textacy Documentation

Release 0.4.1

Burton DeWilde

Aug 10, 2017

Contents

`textacy` is a Python library for performing higher-level natural language processing (NLP) tasks, built on the high-performance spaCy library. With the basics — tokenization, part-of-speech tagging, dependency parsing, etc. — offloaded to another library, `textacy` focuses on tasks facilitated by the ready availability of tokenized, POS-tagged, and parsed text.

CHAPTER 1

Features

- Stream text, json, csv, and spaCy binary data to and from disk
- Clean and normalize raw text, *before* analyzing it
- Explore a variety of included datasets, with both text data and metadata from Congressional speeches to historical literature to Reddit comments
- Access and filter basic linguistic elements, such as words and ngrams, noun chunks and sentences
- Extract named entities, acronyms and their definitions, direct quotations, key terms, and more from documents
- Compare strings, sets, and documents by a variety of similarity metrics
- Transform documents and corpora into vectorized and semantic network representations
- Train, interpret, visualize, and save `sklearn`-style topic models using LSA, LDA, or NMF methods
- Identify a text's language, display key words in context (KWIC), true-case words, and navigate a parse tree

... and more!

CHAPTER 2

Project Links

- textacy @ PyPi
- textacy @ GitHub
- textacy @ ReadTheDocs

CHAPTER 3

Installation

The simple way to install `textacy` is via `pip`:

```
$ pip install textacy
```

or `conda`:

```
$ conda install -c conda-forge textacy
```

Note: If you use `pip`, some dependencies have been made optional, because they can be difficult to install and/or are only needed in certain uses cases. To use visualization functions, you'll need `matplotlib` installed; you can do so via `pip install textacy[viz]`. For automatic language detection, you'll need `cld2-cffi` installed; do `pip install textacy[lang]`. To install all optional dependencies:

```
$ pip install textacy[all]
```

Otherwise, you can download and unzip the source `tar.gz` from PyPi, then install manually:

```
$ python setup.py install
```

spaCy Language Data

For most uses of `textacy`, language-specific model data for `spacy` must first be downloaded. Follow the directions here.

Currently available language models are listed here.

CHAPTER 4

Usage Example

```
>>> import textacy
>>> import textacy.datasets
```

Efficiently stream documents from disk and into a processed corpus:

```
>>> cw = textacy.datasets.CapitolWords()
>>> records = cw.records(speaker_name={'Hillary Clinton', 'Barack Obama'})
>>> text_stream, metadata_stream = textacy.fileio.split_record_fields(
...     records, 'text')
>>> corpus = textacy.Corpus('en', texts=text_stream, metadatas=metadata_stream)
>>> corpus
Corpus(1241 docs; 857058 tokens)
```

Represent corpus as a document-term matrix, with flexible weighting and filtering:

```
>>> vectorizer = textacy.Vectorizer(
...     weighting='tfidf', normalize=True, smooth_idf=True,
...     min_df=2, max_df=0.95)
>>> doc_term_matrix = vectorizer.fit_transform(
...     (doc.to_terms_list(ngrams=1, named_entities=True, as_strings=True)
...      for doc in corpus))
>>> print(repr(doc_term_matrix))
<1241x11708 sparse matrix of type '<class 'numpy.float64'>'
    with 215182 stored elements in Compressed Sparse Row format>
```

Train and interpret a topic model:

```
>>> model = textacy.TopicModel('nmf', n_topics=10)
>>> model.fit(doc_term_matrix)
>>> doc_topic_matrix = model.transform(doc_term_matrix)
>>> doc_topic_matrix.shape
(1241, 10)
>>> for topic_idx, top_terms in model.top_topic_terms(vectorizer.id_to_term, top_
↪n=10):
...     print('topic', topic_idx, ':', '   '.join(top_terms))
```

```
topic 0 : new   people   's   american   senate   need   iraq   york   americans
↪work
topic 1 : rescind   quorum   order   consent   unanimous   ask   president   mr.
↪madam   aside
topic 2 : dispense   reading   amendment   unanimous   consent   ask   president   mr.
↪   pending   aside
topic 3 : health   care   child   mental   quality   patient   medical   program
↪information   family
topic 4 : student   school   education   college   child   teacher   high   program
↪loan   year
topic 5 : senators   desiring   chamber   vote   4,600   amtrak   rail   airline
↪litigation   expedited
topic 6 : senate   thursday   wednesday   session   unanimous   consent   authorize
↪p.m.   committee   ask
topic 7 : medicare   drug   senior   medicaid   prescription   benefit   plan   cut
↪cost   fda
topic 8 : flu   vaccine   avian   pandemic   roberts   influenza   seasonal
↪outbreak   health   cdc
topic 9 : virginia   west virginia   west   senator   yield   question   thank
↪objection   inquiry   massachusetts
```

Basic indexing as well as flexible selection of documents in a corpus:

```
>>> obama_docs = list(corpus.get(
...     lambda doc: doc.metadata['speaker_name'] == 'Barack Obama'))
>>> len(obama_docs)
411
>>> doc = corpus[-1]
>>> doc
Doc(2999 tokens; "In the Federalist Papers, we often hear the ref...")
```

Preprocess plain text, or highlight particular terms in it:

```
>>> textacy.preprocess_text(doc.text, lowercase=True, no_punct=True)[:70]
'in the federalist papers we often hear the reference to the senates ro'
>>> textacy.text_utils.keyword_in_context(doc.text, 'America', window_width=35)
g on this tiny piece of Senate and  America n history. Some 10 years ago, I ask
o do the hard work in New York and  America , who get up every day and do the v
say: You know, you never can count  America  out. Whenever the chips are down,
 what we know will give our fellow  America ns a better shot at the kind of fut
aith in this body and in my fellow  America ns. I remain an optimist, that Amer
ricans. I remain an optimist, that  America 's best days are still ahead of us.
```

Extract various elements of interest from parsed documents:

```
>>> list(textacy.extract.ngrams(
...     doc, 2, filter_stops=True, filter_punct=True, filter_nums=False))[:15]
[Federalist Papers,
 Senate's,
 's role,
 violent passions,
 pernicious resolutions,
 everlasting credit,
 common ground,
 8 years,
 tiny piece,
 American history,
 10 years,
```

```
 years ago,
 New York,
 fellow New,
 New Yorkers]
>>> list(textacy.extract.ngrams(
...     doc, 3, filter_stops=True, filter_punct=True, min_freq=2))
[fellow New Yorkers,
 World Trade Center,
 Senator from New,
 World Trade Center,
 Senator from New,
 lot of fun,
 fellow New Yorkers,
 lot of fun]
>>> list(textacy.extract.named_entities(
...     doc, drop_determiners=True, exclude_types='numeric'))[:10]
[Senate,
 Senate,
 American,
 New York,
 New Yorkers,
 Senate,
 Barbara Mikulski,
 Senate,
 Pennsylvania Avenue,
 Senate]
>>> pattern = textacy.constants.POS_REGEX_PATTERNS['en']['NP']
>>> pattern
<DET>? <NUM>* (<ADJ> <PUNCT>? <CONJ>?)* (<NOUN>|<PROPN> <PART>?)+
>>> list(textacy.extract.pos_regex_matches(doc, pattern))[:10]
[the Federalist Papers,
 the reference,
 the Senate's role,
 the consequences,
 sudden and violent passions,
 intemperate and pernicious resolutions,
 the everlasting credit,
 wisdom,
 our Founders,
 an effort]
>>> list(textacy.extract.semistructured_statements(doc, 'I', cue='be'))
[(I, was, on the other end of Pennsylvania Avenue),
 (I, was, , a very new Senator, and my city and my State had been devastated),
 (I, am, grateful to have had Senator Schumer as my partner and my ally),
 (I, am, very excited about what can happen in the next 4 years),
 (I, been, a New Yorker, but I know I always will be one)]
>>> textacy.keyterms.textrank(doc, n_keyterms=10)
[('day', 0.016085082758778 94),
 ('people', 0.015079868730811194),
 ('year', 0.012330783590843065),
 ('way', 0.011732786337383587),
 ('colleague', 0.010794482493897155),
 ('new', 0.0104941198408241),
 ('time', 0.010016582029543003),
 ('work', 0.0096498231660789),
 ('lot', 0.008960478625039818),
 ('great', 0.008552318032915361)]
```

Compute basic counts and readability statistics for a given text:

```
>>> ts = textacy.TextStats(doc)
>>> ts.n_unique_words
1107
>>> ts.basic_counts
{'n_chars': 11498,
 'n_long_words': 512,
 'n_monosyllable_words': 1785,
 'n_polysyllable_words': 222,
 'n_sents': 99,
 'n_syllables': 3525,
 'n_unique_words': 1107,
 'n_words': 2516}
>>> ts.flesch_kincaid_grade_level
10.853709110179697
>>> ts.readability_stats
{'automated_readability_index': 12.801546064781363,
 'coleman_liau_index': 9.905629258346586,
 'flesch_kincaid_grade_level': 10.853709110179697,
 'flesch_readability_ease': 62.51222198133965,
 'gulpease_index': 55.10492845786963,
 'gunning_fog_index': 13.69506833036245,
 'lix': 45.76390294037353,
 'smog_index': 11.683781121521076,
 'wiener_sachtextformel': 5.401029023140788}
```

Count terms individually, and represent documents as a bag-of-terms with flexible weighting and inclusion criteria:

```
>>> doc.count('America')
3
>>> bot = doc.to_bag_of_terms(ngrams={2, 3}, as_strings=True)
>>> sorted(bot.items(), key=lambda x: x[1], reverse=True)[:10]
[('new york', 18),
 ('senate', 8),
 ('first', 6),
 ('state', 4),
 ('9/11', 3),
 ('look forward', 3),
 ('america', 3),
 ('new yorkers', 3),
 ('chuck', 3),
 ('lot of fun', 2)]
```

Note: In almost all cases, `textacy` expects to be working with unicode text. Docstrings indicate this as `str`, which is clear and correct for Python 3 but not Python 2. In the latter case, users should cast `str` bytes to `unicode`, as needed.

CHAPTER 5

Authors

- Burton DeWilde (<burton@chartbeat.net>)

CHAPTER 6

Table of Contents

License

```
Copyright 2016 Chartbeat, Inc.

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

  http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
```

API Reference

Doc and Corpus

Load, process, iterate, transform, and save text content paired with metadata — a document.

class `textacy.doc.`**Doc**(*content*, *metadata=None*, *lang=<function detect_language>*)

A text document parsed by spaCy and, optionally, paired with key metadata. Transform `Doc` into an easily-customized list of terms, a bag-of-words or (more general) bag-of-terms, or a semantic network; save and load parsed content and metadata to and from disk; index, slice, and iterate through tokens and sentences; and more.

Initialize from a text and (optional) metadata:

```
>>> content = '''
...     The apparent symmetry between the quark and lepton families of
```

```
...     the Standard Model (SM) are, at the very least, suggestive of
...     a more fundamental relationship between them. In some Beyond the
...     Standard Model theories, such interactions are mediated by
...     leptoquarks (LQs): hypothetical color-triplet bosons with both
...     lepton and baryon number and fractional electric charge.'''
>>> metadata = {
...     'title': 'A Search for 2nd-generation Leptoquarks at s = 7 TeV',
...     'author': 'Burton DeWilde',
...     'pub_date': '2012-08-01'}
>>> doc = textacy.Doc(content, metadata=metadata)
>>> print(doc)
Doc(71 tokens; "The apparent symmetry between the quark and lep...")
```

Transform into other, common formats:

```
>>> doc.to_bag_of_words(lemmatize=False, as_strings=False)
{205123: 1, 21382: 1, 17929: 1, 175499: 2, 396: 1, 29774: 1, 27472: 1,
 4498: 1, 1814: 1, 1176: 1, 49050: 1, 287836: 1, 1510365: 1, 6239: 2,
 3553: 1, 5607: 1, 4776: 1, 49580: 1, 6701: 1, 12078: 2, 63216: 1,
 6738: 1, 83061: 1, 5243: 1, 1599: 1}
>>> doc.to_bag_of_terms(ngrams=2, named_entities=True,
...                     lemmatize=True, as_strings=True)
{'apparent symmetry': 1, 'baryon number': 1, 'electric charge': 1,
 'fractional electric': 1, 'fundamental relationship': 1,
 'hypothetical color': 1, 'lepton family': 1, 'model theory': 1,
 'standard model': 2, 'triplet boson': 1}
```

Doc as sequence of tokens, emulating spaCy's "sequence API":

```
>>> doc[49]  # spacy.Token
leptoquarks
>>> doc[:3]  # spacy.Span
The apparent symmetry
```

Save to and load from disk:

```
>>> doc.save('~/Desktop', name='leptoquarks')
>>> doc = textacy.Doc.load('~/Desktop', name='leptoquarks')
>>> print(doc)
Doc(71 tokens; "The apparent symmetry between the quark and lep...")
```

Parameters

- **content** (str or `spacy.Doc`) – Document content as (unicode) text or an already-parsed `spacy.Doc`. If str, content is processed by models loaded with a `spacy.Language` and assigned to *spacy_doc*.
- **metadata** (*dict*) – Dictionary of relevant information about content. This can be helpful when identifying and filtering documents, as well as when engineering features for model inputs.
- **lang** (str or `spacy.Language` or callable) – Language of document content. If known, pass a standard 2-letter language code (e.g. "en"), or the name of a spacy model for the desired language (e.g. "en_core_web_md"), or an already-instantiated `spacy.Language` object. If not known, pass a function/callable that takes unicode text as input and outputs a standard 2-letter language code.

