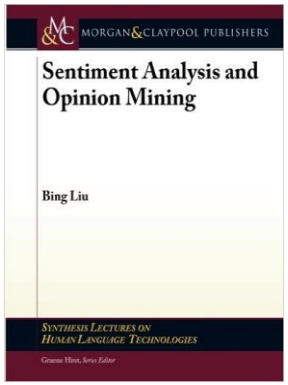
Principles/Social Media Mining

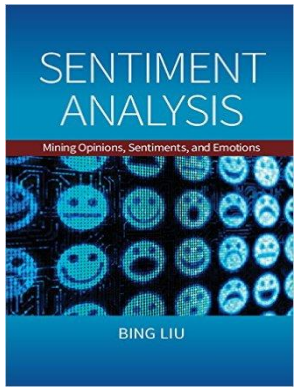
CIS 600

Weeks 10 & 11: Sentiment Analysis, Part 2: NLP & NLTK

Edmund Yu, PhD
Associate Teaching Professor
esyu@syr.edu

October 29, November 3, 2020



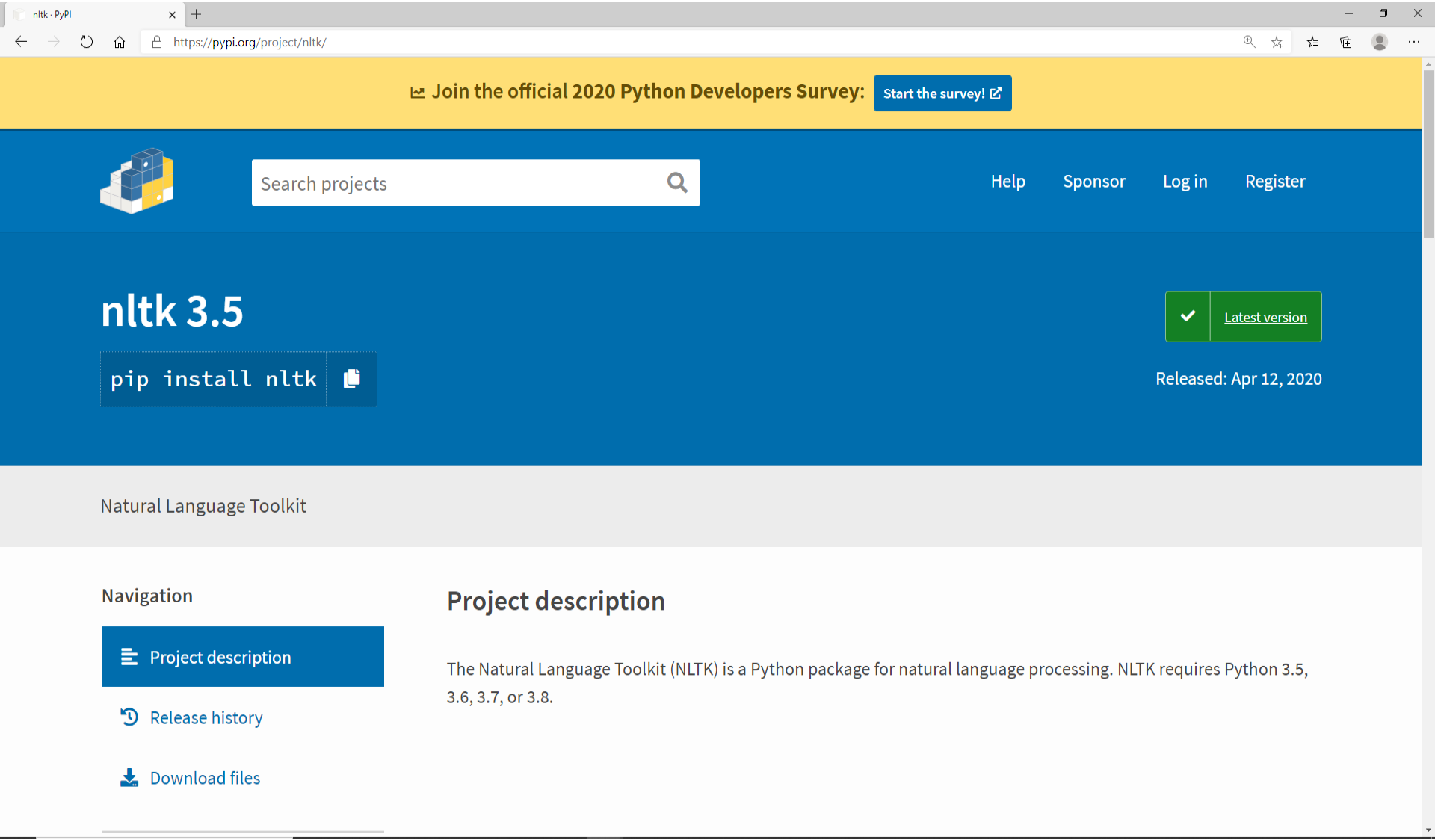


Reading Assignments

- Textbook #3: Sentiment Analysis: Mining Opinions, Sentiments, and Emotions, by Bing Liu, Cambridge University Press, 2015, ISBN: 1107017890. **Chapters 1 & 2**. (Get them at https://www.cs.uic.edu/~liub/FBS/sentiment-opinion-emotion-analysis.html)
- Older version of Textbook #3: Sentiment Analysis & Opinion Mining, by Bing Liu. **Chapters 1 - 6**. (Can be downloaded from www.morganclaypool.com for free, if on campus.)

