.word_tokenize(text)
for w in tokenization:
    print("Stemming for {} is {}".format(w,ps.stem(w)))
```

```
Stemming for studies is studi
Stemming for studying is studi
Stemming for cries is cri
Stemming for cry is cri
```

In [17]:
```
from nltk.stem.lancaster import LancasterStemmer
lc = nltk.LancasterStemmer()
text = "studies studying cries cry"
tokenization = nltk.word_tokenize(text)
for w in tokenization:
    print("Stemming for {} is {}".format(w,lc.stem(w)))
```

```
Stemming for studies is study
Stemming for studying is study
Stemming for cries is cri
Stemming for cry is cry
```

Stemming is a general operation while lemmatization is an intelligent operation where the proper form will be looked in the dictionary. Hence, lemmatization helps in forming better machine learning features.

Use Case of Lemmatizer:

- Lemmatizer minimizes text ambiguity. Example words like bicycle or bicycles are converted to base word bicycle. Basically, it will convert all words having the same meaning but different representation to their base form.
- It reduces the word density in the given text and helps in preparing the accurate features for training machine. Cleaner the data, the more intelligent and accurate your machine learning model, will be.
- Lemmatizer will also saves memory as well as computational cost.

5. Filtering stop words

Stop words are common words that do not contribute to the meaning of a sentence. In general, earch engines and text mining systems filter stop words in the preprocessing step.

We can do that by creating a set of English stop words.

In [18]:
```
from nltk.corpus import stopwords
english_stops = set(stopwords.words('english'))
words = ["Can't", 'is', 'a', 'contraction']
[word for word in words if word not in english_stops]
```

Out[18]:
```
["Can't", 'contraction']
```

In [19]:
```
# listing stop words
set(stopwords.words('english'))
```

```
Out[19]: {'a',
  'about',
  'above',
  'after',
  'again',
  'against',
  'ain',
  'all',
  'am',
  'an',
  'and',
  'any',
  'are',
  'aren',
  "aren't",
  'as',
  'at',
  'be',
  'because',
  'been',
  'before',
  'being',
  'below',
  'between',
  'both',
  'but',
  'by',
  'can',
  'couldn',
  "couldn't",
  'd',
  'did',
  'didn',
  "didn't",
  'do',
  'does',
  'doesn',
  "doesn't",
  'doing',
  'don',
  "don't",
  'down',
  'during',
  'each',
  'few',
  'for',
  'from',
  'further',
  'had',
  'hadn',
  "hadn't",
  'has',
  'hasn',
  "hasn't",
  'have',
  'haven',
  "haven't",
```

```
'having',
'he',
'her',
'here',
'hers',
'herself',
'him',
'himself',
'his',
'how',
'i',
'if',
'in',
'into',
'is',
'isn',
"isn't",
'it',
"it's",
'its',
'itself',
'just',
'll',
'm',
'ma',
'me',
'mightn',
"mightn't",
'more',
'most',
'mustn',
"mustn't",
'my',
'myself',
'needn',
"needn't",
'no',
'nor',
'not',
'now',
'o',
'of',
'off',
'on',
'once',
'only',
'or',
'other',
'our',
'ours',
'ourselves',
'out',
'over',
'own',
're',
's',
'same',
```

```
'shan',
"shan't",
'she',
"she's",
'should',
"should've",
'shouldn',
"shouldn't",
'so',
'some',
'such',
't',
'than',
'that',
"that'll",
'the',
'their',
'theirs',
'them',
'themselves',
'then',
'there',
'these',
'they',
'this',
'those',
'through',
'to',
'too',
'under',
'until',
'up',
've',
'very',
'was',
'wasn',
"wasn't",
'we',
'were',
'weren',
"weren't",
'what',
'when',
'where',
'which',
'while',
'who',
'whom',
'why',
'will',
'with',
'won',
"won't",
'wouldn',
"wouldn't",
'y',
'you',
```

```
"you'd",
"you'll",
"you're",
"you've",
'your',
'yours',
'yourself',
'yourselves'}
```

6. Part-of-speech tagging

Part-of-speech(POS) tagging is the process of marking up a word in a corpus to a corresponding part of a speech tag, based on its context and definition. That it POS tagging is a process of assigning tags to words in a sentence.

This task is not straightforward, as a particular word may have a different part of speech based on the context in which the word is used.

For example: In the sentence "Give me your answer", answer is a Noun, but in the sentence "Answer the question", answer is a verb.

To understand the meaning of any sentence or to extract relationships and build a knowledge graph, POS Tagging is a very important step.

Explore Tagged Corpora (Brown Corpus)

Several corpora included with NLTK have been tagged for their part-of-speech. Brown Corpus is an example. If you open a file in the Brown Corpus, you can see tagged sentences like the following example.

```
The/at Fulton/np-tl County/nn-tl Grand/jj-tl Jury/nn-tl said/vbd Friday/nr an/at investigation/nn of/in Atlanta's/np$ recent/jj primary/nn election/nn produced/vbd ``/`` no/at evidence/nn ''/'' that/cs any/dti irregularities/nns took/vbd place/nn ./.
```

Each token in the sentence is associated with a POS tag.

Now we show some tagged words in the corpus.

In [20]:
```
nltk.download('brown')
```

```
[nltk_data] Downloading package brown to
[nltk_data]     C:\Users\Dileep\AppData\Roaming\nltk_data...
[nltk_data]   Package brown is already up-to-date!
```

Out[20]:
```
True
```

In [21]:
```
nltk.corpus.brown.tagged_words()
```

Out[21]:
```
[('The', 'AT'), ('Fulton', 'NP-TL'), ...]
```

Show tagged words with universal tagset.

In [22]:
```
nltk.download('universal_tagset')
```

```
[nltk_data] Downloading package universal_tagset to
[nltk_data]     C:\Users\Dileep\AppData\Roaming\nltk_data...
[nltk_data]   Package universal_tagset is already up-to-date!
```

Out[22]:
```
True
```

In [23]:
```
nltk.corpus.brown.tagged_words(tagset='universal')
```

Out[23]:
```
[('The', 'DET'), ('Fulton', 'NOUN'), ...]
```

Perform automatic tagging

In `nltk` , we can perform automatic POS tagging as follows.

In [24]:
```
text = word_tokenize("And now for something completely different")
print(nltk.pos_tag(text))
```

```
[('And', 'CC'), ('now', 'RB'), ('for', 'IN'), ('something', 'NN'), ('complete
ly', 'RB'), ('different', 'JJ')]
```

The default pos tagger model using in NLTK is maxent_treebank_pos_tagger model (Maxent model trained on the treebank corpus).

Example: Use of Wordnet Lemmatization and POS Tagging

In [25]:
```
nltk.download('averaged_perceptron_tagger')
```

```
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\Dileep\AppData\Roaming\nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

Out[25]:
```
True
```

References

- Bird, Steven; Klein, Ewan; Loper, Edward (2009). *Natural Language Processing with Python*. [http://www.nltk.org/book/ (http://www.nltk.org/book/)](http://www.nltk.org/book/)

In [ ]:
```

```