NLTK II

Marina Sedinkina
- Folien von Desislava Zhekova -

CIS, LMU
marina.sedinkina@campus.lmu.de

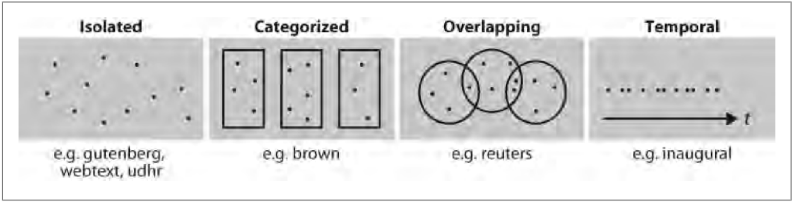
December 11, 2018

Outline

NLP and Corpora

- **Corpora** are large collections of linguistic data
- designed to achieve specific goal in NLP: data should provide best representation for the task. Such tasks are for example:
  - word sense disambiguation:
  - sentiment analysis
  - text categorization
  - part of speech tagging

Corpora Structure



Corpora

- When the `nltk.corpus` module is imported, it automatically creates a set of corpus reader instances that can be used to access the corpora in the NLTK data distribution
- The corpus reader classes may be of several subtypes: `CategorizedTaggedCorpusReader`, `BracketParseCorpusReader`, `WordListCorpusReader`, `PlaintextCorpusReader` ...

```
from nltk.corpus import brown

print(brown)

# prints
# <CategorizedTaggedCorpusReader in '.../corpora/brown' (not
    loaded yet)>
```

Corpus functions

Objects of type `CorpusReader` support the following functions:

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

Corpus functions

`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

Brown Corpus

```python
from nltk.corpus import brown

print(brown.categories())
# ["adventure", "belles_lettres", "editorial", "
    fiction", "government", "hobbies", "humor", "
    learned", "lore", "mystery", "news", "religion",
    "reviews", "romance", "science_fiction"]
```

Brown Corpus

```python
from nltk.corpus import brown

print(brown.categories())
print(brown.words(categories="news"))
# ["The", "Fulton", "County", "Grand", "Jury", "said
    ", ... ]
```

Access the list of words, but restrict them to a specific category.

Brown Corpus

```python
from nltk.corpus import brown

print(brown.categories())
print(brown.words(categories="news"))

print(brown.words(fileids=["cg22"]))
# ["Does", "our", "society", "have", "a", "runaway",
    ",", ... ]
```

Access the list of words, but restrict them to a specific file.

Brown Corpus

```
from nltk.corpus import brown

print(brown.categories())
print(brown.words(categories="news"))
print(brown.words(fileids=["cg22"]))

print(brown.sents(categories=["news", "editorial", "
    reviews"]))
# [["The", "Fulton", "County" ... ], ["The", "jury", "
    further" ... ], ... ]
```

Access the list of sentences, but restrict them to a given list of categories.

Brown Corpus

We can compare genres in their usage of modal verbs:

```
import nltk
from nltk.corpus import brown

news_text = brown.words(categories="news")
fdist = nltk.FreqDist([w.lower() for w in news_text])
modals = ["can", "could", "may", "might", "must", "
    will"]

for m in modals:
    print(m + ":", fdist[m])

# can: 94
# could: 87
# may: 93
# might: 38
```

Gutenberg Corpus

NLTK includes a small selection of texts from the Project Gutenberg electronic text archive, which contains more than 50 000 free electronic books, hosted at `http://www.gutenberg.org`.

Gutenberg Corpus

```
>>> import nltk
>>> nltk.corpus.gutenberg.fileids()
["austen-emma.txt", "austen-persuasion.txt", "austen-sense.txt", "bible-kjv.txt", "blake-poems.txt", "bryant-stories.txt", "burgess-busterbrown.txt", "carroll-alice.txt", "chesterton-ball.txt", "chesterton-brown.txt", "chesterton-thursday.txt", "edgeworth-parents.txt", "melville-moby_dick.txt", "milton-paradise.txt", "shakespeare-caesar.txt", "shakespeare-hamlet.txt", "shakespeare-macbeth.txt", "whitman-leaves.txt"]
```

Inaugural Address Corpus

Time dimension property:

```
>>> from nltk.corpus import inaugural
>>> inaugural.fileids()
["1789-Washington.txt", "1793-Washington.txt", "1797-
    Adams.txt", ... ]
>>> [fileid[:4] for fileid in inaugural.fileids()]
["1789", "1793", "1797", "1801", "1805", "1809", "
    1813", "1817", "1821", ... ]
```

Gutenberg Corpus

Extract statistics about the corpus:

```
for fileid in gutenberg.fileids():

  num_vocab=len(set([w.lower() for w in gutenberg.words(fileid)]))

  x1= len(gutenberg.raw(fileid))/len(gutenberg.words(fileid))
  x2= len(gutenberg.words(fileid))/len(gutenberg.sents(fileid))
  x3= len(gutenberg.words(fileid))/num_vocab
```

???

