Introduction to NLTK

(A Tool Kit for Natural Language Processing)

Mir Tafseer Nayeem

University of Lethbridge

Alberta, Canada

mir.nayeem@uleth.ca

Natural Language Toolkit (NLTK)

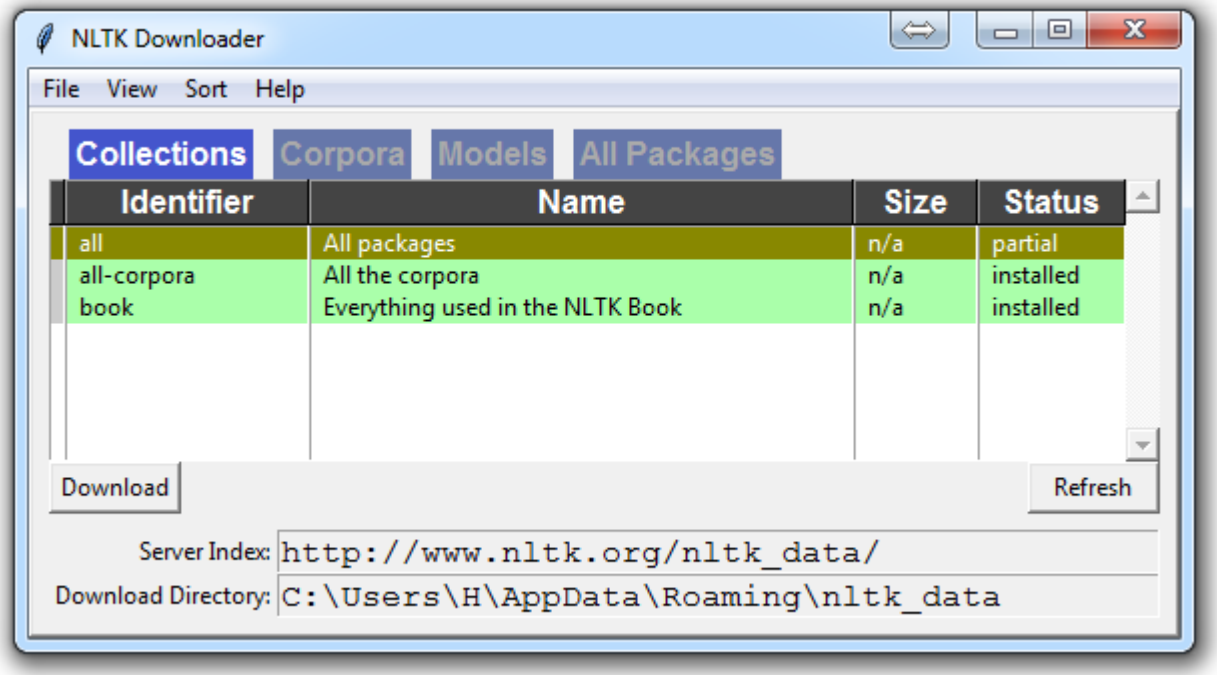
- A collection of Python programs, modules, data set and tutorial to support research and development in Natural Language Processing (NLP).
- NLTK is
  - Free and Open source
  - Easy to use
  - Modular
  - Well documented
  - Simple and extensible
- http://www.nltk.org/

Installing NLTK

- If you do not have Python yet, go to Python.org and download the Python.
- The easiest method to install the NLTK module is with pip.
    - *pip install nltk*

Download NLTK data

- import nltk
- nltk.download()



Some definitions

- **Corpus** - Body of text. Corpora is the plural of Corpus.
  - Example: A collection of news documents.
- **Lexicon** - Words and their meanings.
  - Example: English dictionary.
- **Token -** Each "entity" that is a part of whatever was split up based on rules.
  - For examples, each word is a token when a sentence is "tokenized" into words. Each sentence can also be a token, if you tokenized the sentences out of a paragraph.