The given or detected language str is used to instantiate a corresponding `spacy.Language` with all models loaded by default, and the appropriate 2-letter lang code is assigned to *`Doc.lang`*.

Note: The `spacy.Language` object parses `content` (if str) and sets the *`spacy_vocab`* and *`spacy_stringstore`* attributes. See https://spacy.io/docs/usage/models#available for available spacy models.

lang
str – 2-letter code for language of `Doc`.

metadata
dict – Dictionary of relevant information about content.

spacy_doc
`spacy.Doc` – https://spacy.io/docs#doc

spacy_vocab
`spacy.Vocab` – https://spacy.io/docs#vocab

spacy_stringstore
`spacy.StringStore` – https://spacy.io/docs#stringstore

count(*term*)
Get the number of occurrences (i.e. count) of `term` in `Doc`.

Parameters **term** (str or int or `spacy.Token` or `spacy.Span`) – The term to be counted can be given as a string, a unique integer id, a spacy token, or a spacy span. Counts for the same term given in different forms are the same!

Returns Count of `term` in `Doc`.

Return type int

Tip: Counts are cached. The first time a single word's count is looked up, *all* words' counts are saved, resulting in a slower runtime the first time but orders of magnitude faster runtime for subsequent calls for this or any other word. Similarly, if a bigram's count is looked up, all bigrams' counts are stored — etc. If spans are merged using *`Doc.merge()`*, all cached counts are deleted, since merging spans will invalidate many counts. Better to merge first, count second!

---

classmethod load(*path*, *name=None*)
Load content and metadata from disk, and initialize a `Doc`.

Parameters

- **path** (*`str`*) – Directory on disk where content and metadata are saved.
- **name** (*`str`*) – Identifying/uniquifying name prepended to the default filenames 'spacy_doc.bin' and 'metadata.json', used when doc was saved to disk via *`Doc.save()`*.

Returns *`textacy.Doc`*

Warning: If the `spacy.Vocab` object used to save this document is not the same as the one used to load it, there will be problems! Consequently, this functionality is only useful as short-term but not long-term storage.

merge(*spans*)
Merge spans *in-place* within `Doc` so that each takes up a single token. Note: All cached counts on this doc are cleared after a merge.

Parameters **spans** (Iterable[`spacy.Span`]) – for example, the results from *`extract.named_entities()`* or *`extract.pos_regex_matches()`*

n_sents
The number of sentences in the document.

n_tokens
The number of tokens in the document — including punctuation.

pos_tagged_text
Return text as an ordered, nested list of (token, POS) pairs per sentence.

save(*path*, *name=None*)
Save `Doc` content and metadata to disk.

Parameters

- **path** (*`str`*) – Directory on disk where content + metadata will be saved.
- **name** (*`str`*) – Prepend default filenames 'spacy_doc.bin' and 'metadata.json' with a name to identify/uniquify this particular document.

Warning: If the `spacy.Vocab` object used to save this document is not the same as the one used to load it, there will be problems! Consequently, this functionality is only useful as short-term but not long-term storage.

sents
Yield the document's sentences, as segmented by spaCy.

text
Return the document's raw text.

to_bag_of_terms(*ngrams=(1, 2, 3)*, *named_entities=True*, *normalize=u'lemma'*, *lemmatize=None*, *lowercase=None*, *weighting=u'count'*, *as_strings=False*, ***kwargs*)
Transform `Doc` into a bag-of-terms: the set of unique terms in `Doc` mapped to their frequency of occurrence, where "terms" includes ngrams and/or named entities.

Parameters

- **ngrams** (*`int or Set[int]`*) – n of which n-grams to include; `(1, 2, 3)` (default) includes unigrams (words), bigrams, and trigrams; *2* if only bigrams are wanted; falsy (e.g. False) to not include any
- **named_entities** (*`bool`*) – if True (default), include named entities; note: if ngrams are also included, any ngrams that exactly overlap with an entity are skipped to prevent double-counting
- **lemmatize** (*`bool`*) – *deprecated* if True, words are lemmatized before counting; for example, 'happy', 'happier', and 'happiest' would be grouped together as 'happy', with a count of 3
- **lowercase** (*`bool`*) – *deprecated* if True and `lemmatize` is False, words are lowercased before counting; for example, 'happy' and 'Happy' would be grouped together as 'happy', with a count of 2
- **normalize** (*`str or callable`*) – if 'lemma', lemmatize terms; if 'lower', lowercase terms; if false-y, use the form of terms as they appear in doc; if a callable, must accept

a `spacy.Token` or `spacy.Span` and return a str, e.g. *textacy.spacy_utils.normalized_str()*

- **weighting** (*{'count', 'freq', 'binary'}*) – Type of weight to assign to terms. If 'count' (default), weights are the absolute number of occurrences (count) of term in doc. If 'binary', all counts are set equal to 1. If 'freq', term counts are normalized by the total token count, giving their relative frequency of occurrence.
- **as_strings** (*bool*) – if True, words are returned as strings; if False (default), words are returned as their unique integer ids
- **kwargs** –
  - filter_stops (bool)
  - filter_punct (bool)
  - filter_nums (bool)
  - include_pos (str or Set[str])
  - exclude_pos (str or Set[str])
  - min_freq (int)
  - include_types (str or Set[str])
  - exclude_types (str or Set[str]
  - drop_determiners (bool)

  See *extract.words()*, *extract.ngrams()*, and *extract.named_entities()* for more information on these parameters.

Returns

mapping of a unique term id or string (depending on the value of `as_strings`) to its absolute, relative, or binary frequency of occurrence (depending on the value of `weighting`).

Return type dict

See also:

Doc.to_terms_list()

to_bag_of_words(*normalize=u'lemma'*, *lemmatize=None*, *lowercase=None*, *weighting=u'count'*, *as_strings=False*)

Transform `Doc` into a bag-of-words: the set of unique words in `Doc` mapped to their absolute, relative, or binary frequency of occurrence.

Parameters

- **normalize** (*str*) – if 'lemma', lemmatize words before counting; if 'lower', lowercase words before counting; otherwise, words are counted using the form with which they they appear in doc
- **lemmatize** (*bool*) – if True, words are lemmatized before counting; for example, 'happy', 'happier', and 'happiest' would be grouped together as 'happy', with a count of 3 (*DEPRECATED*)
- **lowercase** (*bool*) – *deprecated* if True and `lemmatize` is False, words are lowercased before counting; for example, 'happy' and 'Happy' would be grouped together as 'happy', with a count of 2

- **weighting** (*{'count', 'freq', 'binary'}*) – Type of weight to assign to words. If 'count' (default), weights are the absolute number of occurrences (count) of word in doc. If 'binary', all counts are set equal to 1. If 'freq', word counts are normalized by the total token count, giving their relative frequency of occurrence. Note: The resulting set of frequencies won't (necessarily) sum to 1.0, since punctuation and stop words are filtered out after counts are normalized.
- **as_strings** (*bool*) – if True, words are returned as strings; if False (default), words are returned as their unique integer ids

Returns

mapping of a unique word id or string (depending on the value of `as_strings`) to its absolute, relative, or binary frequency of occurrence (depending on the value of `weighting`).

Return type dict

to_semantic_network(*nodes=u'words'*, *normalize=u'lemma'*, *edge_weighting=u'default'*, *window_width=10*)

Transform `Doc` into a semantic network, where nodes are either 'words' or 'sents' and edges between nodes may be weighted in different ways.

Parameters

- **nodes** (*{'words', 'sents'}*) – type of doc component to use as nodes in the semantic network
- **normalize** (*str or callable*) – if 'lemma', lemmatize terms; if 'lower', lowercase terms; if false-y, use the form of terms as they appear in doc; if a callable, must accept a `spacy.Token` or `spacy.Span` (if `nodes` = 'words' or 'sents', respectively) and return a str, e.g. *textacy.spacy_utils.normalized_str()*
- **edge_weighting** (*str*) – type of weighting to apply to edges between nodes; if `nodes == 'words'`, options are {'cooc_freq', 'binary'}, if `nodes == 'sents'`, options are {'cosine', 'jaccard'}; if 'default', 'cooc_freq' or 'cosine' will be automatically used
- **window_width** (*int*) – size of sliding window over terms that determines which are said to co-occur; only applicable if 'words'

Returns

where nodes represent either terms or sentences in doc; edges, the relationships between them

Return type `networkx.Graph`

Raises `ValueError` – if `nodes` is neither 'words' nor 'sents'

See also:

terms_to_semantic_network() sents_to_semantic_network()

to_terms_list(*ngrams=(1, 2, 3)*, *named_entities=True*, *normalize=u'lemma'*, *lemmatize=None*, *lowercase=None*, *as_strings=False*, ***kwargs*)

Transform `Doc` into a sequence of ngrams and/or named entities, which aren't necessarily in order of appearance, where each term appears in the list with the same frequency that it appears in `Doc`.

Parameters

- **ngrams** (*int or Set[int]*) – n of which n-grams to include; `(1, 2, 3)` (default) includes unigrams (words), bigrams, and trigrams; *2* if only bigrams are wanted; falsy (e.g. False) to not include any

- **named_entities** (*bool*) – if True (default), include named entities in the terms list; note: if ngrams are also included, named entities are added *first*, and any ngrams that exactly overlap with an entity are skipped to prevent double-counting
- **lemmatize** (*bool*) – *deprecated* if True (default), lemmatize all terms
- **lowercase** (*bool*) – *deprecated* if True and *lemmatize* is False, words are lower-cased
- **normalize** (*str or callable*) – if 'lemma', lemmatize terms; if 'lower', lowercase terms; if false-y, use the form of terms as they appear in doc; if a callable, must accept a `spacy.Token` or `spacy.Span` and return a str, e.g. *textacy.spacy_utils.normalized_str()*
- **as_strings** (*bool*) – if True, terms are returned as strings; if False (default), terms are returned as their unique integer ids
- **kwargs** –
  - filter_stops (bool)
  - filter_punct (bool)
  - filter_nums (bool)
  - include_pos (str or Set[str])
  - exclude_pos (str or Set[str])
  - min_freq (int)
  - include_types (str or Set[str])
  - exclude_types (str or Set[str]
  - drop_determiners (bool)

  see *extract.words*, *extract.ngrams*, and *extract.named_entities* for more information on these parameters

Yields *int or str* –

the next term in the terms list, either as a unique integer id or as a string

Raises `ValueError` – if neither `named_entities` nor `ngrams` are included

Note: Despite the name, this is a generator function; to get an actual list of terms, call `list(doc.to_terms_list())`.

---

tokenized_text
Return text as an ordered, nested list of tokens per sentence.

tokens
Yield the document's tokens, as tokenized by spaCy. Equivalent to iterating directly: `for token in Doc: <do stuff>`

Load, process, iterate, transform, and save a collection of documents — a corpus.

class `textacy.corpus.`**Corpus**(*lang*, *texts=None*, *docs=None*, *metadatas=None*)
An ordered collection of `textacy.Doc` s, all of the same language and sharing the same `spacy.Language` models and vocabulary. Track corpus statistics; flexibly add, iterate through, filter for, and remove documents; save and load parsed content and metadata to and from disk; and more.