Install NLTK

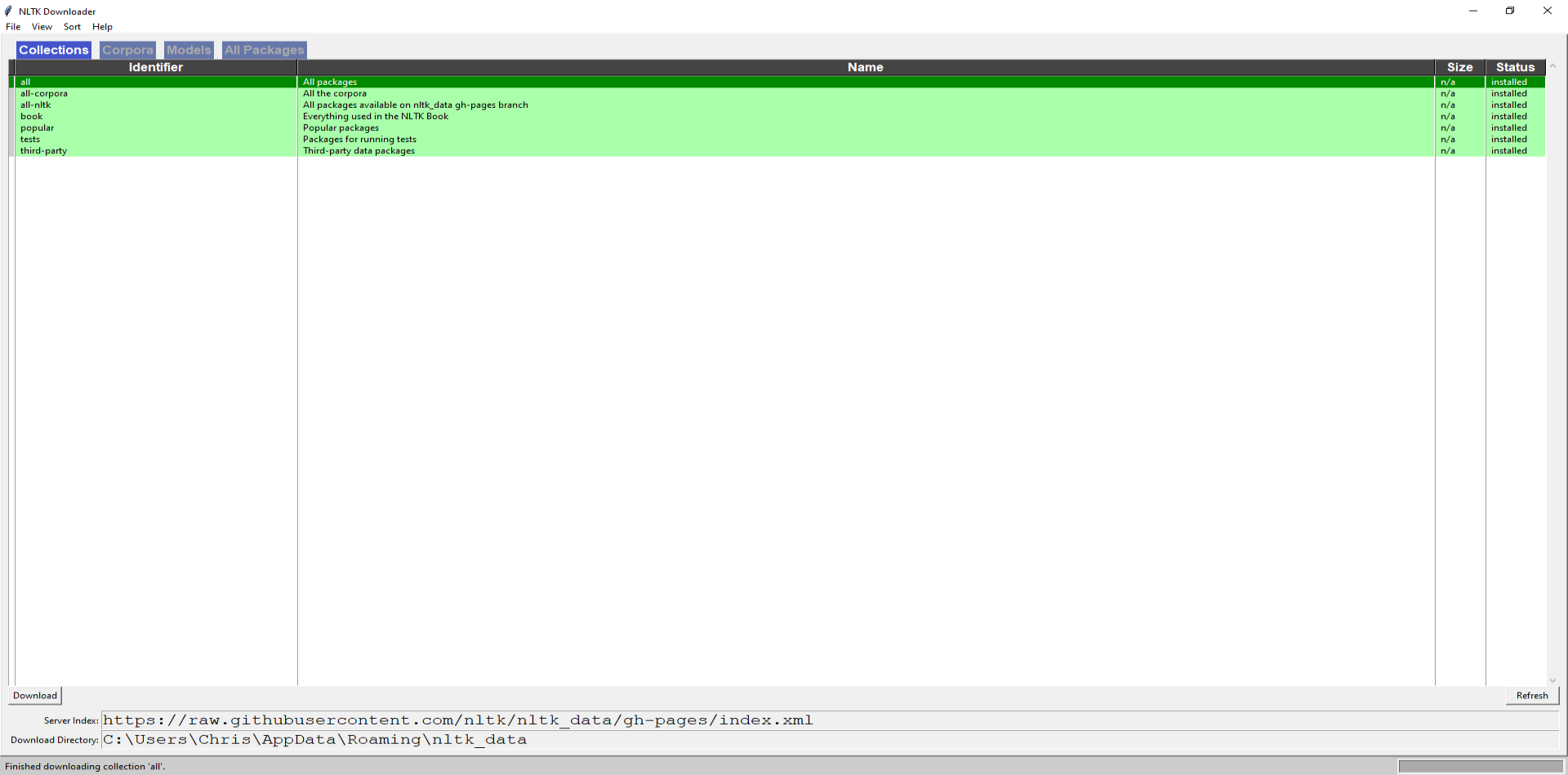
- pip install nltk

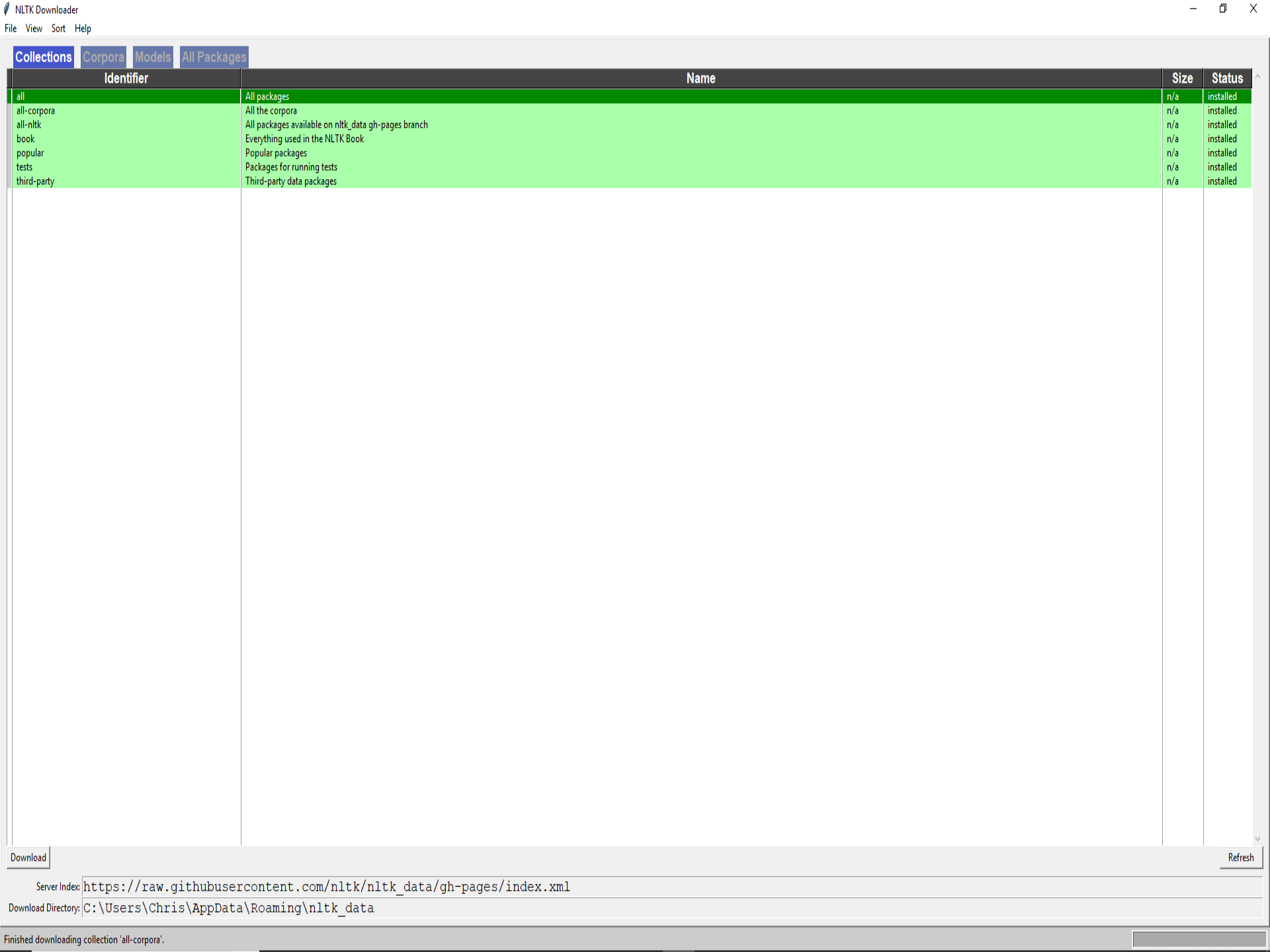


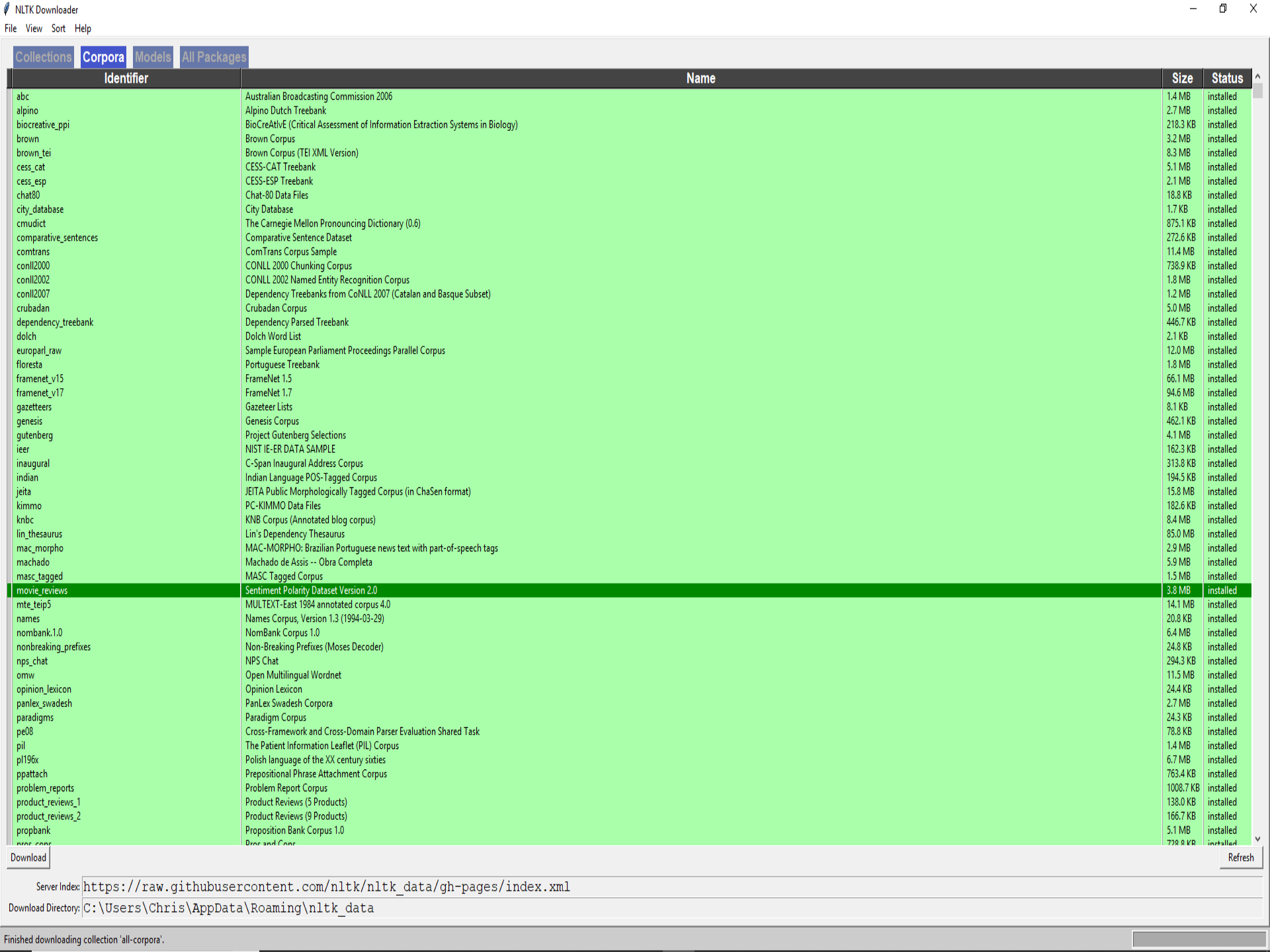
NLTK

- Once you've installed NLTK, install the data required for the book by typing the following two commands:

```
>>> import nltk
>>> nltk.download()
```




NLTK Downloader
File View Sort Help
Collections Corpora Models All Packages
Identifier Name Size Status
all All packages n/a installed
all-corpora All the corpora n/a installed
all-nltk All packages available on nltk_data gh-pages branch n/a installed
book Everything used in the NLTK Book n/a installed
popular Popular packages n/a installed
tests Packages for running tests n/a installed
third-party Third-party data packages n/a installed
Download
Refresh
Server Index: https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/index.xml
Download Directory: C:\Users\Chris\AppData\Roaming\nltk_data
Finished downloading collection 'all-corpora'.


NLTK Downloader
File View Sort Help
Collections Corpora Models All Packages
Identifier Name Size Status
abc Australian Broadcasting Commission 2006 1.4 MB installed
alpino Alpino Dutch Treebank 2.7 MB installed
biocreative_ppi BioCreAtIvE (Critical Assessment of Information Extraction Systems in Biology) 218.3 KB installed
brown Brown Corpus 3.2 MB installed
brown_tei Brown Corpus (TEI XML Version) 8.3 MB installed
cess_cat CESS-CAT Treebank 5.1 MB installed
cess_esp CESS-ESP Treebank 2.1 MB installed
chat80 Chat-80 Data Files 18.8 KB installed
city_database City Database 1.7 KB installed
cmudict The Carnegie Mellon Pronouncing Dictionary (0.6) 875.1 KB installed
comparative_sentences Comparative Sentence Dataset 272.6 KB installed
comtrans ComTrans Corpus Sample 11.4 MB installed
conll2000 CONLL 2000 Chunking Corpus 738.9 KB installed
conll2002 CONLL 2002 Named Entity Recognition Corpus 1.8 MB installed
conll2007 Dependency Treebanks from CoNLL 2007 (Catalan and Basque Subset) 1.2 MB installed
crubadan Crubadan Corpus 5.0 MB installed
dependency_treebank Dependency Parsed Treebank 446.7 KB installed
dolch Dolch Word List 2.1 KB installed
europarl_raw Sample European Parliament Proceedings Parallel Corpus 12.0 MB installed
floresta Portuguese Treebank 1.8 MB installed
framenet_v15 FrameNet 1.5 66.1 MB installed
framenet_v17 FrameNet 1.7 94.6 MB installed
gazetteers Gazeteer Lists 8.1 KB installed
genesis Genesis Corpus 462.1 KB installed
gutenberg Project Gutenberg Selections 4.1 MB installed
ieer NIST IE-ER DATA SAMPLE 162.3 KB installed
inaugural C-Span Inaugural Address Corpus 313.8 KB installed
indian Indian Language POS-Tagged Corpus 194.5 KB installed
jeita JEITA Public Morphologically Tagged Corpus (in ChaSen format) 15.8 MB installed
kimmo PC-KIMMO Data Files 182.6 KB installed
knbc KNB Corpus (Annotated blog corpus) 8.4 MB installed
lin_thesaurus Lin's Dependency Thesaurus 85.0 MB installed
mac_morpho MAC-MORPHO: Brazilian Portuguese news text with part-of-speech tags 2.9 MB installed
machado Machado de Assis -- Obra Completa 5.9 MB installed
masc_tagged MASC Tagged Corpus 1.5 MB installed
movie_reviews Sentiment Polarity Dataset Version 2.0 3.8 MB installed
mte_teip5 MULTEXT-East 1984 annotated corpus 4.0 14.1 MB installed
names Names Corpus, Version 1.3 (1994-03-29) 20.8 KB installed
nombank.1.0 NomBank Corpus 1.0 6.4 MB installed
nonbreaking_prefixes Non-Breaking Prefixes (Moses Decoder) 24.8 KB installed
nps_chat NPS Chat 294.3 KB installed
omw Open Multilingual Wordnet 11.5 MB installed
opinion_lexicon Opinion Lexicon 24.4 KB installed
panlex_swadesh PanLex Swadesh Corpora 2.7 MB installed
paradigms Paradigm Corpus 24.3 KB installed
pe08 Cross-Framework and Cross-Domain Parser Evaluation Shared Task 78.8 KB installed
pil The Patient Information Leaflet (PIL) Corpus 1.4 MB installed
pl196x Polish language of the XX century sixties 6.7 MB installed
ppattach Prepositional Phrase Attachment Corpus 763.4 KB installed
problem_reports Problem Report Corpus 1008.7 KB installed
product_reviews_1 Product Reviews (5 Products) 138.0 KB installed
product_reviews_2 Product Reviews (9 Products) 166.7 KB installed
propbank Proposition Bank Corpus 1.0 5.1 MB installed
Download
Refresh
Server Index: https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/index.xml
Download Directory: C:\Users\Chris\AppData\Roaming\nltk_data
Finished downloading collection 'all-corpora'.

Natural Language Toolkit

NLTK is a leading platform for building Python programs to work with human language data. 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, and an active discussion forum.

Thanks to a hands-on guide introducing programming fundamentals alongside topics in computational linguistics, plus comprehensive API documentation, NLTK is suitable for linguists, engineers, students, educators, researchers, and industry users alike. NLTK is available for Windows, Mac OS X, and Linux. Best of all, NLTK is a free, open source, community-driven project.

NLTK has been called "a wonderful tool for teaching, and working in, computational linguistics using Python," and "an amazing library to play with natural language."

Natural Language Processing with Python provides a practical introduction to programming for language processing. Written by the creators of NLTK, it guides the reader through the fundamentals of writing Python programs, working with corpora, categorizing text, analyzing linguistic structure, and more. The online version of the book has been been updated for Python 3 and NLTK 3. (The original Python 2 version is still available at http://nltk.org/book_1ed.)

Some simple things you can do with NLTK

SEARCH

Go

Natural Language Processing with Python

– Analyzing Text with the Natural Language Toolkit

Steven Bird, Ewan Klein, and Edward Loper

This version of the NLTK book is updated for Python 3 and NLTK 3. The first edition of the book, published by O'Reilly, is available at http://nltk.org/book_1ed/. (There are currently no plans for a second edition of the book.)

Reading Assignment:
Chapters 1 - 6

Corpus	Compiler	Contents
Brown Corpus	Francis, Kucera	15 genres, 1.15M words, tagged, categorized
CESS Treebanks	CLiC-UB	1M words, tagged and parsed (Catalan, Spanish)
Chat-80 Data Files	Pereira & Warren	World Geographic Database
CMU Pronouncing Dictionary	CMU	127k entries
CoNLL 2000 Chunking Data	CoNLL	270k words, tagged and chunked
CoNLL 2002 Named Entity	CoNLL	700k words, pos- and named-entity-tagged (Dutch, Spanish)
CoNLL 2007 Dependency Treebanks (sel)	CoNLL	150k words, dependency parsed (Basque, Catalan)
Dependency Treebank	Narad	Dependency parsed version of Penn Treebank sample
FrameNet	Fillmore, Baker et al	10k word senses, 170k manually annotated sentences
Floresta Treebank	Diana Santos et al	9k sentences, tagged and parsed (Portuguese)
Gazetteer Lists	Various	Lists of cities and countries
Genesis Corpus	Misc web sources	6 texts, 200k words, 6 languages
Gutenberg (selections)	Hart, Newby, et al	18 texts, 2M words
Inaugural Address Corpus	CSpan	US Presidential Inaugural Addresses (1789-present)
Indian POS-Tagged Corpus	Kumaran et al	60k words, tagged (Bangla, Hindi, Marathi, Telugu)
MacMorpho Corpus	NILC, USP, Brazil	1M words, tagged (Brazilian Portuguese)
Movie Reviews	Pang, Lee	2k movie reviews with sentiment polarity classification
Names Corpus	Kantrowitz, Ross	8k male and female names
NIST 1999 Info Extr (selections)	Garofolo	63k words, newswire and named-entity SGML markup
Nombank	Meyers	115k propositions, 1400 noun frames
NPS Chat Corpus	Forsyth, Martell	10k IM chat posts, POS-tagged and dialogue-act tagged
Open Multilingual WordNet	Bond et al	15 languages, aligned to English WordNet
PP Attachment Corpus	Ratnaparkhi	28k prepositional phrases, tagged as noun or verb modifiers
Proposition Bank	Palmer	113k propositions, 3300 verb frames
Question Classification	Li, Roth	6k questions, categorized
Reuters Corpus	Reuters	1.3M words, 10k news documents, categorized
Roget's Thesaurus	Project Gutenberg	200k words, formatted text
RTE Textual Entailment	Dagan et al	8k sentence pairs, categorized
SEMCOR	Rus, Mihalcea	880k words, part-of-speech and sense tagged
Senseval 2 Corpus	Pedersen	600k words, part-of-speech and sense tagged
SentiWordNet	Esuli, Sebastiani	sentiment scores for 145k WordNet synonym sets
Shakespeare texts (selections)	Bosak	8 books in XML format
State of the Union Corpus	CSPAN	485k words, formatted text
Stopwords Corpus	Porter et al	2,400 stopwords for 11 languages
Swadesh Corpus	Wiktionary	comparative wordlists in 24 languages
Switchboard Corpus (selections)	LDC	36 phonecalls, transcribed, parsed
Univ Decl of Human Rights	United Nations	480k words, 300+ languages
Penn Treebank (selections)	LDC	40k words, tagged and parsed
TIMIT Corpus (selections)	NIST/LDC	audio files and transcripts for 16 speakers
VerbNet 2.1	Palmer et al	5k verbs, hierarchically organized, linked to WordNet
Wordlist Corpus	OpenOffice.org et al	960k words and 20k affixes for 8 languages
WordNet 3.0 (English)	Miller, Fellbaum	145k synonym sets