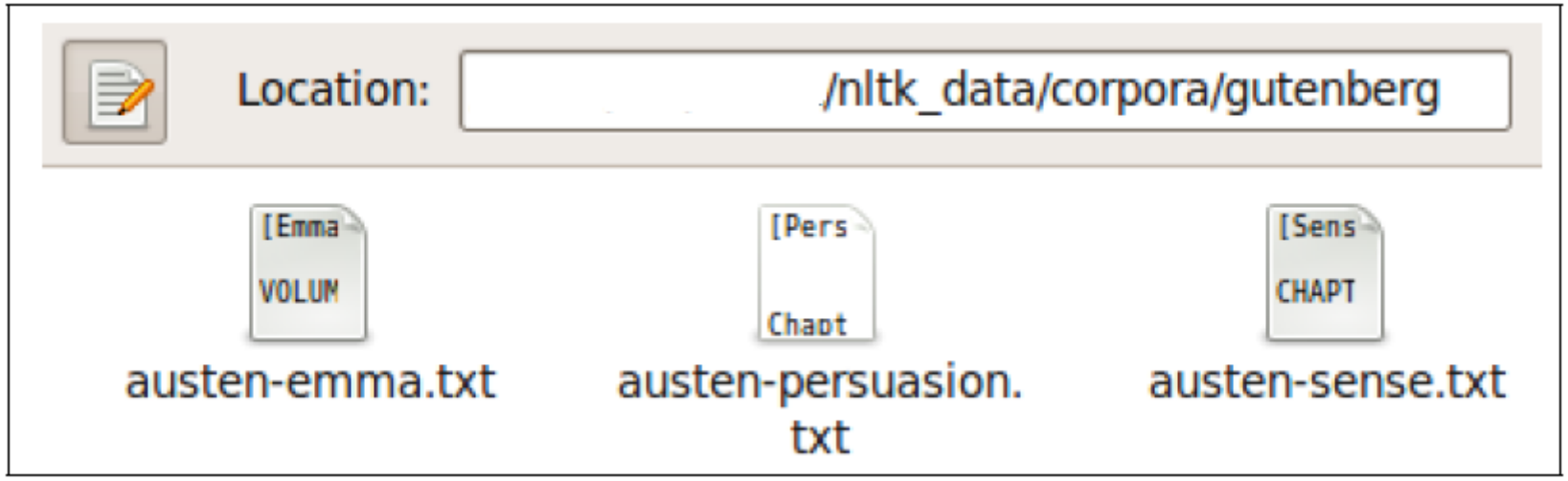
NLTK Modules & Functionality

NLTK Modules	Functionality
nltk.corpus	Corpus
nltk.tokenize, nltk.stem	Tokenizers, stemmers
nltk.collocations	t-test, chi-squared, mutual-info
nltk.tag	n-gram, backoff,Brill, HMM, TnT
nltk.classify, nltk.cluster	Decision tree, Naive bayes, K-means
nltk.chunk	Regex,n-gram, named entity
nltk.parsing	Parsing
nltk.sem, nltk.interence	Semantic interpretation
nltk.metrics	Evaluation metrics
nltk.probability	Probability & Estimation
nltk.app, nltk.chat	Applications

Accessing Corpora

- NLTK provides over 50 corpora and lexical resources

```
>> from nltk.corpus import Gutenberg
>> print(gutenberg.fileids())
>>> ['austen-emma.txt', 'austen-persuasion.txt', 8 'austen-sense.txt', 'bible-kjv.txt',
.....]
```



Accessing Corpora (contd..)

- Accessing Corpora: Raw text

```
>>> emmaText = gutenberg.raw("austen-emma.txt")
>>> emmaText[:200]
```

- Accessing Corpora: Words

```
>>> emmaWords = gutenberg.words("austen-emma.txt")
>>> print(emmaWords[:30])
```

- Accessing Corpora: Sentences

```
>>> senseSents = gutenberg.sents("austen-sense.txt")
>>> print(senseSents[:5])
```