Initialize from a stream of texts and corresponding metadatas:

```
>>> cw = textacy.datasets.CapitolWords()
>>> records = cw.docs(limit=50)
>>> text_stream, metadata_stream = textacy.fileio.split_record_fields(
...     records, 'text')
>>> corpus = textacy.Corpus(
...     'en', texts=text_stream, metadatas=metadata_stream)
>>> print(corpus)
Corpus(50 docs; 32163 tokens)
```

Index, slice, and flexibly get particular documents:

```
>>> corpus[0]
Doc(159 tokens; "Mr. Speaker, 480,000 Federal employees are work...")
>>> corpus[:3]
[Doc(159 tokens; "Mr. Speaker, 480,000 Federal employees are work..."),
 Doc(219 tokens; "Mr. Speaker, a relationship, to work and surviv..."),
 Doc(336 tokens; "Mr. Speaker, I thank the gentleman for yielding...")]
>>> match_func = lambda doc: doc.metadata['speaker_name'] == 'Bernie Sanders'
>>> for doc in corpus.get(match_func, limit=3):
...     print(doc)
Doc(159 tokens; "Mr. Speaker, 480,000 Federal employees are work...")
Doc(336 tokens; "Mr. Speaker, I thank the gentleman for yielding...")
Doc(177 tokens; "Mr. Speaker, if we want to understand why in th...")
```

Add and remove documents, with automatic updating of corpus statistics:

```
>>> records = cw.docs(congress=114, limit=25)
>>> text_stream, metadata_stream = textacy.fileio.split_record_fields(
...     records, 'text')
>>> corpus.add_texts(text_stream, metadatas=metadata_stream, n_threads=4)
>>> print(corpus)
Corpus(75 docs; 55869 tokens)
>>> corpus.remove(lambda doc: doc.metadata['speaker_name'] == 'Rick Santorum')
>>> print(corpus)
Corpus(60 docs; 48532 tokens)
>>> del corpus[:5]
>>> print(corpus)
Corpus(55 docs; 47444 tokens)
```

Get word and doc frequencies in absolute, relative, or binary form:

```
>>> counts = corpus.word_freqs(lemmatize=True, weighting='count')
>>> idf = corpus.word_doc_freqs(lemmatize=True, weighting='idf')
```

Save to and load from disk:

```
>>> corpus.save('~/Desktop', name='congress', compression='gzip')
>>> corpus = textacy.Corpus.load(
...     '~/Desktop', name='congress', compression='gzip')
>>> print(corpus)
Corpus(55 docs; 47444 tokens)
```

Parameters

- **lang** (str or `spacy.Language`) – Language of content for all docs in corpus. Pass a standard 2-letter language code (e.g. "en") or the name of a spacy model for the desired language (e.g. "en_core_web_md") or an already-instantiated `spacy.Language` object. If

a str, the value is used to instantiate the corresponding `spacy.Language` with all models loaded by default, and the appropriate 2-letter lang code is assigned to *`Corpus.lang`*.

Note: The `spacy.Language` object parses all documents contents and sets the *`spacy_vocab`* and *`spacy_stringstore`* attributes. See https://spacy.io/docs/usage/models#available for available spacy models.

- **`texts`** (*`Iterable[str]`*) – Stream of documents as (unicode) text, to be processed by spaCy and added to the corpus as `textacy.Doc` s.
- **`docs`** (Iterable[`textacy.Doc`] or Iterable[`spacy.Doc`]) – Stream of documents already-processed by spaCy alone or via textacy.
- **`metadatas`** (*`Iterable[dict]`*) – Stream of dictionaries of relevant doc metadata. **Note:** This stream must align exactly with `texts` or `docs`, or else metadata will be misassigned. More concretely, the first item in `metadatas` will be assigned to the first item in `texts` or `docs`, and so on from there.

`lang`
str – 2-letter code for language of documents in `Corpus`.

`n_docs`
int – Number of documents in `Corpus`.

`n_tokens`
int – Total number of tokens of all documents in `Corpus`.

`n_sents`
int – Total number of sentences of all documents in `Corpus`. If the `spacy.Language` used to process documents did not include a syntactic parser, upon which sentence segmentation relies, this value will be null.

`docs`
List[`textacy.Doc`] – List of documents in `Corpus`. In 99% of cases, you should never have to interact directly with this list; instead, index and slice directly on `Corpus` or use the flexible *`Corpus.get()`* and `Corpus.remove() <Corpus.remove()` methods.

`spacy_lang`
`spacy.Language` – http://spacy.io/docs/#english

`spacy_vocab`
`spacy.Vocab` – https://spacy.io/docs#vocab

`spacy_stringstore`
`spacy.StringStore` – https://spacy.io/docs#stringstore

`add_doc`(*doc*, *metadata=None*)
Add an existing *`textacy.Doc`* or initialize a new one from a `spacy.Doc` to the corpus.

Parameters

- **`doc`** (`textacy.Doc` or `spacy.Doc`) –
- **`metadata`** (*`dict`*) – Dictionary of relevant document metadata. If `doc` is a `spacy.Doc`, it will be paired as usual; if `doc` is a `textacy.Doc`, it will *overwrite* any existing metadata.

Warning: If `doc` was already added to this or another `Corpus`, it will be deep-copied and then added as if a new document. A warning message will be logged. This is probably not a thing you should do.

add_text(*text*, *metadata=None*)
Create a *textacy.Doc* from `text` and `metadata`, then add it to the corpus.

Parameters

- **text** (*str*) – Document (text) content to add to corpus as a `Doc`.
- **metadata** (*dict*) – Dictionary of relevant document metadata.

add_texts(*texts*, *metadatas=None*, *n_threads=3*, *batch_size=1000*)
Process a stream of texts (and a corresponding stream of metadata dicts, optionally) in parallel with spaCy; add as *textacy.Doc* s to the corpus.

Parameters

- **texts** (*Iterable[str]*) – Stream of texts to add to corpus as `Doc` s
- **metadatas** (*Iterable[dict]*) – Stream of dictionaries of relevant document metadata. **Note:** This stream must align exactly with `texts`, or metadata will be mis-assigned to texts. More concretely, the first item in `metadatas` will be assigned to the first item in `texts`, and so on from there.
- **n_threads** (*int*) – Number of threads to use when processing `texts` in parallel, if available.
- **batch_size** (*int*) – Number of texts to process at a time.

See also:

`fileio.split_record_fields()` http://spacy.io/docs/#multi-threaded

get(*match_func*, *limit=-1*)
Iterate over docs in `Corpus` and return all (or N <= `limit`) for which `match_func(doc)` is True.

Parameters

- **match_func** (*func*) – Function that takes a `textacy.Doc` as input and returns a boolean value. For example:

```
Corpus.get(lambda x: len(x) >= 100)
```

gets all docs with 100+ tokens. And:

```
Corpus.get(lambda x: x.metadata['author'] == 'Burton DeWilde')
```

gets all docs whose author was given as 'Burton DeWilde'.

- **limit** (*int*) – Maximum number of matched docs to return.

Yields *textacy.Doc* –

next document passing `match_func` up to `limit` docs

See also:

Corpus.remove()

Tip: To get doc(s) by index, treat `Corpus` as a list and use Python's usual indexing and slicing: `Corpus[0]` gets the first document in the corpus; `Corpus[:5]` gets the first 5; etc.

---

classmethod load(*path*, *name=None*, *compression=None*)
Load content and metadata from disk, and initialize a `Corpus`.

Parameters

- **path** (*str*) – Directory on disk where content + metadata are saved.
- **name** (*str*) – Identifying/uniquifying name prepended to the default filenames 'spacy_docs.bin', 'metadatas.json', and 'info.json', used when corpus was saved to disk via *Corpus.save()*.
- **compression** (*{'gzip', 'bz2', 'lzma'} or None*) – Type of compression used to reduce size of 'metadatas.json' file when saved, if any.

Returns *textacy.Corpus*

Warning: If the `spacy.Vocab` object used to save this document is not the same as the one used to load it, there will be problems! Consequently, this functionality is only useful as short-term but not long-term storage.

remove(*match_func*, *limit=-1*)

Remove all (or N <= `limit`) docs in `Corpus` for which `match_func(doc)` is True. Corpus doc/sent/token counts are adjusted accordingly, as are the `Doc.corpus_index` attributes on affected documents.

Parameters

- **match_func** (*func*) – Function that takes a *textacy.Doc* and returns a boolean value. For example:

```
Corpus.remove(lambda x: len(x) >= 100)
```

removes docs with 100+ tokens. And:

```
Corpus.remove(lambda x: x.metadata['author'] == 'Burton DeWilde')
```

removes docs whose author was given as 'Burton DeWilde'.

- **limit** (*int*) – Maximum number of matched docs to remove.

See also:

Corpus.get()

Tip: To remove doc(s) by index, treat `Corpus` as a list and use Python's usual indexing and slicing: `del Corpus[0]` removes the first document in the corpus; `del Corpus[:5]` removes the first 5; etc.

save(*path*, *name=None*, *compression=None*)

Save `Corpus` content and metadata to disk.

Parameters

- **path** (*str*) – Directory on disk where content + metadata will be saved.
- **name** (*str*) – Prepend default filenames 'spacy_docs.bin', 'metadatas.json', and 'info.json' with a name to identify/uniquify this particular corpus.
- **compression** (*{'gzip', 'bz2', 'lzma'} or None*) – Type of compression used to reduce size of 'metadatas.json' file, if any.

> **Warning:** If the `spacy.Vocab` object used to save this corpus is not the same as the one used to load it, there will be problems! Consequently, this functionality is only useful as short-term but not long-term storage.

vectors
Constituent docs' word vectors stacked together in a matrix.

word_doc_freqs(*normalize=u'lemma'*, *lemmatize=None*, *lowercase=None*, *weighting=u'count'*, *smooth_idf=True*, *as_strings=False*)
Map the set of unique words in `Corpus` to their *document* counts as absolute, relative, inverse, or binary frequencies of occurence.

Parameters

- **normalize** (*str*) – if 'lemma', lemmatize words before counting; if 'lower', lowercase words before counting; otherwise, words are counted using the form with which they they appear in docs
- **lemmatize** (*bool*) – if True, words are lemmatized before counting; for example, 'happy', 'happier', and 'happiest' would be grouped together as 'happy', with a count of 3 (*DEPRECATED*)
- **lowercase** (*bool*) – if True and `lemmatize` is False, words are lower- cased before counting; for example, 'happy' and 'Happy' would be grouped together as 'happy', with a count of 2 (*DEPRECATED*)
- **weighting** (*{'count', 'freq', 'idf', 'binary'}*) – Type of weight to assign to words. If 'count' (default), weights are the absolute number (count) of documents in which word appears. If 'binary', all counts are set equal to 1. If 'freq', word doc counts are normalized by the total document count, giving their relative frequency of occurrence. If 'idf', weights are the log of the inverse relative frequencies: `log(n_docs / word_doc_count)` or `log(1 + n_docs / word_doc_count)` if `smooth_idf` is True.
- **smooth_idf** (*bool*) – if True, add 1 to all document frequencies when calculating 'idf' weighting, equivalent to adding a single document to the corpus containing every unique word
- **as_strings** (*bool*) – if True, words are returned as strings; if False (default), words are returned as their unique integer ids

Returns

mapping of a unique word id or string (depending on the value of `as_strings`) to the number of documents in which it appears weighted as absolute, relative, or binary frequencies (depending on the value of `weighting`).

Return type dict

See also:

`vsm.get_doc_freqs() <textacy.vsm.get_doc_freqs>`()

word_freqs(*normalize=u'lemma'*, *lemmatize=None*, *lowercase=None*, *weighting=u'count'*, *as_strings=False*)
Map the set of unique words in `Corpus` to their counts as absolute, relative, or binary frequencies of occurence. This is akin to *Doc.to_bag_of_words()*.