What is calculated here?

Annotated Text Corpora

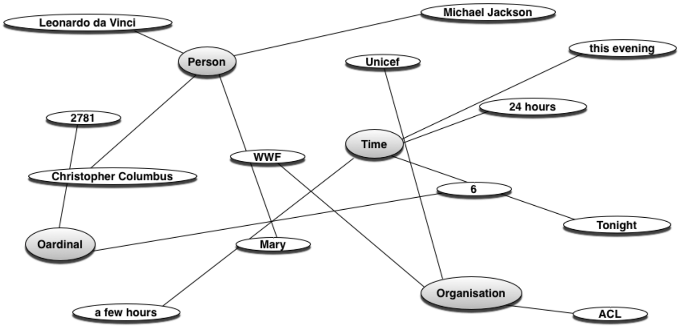
Many text corpora contain linguistic annotations:

- part-of-speech tags
- named entities
- syntactic structures
- semantic roles

Annotated Text Corpora

```
#begin document <document ID>
<sentence>

<sentence>
...
<sentence>

#end document <document ID>
...
#begin document <document ID>
<sentence>

<sentence>
...
<sentence>

#end document <document ID>
```

Annotated Text Corpora



Annotated Text Corpora

Word#	Word	POS	ParseBit	PredLemma	PFID	WS	SA	NE	PredArgs	PredArgs	Coref
0	It	PRP	(TOP(S(NP*)	-	-	-	Speaker#1	*	*	(ARG1*)	(22)
1	is	VBZ	(VP*	-	03	-	Speaker#1	*	(V*)	*	-
2	composed	VBN	(VP*	-	01	2	Speaker#1	*	*	(V*)	-
3	of	IN	(PP*	-	-	-	Speaker#1	*	*	(ARG2*	-
4	a	DT	(NP(NP*	-	-	-	Speaker#1	*	*	*	(24
5	primary	JJ	*	-	-	-	Speaker#1	*	*	*	-
6	stele	NN	*)	-	-	-	Speaker#1	*	*	*	24)
7	,	,	*	-	-	-	Speaker#1	*	*	*	-
8	secondary	JJ	(NP*	-	-	-	Speaker#1	*	*	*	(13
9	steles	NNS	*)	-	-	-	Speaker#1	*	*	*	13)
10	,	,	*	-	-	-	Speaker#1	*	*	*	-
11	a	DT	(NP*	-	-	-	Speaker#1	*	*	*	-
12	huge	JJ	*	-	-	-	Speaker#1	*	*	*	-
13	round	NN	*	-	-	-	Speaker#1	*	*	*	-
14	sculpture	NN	(NML(NML*)	-	-	-	Speaker#1	*	*	*	-
15	and	CC	*	-	-	-	Speaker#1	*	*	*	-
16	beacon	NN	(NML*	-	-	-	Speaker#1	*	*	*	-
17	tower	NN	*)))	-	-	-	Speaker#1	*	*	*	-
18	,	,	*	-	-	-	Speaker#1	*	*	*	-
19	and	CC	*	-	-	-	Speaker#1	*	*	*	-
20	the	DT	(NP*	-	-	-	Speaker#1	(WORK_OF_ART*	*	*	-
21	Great	NNP	*	-	-	-	Speaker#1	*	*	*	-
22	Wall	NNP	*)	-	-	-	Speaker#1	*)	*	*	-
23	,	,	*	-	-	-	Speaker#1	*	*	*	-
24	among	IN	(PP*	-	-	-	Speaker#1	*	*	*	-
25	other	JJ	(NP*	-	-	-	Speaker#1	*	*	*	-
26	things	NNS	*))))))	-	-	-	Speaker#1	*	*	*)	-
27	.	.	*))	-	-	-	Speaker#1	*	*	*	-

Annotated Text Corpora

download required corpus via nltk.download()

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 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

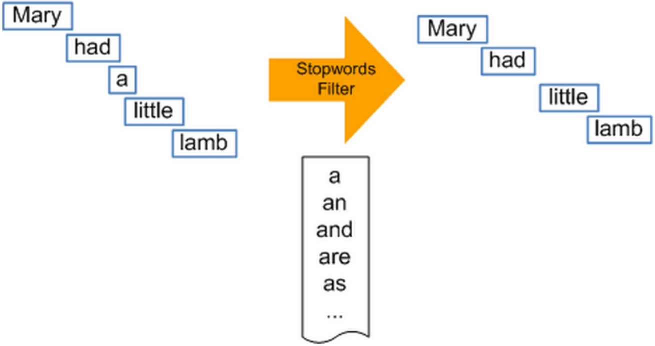
Wordlist Corpora

Word lists are a type of lexical resources. NLTK includes some examples:

- `nltk.corpus.stopwords`
- `nltk.corpus.names`
- `nltk.corpus.swadesh`
- `nltk.corpus.words`

Stopwords

Stopwords are high-frequency words with little lexical content such as **the**, **to**,**and**.