Frequency Distribution

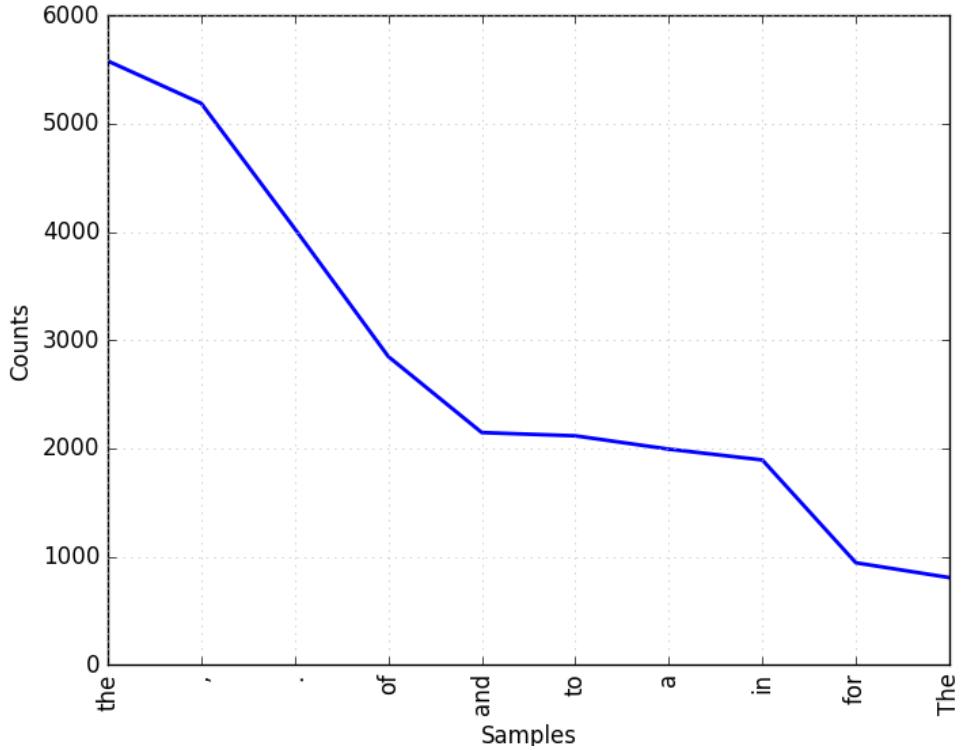
- Records how often each item occurs in a list of words.
- Frequency distribution over words.
- Basically a dictionary with some extra functionality.

```
>>> import nltk
>>> from nltk.corpus import brown
>>> news_words = brown.words(categories = "news")
>>> fdist = nltk.FreqDist(news_words)
>>> print("shoe:", fdist["shoe"])   Output: ('shoe:', 1)
>>> print("the: ", fdist["the"])     Output: ('the: ', 5580)
>>> fdist.tabulate(10)
 the     ,     .    of   and    to     a    in   for  The
5580  5188  4030  2849  2146  2116  1993  1893   943   806
```

Plotting Frequency Distribution

create a plot of the 10 most frequent words

```
>>> fdist.plot(10)
```



Tokenization

- **Tokenization** is the process of breaking a stream of text up into words, phrases, symbols, or other meaningful elements called tokens.

```
>>> from nltk.tokenize import word_tokenize, wordpunct_tokenize, sent_tokenize
>>> s = '''Good muffins cost $3.88\nin New York. Please buy me two of them.\n\nThanks.'''
```

- Word Punctuation Tokenization

```
>>> wordpunct_tokenize(s)
['Good', 'muffins', 'cost', '$', '3', '.', '88', 'in', 'New', 'York', '.', 'Please', 'buy', 'me', 'two', 'of', 'them', '.', 'Thanks', '.']
```

- Sentence Tokenization

```
>>> sent_tokenize(s)
['Good muffins cost $3.88\nin New York.', 'Please buy me\ntwo of them.', 'Thanks.']
```

- Word Tokenization

```
>>> [word_tokenize(t) for t in sent_tokenize(s)]
[['Good', 'muffins', 'cost', '$', '3.88', 'in', 'New', 'York', '.'], ['Please', 'buy', 'me', 'two', 'of','them', '.'], ['Thanks', '.']]
```