Parameters

- **normalize** (*str*) – if 'lemma', lemmatize words before counting; if 'lower', lowercase words before counting; otherwise, words are counted using the form with which they they appear in docs
- **lemmatize** (*bool*) – if True, words are lemmatized before counting; for example, 'happy', 'happier', and 'happiest' would be grouped together as 'happy', with a count of 3 (*DEPRECATED*)
- **lowercase** (*bool*) – if True and `lemmatize` is False, words are lower- cased before counting; for example, 'happy' and 'Happy' would be grouped together as 'happy', with a count of 2 (*DEPRECATED*)
- **weighting** (*{'count', 'freq', 'binary'}*) – Type of weight to assign to words. If 'count' (default), weights are the absolute number of occurrences (count) of word in corpus. If 'binary', all counts are set equal to 1. If 'freq', word counts are normalized by the total token count, giving their relative frequency of occurrence. Note: The resulting set of frequencies won't (necessarily) sum to 1.0, since punctuation and stop words are filtered out after counts are normalized.
- **as_strings** (*bool*) – if True, words are returned as strings; if False (default), words are returned as their unique integer ids

Returns

mapping of a unique word id or string (depending on the value of `as_strings`) to its absolute, relative, or binary frequency of occurrence (depending on the value of `weighting`).

Return type dict

See also:

`vsm.get_term_freqs() <textacy.vsm.get_term_freqs>`()

Text Preprocessing

Functions that modify raw text *in-place*, replacing contractions, URLs, emails, phone numbers, and currency symbols with standardized forms. These should be applied before processing by Spacy, but be warned: preprocessing may affect the interpretation of the text – and spacy's processing of it.

`textacy.preprocess.`**fix_bad_unicode**(*text*, *normalization=u'NFC'*)

Fix unicode text that's "broken" using ftfy; this includes mojibake, HTML entities and other code cruft, and non-standard forms for display purposes.

Parameters

- **text** (*str*) – raw text
- **normalization** (*{'NFC', 'NFKC', 'NFD', 'NFKD'}*) – if 'NFC', combines characters and diacritics written using separate code points, e.g. converting "e" plus an acute accent modifier into "é"; unicode can be converted to NFC form without any change in its meaning! if 'NFKC', additional normalizations are applied that can change the meanings of characters, e.g. ellipsis characters will be replaced with three periods

Returns str

`textacy.preprocess.`**normalize_whitespace**(*text*)

Given `text` str, replace one or more spacings with a single space, and one or more linebreaks with a single newline. Also strip leading/trailing whitespace.

textacy.preprocess.**preprocess_text**(*text*, *fix_unicode=False*, *lowercase=False*, *transliterate=False*, *no_urls=False*, *no_emails=False*, *no_phone_numbers=False*, *no_numbers=False*, *no_currency_symbols=False*, *no_punct=False*, *no_contractions=False*, *no_accents=False*)

Normalize various aspects of a raw text doc before parsing it with Spacy. A convenience function for applying all other preprocessing functions in one go.

Parameters

- **text** (*str*) – raw text to preprocess
- **fix_unicode** (*bool*) – if True, fix "broken" unicode such as mojibake and garbled HTML entities
- **lowercase** (*bool*) – if True, all text is lower-cased
- **transliterate** (*bool*) – if True, convert non-ascii characters into their closest ascii equivalents
- **no_urls** (*bool*) – if True, replace all URL strings with '*URL*'
- **no_emails** (*bool*) – if True, replace all email strings with '*EMAIL*'
- **no_phone_numbers** (*bool*) – if True, replace all phone number strings with '*PHONE*'
- **no_numbers** (*bool*) – if True, replace all number-like strings with '*NUMBER*'
- **no_currency_symbols** (*bool*) – if True, replace all currency symbols with their standard 3-letter abbreviations
- **no_punct** (*bool*) – if True, remove all punctuation (replace with empty string)
- **no_contractions** (*bool*) – if True, replace *English* contractions with their unshortened forms
- **no_accents** (*bool*) – if True, replace all accented characters with unaccented versions; NB: if *transliterate* is True, this option is redundant

Returns input `text` processed according to function args

Return type str

> **Warning:** These changes may negatively affect subsequent NLP analysis performed on the text, so choose carefully, and preprocess at your own risk!

textacy.preprocess.**remove_accents**(*text*, *method='unicode'*)

Remove accents from any accented unicode characters in `text` str, either by transforming them into ascii equivalents or removing them entirely.

Parameters

- **text** (*str*) – raw text
- **method** (*{'unicode', 'ascii'}*) – if 'unicode', remove accented char for any unicode symbol with a direct ASCII equivalent; if 'ascii', remove accented char for any unicode symbol

  NB: the 'ascii' method is notably faster than 'unicode', but less good

Returns str

Raises `ValueError` – if `method` is not in {'unicode', 'ascii'}

textacy.preprocess.**remove_punct**(*text*, *marks=None*)
Remove punctuation from `text` by replacing all instances of `marks` with an empty string.

Parameters

- **text** (*str*) – raw text
- **marks** (*str*) – If specified, remove only the characters in this string, e.g. `marks=',;:'` removes commas, semi-colons, and colons. Otherwise, all punctuation marks are removed.

Returns str

Note: When `marks=None`, Python's built-in `str.translate()` is used to remove punctuation; otherwise,, a regular expression is used instead. The former's performance is about 5-10x faster.

---

textacy.preprocess.**replace_currency_symbols**(*text*, *replace_with=None*)
Replace all currency symbols in `text` str with string specified by `replace_with` str.

Parameters

- **text** (*str*) – raw text
- **replace_with** (*str*) – if None (default), replace symbols with their standard 3-letter abbreviations (e.g. '$' with 'USD', '£' with 'GBP'); otherwise, pass in a string with which to replace all symbols (e.g. "*CURRENCY*")

Returns str

textacy.preprocess.**replace_emails**(*text*, *replace_with=u'*EMAIL*'*)
Replace all emails in `text` str with `replace_with` str.

textacy.preprocess.**replace_numbers**(*text*, *replace_with=u'*NUMBER*'*)
Replace all numbers in `text` str with `replace_with` str.

textacy.preprocess.**replace_phone_numbers**(*text*, *replace_with=u'*PHONE*'*)
Replace all phone numbers in `text` str with `replace_with` str.

textacy.preprocess.**replace_urls**(*text*, *replace_with=u'*URL*'*)
Replace all URLs in `text` str with `replace_with` str.

textacy.preprocess.**transliterate_unicode**(*text*)
Try to represent unicode data in ascii characters similar to what a human with a US keyboard would choose.

Works great for languages of Western origin, worse the farther the language gets from Latin-based alphabets. It's based on hand-tuned character mappings that also contain ascii approximations for symbols and non-Latin alphabets.

textacy.preprocess.**unpack_contractions**(*text*)
Replace *English* contractions in `text` str with their unshortened forms. N.B. The "'d" and "'s" forms are ambiguous (had/would, is/has/possessive), so are left as-is.

Information Extraction

Functions to extract various elements of interest from documents already parsed by spaCy, such as n-grams, named entities, subject-verb-object triples, and acronyms.

textacy.extract.**acronyms_and_definitions**(*doc*, *known_acro_defs=None*)
Extract a collection of acronyms and their most likely definitions, if available, from a spacy-parsed doc. If multiple definitions are found for a given acronym, only the most frequently occurring definition is returned.

Parameters

- **doc** (textacy.Doc or spacy.Doc or spacy.Span) –
- **known_acro_defs** (*dict, optional*) – if certain acronym/definition pairs are known, pass them in as {acronym (str): definition (str)}; algorithm will not attempt to find new definitions

Returns unique acronyms (keys) with matched definitions (values)

Return type dict

References

Taghva, Kazem, and Jeff Gilbreth. "Recognizing acronyms and their definitions." International Journal on Document Analysis and Recognition 1.4 (1999): 191-198.

textacy.extract.**direct_quotations**(*doc*)

Baseline, not-great attempt at direction quotation extraction (no indirect or mixed quotations) using rules and patterns. English only.

Parameters **doc** (textacy.Doc or spacy.Doc) –

Yields (spacy.Span, spacy.Token, spacy.Span) –

next quotation in doc represented as a (speaker, reporting verb, quotation) 3-tuple

Notes

Loosely inspired by Krestel, Bergler, Witte. "Minding the Source: Automatic Tagging of Reported Speech in Newspaper Articles".

TODO: Better approach would use ML, but needs a training dataset.

textacy.extract.**named_entities**(*doc*, *include_types=None*, *exclude_types=None*, *drop_determiners=True*, *min_freq=1*)

Extract an ordered sequence of named entities (PERSON, ORG, LOC, etc.) from a spacy-parsed doc, optionally filtering by entity types and frequencies.

Parameters

- **doc** (textacy.Doc or spacy.Doc) –
- **include_types** (*str or Set[str]*) – remove named entities whose type IS NOT in this param; if "NUMERIC", all numeric entity types ("DATE", "MONEY", "ORDINAL", etc.) are included
- **exclude_types** (*str or Set[str]*) – remove named entities whose type IS in this param; if "NUMERIC", all numeric entity types ("DATE", "MONEY", "ORDINAL", etc.) are excluded
- **drop_determiners** (*bool*) – remove leading determiners (e.g. "the") from named entities (e.g. "the United States" => "United States")
- **min_freq** (*int*) – remove named entities that occur in *doc* fewer than *min_freq* times

Yields spacy.Span –

the next named entity from doc passing all specified filters in order of appearance in the document

Raise:

TypeError: if *include_types* or *exclude_types* is not a str, a set of str, or a falsy value

textacy.extract.**ngrams**(*doc*, *n*, *filter_stops=True*, *filter_punct=True*, *filter_nums=False*, *include_pos=None*, *exclude_pos=None*, *min_freq=1*)

Extract an ordered sequence of n-grams (`n` consecutive words) from a spacy-parsed doc, optionally filtering n-grams by the types and parts-of-speech of the constituent words.

Parameters

- **doc** (`textacy.Doc`, `spacy.Doc`, or `spacy.Span`) –
- **n** (*int*) – number of tokens per n-gram; 2 => bigrams, 3 => trigrams, etc.
- **filter_stops** (*bool*) – if True, remove ngrams that start or end with a stop word
- **filter_punct** (*bool*) – if True, remove ngrams that contain any punctuation-only tokens
- **filter_nums** (*bool*) – if True, remove ngrams that contain any numbers or number-like tokens (e.g. 10, 'ten')
- **include_pos** (*str or Set[str]*) – remove ngrams if any of their constituent tokens' part-of-speech tags ARE NOT included in this param
- **exclude_pos** (*str or Set[str]*) – remove ngrams if any of their constituent tokens' part-of-speech tags ARE included in this param
- **min_freq** (*int, optional*) – remove ngrams that occur in *doc* fewer than *min_freq* times

Yields `spacy.Span` –

the next ngram from `doc` passing all specified filters, in order of appearance in the document

Raises

- `ValueError` – if `n` < 1
- `TypeError` – if *include_pos* or *exclude_pos* is not a str, a set of str, or a falsy value

---

Note: Filtering by part-of-speech tag uses the universal POS tag set, http://universaldependencies.org/u/pos/

---

textacy.extract.**noun_chunks**(*doc*, *drop_determiners=True*, *min_freq=1*)

Extract an ordered sequence of noun chunks from a spacy-parsed doc, optionally filtering by frequency and dropping leading determiners.

Parameters

- **doc** (`textacy.Doc` or `spacy.Doc`) –
- **drop_determiners** (*bool*) – remove leading determiners (e.g. "the") from phrases (e.g. "the quick brown fox" => "quick brown fox")
- **min_freq** (*int*) – remove chunks that occur in *doc* fewer than *min_freq* times

Yields `spacy.Span` –

the next noun chunk from `doc` in order of appearance in the document

textacy.extract.**pos_regex_matches**(*doc*, *pattern*)

Extract sequences of consecutive tokens from a spacy-parsed doc whose part-of-speech tags match the specified regex pattern.

Parameters

- **doc** (textacy.Doc or spacy.Doc or spacy.Span) –
- **pattern** (*str*) – Pattern of consecutive POS tags whose corresponding words are to be extracted, inspired by the regex patterns used in NLTK's *nltk.chunk.regexp*. Tags are uppercase, from the universal tag set; delimited by < and >, which are basically converted to parentheses with spaces as needed to correctly extract matching word sequences; white space in the input doesn't matter.