NLTK: Reuters

- This Reuters Corpus contains **10,788** news documents totaling **1.3 million** words.
- The documents have been classified into **90** topics, and grouped into two sets, called **training** (7769) and **test** (3019):
  - fileid 'test/14826‘ is a document from the test set.
- This split is for training and testing algorithms that automatically detect the topic of a document.

```
>>> from nltk.corpus import reuters
>>> reuters.fileids()
['test/14826', 'test/14828', 'test/14829', 'test/14832', ...]
>>> reuters.categories()
['acq', 'alum', 'barley', 'bop', 'carcass', 'castor-oil', 'cocoa',
'coconut', 'coconut-oil', 'coffee', 'copper', 'copra-cake', 'corn',
'cotton', 'cotton-oil', 'cpi', 'cpu', 'crude', 'dfl', 'dlr', ...]
```

NLTK: Reuters

- Categories in the Reuters Corpus overlap with each other, simply because a news story often covers multiple topics. We can ask for the topics covered by one or more documents, or for the documents included in one or more categories. For convenience, the corpus methods accept a single fileid or a list of fileids.

```
>>> reuters.categories('training/9865')
['barley', 'corn', 'grain', 'wheat']
>>> reuters.categories(['training/9865', 'training/9880'])
['barley', 'corn', 'grain', 'money-fx', 'wheat']
>>> reuters.fileids('barley')
['test/15618', 'test/15649', 'test/15676', 'test/15728', 'test/15871', ...]
>>> reuters.fileids(['barley', 'corn'])
['test/14832', 'test/14858', 'test/15033', 'test/15043', 'test/15106',…]
```

NLTK: Reuters

- Similarly, we can specify the words or sentences we want in terms of files or categories. The first handful of words in each of these texts are the titles, which by convention are stored as uppercase.

```
>>> reuters.words('training/9865')[:14]
['FRENCH', 'FREE', 'MARKET', 'CEREAL', 'EXPORT', 'BIDS',
'DETAILED', 'French', 'operators', 'have', 'requested', 'licences', 'to', 'export']
>>> reuters.words(['training/9865', 'training/9880'])
['FRENCH', 'FREE', 'MARKET', 'CEREAL', 'EXPORT', ...]
>>> reuters.words(categories='barley')
['FRENCH', 'FREE', 'MARKET', 'CEREAL', 'EXPORT', ...]
>>> reuters.words(categories=['barley', 'corn'])
['THAI', 'TRADE', 'DEFICIT', 'WIDENS', 'IN', 'FIRST', ...]
```

Table 2-3. Basic corpus functionality defined in NLTK: More documentation can be found using help(nltk.corpus.reader) and by reading the online Corpus HOWTO at http://www.nltk.org/howto.

Example	Description
`fileids()`	The files of the corpus
`fileids([categories])`	The files of the corpus corresponding to these categories
`categories()`	The categories of the corpus
`categories([fileids])`	The categories of the corpus corresponding to these files
`raw()`	The raw content of the corpus
`raw(fileids=[f1,f2,f3])`	The raw content of the specified files
`raw(categories=[c1,c2])`	The raw content of the specified categories
`words()`	The words of the whole corpus
`words(fileids=[f1,f2,f3])`	The words of the specified fileids
`words(categories=[c1,c2])`	The words of the specified categories
`sents()`	The sentences of the specified categories
`sents(fileids=[f1,f2,f3])`	The sentences of the specified fileids
`sents(categories=[c1,c2])`	The sentences of the specified categories
`abspath(fileid)`	The location of the given file on disk
`encoding(fileid)`	The encoding of the file (if known)
`open(fileid)`	Open a stream for reading the given corpus file
`root()`	The path to the root of locally installed corpus
`readme()`	The contents of the README file of the corpus

Accessing The Reuters Dataset in NLTK

```
from nltk import *  # easy_install nltk (or, pip install nltk)

# Reuters
from nltk.corpus import reuters

print "#Files: ", len(reuters.fileids())
print "File List: ", reuters.fileids()[:10]
print "#Categories: ", len(reuters.categories())
print "Category List: ", reuters.categories()[:10]
print "Raw text in this file: ", reuters.raw('test/14826')
print "Words in this file: ", reuters.words('test/14826')[:100]
print "Categories: ", reuters.categories('test/14826')
print "All files in this category: ", reuters.fileids('barley')
```

Corpus	Compiler	Contents
CoNLL 2002 Named Entity	CoNLL	700k words, POS and named entity tagged (Dutch, Spanish)
CoNLL 2007 Dependency Parsed Tree-banks (selections)	CoNLL	150k words, dependency parsed (Basque, Catalan)
Dependency Treebank	Narad	Dependency parsed version of Penn Treebank sample
Floresta Treebank	Diana Santos et al.	9k sentences, tagged and parsed (Portuguese)
Gazetteer Lists	Various	Lists of cities and countries
Genesis Corpus	Misc web sources	6 texts, 200k words, 6 languages
Gutenberg (selections)	Hart, Newby, et al.	18 texts, 2M words
Inaugural Address Corpus	CSpan	U.S. Presidential Inaugural Addresses (1789–present)
Indian POS Tagged Corpus	Kumaran et al.	60k words, tagged (Bangla, Hindi, Marathi, Telugu)
MacMorpho Corpus	NILC, USP, Brazil	1M words, tagged (Brazilian Portuguese)
Movie Reviews	Pang, Lee	2k movie reviews with sentiment polarity classification
Names Corpus	Kantrowitz, Ross	8k male and female names
NIST 1999 Info Extr (selections)	Garofolo	63k words, newswire and named entity SGML markup
NPS Chat Corpus	Forsyth, Martell	10k IM chat posts, POS and dialogue-act tagged
Penn Treebank (selections)	LDC	40k words, tagged and parsed
PP Attachment Corpus	Ratnaparkhi	28k prepositional phrases, tagged as noun or verb modifiers
Proposition Bank	Palmer	113k propositions, 3,300 verb frames
Question Classification	Li, Roth	6k questions, categorized
Reuters Corpus	Reuters	1.3M words, 10k news documents, categorized
Roget's Thesaurus	Project Gutenberg	200k words, formatted text

NLTK: Movie Reviews

```
>>> from nltk.corpus import movie_reviews
>>> movie_reviews.fileids()
['neg/cv000_29416.txt', 'neg/cv001_19502.txt', 'neg/cv002_17424.txt',
'neg/cv003_12683.txt', 'neg/cv004_12641.txt', 'neg/cv005_29357.txt',
'neg/cv006_17022.txt', 'neg/cv007_4992.txt', 'neg/cv008_29326.txt',
'neg/cv009_29417.txt', 'neg/cv010_29063.txt', 'neg/cv011_13044.txt',
'neg/cv012_29411.txt', 'neg/cv013_10494.txt', 'neg/cv014_15600.txt',
'neg/cv015_29356.txt', 'neg/cv016_4348.txt', 'neg/cv017_23487.txt',
'neg/cv018_21672.txt', 'neg/cv019_16117.txt', 'neg/cv020_9234.txt',
'neg/cv021_17313.txt', 'neg/cv022_14227.txt', 'neg/cv023_13847.txt',
'neg/cv024_7033.txt', 'neg/cv025_29825.txt', 'neg/cv026_29229.txt',
'neg/cv027_26270.txt', 'neg/cv028_26964.txt', 'neg/cv029_19943.txt',
'neg/cv030_22893.txt', 'neg/cv031_19540.txt', 'neg/cv032_23718.txt',
'neg/cv033_25680.txt', 'neg/cv034_29446.txt', 'neg/cv035_3343.txt',
'neg/cv036_18385.txt', 'neg/cv037_19798.txt', 'neg/cv038_9781.txt',
'neg/cv039_5963.txt', 'neg/cv040_8829.txt', 'neg/cv041_22364.txt', …
```

NLTK: Movie Reviews

```
>>> movie_reviews.categories()
['neg', 'pos']

>>> movie_reviews.categories('neg/cv000_29416.txt')
['neg']

>>> movie_reviews.fileids(‘neg’)
['neg/cv000_29416.txt', 'neg/cv001_19502.txt', 'neg/cv002_17424.txt', …]

>>> movie_reviews.fileids(‘pos’)
['pos/cv000_29590.txt', 'pos/cv001_18431.txt', 'pos/cv002_15918.txt', …]
```

NLTK: Movie Reviews

```
>>> movie_reviews.raw('neg/cv000_29416.txt')
'plot : two teen couples go to a church party , drink and then drive . \nthey
    get into an accident . \none of the guys dies , but his girlfriend continues
    to see him in her life , and has nightmares . \nwhat\'s the deal ? \nwatch
    the movie and " sorta " find out . . . \ncritique : a mind-fuck movie for
    the teen generation that touches on a very cool idea , but presents it in a
    very bad package . \nwhich is what makes this review an even harder one
    to write , since i generally applaud films which attempt to break the mold
    , mess with your head and such ( lost highway & memento ) , but there
    are good and bad ways of making all types of films , and these folks just
    didn\'t snag this one correctly . \nthey seem to have taken this pretty neat
    concept , but executed it terribly . \nso what are the problems with the
    movie ? \nwell , its main problem is that it\'s simply too jumbled . \n…

>>> movie_reviews.words('neg/cv000_29416.txt')
['plot', ':', 'two', 'teen', 'couples', 'go', 'to', ...]
```

Movie Review Data

This page is a distribution site for movie-review data for use in sentiment-analysis experiments. Available are collections of movie-review documents labeled with respect to their overall *sentiment polarity* (positive or negative) or *subjective rating* (e.g., "two and a half stars") and sentences labeled with respect to their *subjectivity status* (subjective or objective) or *polarity*. These data sets were introduced in the following papers:

- Bo Pang, Lillian Lee, and Shivakumar Vaithyanathan, Thumbs up? Sentiment Classification using Machine Learning Techniques, *Proceedings of EMNLP 2002.*
- Bo Pang and Lillian Lee, A Sentimental Education: Sentiment Analysis Using Subjectivity Summarization Based on Minimum Cuts, *Proceedings of ACL 2004.*
- Bo Pang and Lillian Lee, Seeing stars: Exploiting class relationships for sentiment categorization with respect to rating scales, *Proceedings of ACL 2005.*

Until April 2012 (but no longer), we maintained a list for of other papers using our data the purposes of facilitating comparison of results.

Please cite the version number of the dataset you used in any publications, in order to facilitate comparison of results. Thank you.

Sentiment polarity datasets

- polarity dataset v2.0 (3.0Mb) (includes README v2.0): 1000 positive and 1000 negative processed reviews. Introduced in Pang/Lee ACL 2004. Released June 2004.
- Pool of 27886 unprocessed html files (81.1Mb) from which the polarity dataset v2.0 was derived. (This file is identical to movie.zip from data release v1.0.)
- sentence polarity dataset v1.0 (includes sentence polarity dataset README v1.0: 5331 positive and 5331 negative processed sentences / snippets. Introduced in Pang/Lee ACL 2005. Released July 2005.

