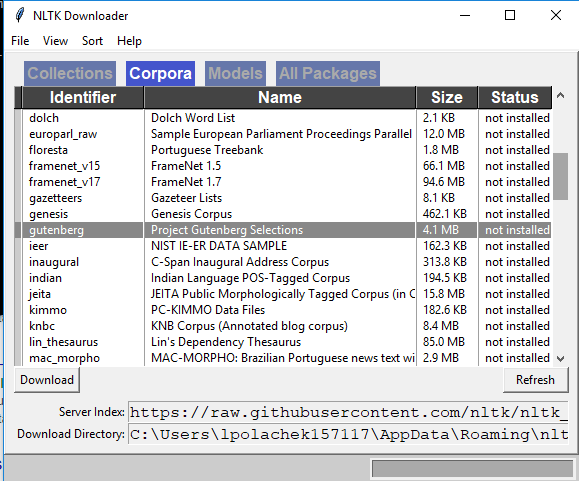
Today we will learn how to use the nltk (natural language toolkit) python library. This library enables us to perform numerous operations connected to natural language processing. It contains corpora (databases of texts) and dictionaries in different langauges. This library is the first and basic tool for nlp in python.

1. Install the Gutenberg corprus from nltk by typing nltk.download() and then choosing and installing the appropriate corpora from the popup GUI as in the image below:



1. Explore file austen-persuasion.txt from the gutenberg corpus. How many words does the file have? How many different words are there? Use regular python commands on the contents of the file.
   1. to import the corpora use: from nltk.corpus import gutenber
   2. to read content of a file as characters use the command raw: gutenberg.raw(file\_name)
   3. to read the word content of a file use: gutenberg.words(file\_name)
   4. to read the sentence content of a file use: gutenberg.sents(file\_name)
2. A **trigram** is a tuple of three words. You can generate trigrams for any text. Use nltk.trigram to produce trigram from the top sentence: <http://www.nltk.org/api/nltk.html#nltk.util.trigrams>
3. A **frequency distribution** tells you how often a given word appears in a corpus. NLTK library has a built-in class to compute frequency distribution: FreqDist. How many times does the word **the** appear in the bible-kjv.txt file? <http://www.nltk.org/api/nltk.html?highlight=freqdist>
   1. Use fd.freq()
4. What are the ten most common word in the file?
   1. Use fd.most\_common()
5. Find all words that occur at least ten times in brown corpus.
   1. Use fd.iteritems() to iterate over all words in the corpus
6. How frequently do letters appear in English text?
   1. Use swadesh corpora. Choose the file ‘en’ which contains the English version of the data.
   2. To see the list of all files in the corpus use: swadesh.fileids()
   3. Use isalpha() to make sure a character is a letter and not a punctuation or space

**WordNet** is a semantically oriented dictionary of English. It’s aware that some words are synonyms.

from nltk.corpus import wordnet as wn

We can find all **synsets** of a word. A synset is a collections of synonyms.

wn.synsets('motorcar')

[Synset('car.n.01')]

Let’s see what are the synonyms in this synset:

WordNet contains much more data, like examples for given synset:

And even definition:

WordNet is aware of hierarchies of words.

**Hyponyms** are more specific concepts of a general concept. For example, ambulance is a hyponym of car.

hyponyms = motorcar.hyponyms()

hyponyms[0]

Synset('ambulance.n.01')

We can display all words that are more specific than motorcar:

A **hypernym** is the opposite to a hyponym. For example, car is a hypernym of ambulance.

1. What percentage of noun synsets have no hyponyms?

You can get all noun synsets using **wn.all\_synsets('n')**.

1. What is the branching factor of noun hypernyms? That is, how many hyponyms on average has each noun synset?

Given the following paragraph:

In [computer science](https://en.wikipedia.org/wiki/Computer_science), **lexical analysis**, **lexing** or **tokenization** is the process of converting a sequence of characters (such as in a computer program or web page) into a sequence of tokens ([strings](https://en.wikipedia.org/wiki/String_(computer_science)) with an assigned and thus identified meaning). A program that performs lexical analysis may be termed a *lexer*, *tokenizer*,[[1]](https://en.wikipedia.org/wiki/Lexical_analysis#cite_note-1) or *scanner*, though *scanner* is also a term for the first stage of a lexer. A lexer is generally combined with a [parser](https://en.wikipedia.org/wiki/Parser), which together analyze the [syntax of programming languages](https://en.wikipedia.org/wiki/Syntax_(programming_languages)), web pages, and so forth.

1. Tokenize (sent\_tokenize) this paragraph to sentences and then the results to words (word\_tokenize) and then stem every word in the result(nltk. PorterStemmer)
   1. Use from nltk.tokenize import sent\_tokenize, word\_tokenize
   2. Use porter = nltk.PorterStemmer(), porter.stem(t)
2. Tag the head paragraph using pos\_tag
   1. from nltk import pos\_tag
3. Use WordNetLemmatizer to lemmatize every word.
4. Use the movie\_reviews nltk corpus and calculate tf-idf on it. Download it first if needed. Use the term frequency divided by document length definition of tf. Don’t forget to lemmatize the words before running them in the algorithm. Use a loop over all documents.
5. Use the nltk Naïve Bayes classification to train and test the prediction of movie review sentiment from the movie\_reviews dataset:

<https://pythonprogramming.net/naive-bayes-classifier-nltk-tutorial/>

<https://www.nltk.org/book/ch06.html>

<http://pythonforengineers.com/build-a-sentiment-analysis-app-with-movie-reviews/>

1. Test your accuracy on your testing data.
2. Write a good movie review and a bad movie review and see if this classifier knows how to classify it well.