  Examples (see constants.POS_REGEX_PATTERNS):

  - noun phrase: r'<DET>? (<NOUN>+ <ADP|CONJ>)* <NOUN>+'
  - compound nouns: r'<NOUN>+'
  - verb phrase: r'<VERB>?<ADV>*<VERB>+'
  - prepositional phrase: r'<PREP> <DET>? (<NOUN>+<ADP>)* <NOUN>+'

Yields spacy.Span –

the next span of consecutive tokens from doc whose parts-of-speech match pattern, in order of apperance

textacy.extract.**semistructured_statements**(*doc*, *entity*, *cue='be'*, *ignore_entity_case=True*, *min_n_words=1*, *max_n_words=20*)

Extract "semi-structured statements" from a spacy-parsed doc, each as a (entity, cue, fragment) triple. This is similar to subject-verb-object triples.

Parameters

- **doc** (textacy.Doc or spacy.Doc) –
- **entity** (*str*) – a noun or noun phrase of some sort (e.g. "President Obama", "global warming", "Python")
- **cue** (*str, optional*) – verb lemma with which *entity* is associated (e.g. "talk about", "have", "write")
- **ignore_entity_case** (*bool, optional*) – if True, entity matching is case-independent
- **min_n_words** (*int, optional*) – min number of tokens allowed in a matching fragment
- **max_n_words** (*int, optional*) – max number of tokens allowed in a matching fragment

Yields (spacy.Span or spacy.Token, spacy.Span or spacy.Token, spacy.Span) – where each element is a matching (entity, cue, fragment) triple

Notes

Inspired by N. Diakopoulos, A. Zhang, A. Salway. Visual Analytics of Media Frames in Online News and Blogs. IEEE InfoVis Workshop on Text Visualization. October, 2013.

Which itself was inspired by by Salway, A.; Kelly, L.; Skadiņa, I.; and Jones, G. 2010. Portable Extraction of Partially Structured Facts from the Web. In Proc. ICETAL 2010, LNAI 6233, 345-356. Heidelberg, Springer.

textacy.extract.**subject_verb_object_triples**(*doc*)

Extract an ordered sequence of subject-verb-object (SVO) triples from a spacy-parsed doc. Note that this only works for SVO languages.

Parameters **doc** (textacy.Doc or spacy.Doc or spacy.Span) –

Yields Tuple[spacy.Span, spacy.Span, spacy.Span] –

the next 3-tuple of spans from doc representing a (subject, verb, object) triple, in order of appearance

textacy.extract.**words**(*doc*, *filter_stops=True*, *filter_punct=True*, *filter_nums=False*, *include_pos=None*, *exclude_pos=None*, *min_freq=1*)

Extract an ordered sequence of words from a document processed by spaCy, optionally filtering words by part-of-speech tag and frequency.

Parameters

- **doc** (textacy.Doc, spacy.Doc, or spacy.Span) –
- **filter_stops** (*bool*) – if True, remove stop words from word list
- **filter_punct** (*bool*) – if True, remove punctuation from word list
- **filter_nums** (*bool*) – if True, remove number-like words (e.g. 10, 'ten') from word list
- **include_pos** (*str or Set[str]*) – remove words whose part-of-speech tag IS NOT included in this param
- **exclude_pos** (*str or Set[str]*) – remove words whose part-of-speech tag IS in the specified tags
- **min_freq** (*int*) – remove words that occur in *doc* fewer than *min_freq* times

Yields spacy.Token –

the next token from doc passing specified filters in order of appearance in the document

Raises TypeError – if *include_pos* or *exclude_pos* is not a str, a set of str, or a falsy value

---

Note: Filtering by part-of-speech tag uses the universal POS tag set, http://universaldependencies.org/u/pos/

Functions for unsupervised automatic key term extraction, both specific algorithms (SGRank, TextRank, SingleRank) and a generalization of semantic network-based methods. Also includes a function to aggregate common key term variants of the same idea.

textacy.keyterms.**aggregate_term_variants**(*terms*, *acro_defs=None*, *fuzzy_dedupe=True*)

Take a set of unique terms and aggregate terms that are symbolic, lexical, and ordering variants of each other, as well as acronyms and fuzzy string matches.

Parameters

- **terms** (*Set[str]*) – set of unique terms with potential duplicates
- **acro_defs** (*dict*) – if not None, terms that are acronyms will be aggregated with their definitions and terms that are definitions will be aggregated with their acronyms
- **fuzzy_dedupe** (*bool*) – if True, fuzzy string matching will be used to aggregate similar terms of a sufficient length

Returns each item is a set of aggregated terms

Return type List[Set[str]]

Notes

Partly inspired by aggregation of variants discussed in Park, Youngja, Roy J. Byrd, and Branimir K. Boguraev. "Automatic glossary extraction: beyond terminology identification." Proceedings of the 19th international conference on Computational linguistics-Volume 1. Association for Computational Linguistics, 2002.

textacy.keyterms.**key_terms_from_semantic_network**(*doc*, *normalize=u'lemma'*, *window_width=2*, *edge_weighting=u'binary'*, *ranking_algo=u'pagerank'*, *join_key_words=False*, *n_keyterms=10*, ***kwargs*)

Extract key terms from a document by ranking nodes in a semantic network of terms, connected by edges and weights specified by parameters.

Parameters

- **doc** (textacy.Doc or spacy.Doc) –
- **normalize** (*str or callable*) – if 'lemma', lemmatize terms; if 'lower', lowercase terms; if None, use the form of terms as they appeared in doc; if a callable, must accept a spacy.Token and return a str, e.g. *textacy.spacy_utils.normalized_str()*
- **window_width** (*int*) – width of sliding window in which term co-occurrences are said to occur
- **edge_weighting** (*{'binary', 'cooc_freq'}*) – method used to determine weights of edges between nodes in the semantic network; if 'binary', edge weight is set to 1 for any two terms co-occurring within *window_width* terms; if 'cooc_freq', edge weight is set to the number of times that any two terms co-occur
- **ranking_algo** (*{'pagerank', 'divrank', 'bestcoverage'}*) – algorithm with which to rank nodes in the semantic network; *pagerank* is the canonical (and default) algorithm, but it prioritizes node centrality at the expense of node diversity; the other two attempt to balance centrality with diversity
- **join_key_words** (*bool*) – if True, join consecutive key words together into longer key terms, taking the sum of the constituent words' scores as the joined key term's combined score
- **n_keyterms** (*int or float*) – if int, number of top-ranked terms to return as keyterms; if float, must be in the open interval (0, 1), is converted to an integer by round(len(doc) * n_keyterms)

Returns

sorted list of top n_keyterms key terms and their corresponding ranking scores

Return type List[Tuple[str, float]]

Raises ValueError – if n_keyterms is a float but not in (0.0, 1.0]

textacy.keyterms.**most_discriminating_terms**(*terms_lists*, *bool_array_grp1*, *max_n_terms=1000*, *top_n_terms=25*)

Given a collection of documents assigned to 1 of 2 exclusive groups, get the *top_n_terms* most discriminating terms for group1-and-not-group2 and group2-and-not-group1.

Parameters

- **terms_lists** (*Iterable[Iterable[str]]*) – a sequence of documents, each as a sequence of (str) terms; used as input to doc_term_matrix()

- **bool_array_grp1** (*Iterable[bool]*) – an ordered sequence of True/False values, where True corresponds to documents falling into "group 1" and False corresponds to those in "group 2"
- **max_n_terms** (*int*) – only consider terms whose document frequency is within the top *max_n_terms* out of all distinct terms; must be > 0
- **top_n_terms** (*int or float*) – if int (must be > 0), the total number of most discriminating terms to return for each group; if float (must be in the interval (0, 1)), the fraction of *max_n_terms* to return for each group

Returns top *top_n_terms* most discriminating terms for grp1-not-grp2 List[str]: top *top_n_terms* most discriminating terms for grp2-not-grp1

Return type List[str]

References

King, Gary, Patrick Lam, and Margaret Roberts. "Computer-Assisted Keyword and Document Set Discovery from Unstructured Text." (2014). http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.458.1445&rep=rep1&type=pdf

textacy.keyterms.**rank_nodes_by_bestcoverage**(*graph*, *k*, *c=1*, *alpha=1.0*)

Rank nodes in a network using the *[BestCoverage]* algorithm that attempts to balance between node centrality and diversity.

Parameters

- **graph** (networkx.Graph) –
- **k** (*int*) – number of results to return for top-k search
- **c** (*int*) – *l* parameter for *l*-step expansion; best if 1 or 2
- **alpha** (*float*) – float in [0.0, 1.0] specifying how much of central vertex's score to remove from its *l*-step neighbors; smaller value puts more emphasis on centrality, larger value puts more emphasis on diversity

Returns

top k nodes as ranked by bestcoverage algorithm; keys as node identifiers, values as corresponding ranking scores

Return type dict

References

textacy.keyterms.**rank_nodes_by_divrank**(*graph*, *r=None*, *lambda_=0.5*, *alpha=0.5*)

Rank nodes in a network using the *[DivRank]* algorithm that attempts to balance between node centrality and diversity.

Parameters

- **graph** (networkx.Graph) –
- **r** (numpy.array,) – the "personalization vector"; by default, `r = ones(1, n)/n`
- **lambda** (*float*) – must be in [0.0, 1.0]
- **alpha** (*float*) – controls the strength of self-links; must be in [0.0, 1.0]

Returns list of (node, score) tuples ordered by desc. divrank score

Return type List[Tuple[str, float]]

References

textacy.keyterms.**sgrank**(*doc*, *ngrams=(1, 2, 3, 4, 5, 6)*, *normalize=u'lemma'*, *window_width=1500*, *n_keyterms=10*, *idf=None*)

Extract key terms from a document using the *[SGRank]* algorithm.

Parameters

- **doc** (textacy.Doc or spacy.Doc) –
- **ngrams** (*int or Set[int]*) – n of which n-grams to include; `(1, 2, 3, 4, 5, 6)` (default) includes all ngrams from 1 to 6; *2* if only bigrams are wanted
- **normalize** (*str or callable*) – If 'lemma', lemmatize terms; if 'lower', lowercase terms; if None, use the form of terms as they appeared in `doc`; if a callable, must accept a `spacy.Span` and return a str, e.g. *textacy.spacy_utils.normalized_str()*
- **window_width** (*int*) – Width of sliding window in which term co-occurrences are determined to occur. Note: Larger values may dramatically increase runtime, owing to the larger number of co-occurrence combinations that must be counted.
- **n_keyterms** (*int or float*) – Number of top-ranked terms to return as keyterms. If int, represents the absolute number; if float, must be in the open interval (0.0, 1.0), and is converted to an integer by `int(round(len(doc) * n_keyterms))`
- **idf** (*dict*) – Mapping of `normalize(term)` to inverse document frequency for re-weighting of unigrams (n-grams with n > 1 have df assumed = 1). NOTE: Results are typically better with idf information.

Returns

sorted list of top n_keyterms key terms and their corresponding SGRank scores

Return type List[Tuple[str, float]]

Raises `ValueError` – If `n_keyterms` is a float but not in (0.0, 1.0] or `window_width` < 2.

References

textacy.keyterms.**singlerank**(*doc*, *normalize=u'lemma'*, *n_keyterms=10*)

Convenience function for calling *key_terms_from_semantic_network* with the parameter values used in the *[SingleRank]* algorithm.

Parameters

- **doc** (textacy.Doc or spacy.Doc) –
- **normalize** (*str or callable*) – if 'lemma', lemmatize terms; if 'lower', lowercase terms; if None, use the form of terms as they appeared in `doc`; if a callable, must accept a `spacy.Token` and return a str, e.g. *textacy.spacy_utils.normalized_str()*
- **n_keyterms** (*int or float*) – if int, number of top-ranked terms to return as keyterms; if float, must be in the open interval (0, 1), representing the fraction of top-ranked terms to return as keyterms

Returns see *key_terms_from_semantic_network()*.

References

textacy.keyterms.**textrank**(*doc*, *normalize='lemma'*, *n_keyterms=10*)

Convenience function for calling *key_terms_from_semantic_network* with the parameter values used in the *[TextRank]* algorithm.