- archive:
  - polarity dataset v1.0 (2.8Mb) (includes README): 700 positive and 700 negative processed reviews. Released July 2002.
  - polarity dataset v1.1 (2.2Mb) (includes README.1.1): approximately 700 positive and 700 negative processed reviews. Released November 2002. This alternative version was created by Nathan Treloar, who removed a few non-English/incomplete reviews and changing some of the labels (judging some

The NLTK Tweet Corpus

NLTK's Twitter corpus currently contains a sample of 20k Tweets (named ' twitter_samples ') retrieved from the Twitter Streaming API, together with another 10k which are divided according to sentiment into negative and positive.

The NLTK Tweet Corpus

```
# 3 Files: negative_tweets.json (5K), positive_tweets.json (5K),
# & tweets.20150430-223406.json (20K)

from nltk.corpus import twitter_samples

print twitter_samples.fileids()

strings = twitter_samples.strings('positive_tweets.json')

for string in strings[:10]:
    print string
```

Twitter HOWTO

Overview

This document is an overview of how to use NLTK to collect and process Twitter data. It was written as an IPython notebook, and if you have IPython installed, you can download the source of the notebook from the NLTK GitHub repository and run the notebook in interactive mode.

Most of the tasks that you might want to carry out with 'live' Twitter data require you to authenticate your request by registering for API keys. This is usually a once-only step. When you have registered your API keys, you can store them in a file on your computer, and then use them whenever you want. We explain what's involved in the section First Steps.

If you have already obtained Twitter API keys as part of some earlier project, storing your keys explains how to save them to a file that NLTK will be able to find. Alternatively, if you just want to play around with the Twitter data that is distributed as part of NLTK, head over to the section on using the `twitter-samples` corpus reader.

Once you have got authentication sorted out, we'll show you how to use NLTK's `Twitter` class. This is made as simple as possible, but deliberately limits what you can do.

First Steps

As mentioned above, in order to collect data from Twitter, you first need to register a new *application* — this is Twitter's way of referring to any computer program that interacts with the Twitter API. As long as you save your registration information correctly, you should only need to do this once, since the information should work for any NLTK code that you write. You will need to have a Twitter account before you can register. Twitter also insists that you add a mobile phone number to your Twitter profile before you will be allowed to register an application.

These are the steps you need to carry out.

Getting your API keys from Twitter

NLTK SentiWordNet (Revisited)

```
# SentiWordNet
from nltk.corpus import sentiwordnet as swn

breakdown = swn.senti_synset('breakdown.n.03')
print(breakdown)
print(breakdown.pos_score())
print(breakdown.neg_score())
print(breakdown.obj_score())

for i in swn.senti_synsets('slow’): print(i)
for i in swn.senti_synsets('happy', 'a’): print(i)
for i in swn.all_senti_synsets(): print(i)
```

Texts to Numbers

From Texts to Numbers

- Once you have a collection of text data appropriate for your application (such as tweets), you need to process it into a form that data-mining procedures can use.

- **Tokenization** usually is the first step in this process.

Tokenization

- Breaking a stream of characters into tokens is trivial for a person familiar with the language structure, but not for a computer program.
  - The reason is that certain characters are sometimes token delimiters and sometimes not, depending on the application.
  - The characters **space**, **tab**, and **newline** (white space) we assume are always delimiters and are not counted as tokens.
  - The characters ***()*** **<>!?”** are always delimiters and may also be tokens. (See next slide)
  - The characters **. , : -** ’ may or may not be delimiters, depending on their environment:

    “We bought apples, oranges, etc.”

Uniform Resource Identifiers (URI): Generic Syntax

2.2. **Reserved Characters**

Many URI include components consisting of or delimited by, certain special characters. These characters are called "reserved", since their usage within the URI component is limited to their reserved purpose. If the data for a URI component would conflict with the reserved purpose, then the conflicting data must be escaped before forming the URI.

reserved = ";" | "/" | "?" | ":" | "@" | "&" | "=" | "+" | "$" | ","

Tokenization: The Pseudocode

```
Initialize:
    Set Stream to the input text string
    Set currentPosition to 0 and internalQuoteFlag to false
    Set delimiterSet to ' , . ; : ! ? ( ) <> + " \n \t space
    Set whiteSpace to \t\n space
Procedure getNextToken:
    L1: cursor := currentPosition; ch := charAt(cursor);
        If ch = endOfStream then return null; endif
    L2: while ch is not endOfStream nor instanceOf(delimiterSet) do
            increment cursor by 1; ch := charAt(cursor);
        endwhile
        If ch = endOfStream then
            If cursor = currentPosition then return null; endif
        endif
        If ch is whiteSpace then
            If currentPosition = cursor then
                increment currentPosition by 1 and goto L1;
            else
                Token := substring(Stream,currentPosition,cursor-1);
                currentPosition := cursor+1; return Token;
            endif
        endif
        If ch = ' then
            If charAt(cursor-1) = instanceOf(delimiterSet) then
                internalQuoteFlag := true; increment currentPosition by 1; goto L1;
            endif
            If charAt(cursor+1) != instanceof(delimiterSet) then
                increment cursor by 1; ch := charAt(cursor); goto L2;
            elseif internalQuoteFlag = true then
                Token := substring(Stream,currentPosition,cursor-1);
                internalQuoteFlag := false;
            else
                Token := substring(Stream,currentPosition,cursor);
            endif
            currentPosition := cursor+1; return Token;
        endif
        If cursor = currentPosition then
            Token := ch; currentPosition := cursor+1;
        else
            Token := substring(Stream,currentPosition,cursor-1);
            currentPosition := cursor;
        endif
        return Token;
endprocedure
```

Tokenization: NLTK

```
from nltk import *

s = 'We bought apples, oranges, etc., etc.'
t = '#qcpoli enjoyed a hearty laugh today with #plq debate audience
        for @jflisee #notrehome tune was that the intended reaction?'


tokens = tokenize.word_tokenize(s) # a regular tokenizer
print(tokens)

tt = TweetTokenizer(t) # a Tweet specific tokenizer
tokens2 = tt.tokenize(t)
print(tokens2)
```

nltk.tokenize package

Submodules

nltk.tokenize.api module

Tokenizer Interface

class nltk.tokenize.api.**StringTokenizer** [source]

Bases: nltk.tokenize.api.TokenizerI

A tokenizer that divides a string into substrings by splitting on the specified string (defined in subclasses).

span_tokenize(*s*) [source]

Identify the tokens using integer offsets `(start_i, end_i)`, where `s[start_i:end_i]` is the corresponding token.

Return type

iter(tuple(int, int))

tokenize(*s*) [source]

Return a tokenized copy of *s*.

Return type

Stemming/Lemmatization

- Once a character stream (string) has been segmented into a sequence of tokens, the next step is to convert each of the tokens to a standard form (**text normalization**), a process usually referred to as **stemming** or **lemmatization**
  - *likes → like, carries → carry, books → book, etc.*
  - Stemming → stripping off any prefix/suffix
  - Lemmatization → stripping off any prefix/suffix so that the resulting form is a known word in a dictionary
- For classification algorithms that take **frequency** into account, this step can sometimes make a difference.
- In some other application, the extra processing may not provide any significant gains.

Stemming/Lemmatization

- When the normalization is confined to regularizing grammatical variants such as singular/plural and present/past, the process is called **inflectional stemming**. → **morphological analysis**
  - In some languages, morphological analysis is comparatively simple.
  - In English, which has many irregular word forms, it is more difficult.
    - *sought* → *seek* , *rebelled* → *rebel, belled* → *bell*

Stemming/Lemmatization

- An algorithm for inflectional stemming must be part **rule-based** and part **dictionary-based**.
- Any stemming algorithm for English that operates only on tokens, without more grammatical information such as part-of-speech, will make some mistakes because of ambiguity.
  - bored (adjective) → bored; bored (verb) → ? (bore or bear)
  - In the absence of some often complicated disambiguation process, a stemming algorithm should probably just pick the most frequent choice.
- See next slide for the Pseudocode for a somewhat simplified inflectional stemmer for English.
  - Notice how the algorithm consists of rules that are applied in sequence until one of them is satisfied, and the use of a dictionary, usually referred to as **a stemming dictionary**.

Stemming: The Pseudocode

Input: a text token and a dictionary
Doubling consonants: b d g k m n p r l t

Rules:

If token length < 4 **return** token
If token is number **return** token
If token is acronym **return** token
If token in dictionary **return** the stored stem
If token ends in s'
    strip the ' and **return** stripped token
If token ends in 's
    strip the 's and **return** stripped token
If token ends in "is", "us", or "ss" **return** token
If token ends in s
    strip s, check in dictionary, and **return** stripped token if there
If token ends with es
    strip es, check in dictionary, and **return** stripped token if there
If token ends in ies
    replace ies by y and **return** changed token
If token ends in s
    strip s and **return** stripped token
If token doesn't end with ed or ing **return** token
If token ends with ed
    strip ed, check in dictionary and **return** stripped token if there
If token ends in ied
    replace ied by y and **return** changed token
If token ends in eed
    remove d and **return** stripped token if in dictionary
If token ends with ing
    strip ing (if length > 5) and **return** stripped token if in dictionary
If token ends with ing and length ≤ 5 **return** token
// Now we have SS, the stripped stem, without ed or ing and it's
// not in the dictionary (otherwise algorithm would terminate)
If SS ends in doubling consonant
    strip final consonant and **return** the changed SS if in dictionary
If doubling consonant was l **return** original SS
If no doubled consonants in SS
    add e and **return** changed SS if in dictionary
If SS ends in c or z, or there is a g or l before the final doubling consonant
    add e and **return** changed SS
If SS ends in any consonant that is preceded by a single vowel
    add e and **return** changed SS
return SS

Aggressive Stemming

- Some practitioners have felt that more aggressive normalization is advantageous for at least some text-mining applications
  - The intent of these stemmers is to reach a root form with no inflectional or derivational prefixes and suffixes:

    denormalization → norm

    - This reduces the number of 'roots' in a text collection very drastically, making distributional statistics more reliable.
    - This also merges words with the same core meaning:
      - reapplied → apply

Stemming: NLTK

```
from nltk import *

s = 'We bought apples, oranges, etc., etc.'
t = '#qcpoli enjoyed a hearty laugh today with #plq debate audience
        for @jflisee #notrehome tune was that the intended reaction?‘

tokens = word_tokenize(s)
porter = PorterStemmer()
stems = [porter.stem(t) for t in tokens]
print(stems)

lancaster = LancasterStemmer()
stems = [lancaster.stem(t) for t in tokens]
print(stems)
```

nltk.stem package

Submodules

nltk.stem.api module

class `nltk.stem.api.StemmerI` [source]

Bases: `object`

A processing interface for removing morphological affixes from words. This process is known as stemming.

abstract `stem`(*token*) [source]

Strip affixes from the token and return the stem.

Parameters

token (*str*) – The token that should be stemmed.

nltk.stem.arlstem module

ARLSTem Arabic Stemmer The details about the implementation of this algorithm are described in: K. Abainia, S. Ouamour and H. Sayoud, A Novel Robust Arabic Light Stemmer , Journal of Experimental & Theoretical Artificial Intelligence (JETAI'17), Vol. 29, No. 3, 2017,

nltk.stem.snowball module

Snowball stemmers

This module provides a port of the Snowball stemmers developed by Martin Porter.

There is also a demo function: *snowball.demo()*.

class `nltk.stem.snowball.ArabicStemmer`(*ignore_stopwords=False*) [source]

Bases: `nltk.stem.snowball._StandardStemmer`

https://github.com/snowballstem/snowball/blob/master/algorithms/arabic/stem_Unicode.sbl
(Original Algorithm) The Snowball Arabic light Stemmer Algorithm : Assem Chelli

Abdelkrim Aries Lakhdar Benzahia

Nltk Version Author : Lakhdar Benzahia

`is_defined` = *False*

`is_noun` = *True*

`is_verb` = *True*

`prefix_step2a_success` = *False*

`prefix_step3a_noun_success` = *False*

`prefix_step3b_noun_success` = *False*

`stem`(*word*) [source]

Stem an Arabic word and return the stemmed form.