Wordlists: Stopwords

```
>>> from nltk.corpus import stopwords
>>> stopwords.words("english")
["a", "a's", "able", "about", "above", "according", "accordingly",
    "across", "actually", "after", "afterwards", "again", "
   against", "ain't", "all", "allow", "allows", "almost", "alone
   ", "along", "already", "also", "although", "always", ... ]
```

Also available for: Danish, Dutch, English, Finnish, French, German, Hungarian, Italian, Norwegian, Portuguese, Russian, Spanish, Swedish and Turkish

Wordlists: Names

- Names Corpus is a wordlist corpus, containing 8,000 first names categorized by gender.
- The male and female names are stored in separate files.

Wordlists

```
import nltk

names = nltk.corpus.names
print(names.fileids())
# ["female.txt", "male.txt"]

female_names = names.words(names.fileids()[0])
male_names = names.words(names.fileids()[1])

print([w for w in male_names if w in female_names])
#["Abbey", "Abbie", "Abby", "Addie", "Adrian", "Adrien", "Ajay", "
    Alex", "Alexis", "Alfie", "Ali", "Alix", "Allie", "Allyn", "
    Andie", "Andrea", "Andy", "Angel", "Angie", "Ariel", "Ashley
    ", "Aubrey", "Augustine", "Austin", "Averil", ... ]
```

Wordlists

NLP application for which gender information would be helpful

Anaphora Resolution:

Adrian drank from the cup. **He** liked the tea.

Note

Both **he** as well as **she** will be possible solutions when Adrian is the antecedent, since this name occurs in both lists: female and male names.

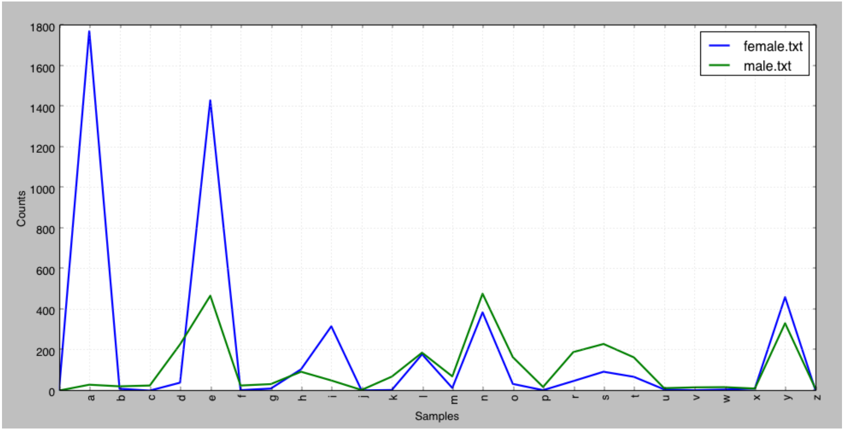
Wordlists

```python
import nltk
names = nltk.corpus.names

cfd = nltk.ConditionalFreqDist(
        (fileid, name[-1])
        for fileid in names.fileids()
        for name in names.words(fileid))
```

???

What will be calculated for the conditional frequency distribution stored in `cfd`?

Wordlists


1800
1600
1400
1200
1000
800
600
400
200
0
Counts
a b c d e f g h i j k l m n o p r s t u v w x y z
Samples
female.txt
male.txt

Wordlists: Swadesh

- comparative wordlist
- lists about 200 common words in several languages.

Comparative Wordlists

```
>>> from nltk.corpus import swadesh
>>> swadesh.fileids()
["be", "bg", "bs", "ca", "cs", "cu", "de", "en", "es", "fr", "hr",
    "it", "la", "mk", "nl", "pl", "pt", "ro", "ru", "sk", "sl",
   "sr", "sw", "uk"]
>>> swadesh.words("en")
["I", "you (singular), thou", "he", "we", "you (plural)", "they",
   "this", "that", "here", "there", "who", "what", "where", "
   when", "how", "not", "all", "many", "some", "few", "other", "
   one", "two", "three", "four", "five", "big", "long", "wide",
    ... ]
```

Comparative Wordlists

```
>>> fr2en = swadesh.entries(["fr", "en"])
>>> fr2en
[("je", "I"), ("tu, vous", "you (singular), thou"), ("il", "he"),
     ... ]
>>> translate = dict(fr2en)
>>> translate["chien"]
"dog"
>>> translate["jeter"]
"throw"
```

Comparative Wordlists

```
>>> de2en = swadesh.entries(["de", "en"]) # German-English
>>> es2en = swadesh.entries(["es", "en"]) # Spanish-English
>>> translate.update(dict(de2en))
>>> translate.update(dict(es2en))
>>> translate["Hund"] "dog"
>>> translate["perro"] "dog"
```

Words Corpus

- NLTK includes some corpora that are nothing more than wordlists.
- The Words Corpus `/usr/share/dict/words` from Unix is used by some spell checkers.

```

def method_x(text):
  text_vocab=set(w.lower() for w in text if w.isalpha())
  english_vocab=set(w.lower() for w in nltk.corpus.words.words())
  x=text_vocab - english_vocab
  return sorted(x)

>>> method_x(nltk.corpus.gutenberg.words('austen-sense.txt'))
```

???