Parameters

- **doc** (textacy.Doc or spacy.Doc) –
- **normalize** (*str or callable*) – if 'lemma', lemmatize terms; if 'lower', lowercase terms; if None, use the form of terms as they appeared in doc; if a callable, must accept a spacy.Token and return a str, e.g. *textacy.spacy_utils.normalized_str()*
- **n_keyterms** (*int or float*) – if int, number of top-ranked terms to return as keyterms; if float, must be in the open interval (0, 1), representing the fraction of top-ranked terms to return as keyterms

Returns See *key_terms_from_semantic_network()*.

References

Topic Modeling

Convenient and consolidated topic-modeling, built on scikit-learn.

class textacy.tm.topic_model.**TopicModel**(*model*, *n_topics=10*, ***kwargs*)

Train and apply a topic model to vectorized texts using scikit-learn's implementations of LSA, LDA, and NMF models. Inspect and visualize results. Save and load trained models to and from disk.

Prepare a vectorized corpus (i.e. document-term matrix) and corresponding vocabulary (i.e. mapping of term strings to column indices in the matrix). See *textacy.vsm.Vectorizer* for details. In short:

```
>>> vectorizer = Vectorizer(
...     weighting='tfidf', normalize=True, smooth_idf=True,
...     min_df=3, max_df=0.95, max_n_terms=100000)
>>> doc_term_matrix = vectorizer.fit_transform(terms_list)
```

Initialize and train a topic model:

```
>>> model = textacy.tm.TopicModel('nmf', n_topics=20)
>>> model.fit(doc_term_matrix)
>>> model
TopicModel(n_topics=10, model=NMF)
```

Transform the corpus and interpret our model:

```
>>> doc_topic_matrix = model.transform(doc_term_matrix)
>>> for topic_idx, top_terms in model.top_topic_terms(vectorizer.id_to_term,␣
↪topics=[0,1]):
...     print('topic', topic_idx, ':', '   '.join(top_terms))
topic 0 : people   american   go   year   work   think   $   today   money   ␣
↪america
topic 1 : rescind   quorum   order   unanimous   consent   ask   president   mr.  ␣
↪ madam   absence
>>> for topic_idx, top_docs in model.top_topic_docs(doc_topic_matrix, topics=[0,
↪1], top_n=2):
...     print(topic_idx)
```

```
...     for j in top_docs:
...         print(corpus[j].metadata['title'])
0
THE MOST IMPORTANT ISSUES FACING THE AMERICAN PEOPLE
55TH ANNIVERSARY OF THE BATTLE OF CRETE
1
CHEMICAL WEAPONS CONVENTION
MFN STATUS FOR CHINA
>>> for doc_idx, topics in model.top_doc_topics(doc_topic_matrix, docs=range(5),␣
↪top_n=2):
...     print(corpus[doc_idx].metadata['title'], ':', topics)
JOIN THE SENATE AND PASS A CONTINUING RESOLUTION : (9, 0)
MEETING THE CHALLENGE : (2, 0)
DISPOSING OF SENATE AMENDMENT TO H.R. 1643, EXTENSION OF MOST-FAVORED- NATION␣
↪TREATMENT FOR BULGARIA : (0, 9)
EXAMINING THE SPEAKER'S UPCOMING TRAVEL SCHEDULE : (0, 9)
FLOODING IN PENNSYLVANIA : (0, 9)
>>> for i, val in enumerate(model.topic_weights(doc_topic_matrix)):
...     print(i, val)
0 0.302796022302
1 0.0635617650602
2 0.0744927472417
3 0.0905778808867
4 0.0521162262192
5 0.0656303769725
6 0.0973516532757
7 0.112907245542
8 0.0680659204364
9 0.0725001620636
```

Visualize the model:

```
>>> model.termite_plot(doc_term_matrix, vectorizer.id_to_term,
...                    topics=-1,  n_terms=25, sort_terms_by='seriation')
```

Persist our topic model to disk:

```
>>> model.save('nmf-10topics.pkl')
```

Parameters

- **model** ({'nmf', 'lda', 'lsa'} or `sklearn.decomposition.<model>`) –
- **n_topics** (*int*) – number of topics in the model to be initialized
- ****kwargs** – variety of parameters used to initialize the model; see individual sklearn pages for full details

Raises `ValueError` – if `model` not in `{'nmf', 'lda', 'lsa'}` or is not an NMF, LatentDirichletAllocation, or TruncatedSVD instance

See also:

- http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.NMF.html
- http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.LatentDirichletAllocation.html
- http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.TruncatedSVD.html

get_doc_topic_matrix(*doc_term_matrix*, *normalize=True*)

Transform a document-term matrix into a document-topic matrix, where rows correspond to documents and columns to the topics in the topic model.

Parameters

- **doc_term_matrix** (*array-like or sparse matrix*) – corpus represented as a document-term matrix with shape (n_docs, n_terms); NOTE: LDA expects tf-weighting, while NMF and LSA may do better with tfidf-weighting!
- **normalize** (*bool*) – if True, the values in each row are normalized, i.e. topic weights on each document sum to 1

Returns document-topic matrix with shape (n_docs, n_topics)

Return type `numpy.ndarray`

termite_plot(*doc_term_matrix*, *id2term*, *topics=-1*, *sort_topics_by=u'index'*, *highlight_topics=None*, *n_terms=25*, *rank_terms_by=u'topic_weight'*, *sort_terms_by=u'seriation'*, *save=False*)

Make a "termite" plot for assessing topic models using a tabular layout to promote comparison of terms both within and across topics.

Parameters

- **doc_term_matrix** (`np.ndarray`-like or sparse matrix) – corpus represented as a document-term matrix with shape (n_docs, n_terms); may have tf- or tfidf-weighting
- **id2term** (*List[str] or dict*) – object that returns the term string corresponding to term id `i` through `id2term[i]`; could be a list of strings where the index represents the term id, such as that returned by `sklearn.feature_extraction.text.CountVectorizer.get_feature_names()`, or a mapping of term id: term string
- **topics** (*int or Sequence[int]*) – topic(s) to include in termite plot; if -1, all topics are included
- **sort_topics_by** (*{'index', 'weight'}*) –
- **highlight_topics** (*int or Sequence[int]*) – indices for up to 6 topics to visually highlight in the plot with contrasting colors
- **n_terms** (*int*) – number of top terms to include in termite plot
- **rank_terms_by** (*{'topic_weight', 'corpus_weight'}*) – value used to rank terms; the top-ranked `n_terms` are included in the plot
- **sort_terms_by** (*{'seriation', 'weight', 'index', 'alphabetical'}*) – method used to vertically sort the selected top `n_terms` terms; the default ("seriation") groups similar terms together, which facilitates cross-topic assessment
- **save** (*str*) – give the full /path/to/fname on disk to save figure

Returns axis on which termite plot is plotted

Return type `matplotlib.axes.Axes.axis`

Raises `ValueError` – if more than 6 topics are selected for highlighting, or an invalid value is passed for the sort_topics_by, rank_terms_by, and/or sort_terms_by params

References

See also:

viz.termite_plot

TODO: *rank_terms_by* other metrics, e.g. topic salience or relevance

top_doc_topics(*doc_topic_matrix*, *docs=-1*, *top_n=3*, *weights=False*)
Get the top top_n topics by weight per doc for docs in doc_topic_matrix.

Parameters

- **doc_topic_matrix** (numpy.ndarray) – document-topic matrix with shape (n_docs, n_topics), the result of calling get_doc_topic_matrix()
- **docs** (*int or Sequence[int]*) – docs for which to return top topics; if -1, all docs' top topics are returned
- **top_n** (*int*) – number of top topics to return per doc
- **weights** (*bool*) – if True, docs are returned with their corresponding (normalized) topic weights; otherwise, docs are returned without weights

Yields *Tuple[int, Tuple[int]] or Tuple[int, Tuple[Tuple[int, float]]]* – next tuple corresponding to a doc; the first element is the doc's index; if weights is False, the second element is a tuple of ints representing the top top_n related topics; otherwise, the second is a tuple of (int, float) pairs representing the top top_n related topics and their associated weights wrt the doc; for example:

```
>>> list(TopicModel.top_doc_topics(dtm, docs=(0, 1), top_n=2,␣
↪weights=False))
[(0, (1, 4)), (1, (3, 2))]
>>> list(TopicModel.top_doc_topics(dtm, docs=0, top_n=2,␣
↪weights=True))
[(0, ((1, 0.2855), (4, 0.2412)))]
```

top_topic_docs(*doc_topic_matrix*, *topics=-1*, *top_n=10*, *weights=False*)
Get the top top_n docs by weight per topic in doc_topic_matrix.

Parameters

- **doc_topic_matrix** (numpy.ndarray) – document-topic matrix with shape (n_docs, n_topics), the result of calling get_doc_topic_matrix()
- **topics** (*int or Sequence[int]*) – topic(s) for which to return top docs; if -1, all topics' docs are returned
- **top_n** (*int*) – number of top docs to return per topic
- **weights** (*bool*) – if True, docs are returned with their corresponding (normalized) topic weights; otherwise, docs are returned without weights

Yields *Tuple[int, Tuple[int]] or Tuple[int, Tuple[Tuple[int, float]]]* – next tuple corresponding to a topic; the first element is the topic's index; if weights is False, the second element is a tuple of ints representing the top top_n related docs; otherwise, the second is a tuple of (int, float) pairs representing the top top_n related docs and their associated weights wrt the topic; for example:

```
>>> list(TopicModel.top_doc_terms(dtm, topics=(0, 1), top_n=2,␣
↪weights=False))
[(0, (4, 2)), (1, (1, 3))]
>>> list(TopicModel.top_doc_terms(dtm, topics=0, top_n=2,␣
↪weights=True))
[(0, ((4, 0.3217), (2, 0.2154)))]
```

top_topic_terms(*id2term*, *topics=-1*, *top_n=10*, *weights=False*)
Get the top `top_n` terms by weight per topic in `model`.

Parameters

- **id2term** (*list(str) or dict*) – object that returns the term string corresponding to term id `i` through `id2term[i]`; could be a list of strings where the index represents the term id, such as that returned by `sklearn.feature_extraction.text.CountVectorizer.get_feature_names()`, or a mapping of term id: term string
- **topics** (*int or Sequence[int]*) – topic(s) for which to return top terms; if -1 (default), all topics' terms are returned
- **top_n** (*int*) – number of top terms to return per topic
- **weights** (*bool*) – if True, terms are returned with their corresponding topic weights; otherwise, terms are returned without weights

Yields *Tuple[int, Tuple[str]] or Tuple[int, Tuple[Tuple[str, float]]]* – next tuple corresponding to a topic; the first element is the topic's index; if `weights` is False, the second element is a tuple of str representing the top `top_n` related terms; otherwise, the second is a tuple of (str, float) pairs representing the top `top_n` related terms and their associated weights wrt the topic; for example:

```
>>> list(TopicModel.top_topic_terms(id2term, topics=(0, 1), top_n=2,
↪ weights=False))
[(0, ('foo', 'bar')), (1, ('bat', 'baz'))]
>>> list(TopicModel.top_topic_terms(id2term, topics=0, top_n=2,␣
↪weights=True))
[(0, (('foo', 0.1415), ('bar', 0.0986)))]
```

topic_weights(*doc_topic_matrix*)
Get the overall weight of topics across an entire corpus. Note: Values depend on whether topic weights per document in `doc_topic_matrix` were normalized, or not. I suppose either way makes sense... o_O

Parameters **doc_topic_matrix** (`numpy.ndarray`) – document-topic matrix with shape (n_docs, n_topics), the result of calling `get_doc_topic_matrix()`

Returns the ith element is the ith topic's overall weight

Return type `numpy.ndarray`

Representations

Represent documents as semantic networks, where nodes are individual terms or whole sentences.

`textacy.network.`**sents_to_semantic_network**(*sents*, *normalize='lemma'*, *edge_weighting='cosine'*)
Convert a list of sentences into a semantic network, where each sentence is represented by a node with edges linking it to other sentences weighted by the (cosine or jaccard) similarity of their constituent words.

Parameters

- **sents** (List[str] or List[`spacy.Span`]) –
- **normalize** (*str or callable*) – if 'lemma', lemmatize words in sents; if 'lower', lowercase word in sents; if false-y, use the form of words as they appear in sents; if a callable, must accept a `spacy.Token` and return a str, e.g. *textacy.spacy_utils.normalized_str()*; only applicable if `sents` is a List[`spacy.Span`]

- **edge_weighting** (*str {'cosine', 'jaccard'}, optional*) – similarity metric to use for weighting edges between sentences; if 'cosine', use the cosine similarity between sentences represented as tf-idf word vectors; if 'jaccard', use the set intersection divided by the set union of all words in a given sentence pair

Returns

nodes are the integer indexes of the sentences in the input `sents` list, *not* the actual text of the sentences!