Algorithms
Download
Mailing Lists
License
Credits
Projects
Source on github

We present stemming algorithms (with implementations in Snowball) for the following languages:

- English (porter)
- English (porter2)
- A note on early English
- Romance stemmers:
  - French
  - Spanish
  - Catalan
  - Portuguese
  - Italian
  - Romanian
- Germanic stemmers
  - German
  - (German variant)
  - Dutch
- Scandinavian stemmers
  - Swedish
  - Norwegian (Bokmål)
  - Danish
- Russian
- Finnish
- Basque

There are two English stemmers, the original Porter stemmer, and an improved stemmer which has been called Porter2. Read the accounts of them to learn a bit more about using Snowball.

Each formal algorithm should be compared with the corresponding Snowball program.

Surprisingly, among the Indo-European languages (*), the French stemmer turns out to be the most complicated, whereas the

Lemmatization: NLTK

```
from nltk import *


# Lemmatization
wnl = WordNetLemmatizer()
print([wnl.lemmatize(t) for t in tokens])
```

Find Synonyms Using WordNet

- WordNet is a semantically oriented dictionary of English, similar to a traditional thesaurus but with a richer structure. Most popular in NLP.
- NLTK includes the English WordNet, with 155,287 words and 117,659 synonym sets (synsets).
  - Consider the sentence in (1a). If we replace the word *motorcar in (1a) with automobile*, to get (1b), the meaning of the sentence stays pretty much the same:

    *(1) a. Benz is credited with the invention of the **motorcar**.*

    *b. Benz is credited with the invention of the **automobile**.*

- We can explore these words with the help of WordNet:

```
>>> from nltk.corpus import wordnet as wn
>>> wn.synsets('motorcar')
[Synset('car.n.01')]
```

WordNet Synsets

- The entity car.n.01 is called a synset, or "synonym set," which is a collection of synonymous words (or "lemmas"):

```
>>> wn.synset('car.n.01').lemma_names()
['car', 'auto', 'automobile', 'machine', 'motorcar']
```

- Each word of a synset can have several meanings, e.g., car can also signify a train carriage, a gondola, or an elevator car.

```
>>> wn.synset('car.n.01').definition()
'a motor vehicle with four wheels; usually propelled by an internal
   combustion engine'
>>> wn.synset('car.n.01').examples()
['he needs a car to get to work']
```

WordNet Synsets

- Unlike the words *automobile* and *motorcar*, which are unambiguous and have one synset, the word *car* is ambiguous, having five synsets:

```
>>> wn.synsets('car')
[Synset('car.n.01'), Synset('car.n.02'), Synset('car.n.03'),
   Synset('car.n.04'), Synset('cable_car.n.01')]
>>> for synset in wn.synsets('car'):
...         print synset.lemma_names()
...
['car', 'auto', 'automobile', 'machine', 'motorcar']
['car', 'railcar', 'railway_car', 'railroad_car']
['car', 'gondola']
['car', 'elevator_car']
['cable_car', 'car']
```

WordNet: Antonyms

- Some lexical relationships hold between lemmas, e.g., **antonymy:**

```
>>> wn.lemma('supply.n.02.supply').antonyms()
[Lemma('demand.n.02.demand')]
>>> wn.lemma('rush.v.01.rush').antonyms()
[Lemma('linger.v.04.linger')]
>>> wn.lemma('horizontal.a.01.horizontal').antonyms()
[Lemma('vertical.a.01.vertical'), Lemma('inclined.a.02.inclined')]
>>> wn.lemma('staccato.r.01.staccato').antonyms()
[Lemma('legato.r.01.legato')]
```

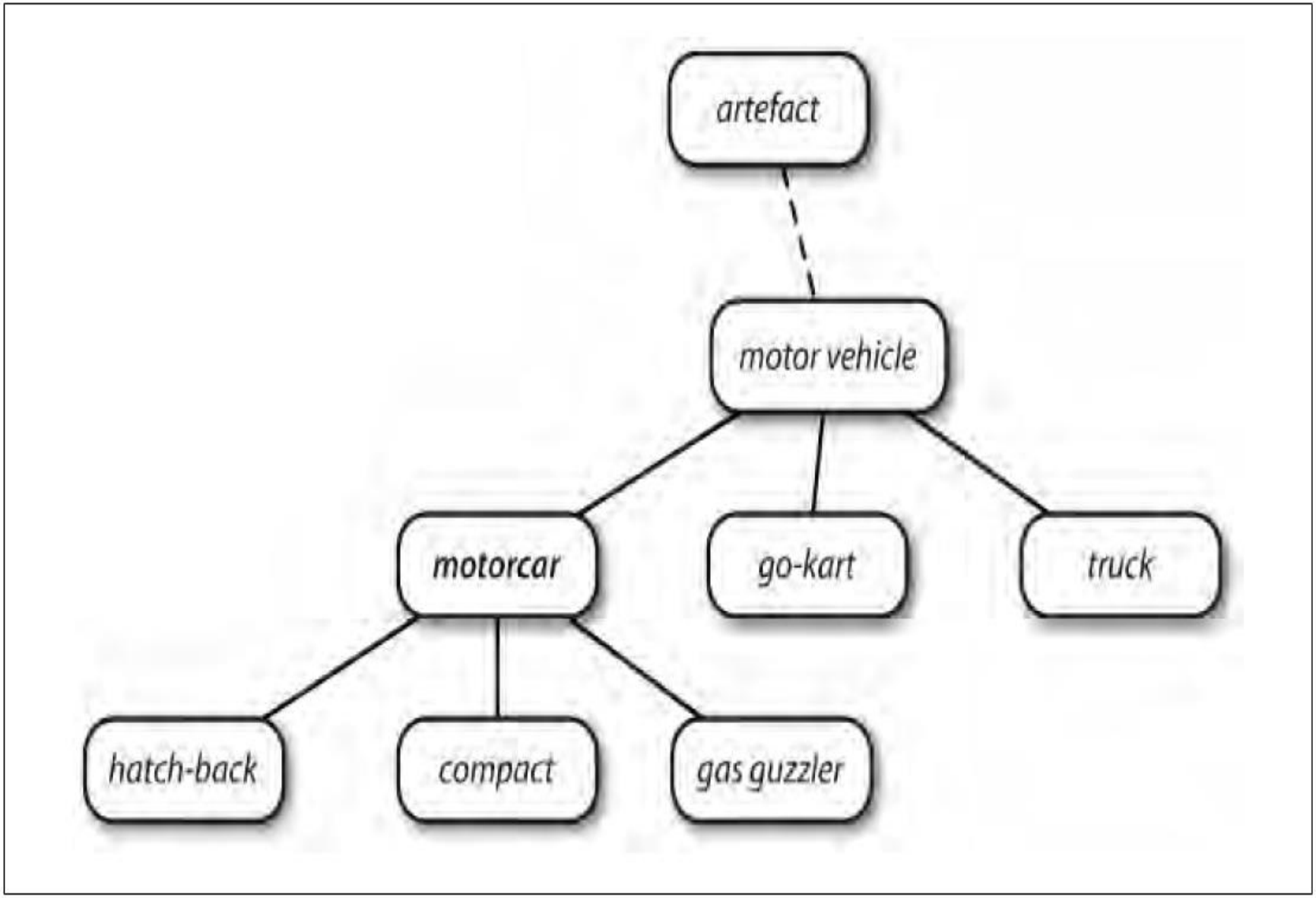


Figure 2-8. Fragment of WordNet concept hierarchy: Nodes correspond to synsets; edges indicate the hypernym/hyponym relation, i.e., the relation between superordinate and subordinate concepts.

WordNet Interface

WordNet is just another NLTK corpus reader, and can be imported like this:

```
>>> from nltk.corpus import wordnet
```

For more compact code, we recommend:

```
>>> from nltk.corpus import wordnet as wn
```

Words

Look up a word using `synsets()`; this function has an optional `pos` argument which lets you constrain the part of speech of the word:

```
>>> wn.synsets('dog') # doctest: +ELLIPSIS +NORMALIZE_WHITESPACE
[Synset('dog.n.01'), Synset('frump.n.01'), Synset('dog.n.03'), Synset('cad.n.01'),
Synset('frank.n.02'), Synset('pawl.n.01'), Synset('andiron.n.01'), Synset('chase.v.01')]
>>> wn.synsets('dog', pos=wn.VERB)
[Synset('chase.v.01')]
```

The other parts of speech are `NOUN`, `ADJ` and `ADV`. A synset is identified with a 3-part name of the form: word.pos.nn:

```
>>> wn.synset('dog.n.01')
Synset('dog.n.01')
>>> print(wn.synset('dog.n.01').definition())
a member of the genus Canis (probably descended from the common wolf) that has been domesticated by man since prehistoric t
>>> len(wn.synset('dog.n.01').examples())
1
>>> print(wn.synset('dog.n.01').examples()[0])
the dog barked all night
```

Stopwords

- **Stopwords** are common words that almost never have any predictive power for classifying texts, such as articles **a** and **the** and pronouns such as **it** and **they**.

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link

Stop words

From Wikipedia, the free encyclopedia

Not to be confused with Safeword.

In computing, **stop words** are words which are filtered out before or after processing of natural language data (text).[1] Though **stop words** usually refer to the most common words in a language, there is no single universal list of stop words used by all natural language processing tools, and indeed not all tools even use such a list. Some tools specifically avoid removing these **stop words** to support phrase search.

Any group of words can be chosen as the stop words for a given purpose. For some search engines, these are some of the most common, short function words, such as *the*, *is*, *at*, *which*, and *on*. In this case, stop words can cause problems when searching for phrases that include them, particularly in names such as "The Who", "The The", or "Take That". Other search engines remove some of the most common words—including lexical words, such as "want"—from a query in order to improve performance.[2]

Hans Peter Luhn, one of the pioneers in information retrieval, is credited with coining the phrase and using the concept. [3]

See also [edit]

- Text mining
- Concept mining
- Information extraction
- Natural language processing
- Query expansion
- Stemming
- Search engine indexing
- Poison words
- Function words

Table 1.2:

Some of the Corpora and Corpus Samples Distributed with NLTK: For information about downloading and using them, please consult the NLTK website.

Corpus	Compiler	Contents
Brown Corpus	Francis, Kucera	15 genres, 1.15M words, tagged, categorized
CESS Treebanks	CLiC-UB	1M words, tagged and parsed (Catalan, Spanish)
Chat-80 Data Files	Pereira & Warren	World Geographic Database
CMU Pronouncing Dictionary	CMU	127k entries
CoNLL 2000 Chunking Data	CoNLL	270k words, tagged and chunked
CoNLL 2002 Named Entity	CoNLL	700k words, pos- and named-entity-tagged (Dutch, Spanish)
CoNLL 2007 Dependency Treebanks (sel)	CoNLL	150k words, dependency parsed (Basque, Catalan)
Dependency Treebank	Narad	Dependency parsed version of Penn Treebank sample
FrameNet	Fillmore, Baker et al	10k word senses, 170k manually annotated sentences
Floresta Treebank	Diana Santos et al	9k sentences, tagged and parsed (Portuguese)
Gazetteer Lists	Various	Lists of cities and countries
Genesis Corpus	Misc web sources	6 texts, 200k words, 6 languages
Gutenberg (selections)	Hart, Newby, et al	18 texts, 2M words
Inaugural Address Corpus	CSpan	US Presidential Inaugural Addresses (1789-present)
Indian POS-Tagged Corpus	Kumaran et al	60k words, tagged (Bangla, Hindi, Marathi, Telugu)
MacMorpho Corpus	NILC, USP, Brazil	1M words, tagged (Brazilian Portuguese)
Movie Reviews	Pang, Lee	2k movie reviews with sentiment polarity classification
Names Corpus	Kantrowitz, Ross	8k male and female names
NIST 1999 Info Extr (selections)	Garofolo	63k words, newswire and named-entity SGML markup
Nombank	Meyers	115k propositions, 1400 noun frames
NPS Chat Corpus	Forsyth, Martell	10k IM chat posts, POS-tagged and dialogue-act tagged
Open Multilingual WordNet	Bond et al	15 languages, aligned to English WordNet
PP Attachment Corpus	Ratnaparkhi	28k prepositional phrases, tagged as noun or verb modifiers
Proposition Bank	Palmer	113k propositions, 3300 verb frames
Question Classification	Li, Roth	6k questions, categorized
Reuters Corpus	Reuters	1.3M words, 10k news documents, categorized
Roget's Thesaurus	Project Gutenberg	200k words, formatted text
RTE Textual Entailment	Dagan et al	8k sentence pairs, categorized
SEMCOR	Rus, Mihalcea	880k words, part-of-speech and sense tagged
Senseval 2 Corpus	Pedersen	600k words, part-of-speech and sense tagged
SentiWordNet	Esuli, Sebastiani	sentiment scores for 145k WordNet synonym sets
Shakespeare texts (selections)	Bosak	8 books in XML format
State of the Union Corpus	CSPAN	485k words, formatted text
Stopwords Corpus	Porter et al	2,400 stopwords for 11 languages
Swadesh Corpus	Wiktionary	comparative wordlists in 24 languages
Switchboard Corpus (selections)	LDC	36 phonecalls, transcribed, parsed
Univ Decl of Human Rights	United Nations	480k words, 300+ languages
Penn Treebank (selections)	LDC	40k words, tagged and parsed
TIMIT Corpus (selections)	NIST/LDC	audio files and transcripts for 16 speakers
VerbNet 2.1	Palmer et al	5k verbs, hierarchically organized, linked to WordNet

Stopwords

from nltk.corpus import **stopwords**

import **string**

stop = stopwords.**words('english')**

print(stop) # if you want to see them

tokens_filtered = [w for w in tokens if w.lower() not in **stop** and w.lower() not in **string.punctuation**]

print(tokens_filtered)`

print(string.punctuation) # if you want to see them

Find our which language is supported

print(stopwords.**fileids**())

Frequency Information

- Frequency information on the word counts can be quite useful in reducing dictionary size and can sometimes improve predictive performance for some methods.
  - The most frequent words are often stopwords and can be deleted.
  - The remaining most frequently used words are often the important words that should remain in the dictionary.
  - The very rare words are usually typos and can also be dismissed.
  - For some learning methods, a dictionary of the most frequent words, perhaps less than **200** words, can be surprisingly effective.