What is calculated in this function?

Words Corpus

```

def method_x(text):
  text_vocab=set(w.lower() for w in text if w.isalpha())
  english_vocab=set(w.lower() for w in nltk.corpus.words.words())
  x= text_vocab - english_vocab
  return sorted(x)

>>> method_x(nltk.corpus.gutenberg.words('austen-sense.txt'))
['abbeyland', 'abhorred', 'abilities', 'abounded', ... ]
```

Preprocessing

Original	The boy’s cars are different colors.
Tokenized	["The", "boy’s", "cars", "are", "different", "colors."]
Punctuation removal	["The", "boy’s", "cars", "are", "different", "colors"]
Lowecased	["the", "boy’s", "cars", "are", "different", "colors"]
Stemmed	["the", "boy’s", "car", "are", "differ", "color"]
Lemmatized	["the", "boy’s", "car", "are", "different", "color"]
Stopword removal	["boy’s", "car", "different", "color"]

Tokenization

- Tokenization is the process of breaking raw text into its building parts: words, phrases, symbols, or other meaningful elements called tokens.
- A list of tokens is almost always the first step to any other NLP task, such as part-of-speech tagging and named entity recognition.

Tokenization

- **token** – is an instance of a sequence of characters in some particular document that are grouped together as a useful semantic unit for processing
- **type** – is the class of all tokens containing the same character sequence

Tokenization

- What is Token?
- Fairly trivial: chop on whitespace and throw away punctuation characters.
- Tricky cases: various uses of the apostrophe for possession and contractions?

Tokenization

`Mrs. O'Reilly said that the girls' bags from H&M's shop in New York aren't cheap.`

Mrs.	"Mrs."; "Mrs", "."
O'Reilly	"O'Reilly"; "OReilly"; "O'", "Reilly"; "O", "'", "Reilly";
aren't	"aren't"; "arent"; "are", "n't"; "aren", "t"
...	...

Tokenization

???

Tokenize manually the following sentence. How many tokens do you get?

`Mrs. O'Reilly said that the girls' bags from H&M's shop in New York aren't cheap.`

Tokenization

???

Tokenize manually the following sentence:

`Mrs. O'Reilly said that the girls' bags from H&M's shop in New York aren't cheap.`

Answer

NLTK returns the following 20 tokens:

```
["Mrs.", "O'Reilly", "said", "that", "the",
"girls", "'", "bags", "from", "H", "&", "M",
"'s", "shop", "in", "New", "York", "are",
"n't", "cheap", "."]
```

Tokenization

Most decisions need to be met depending on the language at hand. Some problematic cases for English include:

- hyphenation – *ex-wife*, *Cooper-Hofstadter*, *the bazinga-him-again maneuver*
- internal white spaces – *New York*, *+49 89 21809719*, *January 1, 1995*, *San Francisco-Los Angeles*
- apostrophe – *O'Reilly*, *aren't*
- other cases – *H&M's*

Sentence Segmentation

Tokenization can be approached at any level:

- word segmentation
- sentence segmentation
- paragraph segmentation
- other elements of the text

Segmentation

NLTK comes with a whole bunch of tokenization possibilities:

```
>>> from nltk import word_tokenize ,
    wordpunct_tokenize
>>> s = "Good muffins cost $3.88\nin New York.
    Please buy me\n two of them.\n\nThanks."
>>> word_tokenize(s)
['Good', 'muffins', 'cost', '$', '3.88', 'in', 'New',
     'York', '.', 'Please', 'buy', 'me', 'two', 'of',
     'them', '.', 'Thanks', '.']
>>> wordpunct_tokenize(s)
['Good', 'muffins', 'cost', '$', '3', '.', '88', 'in'
    , 'New', 'York', '.', 'Please', 'buy', 'me', 'two
    ', 'of', 'them', '.', 'Thanks', '.']
```

Segmentation

NLTK comes with a whole bunch of tokenization possibilities:

```
>>> from nltk.tokenize import *
>>> # same as s.split():
>>> WhitespaceTokenizer().tokenize(s)
['Good', 'muffins', 'cost', '$3.88', 'in', 'New', '
    York.', 'Please', 'buy', 'me', 'two', 'of', '
    them.', 'Thanks.']
>>> # same as s.split(" "):
>>> SpaceTokenizer().tokenize(s)
['Good', 'muffins', 'cost', '$3.88\nin', 'New', 'York
    .', '', 'Please', 'buy', 'me\ntwo', 'of', 'them
    .\n\nThanks.']
```