Return type `networkx.Graph`

Notes

•If passing sentences as strings, be sure to filter out stopwords, punctuation, certain parts of speech, etc. beforehand

•Consider normalizing the strings so that like terms are counted together (see *normalized_str()*)

`textacy.network.`**terms_to_semantic_network**(*terms*, *normalize='lemma'*, *window_width=10*, *edge_weighting='cooc_freq'*)

Convert an ordered list of non-overlapping terms into a semantic network, where each term is represented by a node with edges linking it to other terms that co-occur within `window_width` terms of itself.

Parameters

- **terms** (List[str] or List[`spacy.Token`]) –
- **normalize** (*str or callable*) – if 'lemma', lemmatize terms; if 'lower', lowercase terms; if false-y, use the form of terms as they appear in doc; if a callable, must accept a `spacy.Token` and return a str, e.g. *textacy.spacy_utils.normalized_str()*; only applicable if `terms` is a List[`spacy.Token`]
- **window_width** (*int, optional*) – size of sliding window over *terms* that determines which are said to co-occur; if = 2, only adjacent terms will have edges in network
- **edge_weighting** (*str {'cooc_freq', 'binary'}, optional*) – if 'binary', all co-occurring terms will have network edges with weight = 1; if 'cooc_freq', edges will have a weight equal to the number of times that the connected nodes co-occur in a sliding window

Returns

nodes are terms, edges are for co-occurrences of terms

Return type `networkx.Graph`

Notes

•Be sure to filter out stopwords, punctuation, certain parts of speech, etc. from the terms list before passing it to this function

•Multi-word terms, such as named entities and compound nouns, must be merged into single strings or spacy.Tokens beforehand

•If terms are already strings, be sure to have normalized them so that like terms are counted together; for example, by applying *normalized_str()*

Represent a collection of spacy-processed texts as a document-term matrix of shape (# docs, # unique terms), with a variety of filtering, normalization, and term weighting schemes for the values.

class textacy.vsm.**Vectorizer**(*weighting=u'tf'*, *normalize=False*, *sublinear_tf=False*, *smooth_idf=True*, *vocabulary=None*, *min_df=1*, *max_df=1.0*, *min_ic=0.0*, *max_n_terms=None*)

Transform one or more tokenized documents into a document-term matrix of shape (# docs, # unique terms), with tf-, tf-idf, or binary-weighted values.

Stream a corpus with metadata from disk:

```
>>> cw = textacy.datasets.CapitolWords()
>>> text_stream, metadata_stream = textacy.fileio.split_record_fields(
...     cw.records(limit=1000), 'text', itemwise=False)
>>> corpus = textacy.Corpus('en', texts=text_stream, metadatas=metadata_stream)
>>> corpus
Corpus(1000 docs; 537742 tokens)
```

Tokenize and vectorize (the first half of) a corpus:

```
>>> terms_list = (doc.to_terms_list(ngrams=1, named_entities=True, as_
↪strings=True)
                  for doc in corpus[:500])
>>> vectorizer = Vectorizer(
...     weighting='tfidf', normalize=True, smooth_idf=True,
...     min_df=3, max_df=0.95, max_n_terms=100000)
>>> doc_term_matrix = vectorizer.fit_transform(terms_list)
>>> doc_term_matrix
<500x3811 sparse matrix of type '<class 'numpy.float64'>'
       with 54530 stored elements in Compressed Sparse Row format>
```

Tokenize and vectorize (the *other* half of) a corpus, using only the terms and weights learned in the previous step:

```
>>> terms_list = (doc.to_terms_list(ngrams=1, named_entities=True, as_
↪strings=True)
...                for doc in corpus[:500])
>>> doc_term_matrix = vectorizer.transform(terms_list)
>>> doc_term_matrix
<500x3811 sparse matrix of type '<class 'numpy.float64'>'
       with 44788 stored elements in Compressed Sparse Row format>
```

Parameters

- **weighting** (*{'tf', 'tfidf', 'binary'}*) – Weighting to assign to terms in the doc-term matrix. If 'tf', matrix values (i, j) correspond to the number of occurrences of term j in doc i; if 'tfidf', term frequencies (tf) are multiplied by their corresponding inverse document frequencies (idf); if 'binary', all non-zero values are set equal to 1.
- **normalize** (*bool*) – If True, normalize term frequencies by the L2 norms of the vectors.
- **binarize** (*bool*) – If True, set all term frequencies > 0 equal to 1.
- **sublinear_tf** (*bool*) – If True, apply sub-linear term-frequency scaling, i.e. tf => 1 + log(tf).
- **smooth_idf** (*bool*) – If True, add 1 to all document frequencies, equivalent to adding a single document to the corpus containing every unique term.

- **vocabulary** (*Dict[str, int] or Iterable[str]*) – Mapping of unique term string (str) to unique term id (int) or an iterable of term strings (which gets converted into a suitable mapping).
- **min_df** (*float or int*) – If float, value is the fractional proportion of the total number of documents, which must be in [0.0, 1.0]. If int, value is the absolute number. Filter terms whose document frequency is less than `min_df`.
- **max_df** (*float or int*) – If float, value is the fractional proportion of the total number of documents, which must be in [0.0, 1.0]. If int, value is the absolute number. Filter terms whose document frequency is greater than `max_df`.
- **min_ic** (*float*) – Filter terms whose information content is less than `min_ic`; value must be in [0.0, 1.0].
- **max_n_terms** (*int*) – Only include terms whose document frequency is within the top `max_n_terms`.

vocabulary
Dict[str, int]

is_fixed_vocabulary
bool

id_to_term
Dict[int, str]

feature_names
List[str]

feature_names
Array mapping from feature integer indices to feature name.

fit (*terms_list*)
Count terms and build up a vocabulary based on the terms found in the `terms_list`.

Parameters **terms_list** (*Iterable[Iterable[str]]*) – A sequence of tokenized documents, where each document is a sequence of (str) terms. For example:

```
>>> ([tok.lemma_ for tok in spacy_doc]
```

... for spacy_doc in spacy_docs) >>> ((ne.text for ne in extract.named_entities(doc)) ... for doc in corpus) >>> (tuple(ng.text for ng in

itertools.chain.from_iterable(extract.ngrams(doc, i) for i in range(1, 3)))

... for doc in docs)

Returns The instance that has just been fit.

Return type *Vectorizer*

fit_transform (*terms_list*)
Count terms and build up a vocabulary based on the terms found in the `terms_list`, then transform the `terms_list` into a document-term matrix with values weighted according to the parameters specified in `Vectorizer` initialization.

Parameters **terms_list** (*Iterable[Iterable[str]]*) – A sequence of tokenized documents, where each document is a sequence of (str) terms. For example:

```
>>> ([tok.lemma_ for tok in spacy_doc]
```

... for spacy_doc in spacy_docs) >>> ((ne.text for ne in extract.named_entities(doc)) ... for doc in corpus) >>> (tuple(ng.text for ng in

itertools.chain.from_iterable(extract.ngrams(doc, i) for i in range(1, 3)))

... for doc in docs)

Returns

The transformed document-term matrix. Rows correspond to documents and columns correspond to terms.

Return type `scipy.sparse.csr_matrix`

`id_to_term`

dict – Mapping of unique term id (int) to unique term string (str), i.e. the inverse of *`Vectorizer.vocabulary`*. This attribute is only generated if needed, and it is automatically kept in sync with the corresponding vocabulary.

`transform`(*terms_list*)

Transform the `terms_list` into a document-term matrix with values weighted according to the parameters specified in `Vectorizer` initialization.

Parameters **`terms_list`** (*`Iterable[Iterable[str]]`*) – A sequence of tokenized documents, where each document is a sequence of (str) terms. For example:

```
>>> ([tok.lemma_ for tok in spacy_doc]
```

... for spacy_doc in spacy_docs) >>> ((ne.text for ne in extract.named_entities(doc)) ... for doc in corpus) >>> (tuple(ng.text for ng in

itertools.chain.from_iterable(extract.ngrams(doc, i) for i in range(1, 3)))

... for doc in docs)

Returns

The transformed document-term matrix. Rows correspond to documents and columns correspond to terms.

Return type `scipy.sparse.csr_matrix`

Note: This requires an existing vocabulary, either built when calling *`Vectorizer.fit()`* or provided in `Vectorizer` initialization.

`textacy.vsm.`**`apply_idf_weighting`**(*doc_term_matrix*, *smooth_idf=True*)

Apply inverse document frequency (idf) weighting to a term-frequency (tf) weighted document-term matrix, optionally smoothing idf values.

Parameters

- **`doc_term_matrix`** (`scipy.sparse.csr_matrix <scipy.sparse.csr_matrix`) – M X N matrix, where M is the # of docs and N is the # of unique terms
- **`smooth_idf`** (*`bool`*) – if True, add 1 to all document frequencies, equivalent to adding a single document to the corpus containing every unique term

Returns

sparse matrix of shape (# docs, # unique terms), where value (i, j) is the tfidf weight of term j in doc i

Return type scipy.sparse.csr_matrix

textacy.vsm.**filter_terms_by_df**(*doc_term_matrix*, *term_to_id*, *max_df=1.0*, *min_df=1*, *max_n_terms=None*)

Filter out terms that are too common and/or too rare (by document frequency), and compactify the top max_n_terms in the id_to_term mapping accordingly. Borrows heavily from the sklearn.feature_extraction.text module.

Parameters

- **doc_term_matrix** (scipy.sparse.csr_matrix) – M X N matrix, where M is the # of docs and N is the # of unique terms.
- **term_to_id** (*Dict[str, int]*) – Mapping of term string to unique term id, e.g. *Vectorizer.vocabulary*.
- **min_df** (*float or int*) – if float, value is the fractional proportion of the total number of documents and must be in [0.0, 1.0]; if int, value is the absolute number; filter terms whose document frequency is less than min_df
- **max_df** (*float or int*) – if float, value is the fractional proportion of the total number of documents and must be in [0.0, 1.0]; if int, value is the absolute number; filter terms whose document frequency is greater than max_df
- **max_n_terms** (*int*) – only include terms whose *term* frequency is within the top *max_n_terms*

Returns

sparse matrix of shape (# docs, # unique *filtered* terms), where value (i, j) is the weight of term j in doc i

dict: id to term mapping, where keys are unique *filtered* integers as term ids and values are corresponding strings

Return type scipy.sparse.csr_matrix

Raises ValueError – if max_df or min_df or max_n_terms < 0

textacy.vsm.**filter_terms_by_ic**(*doc_term_matrix*, *term_to_id*, *min_ic=0.0*, *max_n_terms=None*)

Filter out terms that are too common and/or too rare (by information content), and compactify the top max_n_terms in the id_to_term mapping accordingly. Borrows heavily from the sklearn.feature_extraction.text module.

Parameters

- **doc_term_matrix** (scipy.sparse.csr_matrix) – M X N matrix, where M is the # of docs and N is the # of unique terms.
- **term_to_id** (*Dict[str, int]*) – Mapping of term string to unique term id, e.g. *Vectorizer.vocabulary*.
- **min_ic** (*float*) – filter terms whose information content is less than this value; must be in [0.0, 1.0]
- **max_n_terms** (*int*) – only include terms whose information content is within the top max_n_terms

Returns

sparse matrix of shape (# docs, # unique *filtered* terms), where value (i, j) is the weight of term j in doc i

dict: id to term mapping, where keys are unique *filtered* integers as term ids and values are corresponding strings

Return type scipy.sparse.csr_matrix

Raises ValueError – if min_ic not in [0.0, 1.0] or max_n_terms < 0

textacy.vsm.**get_doc_freqs**(*doc_term_matrix*, *normalized=True*)

Compute absolute or relative document frequencies for all terms in a term-document matrix.