Regular Expression Tokenizer

- First you need to decide how you want to tokenize a piece of text then you construct your regular expression. The choices are:
    - Match on the tokens
    - Match on the separators, or gaps

```
>>> from nltk.tokenize import RegexpTokenizer
>>> tokenizer = RegexpTokenizer("[\w']+")
>>> tokenizer.tokenize("Natural Language Processing is very interesting")
    ['Natural', 'Language', 'Processing', 'is', 'very', 'interesting']
```

Filtering stopwords

- Stopwords are common words that generally do not contribute to the meaning of a sentence.
- Most search engines will filter stopwords out of search queries and documents in order to save space in their index.
- Stopwords can be found in the directory
  - **nltk_data/corpora/stopwords/**

```
>>> from nltk.corpus import stopwords
>>> english_stops = set(stopwords.words('english'))
>>> words = ['The', ‘natural', 'language', 'processing', 'is', 'very', 'interesting']
>>> [word for word in words if word.lower() not in english_stops]
    [‘natural', 'language', 'processing', 'interesting']
```

Edit Distance

- The edit distance is the number of character changes necessary to transform the given word into the suggested word.

```
>>> from nltk.metrics import edit_distance
>>> edit_distance("Birthday", "Bday")
4
>>> edit_distance("Addition", "substitution")
7
```

Normalizing Text

- The goal of both stemming and lemmatization is to "normalize" words to their common base form, which is useful for many text-processing applications.
- **Stemming** = heuristically removing the affixes of a word, to get its stem (root).
- **Lemmatization** = Lemmatization process involves first determining the part of speech of a word, and applying different normalization rules for each part of speech.

Consider:

- I was taking a ride in the car.
- I was riding in the car.

- Imagine every word in the English language, every possible tense and affix you can put on a word.
- Having individual dictionary entries per version would be highly redundant and inefficient.

Stemming

- One of the most popular stemming algorithms is the Porter stemmer, which has been around since 1979.
- Several other stemming algorithms provided by NLTK are Lancaster Stemmer and Snowball Stemmer.

```
from nltk.stem import PorterStemmer
ps = PorterStemmer()
example_words = ["python","pythoner","pythoning","pythoned","pythonly"]
for w in example_words:
    print(ps.stem(w))
```

Output:

```
python
python
python
python
pythonli
```

Lemmatization

- Lemmatize takes a part of speech parameter, "pos." If not supplied, the default is "noun".

```
>>> from nltk.stem import WordNetLemmatizer
>>> lemmatizer = WordNetLemmatizer()
>>> lemmatizer.lemmatize('cooking')
'cooking'
>>> lemmatizer.lemmatize('cooking', pos='v')
'cook'
```

Comparison between stemming and lemmatizing

- The major difference between these is, as you saw earlier, stemming can often create non-existent words, whereas lemmas are actual words, you can just look up in an English dictionary.

```
>>> stemmer.stem('believes')
'believ'
>>> lemmatizer.lemmatize('believes')
'belief'
```

Part-of-speech Tagging

- Part-of-speech Tagging is the process of marking up a word in a text (corpus) as corresponding to a particular part of speech.

```
>>> from nltk.tokenize import word_tokenize
>>> from nltk.tag import pos_tag
>>> words = word_tokenize('And now for something completely different')
>>> pos_tag(words)
[('And', 'CC'), ('now', 'RB'), ('for', 'IN'), ('something', 'NN'), ('completely', 'RB'), ('different','JJ')]
```

- https://www.ling.upenn.edu/courses/Fall_2003/ling001/penn_treebank_pos.html

Penn Bank Part-of-Speech Tags

CC	Coordinating conjunction
CD	Cardinal number
DT	Determiner
EX	Existential "there"
FW	Foreign word
IN	Prepostion or subordination conjunction
JJ	Adjective
JJR	Adjective- comparative
JJS	Adjective- superlative
LS	List item marker
MD	Modal
NN	Noun- singular or mass
NNS	Noun- plural
NP	Proper noun- singular
NPS	Proper noun- plural