Segmentation

NLTK comes with a whole bunch of tokenization possibilities:

```
>>> # same as s.split('\n'):
>>> LineTokenizer(blanklines='keep').tokenize(s)
['Good muffins cost $3.88', 'in New York.  Please buy
    me', 'two of them.', '', 'Thanks.']
>>> # same as [l for l in s.split('\n') if l.strip()
    ]:
>>> LineTokenizer(blanklines='discard').tokenize(s)
['Good muffins cost $3.88', 'in New York.  Please buy
    me', 'two of them.', 'Thanks.']
 >>> # same as s.split('\t'):
>>> TabTokenizer().tokenize('a\tb c\n\t d')
['a', 'b c\n', ' d']
```

Segmentation

NLTK PunktSentenceTokenizer: divides a text into a list of sentences

```
>>> import nltk.data
>>> text = "Punkt knows that the periods in Mr. Smith and Johann S. Bach do not mark sentence boundaries.  And sometimes sentences can start with non-capitalized words.  i is a good variable name."
>>> sent_detector = nltk.data.load('tokenizers/punkt/english.pickle')
>>> print '\n-----\n'.join(sent_detector.tokenize(text.strip()))
# Punkt knows that the periods in Mr. Smith and Johann S. Bach do not mark sentence boundaries.
# -----
# And sometimes sentences can start with non-capitalized words.
# -----
# i is a good variable name.
```

Normalization

Once the text has been segmented into its tokens (paragraphs, sentences, words), most NLP pipelines do a number of other basic procedures for text normalization, e.g.:

- lowercasing
- stemming
- lemmatization
- stopword removal

Lowercasing

Lowercasing:

```python
import nltk

string = "The boy's cars are different colors."
tokens = nltk.word_tokenize(string)
lower = [x.lower() for x in tokens]
print(" ".join(lower))

# prints
# the boy 's cars are different colors .
```

- Often, however, instead of working with all word forms, we would like to extract and work with their base forms (e.g. lemmas or stems)
- Thus with **stemming** and **lemmatization** we aim to reduce inflectional (and sometimes derivational) forms to their base forms.

Stemming

Stemming: removing morphological affixes from words, leaving only the word stem.

```python
import nltk

string = "The boy's cars are different colors."
tokens = nltk.word_tokenize(string)
lower = [x.lower() for x in tokens]
stemmed = [stem(x) for x in lower]
print(" ".join(stemmed))

def stem(word):
    for suffix in ["ing", "ly", "ed", "ious", "ies", "ive", "es",
            "s", "ment"]:
        if word.endswith(suffix):
            return word[:-len(suffix)]
    return word
# prints
# the boy 's car are different color .
```

Stemming

Stemming:

```python
import nltk
import re

string = "The boy's cars are different colors."
tokens = nltk.word_tokenize(string)
lower = [x.lower() for x in tokens]
stemmed = [stem(x) for x in lower]
print(" ".join(stemmed))

def stem(word):
    regexp = r"^(.*?)(ing|ly|ed|ious|ies|ive|es|s|ment)?$"
    stem, suffix = re.findall(regexp, word)[0]
    return stem

# prints
# the boy 's car are different color .
```

Stemming

NLTK's stemmers:

- **Porter Stemmer** is the oldest stemming algorithm supported in NLTK, originally published in 1979.
  `http://www.tartarus.org/~martin/PorterStemmer/`
- **Lancaster Stemmer** is much newer, published in 1990, and is more aggressive than the Porter stemming algorithm.
- **Snowball stemmer** currently supports several languages: Danish, Dutch, English, Finnish, French, German, Hungarian, Italian, Norwegian, Porter, Portuguese, Romanian, Russian, Spanish, Swedish.
- **Snowball stemmer**: slightly faster computation time than porter.

Stemming

NLTK's stemmers:

```
import nltk

string = "The boy's cars are different colors."
tokens = nltk.word_tokenize(string)
lower = [x.lower() for x in tokens]

porter = nltk.PorterStemmer()
stemmed = [porter.stem(t) for t in lower]
print(" ".join(stemmed))
# prints
# the boy 's car are differ color .

lancaster = nltk.LancasterStemmer()
stemmed = [lancaster.stem(t) for t in lower]
print(" ".join(stemmed))
# prints
# the boy 's car ar diff col .
```

Stemming

NLTK's stemmers:

```python
import nltk

string = "The boy's cars are different colors."
tokens = nltk.word_tokenize(string)
lower = [x.lower() for x in tokens]

snowball = nltk.SnowballStemmer("english")
stemmed = [snowball.stem(t) for t in lower]
print(" ".join(stemmed))
# prints
# the boy 's car are differ color .
```