Frequency Information: NLTK

```
from nltk import *

tokens = tokenize.word_tokenize(reuters.raw('test/14826'))

fdist = FreqDist(tokens) # Create a frequency distribution

print(fdist.most_common(100))

print("Total number of tokens = {}".format(fdist.N()))
print("Total number of unique tokens = {}".format(len(fdist.keys())))

for token in fdist: # Iterate over the samples, in order of decreasing frequency
    print("Term " + token + " occurs " + str(fdist[token]) + " time(s).“)

fdist.plot() # Graphical plot of the frequency distribution
fdist.plot(cumulative=True)   # Cumulative plot of the frequency distribution
```

Table 3.1:

Functions Defined for NLTK's Frequency Distributions

Example	Description
`fdist = FreqDist(samples)`	create a frequency distribution containing the given samples
`fdist[sample] += 1`	increment the count for this sample
`fdist['monstrous']`	count of the number of times a given sample occurred
`fdist.freq('monstrous')`	frequency of a given sample
`fdist.N()`	total number of samples
`fdist.most_common(n)`	the `n` most common samples and their frequencies
`for sample in fdist:`	iterate over the samples
`fdist.max()`	sample with the greatest count
`fdist.tabulate()`	tabulate the frequency distribution
`fdist.plot()`	graphical plot of the frequency distribution
`fdist.plot(cumulative=True)`	cumulative plot of the frequency distribution
`fdist1 |= fdist2`	update `fdist1` with counts from `fdist2`
`fdist1 < fdist2`	test if samples in `fdist1` occur less frequently than in `fdist2`

Sentiment Analysis On Twitter
The Steps

Sentiment Analysis On Twitter

- Twitter is limited to 140 characters which forces the user to focus on one thought at a time
  - Users' tweets are teeming with emotions.
  - Viewing users' emotions or feelings **in aggregate** allows researchers to discover **public sentiments**

- **Step 1. Data preparation**
  - is the process of collecting and extracting usable data to feed into the classifier (classification algorithm).
  - The three steps to prepare data are:
    - Data Gathering
    - Data Selection
    - Data Filtering

Sentiment Analysis On Twitter

Step 1.1 Data gathering

- is the process actually deciding how to acquirer the data.
- Option 1. Directly use Twitter's APIs:
  - Search API & Streaming API
- Option 2. Use a third-party application, which accesses the Twitter APIs directly
  - These applications often will add their own sentiment evaluation, and researchers that are interested in comparing and improving these will applications tend to use them. - sentiment140.com, NLTK Twitter Corpus, etc.

Sentiment140

Search this site

General Information
Site Functionality
For Academics
API
Sentiment Analysis Sites
Contact Us

Return to Sentiment140

API

We provide APIs for classifying tweets. This allows you to integrate our sentiment analysis classifier into your site or product.

Contents

Registration

You may register your application at http://help.sentiment140.com/api/registration.

Please provide an `appid` parameter in your API requests. The `appid` value should be an email address we can contact. For example, if you are using the JSON Bulk Classification API, the HTTP endpoint that you use to send requests may look like this:

```
http://www.sentiment140.com/api/bulkClassifyJson?appid=bob@apple.com
```

where `bob@apple.com` is the main contact. You may URL-encode the `appid` value.

Technically you can use the API without supplying the `appid` parameter. But, we may block the requests that don't have an `appid` specified if we suspect abuse.

Sentiment140

Search this site

For Academics

Is the code open source?

Sentiment140 isn't open source, but there are resources with open source code with a similar implementation:

- Text Classification for Sentiment Analysis by Jacob Perkins
- TwitGraph by Ran Tavory
- Twitter sentiment analysis using Python and NLTK by Laurent Luce
- Twitter Sentiment Corpus by Niek Sanders

What algorithm are you using?

We are using a Maximum Entropy classifier. Read about our machine learning approach.

Where is the training data?

You can download our training data:

- Stanford link
- Google Drive link

What is the format of the training data?

The data is a CSV with emoticons removed. Data file format has 6 fields:

0 - the polarity of the tweet (0 = negative, 2 = neutral, 4 = positive)

SNAP for C++
SNAP for Python
SNAP Datasets
BIOSNAP Datasets
What's new
People
Papers
Projects
Citing SNAP
Links
About
Contact us

Open positions

Open research positions in **SNAP** group are available at undergraduate, graduate and postdoctoral levels.

476 million Twitter tweets

Dataset information

467 million Twitter posts from 20 million users covering a 7 month period from June 1 2009 to December 31 2009. We estimate this is about 20-30% of all public tweets published on Twitter during the particular time frame.

For each public tweet the following information is available:

- Author
- Time
- Content

We have no Twitter social graph (who-follows-whom graph) available. You can find a copy of the graph here (thanks to Haewoon Kwak, et al.).

Dataset statistics	
Number of users	17,069,982
Number of tweets	476,553,560
Number of URLs	181,611,080
Number of Hashtags	49,293,684
Number of re-tweets	71,835,017

Source (citation)

- J. Yang, J. Leskovec. Temporal Variation in Online Media. ACM International Conference on Web Search and Data Mining (WSDM '11), 2011.

As per request from Twitter the data is no longer available.

There are 83 twitter datasets available on data.world.

We're dedicated to providing an online platform for free, open data and these Twitter datasets are no exception.
Looking for data sets from and about Twitter? Explore these data sets, ready for your analysis.

TOP OPEN DATA TOPICS

- census (3600)
- refugees (1776)
- management (2471)
- biota (1691)
- wildlife (1872)
- geospatial (3106)
- water (1682)
- environment (2281)
- oregon (1528)
- cso (3142)
- active (2732)
- transect (2746)
- society (1497)
- hxl (3311)

OPEN

a2liz/favorited_tweets

collects a midwesterner's favorite tweets

Dataset • Updated last year • Public Domain License

twitter, favorites, ifttt

464

OPEN

adamhelsinger/Elon Musk Tweets-Until 4/6/17

Elon Musk's tweets from 2010-06-04 to 2017-04-05.

Dataset • Updated 3 years ago

twitter, tweets, elon musk, iron man, space x, +8

519

How data.world helps you.

Find fantastic data.

Discover the data you need.
Understand it at a glance.
Follow people, topics, and projects.

Discover data →

Data | Kernels (21) | Discussion (2) | Activity | Metadata | Download (493 MB) | New Notebook

Usability 9.1 **License** CC BY-NC-SA 4.0 **Tags** business, computing, computer science, statistics, internet

Description

The Customer Support on Twitter dataset is a large, modern corpus of tweets and replies to aid innovation in natural language understanding and conversational models, and for study of modern customer support practices and impact.

Sentiment Analysis On Twitter

Step 1.2 Data Selection

- Is the process deciding how they are going to conduct their experiments.
- **Volume-based experiments**
  - This approach is taken if you are only interested in processing as much data as they can, for example, to improve your classification algorithm.
- **Time series based experiments**
  - You will take the output from sentiment analysis and compare it against a time series.
  - For example, comparing the results to the Consumer Confidence Index , Dow Jones Index, Gallup polls.

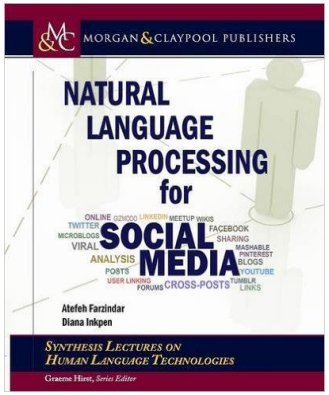
Sentiment Analysis On Twitter

Step 1.3 Data Filtering: Filtering plays a key role in massaging the data before it is fed to the classifier. (Garbage in, garbage out)

- Some typical approaches
  - Keywords – Use this approach to look for something specific in tweets.
    - If you are interested in public opinion of the economy and politicians, look for tweets that contains words like jobs, economy, or political parties.
  - Tweet Structure –If you are interested in tweets that display a subjective nature, such as “I feel” or “I believe.”
  - Emoticons – They are valuable because they display the feelings of the author.
  - Non-English Terms – for multi-lingual sentiment analysis

Noise (Revisited)

- The raw Twitter data is notoriously noisy, which makes filtering a necessary step.
  - The use of social networks has made everybody a potential author, so the language is now **closer to the user** than to any prescribed norms, such as those in newspaper.
  - Tweets, as well as posts & blogs, are written in an **informal**, **conversational** tone (a "stream of consciousness" rather than the meticulously edited work that might be expected in traditional print media.)



Noise (Revisited)

- While most standard NLP techniques were developed for long, structured, grammatical text, Tweets are short, colloquial, and ungrammatical. (See more examples on the following slide.)
  - Users frequently misspell words either unintentionally (teh, waht) or intentionally, by expanding words, abbreviating words, or using lexical/numeric substitutions:
    - E.g. looooovveeeee, rly, c u l8r,…
  - Twitter posts also frequently contain other non-standard tokens, including emoticons
  - Acronyms (lol, smh), hashtags, user mentions, or Twitter specfic terminology indicating re-tweeted posts (RT) and trending topics (TT).

TABLE I
SAMPLE TWEETS

Never say never.....dont let me goo dont let mee gooo dont let me gooooo....
@user13431 when r u commin to Montreal
#bestfeeling is feeling like u mean the world to someone
My work buddy 'go smoke' like 3 times already
mai8mai RT @user1341 : Support Breast Cancer Awareness. Add A #twibbon To Your Avatar Now!!
I'm so #overyou Didn't even know it was possible!!!

Text Normalization

- As a result, two tweets with alternate spellings of some word may not be considered related, when in fact they are.
- **Normalization** of tweets remains a difficult problem, but fortunately some progress has been made, and we will discuss normalizing text, in general, and normalizing tweets, in particular later. (See some examples on the next slide)

Original: @user3419 nay lol y u say dat?&wat u doing 2day?

Post-normalization: No, why did you say that? What you doing today?

Original: 1001 colors: Contemporary art from Iran <URL> #Iran #culture #Art

Post-normalization: 1001 colors: contemporary art from Iran <URL>.

Original: it's soo quiet, it's like I'm goin die

Post-normalization: It is so quiet, it is like i am going to die.

Original: #worstfeeling buyin a fresh laptop..then ur screen blowz out :((

Post-normalization: worst feeling is buying a fresh laptop.. then your screen blowz out.

---

*Original:*This is superb Grape+apple splash with manggo juice, super!

Post-normalization: This is superb grape + Apple splash with mango juice, Super!

Original: @user31903 u n ur fam can n if u interested ill b n touch w u bout it

Post-normalization: You and your family can and if you interested Ill be and touch with you about it.

Original: RT: @user4191 BEAUTIFUL CREATURES has a new #website designed by @user4192!

Post-normalization: Beautiful creatures has a new website designed by @user4192!