Named-entity Recognition

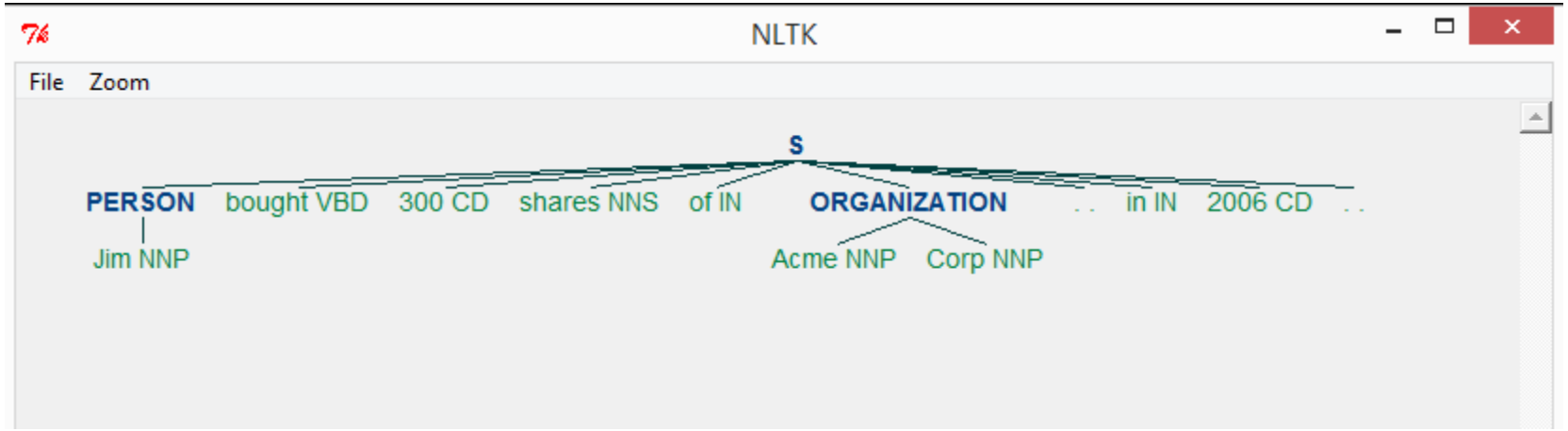
- Named-entity recognition is a subtask of information extraction that seeks to locate and classify elements in text into pre-defined categories such as the names of persons, organizations, locations, expressions of times, quantities, monetary values, percentages, etc.

```
>>> from nltk import pos_tag, ne_chunk
>>> from nltk.tokenize import wordpunct_tokenize
>>> sent = 'Jim bought 300 shares of Acme Corp. in 2006.'
>>> ne_chunk(pos_tag(wordpunct_tokenize(sent)))
Tree('S', [Tree('PERSON', [('Jim', 'NNP')]), ('bought', 'VBD'), ('300', 'CD'), ('shares', 'NNS'),('of', 'IN'),
Tree('ORGANIZATION', [('Acme', 'NNP'), ('Corp', 'NNP')]), ('.', '.'), ('in', 'IN'),('2006', 'CD'), ('.', '.')])
```

NE Type and Examples

ORGANIZATION - Georgia-Pacific Corp., WHO
PERSON - Eddy Bonte, President Obama
LOCATION - Murray River, Mount Everest
DATE - June, 2008-06-29
TIME - two fifty a m, 1:30 p.m.
MONEY - 175 million Canadian Dollars, GBP 10.40
PERCENT - twenty pct, 18.75 %
FACILITY - Washington Monument, Stonehenge
GPE - South East Asia, Midlothian

```
>>> ne_chunk(pos_tag(wordpunct_tokenize(sent))).draw()
```



WordNet

- WordNet is a lexical database for the English language. In other words, it's a dictionary designed specifically for natural language processing, which was created by Princeton, and is part of the NLTK corpus.
- You can use WordNet alongside the NLTK module to find the meanings of words, synonyms, antonyms, similarity and more. Let's cover some examples.