Lemmatization

- stemming can often create non-existent words, whereas lemmas are actual words
- **NLTK WordNet Lemmatizer** uses the WordNet Database to lookup lemmas

```
import nltk
string = "The boy's cars are different colors."
tokens = nltk.word_tokenize(string)
lower = [x.lower() for x in tokens]
porter = nltk.PorterStemmer()
stemmed = [porter.stem(t) for t in lower]
print(" ".join(lemmatized))
# prints the boy 's car are differ color .
wnl = nltk.WordNetLemmatizer()
lemmatized = [wnl.lemmatize(t) for t in lower]
print(" ".join(lemmatized))
# prints the boy 's car are different color .
```

Stopword removal:

Stopword removal:

```
import nltk

string = "The boy's cars are different colors."
tokens = nltk.word_tokenize(string)
lower = [x.lower() for x in tokens]
wnl = nltk.WordNetLemmatizer()
lemmatized = [wnl.lemmatize(t) for t in lower]

content = [x for x in lemmatized if x not in nltk.
    corpus.stopwords.words("english")]
print(" ".join(content))
# prints
# boy 's car different color .
```

spaCy


spaCy
USAGE
MODELS
API
UNIVERSE
Industrial-Strength Natural Language Processing
IN PYTHON
Fastest in the world
spaCy excels at large-scale information extraction tasks. It's written from the ground up in carefully memory-managed Cython. Independent research has confirmed that spaCy is the fastest in the world. If your application needs to process entire web dumps, spaCy is the
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
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

spaCy

- a free, open-source library for advanced **Natural Language Processing** (NLP) in Python
- you can build applications that process and "understand" large volumes of text:
  - information extraction systems
  - natural language understanding systems
  - **pre-process text**
  - deep learning
- **NLTK** - a platform for teaching and research
- **spaCY** - designed specifically for **production use**

spaCy

ADAM: Question Answering System

A question answering system that extracts answers from Wikipedia to questions posed in natural language.

AllenNLP

An open-source NLP research library, built on PyTorch and spaCy

CleanNLP

A tidy data model for NLP in R

displaCy

A modern syntactic dependency visualizer

displaCy ENT

A modern named entity visualizer

Dragonfire

An open-source virtual assistant for Ubuntu based Linux distributions

EpiTator

Extracts case counts, resolved location/species/disease names, date ranges and more

ExcelCy

Excel Integration with spaCy. Training NER using XLSX from PDF, DOCX, PPT, PNG or JPG.

spaCy Features

NAME	DESCRIPTION
Tokenization	Segmenting text into words, punctuations marks etc.
Part-of-speech (POS) **Tagging**	Assigning word types to tokens, like verb or noun.
Dependency Parsing	Assigning syntactic dependency labels, describing the relations between individual tokens, like subject or object.
Lemmatization	Assigning the base forms of words. For example, the lemma of "was" is "be", and the lemma of "rats" is "rat".
Sentence Boundary Detection (SBD)	Finding and segmenting individual sentences.
Named Entity Recognition (NER)	Labelling named "real-world" objects, like persons, companies or locations.
Similarity	Comparing words, text spans and documents and how similar they are to each other.
Text Classification	Assigning categories or labels to a whole document, or parts of a document.
Rule-based Matching	Finding sequences of tokens based on their texts and linguistic annotations, similar to regular expressions.

Statistical models

- some of spaCy's features require **statistical models** to be loaded (e.g. to predict whether a word is a verb or a noun)
- spaCy currently offers statistical models for **8 languages**
- Components:
  - **Binary weights** for the part-of-speech tagger, dependency parser and named entity recognizer
  - **Lexical entries** in the vocabulary, i.e. words and e.g. their shape or spelling.
  - **Word vectors**, i.e. multi-dimensional meaning representations of words
  - **Configuration** options, like the language and processing pipeline settings

Linguistic annotations

spaCy provides a variety of linguistic annotations to give you **insights into a text's grammatical structure.**

```
import spacy

nlp = spacy.load('en_core_web_sm')
doc = nlp(u'Apple is looking at buying U.K. startup for $1
    billion')
for token in doc:
    print(token.text, token.pos_, token.dep_)

>>> Apple PROPN nsubj
    is VERB aux
    looking VERB ROOT
    at ADP prep
    buying VERB pcomp
     ...
```

Part-of-speech tags and dependencies

```
import spacy

nlp = spacy.load('en_core_web_sm')
doc = nlp(u'Apple is looking at buying U.K. startup for $1 billion
    ')

for token in doc:
    print(token.text, token.lemma_, token.pos_, token.tag_,
        token.dep_, token.shape_, token.is_alpha, token.is_stop)
>>>
Apple apple PROPN NNP nsubj Xxxxx True False
is be VERB VBZ aux xx True True
looking look VERB VBG ROOT xxxx True False
at at ADP IN prep xx True True
buying buy VERB VBG pcomp xxxx True False
U.K. u.k. PROPN NNP compound X.X. False False
startup startup NOUN NN dobj xxxx True False
for for ADP IN prep xxx True True
$ $ SYM $ quantmod $ False False
1 1 NUM CD compound d False False
billion billion NUM CD pobj xxxx True False
```