---

researchers studying SMS normalization have chosen to evaluate their results with the BLEU metric, it might not be the best choice. The BLEU scoring metric was designed for evaluating translations from one language to another, not for evaluating the results of noisy text normalization. Because of this, a better BLEU score does not necessarily mean a better translation. For example, the subjectivity of the human annotators could cause substantial variation in BLEU scores. In papers such as [8] their corpora was only annotated by two people. The fact that there were 10 annotators could have lead to inconsistencies in the scoring data. For example, although annotators were instructed to expand contractions, some annotators chose to translate *Im* as *I'm*, instead of *I am*. BLEU scores are obtained by comparing the similarities between *n-grams* of the hypothesized translation and gold standards, so errors such as this could have detrimental effects on the score, despite the fact that "I'm" and "I am" are grammatically equivalent.

Even if we ignore the issue of the applicability of BLEU as an evaluator itself, there are still several problems with the BLEU metric itself. The relationship between BLEU scores and human judgment is questionable. Papers such as [16] have suggested that an increase in BLEU score may not correlate with an increase in translation quality. In fact, on a test of several machine translation systems, the correlation between human and BLEU scores was found to be as low as .38 in some cases. One example where the BLEU score performed poorly was on the tweet *@user12493 I'm following u now should I hold on tight?*. The translation generated by the normalization system was *I'm following you now, should I hold on tight*". This seems like a perfectly acceptable translation. However, the human annotator translated the tweet as *I'm following you. Now, should I hold on tight?*. BLEU scores this translation at .43. However, both translations are acceptable.

Punctuation, Capitalization & Emoticons

- **Inconsistent (or absent) punctuation and capitalization** can make detection of sentence boundaries quite difficult - sometimes even for human readers:
  - **#qcpoli enjoyed a hearty laugh today with #plq debate audience for @jflisee #notrehome tune was that the intended reaction?**
  - → Big problem for end-of-sentence detection, and NER

- **Grammaticality**, or frequent lack thereof, is another concern for any syntactic analyses of social media texts, where fragments can be as commonplace as actual full sentences

Other Forms of Noise

- Social media are also much noisier than traditional print media.
  - Like much else on the Internet, social networks are plagued with **spam**, **ads**, and all manner of other unsolicited, irrelevant, or distracting content. (next slide)
  - Even by ignoring these forms of noise, much of the genuine, legitimate content on social media can be seen as irrelevant with respect to most information needs.
- In a study, the authors collected over 40,000 tweets. Only 36% of the tweets were rated as "worth reading."
  - The least valued tweets were so-called presence maintenance posts (e.g., "Hullo twitter!").
  - Pre-processing to filter out spam and other irrelevant content, or models that are better capable of coping with noise, are essential in mining social media texts.

Twitter bot

From Wikipedia, the free encyclopedia
(Redirected from Twitterbot)

A **Twitter bot** is a type of bot software that controls a Twitter account via the Twitter API.[1] The bot software may autonomously perform actions such as tweeting, retweeting, liking, following, unfollowing, or direct messaging other accounts. The automation of Twitter accounts is governed by a set of automation rules that outline proper and improper uses of automation.[2] Proper usage includes broadcasting helpful information, automatically generating interesting or creative content, and automatically replying to users via direct message.[3][4][5] Improper usage includes circumventing API rate limits, violating user privacy, or spamming.[6]

Features [edit]

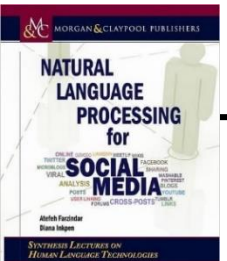
It is sometimes desirable to identify when a Twitter account is controlled by a bot. In a 2012 paper,[1] Chu et al. propose the following criteria that indicate that an account may be a bot (they were designing an automated system):

- "Periodic and regular timing" of tweets;
- Whether the tweet content contains known spam; and
- The ratio of tweets from mobile versus desktop, as compared to an average human Twitter user.

Research shows that humans can view Twitterbots as a credible source of information.[7]

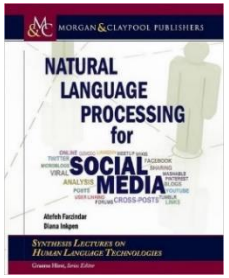
Text Normalization

- Text normalization is a possible solution for reducing noise, which can be approached in two stages:
  - The identification of **orthographic errors** in an input text
  - The correction of these errors
- Normalization approaches typically include a **dictionary** of known correctly spelled terms, and detects in-vocabulary (IV) and **out-of-vocabulary** (OOV) terms with respect to this dictionary.
  - Basic normalization deals with the errors detected at the POS tagging stage, such as **unknown words**, **misspelled words**, etc.
  - Advanced normalization takes a supervised approach, trained on an external dataset - annotated with short forms versus their equivalent long or corrected forms.



Text Normalization for Tweets

- Twitter text normalization into traditional written English is described in [Han and Baldwin, 2011], as not only difficult, but also a "lossy" translation task.



Lexical Normalisation of Short Text Messages: Makn Sens a #twitter

Bo Han and **Timothy Baldwin**
NICTA Victoria Research Laboratory
Department of Computer Science and Software Engineering
The University of Melbourne
hanb@student.unimelb.edu.au tb@ldwin.net

Abstract

Twitter provides access to large volumes of data in real time, but is notoriously noisy, hampering its utility for NLP. In this paper, we target out-of-vocabulary words in short text messages and propose a method for identifying and normalising ill-formed words. Our method uses a classifier to detect ill-formed words, and generates correction candidates based on morphophonemic similarity. Both word similarity and context are then exploited to select the most probable correction candidate for the word. The proposed method doesn't require any annotations, and achieves state-of-the-art performance over an SMS corpus and a novel dataset based on Twitter.

1 Introduction

Twitter and other micro-blogging services are highly

Manning, 2003; de Marneffe et al., 2006) analyses *bout the paper* and *thinkin movies* as a clause and noun phrase, respectively, rather than a prepositional phrase and verb phrase. If there were some way of preprocessing the message to produce a more canonical lexical rendering, we would expect the quality of the parser to improve appreciably. Our aim in this paper is this task of lexical normalisation of noisy English text, with a particular focus on Twitter and SMS messages. In this paper, we will collectively refer to individual instances of typos, ad hoc abbreviations, unconventional spellings, phonetic substitutions and other causes of lexical deviation as "ill-formed words".

The message normalisation task is challenging. It has similarities with spell checking (Peterson, 1980), but differs in that ill-formedness in text messages is often intentional, whether due to the desire

Text Normalization for Tweets

- Key features for determining if an “ill-formed” OOV (Out-Of-Vocabulary) word is similar to a IV (In-Vocabulary) word:
  - **Lexical edit distance**
  - Phonemic edit distance
  - Prefix substring
  - Suffix substring
  - Longest common subsequence (LCS)

Edit Distance

- The most popular approach to determining how similar one string is to another is by **edit distance**.
  - The edit distance between two strings is the number of **edit operations** required to turn one string into the other string.
  - **Edit operations** include insertions, deletions, and substitutions.
    - An **insertion** adds a character to the source string to make it more similar to the target string.
    - A **deletion** removes a character.
    - A **substitution** replaces one character in the source string with another from the target string.
  - The edit distance is the sum of the number of insertions, deletions, and substitutions required to transform one string into another.
    - For example, to turn the string *hello* into *yellow* would require substitution and one insertion, resulting in an edit distance of 2. Turning *here* into *there* is edit distance 1.

Jaro-Winkler Distance

- Another way to look at fuzzy string matching is in terms of **character overlap**.
- Intuitively, strings that share many of the same characters are more similar to one another than strings that share few or no characters.
- The best known approach in this category is the **Jaro-Winkler distance**.

Jaro-Winkler Distance

- One disadvantage of character overlap approaches is that they don't model **character order**.
  - If a word is reversed (*stressed* → *desserts*), its score is identical to a string that matches exactly.
- The Jaro-Winkler distance tries to heuristically address this in three distinct ways:
  1. It limits matching to a window of characters in the second string, based on the length of the larger of the strings.
  2. It factors in the number of transpositions (*ab* → *ba*).
  3. It adds a bonus based on the length of the largest common prefix.
- Studies showed that the Jaro-Winkler distance performs best when dealing with **shorter strings**, such as person names.

Winkler, W. E. (2006). "Overview of Record Linkage and Current Research Directions" (PDF). *Research Report Series, RRS*.

Definition [edit]

https://en.wikipedia.org/wiki/Jaro%E2%80%93Winkler_distance

The Jaro distance d_j of two given strings s_1 and s_2 is

$$d_j = \begin{cases} 0 & \text{if } m = 0 \\ \frac{1}{3}\left(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m}\right) & \text{otherwise} \end{cases}$$

Where:

- m is the number of *matching characters* (see below);
- t is half the number of *transpositions* (see below).

Two characters from s_1 and s_2 respectively, are considered *matching* only if they are the same and not farther than $\left\lfloor \frac{\max(|s_1|,|s_2|)}{2} \right\rfloor - 1$.

Each character of s_1 is compared with all its matching characters in s_2. The number of matching (but different sequence order) characters divided by 2 defines the number of *transpositions*. For example, in comparing CRATE with TRACE, only 'R' 'A' 'E' are the matching characters, i.e. m=3. Although 'C', 'T' appear in both strings, they are farther than 1, i.e., floor(5/2)-1=1. Therefore, t=0 . In DwAyNE versus DuANE the matching letters are already in the same order D-A-N-E, so no transpositions are needed.

Jaro–Winkler distance uses a prefix scale p which gives more favourable ratings to strings that match from the beginning for a set prefix length ℓ. Given two strings s_1 and s_2, their Jaro–Winkler distance d_w is:

$$d_w = d_j + (\ell p(1 - d_j))$$

where:

- d_j is the Jaro distance for strings s_1 and s_2
- ℓ is the length of common prefix at the start of the string up to a maximum of 4 characters
- p is a constant scaling factor for how much the score is adjusted upwards for having common prefixes. p should not exceed 0.25, otherwise the distance can become larger than 1. The standard value for this constant in Winkler's work is $p = 0.1$

jaro-winkler 2.0.0

```
pip install jaro-winkler
```

Latest version

Released: Nov 13, 2019

Original, standard and customisable versions of the Jaro-Winkler functions.

Navigation

Project description

Release history

Download files

Project links

Homepage

Project description

JaroWinkler

pypi package 2.0.0

Original, standard and customisable versions of the Jaro-Winkler functions.

```
>>> import jaro
>>> jaro.jaro_winkler_metric(u'SHACKLEFORD', u'SHACKELFORD')
0.9818181
>>> help(jaro)

Help on package jaro:
```

pyjarowinkler 1.8

```
pip install pyjarowinkler
```

Latest version

Released: Mar 22, 2016

Find the Jaro Winkler Distance which indicates the similarity score between two Strings

Navigation

Project description

Release history

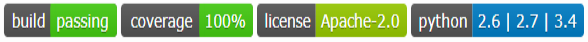
Download files

Project links

Homepage

Download

Project description



Find the Jaro Winkler Distance which indicates the similarity score between two Strings. The Jaro measure is the weighted sum of percentage of matched characters from each file and transposed characters. Winkler increased this measure for matching initial characters.

The Implementation

The original implementation is based on the Jaro Winkler Similarity Algorithm article that can be found on Wikipedia. This Python version of the original implementation is based on the Apache StringUtils library.

Text Normalization for Tweets

- Key features for determining if an "ill-formed" OOV (Out-Of-Vocabulary) word is similar to a IV (In-Vocabulary) word:
  - Lexical edit distance
  - **Phonemic edit distance**
  - Prefix substring
  - Suffix substring
  - Longest common subsequence (LCS)

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes

Soundex

From Wikipedia, the free encyclopedia

This article is about the phonetic algorithm. For the Rock n' Soul band, see the SoundEx.

Soundex is a phonetic algorithm for indexing names by sound, as pronounced in English. The goal is for homophones to be encoded to the same representation so that they can be matched despite minor differences in spelling.[1] The algorithm mainly encodes consonants; a vowel will not be encoded unless it is the first letter. Soundex is the most widely known of all phonetic algorithms (in part because it is a standard feature of popular database software such as DB2, PostgreSQL,[2] MySQL,[3] Ingres, MS SQL Server[4] and Oracle[5]) and is often used (incorrectly) as a synonym for "phonetic algorithm".[*citation needed*] Improvements to Soundex are the basis for many modern phonetic algorithms.[6]

Contents [hide]

py4Soundex 0.0.1

```
pip install py4Soundex
```

Latest version

Released: May 23, 2020

Soundex Python Library

Navigation

Project description

Release history

Download files

Project links

Homepage

Project description

py4Soundex

Soundex Python Library

Getting Started

This project is simply implementation of Soundex algorithm in python programming language.