Some Examples

```
>>> from nltk.corpus import wordnet
>>> syns = wordnet.synsets("program")
>>> print syns
[Synset('plan.n.01'), Synset('program.n.02'), Synset('broadcast.n.02'),
Synset('platform.n.02'), Synset('program.n.05'), Synset('course_of_study.n.01'),
Synset('program.n.07'), Synset('program.n.08'), Synset('program.v.01'),
Synset('program.v.02')]
>>> print(syns[0].name())
plan.n.01
>>> print syns[0].pos()
N
>>> print(syns[0].definition())
a series of steps to be carried out or goals to be accomplished
>>> print(syns[0].examples())
[u'they drew up a six-step plan', u'they discussed plans for a new bond issue']
```

WordNet synset similarity

- Synsets are organized in a hypernym tree. Two synsets are more similar, the closer they are in the tree. According to Wu and Palmer method for semantic related-ness.

```
>>> w1 = wordnet.synset('ship.n.01')
>>> w2 = wordnet.synset('boat.n.01')
>>> print(w1.wup_similarity(w2))
0.909090909090909
>>> w1 = wordnet.synset('ship.n.01')
>>> w2 = wordnet.synset('car.n.01')
>>> print(w1.wup_similarity(w2))
0.6956521739130435
>>> w1 = wordnet.synset('ship.n.01')
>>> w2 = wordnet.synset('cat.n.01')
>>> print(w1.wup_similarity(w2))
0.38095238095238093
```

Bag of Words model

- All texts need to be converted to numbers before starts processing by the machine.
- Consider these two short texts.
  1. Julie loves me more than Linda loves me
  2. Jane likes me more than Julie loves me
- We want to know how similar these texts are, purely in terms of word counts (and ignoring word order). We begin by making a list of the words from both texts:

[me Jane Julie Linda likes loves more than]

- Now we count the number of times each of these words appears in each text.

	S1	S2
me	2	2
Jane	0	1
Julie	1	1
Linda	1	0
likes	0	1
loves	2	1
more	1	1
than	1	1

- We are interested only in those two vertical vectors of counts.
  - a: [2, 0, 1, 1, 0, 2, 1, 1]
  - b: [2, 1, 1, 0, 1, 1, 1, 1]
- The cosine of the angle between them defines the similarity between the texts. [Cosine Similarity]
- The cosine of the angle between them is about 0.822.

Classifying with NLTK

- Supervised Classification Algos in NLTK
    - Naive Bayes
    - Maximum Entropy / Logistic Regression
    - Decision Tree
    - SVM (coming soon) [Can also be used through scikit-learn library]
- **Problem:** Gender identification from name.
- **Relevant feature:** Last Letter

Building Features

- Create a feature set (a dictionary) that maps from features‘ names to their values.

```
def gender_features(word):
    return {'last_letter': word[-1]}
```

- Import names, shuffle them

```
from nltk.corpus import names
import random
names = ([(name, 'male') for name in names.words('male.txt')] + [(name, 'female') for name in names.words('female.txt')])

random.shuffle(names)
```

Train and Test

- Divide list of features into training set and test set

```
featuresets = [(gender_features(n), g) for (n,g) in names]
train_set, test_set = featuresets[500:], featuresets[:500]
```

- Use training set to train a naive Bayes classifier

```
classifier = nltk.NaiveBayesClassifier.train(train_set)
```

- Test the classier on unseen data

```
classifier.classify(gender_features('Neo'))
>>> 'male'
 classifier.classify(gender_features('Trinity'))
>>> 'female'
print nltk.classify.accuracy(classifier, test_set)
>>> 0.744
```

Most Informative Features

- Examine the classier to see which feature is most effective at distinguishing between classes.

```
>>> classifier.show_most_informative_features(5)
Most Informative Features
last_letter = 'a' female : male = 35.7 : 1.0
last_letter = 'k' male : female = 31.7 : 1.0
last_letter = 'f' male : female = 16.6 : 1.0
last_letter = 'p' male : female = 11.9 : 1.0
last_letter = 'v' male : female = 10.5 : 1.0
```

Feeling lonely?

- Eliza is there to talk to you all day!

```
>>> from nltk.chat import eliza
>>> eliza.eliza_chat()
```

……starts the chatbot

```
Therapist
---------
Talk to the program by typing in plain English, using normal upper-
and lower-case letters and punctuation.  Enter "quit" when done.
==========================================================================
Hello.  How are you feeling today?
>
```

Thanks! ☺

Any Questions?