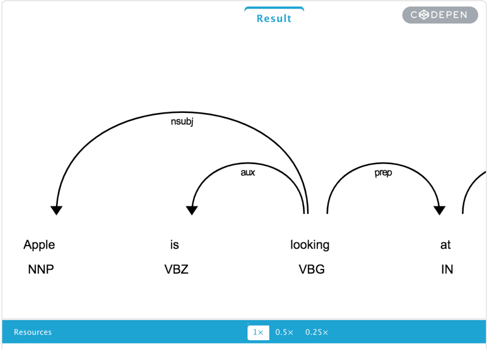
Part-of-speech tags and dependencies

TEXT	LEMMA	POS	TAG	DEP	SHAPE	ALPHA	STOP
Apple	apple	PROPN	NNP	nsubj	Xxxxx	True	False
is	be	VERB	VBZ	aux	xx	True	True
looking	look	VERB	VBG	ROOT	xxxx	True	False
at	at	ADP	IN	prep	xx	True	True
buying	buy	VERB	VBG	pcomp	xxxx	True	False
U.K.	u.k.	PROPN	NNP	compound	X.X.	False	False
startup	startup	NOUN	NN	dobj	xxxx	True	False
for	for	ADP	IN	prep	xxx	True	True
$	$	SYM	$	quantmod	$	False	False
1	1	NUM	CD	compound	d	False	False
billion	billion	NUM	CD	pobj	xxxx	True	False

Part-of-speech tags and dependencies

Use built-in **displaCy visualizer**

```python
from spacy import displacy
displacy.serve(doc, style='dep')
```



Named Entities

```
import spacy

nlp = spacy.load('en_core_web_sm')
doc = nlp(u'Apple is looking at buying U.K. startup for $1 billion
    ')

for ent in doc.ents:
    print(ent.text, ent.start_char, ent.end_char, ent.label_)
>>>
Apple 0 5 ORG
U.K. 27 31 GPE
$1 billion 44 54 MONEY
```

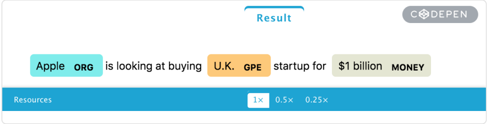
Named Entities

TEXT	START	END	LABEL	DESCRIPTION
Apple	`0`	`5`	`ORG`	Companies, agencies, institutions.
U.K.	`27`	`31`	`GPE`	Geopolitical entity, i.e. countries, cities, states.
$1 billion	`44`	`54`	`MONEY`	Monetary values, including unit.

Named Entities

Use built-in **displaCy visualizer**

```
from spacy import displacy
displacy.serve(doc, style='ent')
```



Word vectors and similarity

```
import spacy

nlp = spacy.load('en_core_web_md')  # make sure to use larger
    model! python -m spacy download en_core_web_md
tokens = nlp(u'dog cat banana')

for token1 in tokens:
    for token2 in tokens:
        print(token1.text, token2.text, token1.similarity(token2))
>>>
dog dog 1.0
dog cat 0.80168545
dog banana 0.24327643
cat dog 0.80168545
cat cat 1.0
cat banana 0.28154364
banana dog 0.24327643
banana cat 0.28154364
banana banana 1.0
```

Word vectors and similarity

	DOG	CAT	BANANA
DOG	1.00	0.80	0.24
CAT	0.80	1.00	0.28
BANANA	0.24	0.28	1.00

BANANA.VECTOR

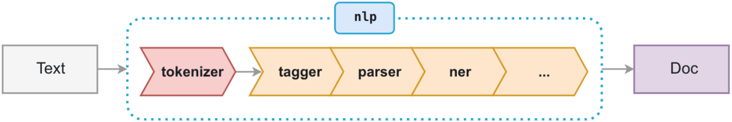
```
array([2.02280000e-01, -7.66180009e-02,  3.70319992e-01,
        3.28450017e-02, -4.19569999e-01,  7.20689967e-02,
       -3.74760002e-01,  5.74599989e-02, -1.24009997e-02,
        5.29489994e-01, -5.23800015e-01, -1.97710007e-01,
       -3.41470003e-01,  5.33169985e-01, -2.53309999e-02,
        1.73800007e-01,  1.67720005e-01,  8.39839995e-01,
        5.51070012e-02,  1.05470002e-01,  3.78719985e-01,
        2.42750004e-01,  1.47449998e-02,  5.59509993e-01,
        1.25210002e-01, -6.75960004e-01,  3.58420014e-01,
```

Word vectors and similarity

```
import spacy

nlp = spacy.load('en_core_web_md')
tokens = nlp(u'dog cat banana afskfsd')

for token in tokens:
    print(token.text, token.has_vector, token.vector_norm,
token.is_oov)
>>>
dog True 7.0336733 False
cat True 6.6808186 False
banana True 6.700014 False
afskfsd False 0.0 True
```

en_vectors_web_lg includes over 1 million unique vectors

Pipelines



"pipeline": ["tagger", "parser", "ner"]