Prerequisites

Sentiment Classification with POS

- The part-of-speech (POS) of each word can be important too.
- Words of different parts of speech (POS) may be treated differently.
  - It was shown that adjectives are important indicators of opinions.
  - Some researchers treated adjectives as special features.
- However, one can also use all POS tags and their n-grams as features.
- Our textbook uses the standard Penn Treebank POS Tags as shown in the following Table (Santorini, 1990).
- The Penn Treebank site is at:

  http://www.cis.upenn.edu/~treebank/home.html.

(This web page is permanently under construction.)

The Penn Treebank Project

The Penn Treebank Project annotates naturally-occuring text for linguistic structure. Most notably, we produce skeletal parses showing rough syntactic and semantic information -- a *bank* of linguistic *trees*. We also annotate text with part-of-speech tags, and for the Switchboard corpus of telephone conversations, dysfluency annotation. We are located in the LINC Laboratory of the Computer and Information Science Department at the University of Pennsylvania.
All data produced by the Treebank is released through the Linguistic Data Consortium.

Descriptions and samples of annotated corpora:

Wall Street Journal | The Brown Corpus | Switchboard | ATIS

On-line tgrep searches are now possible for those with LDC Online access.

Frequently Asked Questions (FAQs)

- tokenization
- NP heads and Base NPs in Treebank II bracketing

Annotation Style Manuals

- Part-of-speech tagging
- Treebank I bracketing was used until 12/92.
- Treebank II bracketing is designed to allow the extraction of simple predicate-argument structure.
- Dysfluency annotation used for Switchboard corpus only

TABLE 3.1: Penn Treebank Part-Of-Speech (POS) tags

TAG	DESCRIPTION	TAG	DESCRIPTION
CC	Coordinating conjunction	PRP$	Possessive pronoun
CD	Cardinal number	RB	Adverb
DT	Determiner	RBR	Adverb, comparative
EX	Existential *there*	RBS	Adverb, superlative
FW	Foreign word	RP	Particle
IN	Preposition or subordinating conjunction	SYM	Symbol
JJ	Adjective	TO	*to*
JJR	Adjective, comparative	UH	Interjection
JJS	Adjective, superlative	VB	Verb, base form
LS	List item marker	VBD	Verb, past tense
MD	Modal	VBG	Verb, gerund or present participle
NN	Noun, singular or mass	VBN	Verb, past participle
NNS	Noun, plural	VBP	Verb, non-3rd person singular present
NNP	Proper noun, singular	VBZ	Verb, 3rd person singular present
NNPS	Proper noun, plural	WDT	Wh-determiner
PDT	Predeterminer	WP	Wh-pronoun
POS	Possessive ending	WP$	Possessive wh-pronoun

:	colon or ellipsis	: ; ...
CC	conjunction, coordinating	& 'n and both but either et for less minus neither nor or plus so therefore times v. versus vs. whether yet
CD	numeral, cardinal	mid-1890 nine-thirty forty-two one-tenth ten million 0.5 one forty-seven 1987 twenty '79 zero two 78-degrees eighty-four IX '60s .025 fifteen 271,124 dozen quintillion DM2,000 ...
DT	determiner	all an another any both del each either every half la many much nary neither no some such that the them these this those
EX	existential there	there
FW	foreign word	gemeinschaft hund ich jeux habeas Haementeria Herr K'ang-si vous lutihaw alai je jour objets salutaris fille quibusdam pas trop Monte terram fiche oui corporis ...
IN	preposition or conjunction, subordinating	astride among uppon whether out inside pro despite on by throughout below within for towards near behind atop around if like until below next into if beside ...
JJ	adjective or numeral, ordinal	third ill-mannered pre-war regrettable oiled calamitous first separable ectoplasmic battery-powered participatory fourth still-to-be-named multilingual multi-disciplinary ...
JJR	adjective, comparative	bleaker braver breezier briefer brighter brisker broader bumper busier calmer cheaper choosier cleaner clearer closer colder commoner costlier cozier creamier crunchier cuter ...
		calmest cheapest choicest classiest cleanest clearest

nltk.tag.**pos_tag**(*tokens*) [source]

Use NLTK's currently recommended part of speech tagger to tag the given list of tokens.

```
>>> from nltk.tag import pos_tag
>>> from nltk.tokenize import word_tokenize
>>> pos_tag(word_tokenize("John's big idea isn't all that bad."))
[('John', 'NNP'), ("'s", 'POS'), ('big', 'JJ'), ('idea', 'NN'), ('is',
'VBZ'), ("n't", 'RB'), ('all', 'DT'), ('that', 'DT'), ('bad', 'JJ'),
('.', '.')]
```

Parameters:tokens (*list(str)*) – Sequence of tokens to be tagged

Returns: The tagged tokens

Return type:list(tuple(str, str))

api Module

Interface for tagging each token in a sentence with supplementary information, such as its part of speech.

class nltk.tag.api.**FeaturesetTaggerI** [source]

Named Entities

- Named entities are definite noun phrases that refer to specific types of individuals, such as organizations, persons, dates,…

Natural Language Processing with Python (2009)+++.pdf - Adobe Reader

File Edit View Window Help

Table 7-3. Commonly used types of named entity

NE type	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*

human-made artifacts in the domains of architecture and civil engineering

geo-political entities such as city, state/province, and country.

7.00 x 9.19 in

NE Recognition with NLTK

- NLTK provides a classifier that has already been trained to recognize named entities, accessed with the function nltk.ne_chunk().
- If we set the parameter *binary=True* , then named entities are just tagged as NE. Otherwise, the classifier adds category labels such as PERSON, ORGANIZATION, and GPE.

```
>>> print nltk.ne_chunk(sent, binary=True)
(S
The/DT
(NE U.S./NNP)
is/VBZ
one/CD
...
according/VBG
to/TO
(NE Brooke/NNP T./NNP Mossman/NNP)
...)
```

SpaCy

Industrial-Strength Natural Language Processing

IN PYTHON

Get things done

spaCy is designed to help you do real work — to build real products, or gather real insights. The library respects your time, and tries to avoid wasting it. It's easy to install, and its API is simple and productive. We like to think of spaCy as the Ruby on Rails of Natural Language Processing.

GET STARTED

Blazing fast

spaCy excels at large-scale information extraction tasks. It's written from the ground up in carefully memory-managed Cython. Independent research in 2015 found spaCy to be the fastest in the world. If your application needs to process entire web dumps, spaCy is the library you want to be using.

FACTS & FIGURES

Deep learning

spaCy is the best way to prepare text for deep learning. It interoperates seamlessly with TensorFlow, PyTorch, scikit-learn, Gensim and the rest of Python's awesome AI ecosystem. With spaCy, you can easily construct linguistically sophisticated statistical models for a variety of NLP problems.

READ MORE

Edit the code & try spaCy spaCy v2.2.0 · Python 3 · via Binder

```
# pip install spacy
# python -m spacy download en_core_web_sm

import spacy

# Load English tokenizer, tagger, parser, NER and word vectors
nlp = spacy.load("en_core_web_sm")

# Process whole documents
text = ("When Sebastian Thrun started working on self-driving cars at "
        "Google in 2007, few people outside of the company took him "
        "seriously. “I can tell you very senior CEOs of major American "
        "car companies would shake my hand and turn away because I wasn’t "
        "worth talking to,” said Thrun, in an interview with Recode earlier "
        "this week.")
doc = nlp(text)

# Analyze syntax
print("Noun phrases:", [chunk.text for chunk in doc.noun_chunks])
print("Verbs:", [token.lemma_ for token in doc if token.pos_ == "VERB"])

# Find named entities, phrases and concepts
for entity in doc.ents:
    print(entity.text, entity.label_)
```

RUN

Features

- Non-destructive **tokenization**
- **Named entity** recognition
- Support for **53+ languages**
- **23 statistical models** for 11 languages
- pretrained **word vectors**
- State-of-the-art speed
- Easy **deep learning** integration
- Part-of-speech tagging
- Labelled dependency parsing
- Syntax-driven sentence segmentation
- Built in **visualizers** for syntax and NER
- Convenient string-to-hash mapping
- Export to numpy data arrays
- Efficient binary serialization
- Easy **model packaging** and deployment
- Robust, rigorously evaluated accuracy

```
# pip install spacy
# python -m spacy download en_core_web_sm

import spacy

# Load English tokenizer, tagger, parser, NER and word vectors
nlp = spacy.load("en_core_web_sm")

# Process whole documents
text = ("When Sebastian Thrun started working on self-driving cars at "
        "Google in 2007, few people outside of the company took him "
        "seriously. “I can tell you very senior CEOs of major American "
        "car companies would shake my hand and turn away because I wasn’t "
        "worth talking to,” said Thrun, in an interview with Recode earlier "
        "this week.")
doc = nlp(text)

# Analyze syntax
print("Noun phrases:", [chunk.text for chunk in doc.noun_chunks])
print("Verbs:", [token.lemma_ for token in doc if token.pos_ == "VERB"])

# Find named entities, phrases and concepts
for entity in doc.ents:
    print(entity.text, entity.label_)
```

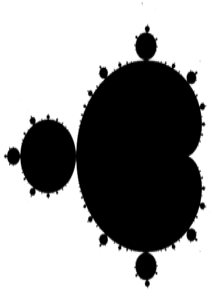
Python 3.7.3 Shell

File Edit Shell Debug Options Window Help

```
Python 3.7.3 (v3.7.3:ef4ec6ed12, Mar 25 2019, 22:22:05) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: \\hd.ad.syr.edu\02\d7fef1\Documents\Desktop\testspacy.py =====
Noun phrases: ['Sebastian Thrun', 'self-driving cars', 'Google', 'few people', 'the company', 'him', 'I', 'you', 'very senior CEOs', 'major American car com
panies', 'my hand', 'I', 'Thrun', 'an interview', 'Recode']
Verbs: ['start', 'work', 'drive', 'take', 'can', 'tell', 'would', 'shake', 'turn', 'be', 'talk', 'say']
Sebastian Thrun PERSON
Google ORG
2007 DATE
American NORP
Thrun PERSON
Recode ORG
earlier this week DATE
>>>
```

Ln: 14 Col: 4

TYPE	DESCRIPTION
PERSON	People, including fictional.
NORP	Nationalities or religious or political groups.
FAC	Buildings, airports, highways, bridges, etc.
ORG	Companies, agencies, institutions, etc.
GPE	Countries, cities, states.
LOC	Non-GPE locations, mountain ranges, bodies of water.
PRODUCT	Objects, vehicles, foods, etc. (Not services.)
EVENT	Named hurricanes, battles, wars, sports events, etc.
WORK_OF_ART	Titles of books, songs, etc.
LAW	Named documents made into laws.
LANGUAGE	Any named language.
DATE	Absolute or relative dates or periods.
TIME	Times smaller than a day.
PERCENT	Percentage, including "%".
MONEY	Monetary values, including unit.
QUANTITY	Measurements, as of weight or distance.
ORDINAL	"first", "second", etc.
CARDINAL	Numerals that do not fall under another type.



TextBlob

TextBlob is a Python (2 and 3) library for processing textual data. It provides a consistent API for diving into common natural language processing (NLP) tasks such as part-of-speech tagging, noun phrase extraction, sentiment analysis, and more.

Useful Links

TextBlob @ PyPI
TextBlob @ GitHub
Issue Tracker

Stay Informed

TextBlob: Simplified Text Processing

Release v0.15.2. (Changelog)

TextBlob is a Python (2 and 3) library for processing textual data. It provides a simple API for diving into common natural language processing (NLP) tasks such as part-of-speech tagging, noun phrase extraction, sentiment analysis, classification, translation, and more.

```
from textblob import TextBlob

text = '''
The titular threat of The Blob has always struck me as the ultimate movie
monster: an insatiably hungry, amoeba-like mass able to penetrate
virtually any safeguard, capable of--as a doomed doctor chillingly
describes it--"assimilating flesh on contact.
Snide comparisons to gelatin be damned, it's a concept with the most
devastating of potential consequences, not unlike the grey goo scenario
proposed by technological theorists fearful of
artificial intelligence run rampant.
'''

blob = TextBlob(text)
blob.tags           # [('The', 'DT'), ('titular', 'JJ'),
                    #  ('threat', 'NN'), ('of', 'IN'), ...]

blob.noun_phrases   # WordList(['titular threat', 'blob',
                    #           'ultimate movie monster',
                    #           'amoeba-like mass', ...])
```