Vocab, hashes and lexemes

- spaCy stores data in a vocabulary **Vocab**
- encodes all string to hash values
- uses **lookup table that works in both directions** - you can look up a string to get its hash, or a hash to get its string

```
import spacy

nlp = spacy.load('en_core_web_sm')
doc = nlp(u'I love coffee')
print(doc.vocab.strings[u'coffee'])  # 3197928453018144401
print(doc.vocab.strings[3197928453018144401])  # 'coffee'
>>>
3197928453018144401
coffee
```

Vocab, hashes and lexemes

- Each entry in the vocabulary is called **Lexeme**

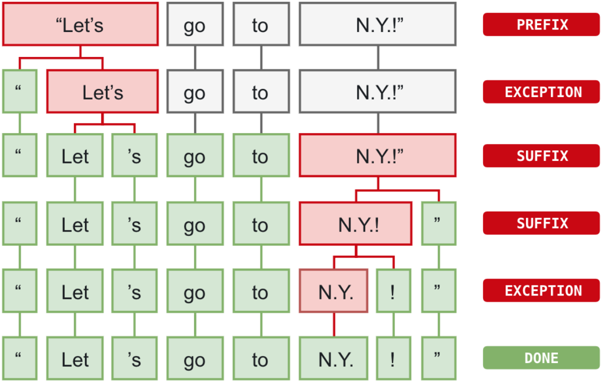
```
import spacy

nlp = spacy.load('en_core_web_sm')
doc = nlp(u'I love coffee')
for word in doc:
    lexeme = doc.vocab[word.text]
    print(lexeme.text, lexeme.orth, lexeme.shape_, lexeme.prefix_,
          lexeme.suffix_, lexeme.is_alpha, lexeme.is_digit,
           lexeme.is_title, lexeme.lang_)
>>>I 4690420944186131903 X I I True False True en
love 3702023516439754181 xxxx l ove True False False en
coffee 3197928453018144401 xxxx c fee True False False en
```

Tokenization with spaCy

```
import spacy

nlp = spacy.load('en_core_web_sm')
doc = nlp(u'Apple is looking at buying U.K. startup for $1 billion
    ')
for token in doc:
    print(token.text)
>>>
    Apple
    is
    looking
    at
    buying
    U.K.
     ...
```

Tokenization with spaCy

- **Does the substring match a tokenizer exception rule?** (U.K.)
- **Can a prefix, suffix or infix be split off?** (e.g. punctuation)



Tokenization with spaCy

- Tokenizer exceptions strongly depend on the specifics of the individual language
- Global and language-specific tokenizer data is supplied via the language data in **spacy/lang**

LANGUAGE	CODE	LANGUAGE DATA	MODELS
English	`en`	`lang/en` </>	4 models
German	`de`	`lang/de` </>	1 model
Spanish	`es`	`lang/es` </>	2 models
Portuguese	`pt`	`lang/pt` </>	1 model
French	`fr`	`lang/fr` </>	2 models
Italian	`it`	`lang/it` </>	1 model

Adding special case tokenization rules

- The tokenizer exceptions define special cases like "don’t" in English, which needs to be split into two tokens: {ORTH: do} and {ORTH: n’t, LEMMA: not}
- Later: also possibility to customize spaCy Tokenizer class

```python
import spacy
from spacy.symbols import ORTH, LEMMA, POS, TAG

# add special case rule
special_case = [{ORTH: u'gim', LEMMA: u'give', POS: u'VERB'},
                {ORTH: u'me'}]
nlp.tokenizer.add_special_case(u'gimme', special_case)

# check new tokenization
print([w.text for w in nlp(u'gimme that')])#['gim','me','that']
```

Other Tokenizers

Tweet NLP

Carnegie Mellon

We provide a tokenizer, a part-of-speech tagger, hierarchical word clusters, and a dependency parser for tweets, along with annotated corpora and web-based annotation tools.

Contributors: Archna Bhatia, Dipanjan Das, Chris Dyer, Jacob Eisenstein, Jeffrey Flanigan, Kevin Gimpel, Michael Heilman, Lingpeng Kong, Daniel Mills, Brendan O'Connor, Olutobi Owoputi, Nathan Schneider, Noah Smith, Swabha Swayamdipta and Dani Yogatama.

Quick Links

- Part-of-speech Tagger and POS annotated data - also Twokenizer: tokenizer software (part of tagger package) and Tagging Models -- Download Link
- Tweeboparser and Tweebank: Dependency parser software and dependency annotated data -- Download Link
- Documentation, annotation guidelines, and papers describing this work
- Hierarchical Twitter Word Clusters

References

- http://www.nltk.org/book/
- https://github.com/nltk/nltk
- https://spacy.io/