Parameters

- **doc_term_matrix** (scipy.sparse.csr_matrix <scipy.sparse.csr_matrix) – M X N matrix, where M is the # of docs and N is the # of unique terms

  Note: Weighting on the terms doesn't matter! Could be 'tf' or 'tfidf' or 'binary' weighting, a term's doc freq will be the same

- **normalized** (*bool*) – if True, return normalized doc frequencies, i.e. doc counts divided by the total number of docs; if False, return absolute doc counts

Returns

array of absolute or relative document frequencies, with length equal to the # of unique terms, i.e. # of columns in doc_term_matrix

Return type numpy.ndarray

Raises ValueError – if doc_term_matrix doesn't have any non-zero entries

textacy.vsm.**get_information_content**(*doc_term_matrix*)

Compute information content for all terms in a term-document matrix. IC is a float in [0.0, 1.0], defined as -df * log2(df) - (1 - df) * log2(1 - df), where df is a term's normalized document frequency.

Parameters **doc_term_matrix** (scipy.sparse.csr_matrix <scipy.sparse.csr_matrix) – M X N matrix, where M is the # of docs and N is the # of unique terms

Note: Weighting on the terms doesn't matter! Could be 'tf' or 'tfidf' or 'binary' weighting, a term's information content will be the same

Returns

array of term information content values, with length equal to the # of unique terms, i.e. # of columns in doc_term_matrix

Return type numpy.ndarray

Raises ValueError – if doc_term_matrix doesn't have any non-zero entries

textacy.vsm.**get_term_freqs**(*doc_term_matrix*, *normalized=True*)

Compute absolute or relative term frequencies for all terms in a document-term matrix.

Parameters

- **doc_term_matrix** (scipy.sparse.csr_matrix <scipy.sparse.csr_matrix) – M X N matrix, where M is the # of docs and N is the # of unique terms

  Note: Weighting on the terms DOES matter! Only absolute term counts (rather than normalized term frequencies) should be used here

- **normalized** (*bool*) – if True, return normalized term frequencies, i.e. term counts divided by the total number of terms; if False, return absolute term counts

Returns

array of absolute or relative term frequencies, with length equal to the # of unique terms, i.e. # of columns in `doc_term_matrix`

Return type `numpy.ndarray`

Raises `ValueError` – if `doc_term_matrix` doesn't have any non-zero entries

Datasets

Capitol Words

A collection of ~11k (almost all) speeches given by the main protagonists of the 2016 U.S. Presidential election that had previously served in the U.S. Congress — including Hillary Clinton, Bernie Sanders, Barack Obama, Ted Cruz, and John Kasich — from January 1996 through June 2016.

Records include the following fields:

- `text`: full text of the Congressperson's remarks
- `title`: title of the speech, in all caps
- `date`: date on which the speech was given, as an ISO-standard string
- `speaker_name`: first and last name of the speaker
- `speaker_party`: political party of the speaker ('R' for Republican, 'D' for Democrat, and 'I' for Independent)
- `congress`: number of the Congress in which the speech was given; ranges continuously between 104 and 114
- `chamber`: chamber of Congress in which the speech was given; almost all are either 'House' or 'Senate', with a small number of 'Extensions'

This dataset was derived from data provided by the (now defunct) Sunlight Foundation's Capitol Words API.

class `textacy.datasets.capitol_words.`**`CapitolWords`**(*data_dir=u'/home/docs/checkouts/readthedocs.org/user_builds/textacy/envs/latest/lib/python2.7/site-packages/textacy-0.4.1-py2.7.egg/textacy/data/capitol_words'*)

Stream Congressional speeches from a compressed json file on disk, either as texts (str) or records (dict) with both text content and metadata.

Download a Python version-specific file from s3:

```
>>> cw = CapitolWords()
>>> cw.download()
>>> cw.info
{'data_dir': 'path/to/textacy/data/capitolwords',
 'description': 'Collection of ~11k speeches in the Congressional Record given by␣
↪notable U.S. politicians between Jan 1996 and Jun 2016.',
 'name': 'capitolwords',
 'site_url': 'http://sunlightlabs.github.io/Capitol-Words/'}
```

Iterate over speeches as plain texts or records with both text and metadata:

```
>>> for text in cw.texts(limit=5):
...     print(text)
>>> for record in cw.records(limit=5):
...     print(record['title'], record['date'])
...     print(record['text'])
```

Filter speeches by a variety of metadata fields and text length:

```
>>> for record in cw.records(speaker_name='Bernie Sanders', limit=1):
...     print(record['date'], record['text'])
>>> for record in cw.records(speaker_party='D', congress={110, 111, 112},
...                          chamber='Senate', limit=5):
...     print(record['speaker_name'], record['title'])
>>> for record in cw.records(speaker_name={'Barack Obama', 'Hillary Clinton'},
...                          date_range=('2002-01-01', '2002-12-31')):
...     print(record['speaker_name'], record['title'], record['date'])
>>> for text in cw.texts(min_len=50000):
...     print(len(text))
```

Stream speeches into a `textacy.Corpus`:

```
>>> text_stream, metadata_stream = textacy.fileio.split_record_fields(
...     cw.records(limit=100), 'text')
>>> c = textacy.Corpus('en', texts=text_stream, metadatas=metadata_stream)
>>> c
Corpus(100 docs; 70500 tokens)
```

Parameters **data_dir** (*str*) – Path to directory on disk under which compressed json files are stored.

min_date
str – Earliest date for which speeches are available, as an ISO-formatted string (YYYY-MM-DD).

max_date
str – Latest date for which speeches are available, as an ISO-formatted string (YYYY-MM-DD).

speaker_names
Set[str] – full names of all speakers included in corpus, e.g. *'Bernie Sanders'*

speaker_parties
Set[str] – all distinct political parties of speakers, e.g. *'R'*

chambers
Set[str] – all distinct chambers in which speeches were given, e.g. *'House'*

congresses
Set[int] – all distinct numbers of the congresses in which speeches were given, e.g. *114*

download(*force=False*)
Download a Python version-specific compressed json file from s3, and save it to disk under the `data_dir` directory.

Parameters **force** (*bool*) – Download the file, even if it already exists on disk.

filename
str – Full path on disk for CapitolWords data as compressed json file. `None` if file is not found, e.g. has not yet been downloaded.

records(*speaker_name=None*, *speaker_party=None*, *chamber=None*, *congress=None*, *date_range=None*, *min_len=None*, *limit=-1*)
Iterate over records (including text and metadata) in this dataset, optionally filtering by a variety of metadata and/or text length, in chronological order.

Parameters

- **speaker_name** (*str or Set[str]*) – Filter records by the speakers' name; see *speaker_names*.

- **speaker_party** (*str or Set[str]*) – Filter records by the speakers' party; see *speaker_parties*.
- **chamber** (*str or Set[str]*) – Filter records by the chamber in which they were given; see *chambers*.
- **congress** (*int or Set[int]*) – Filter records by the congress in which they were given; see *congresses*.
- **date_range** (*List[str] or Tuple[str]*) – Filter records by the date on which they were given. Both start and end date must be specified, but a null value for either will be replaced by the min/max date available in the dataset.
- **min_len** (*int*) – Filter records by the length (number of characters) of their text content.
- **limit** (*int*) – Return no more than `limit` records, in chronological order.

Yields *dict* –

Full text and metadata of next (by chronological order) record in dataset passing all filter params.

Raises `ValueError` – If any filtering options are invalid.

texts(*speaker_name=None*, *speaker_party=None*, *chamber=None*, *congress=None*, *date_range=None*, *min_len=None*, *limit=-1*)

Iterate over texts in this dataset, optionally filtering by a variety of metadata and/or text length, in chronological order.

Parameters

- **speaker_name** (*str or Set[str]*) – Filter texts by the speakers' name; see *speaker_names*.
- **speaker_party** (*str or Set[str]*) – Filter texts by the speakers' party; see *speaker_parties*.
- **chamber** (*str or Set[str]*) – Filter texts by the chamber in which they were given; see *chambers*.
- **congress** (*int or Set[int]*) – Filter texts by the congress in which they were given; see *congresses*.
- **date_range** (*List[str] or Tuple[str]*) – Filter texts by the date on which they were given. Both start and end date must be specified, but a null value for either will be replaced by the min/max date available in the dataset.
- **min_len** (*int*) – Filter texts by the length (number of characters) of their text content.
- **limit** (*int*) – Return no more than `limit` texts, chronological order.

Yields *str* –

Full text of next (by chronological order) text in dataset passing all filter params.

Raises `ValueError` – If any filtering options are invalid.

Supreme Court Decisions

A collection of ~8.4k (almost all) decisions issued by the U.S. Supreme Court from November 1946 through June 2016 — the "modern" era.

Records include the following fields:

- `text`: full text of the Court's decision
- `case_name`: name of the court case, in all caps
- `argument_date`: date on which the case was argued before the Court, as a string with format 'YYYY-MM-DD'
- `decision_date`: date on which the Court's decision was announced, as a string with format 'YYYY-MM-DD'
- `decision_direction`: ideological direction of the majority decision; either 'conservative', 'liberal', or 'unspecifiable'
- `maj_opinion_author`: name of the majority opinion's author, if available and identifiable, as an integer code whose mapping is given in `SupremeCourt.opinion_author_codes`
- `n_maj_votes`: number of justices voting in the majority
- `n_min_votes`: number of justices voting in the minority
- `issue`: subject matter of the case's core disagreement (e.g. affirmative action) rather than its legal basis (e.g. the equal protection clause), as a string code whose mapping is given in `SupremeCourt.issue_codes`
- `issue_area`: higher-level categorization of the issue (e.g. Civil Rights), as an integer code whose mapping is given in `SupremeCourt.issue_area_codes`
- `us_cite_id`: citation identifier for each case according to the official United States Reports; Note: There are ~300 cases with duplicate ids, and it's not clear if that's "correct" or a data quality problem

The text in this dataset was derived from FindLaw's searchable database of court cases: http://caselaw.findlaw.com/court/us-supreme-court

The metadata was extracted without modification from the Supreme Court Database: Harold J. Spaeth, Lee Epstein, et al. 2016 Supreme Court Database, Version 2016 Release 1. http://supremecourtdatabase.org. Its license is CC BY-NC 3.0 US: https://creativecommons.org/licenses/by-nc/3.0/us/

This corpus' creation was inspired by a blog post by Emily Barry: http://www.emilyinamillion.me/blog/2016/7/13/visualizing-supreme-court-topics-over-time

NOTE: The two datasets were merged through much munging and a carefully trained model using the `dedupe` package. The model's duplicate threshold was set so as to maximize the F-score where precision had twice as much weight as recall. Still, given occasionally baffling inconsistencies in case naming, citation ids, and decision dates, a very small percentage of texts may be incorrectly matched to metadata. Sorry.

class `textacy.datasets.supreme_court.`**`SupremeCourt`**(*data_dir=u'/home/docs/checkouts/readthedocs.org/user_builds/textacy/envs/latest/lib/python2.7/site-packages/textacy-0.4.1-py2.7.egg/textacy/data/supreme_court'*)

Stream U.S. Supreme Court decisions from a compressed json file on disk, either as texts (str) or records (dict) with both text content and metadata.

Download a Python version-specific file from s3:

```
>>> sc = SupremeCourt()
>>> sc.download()
>>> sc.info
{'data_dir': 'path/to/textacy/data/supreme_court',
 'description': 'Collection of ~8.4k decisions issued by the U.S. Supreme Court␣
↪between November 1946 and June 2016.',
 'name': 'supreme_court',
 'site_url': 'http://caselaw.findlaw.com/court/us-supreme-court'}
```

Iterate over decisions as plain texts or records with both text and metadata:

```
>>> for text in sc.texts(limit=1):
...     print(text)
>>> for record in sc.records(limit=1):
...     print(record['case_name'], record['decision_date'])
...     print(record['text'])
```

Filter decisions by a variety of metadata fields and text length:

```
>>> for record in sc.records(opinion_author=109, limit=1):  # Notorious RBG!
...     print(record['case_name'], record['decision_direction'], record['n_maj_
↪votes'])
>>> for record in sc.records(decision_direction='liberal',
...                          issue_area={1, 9, 10}, limit=10):
...     print(record['maj_opinion_author'], record['n_maj_votes'])
>>> for record in sc.records(opinion_author=102,
...                          date_range=('1990-01-01', '1999-12-31')):
...     print(record['case_name'], record['decision_date'])
...     print(sc.issue_codes[record['issue']])
>>> for text in sc.texts(min_len=50000):
...     print(len(text))
```

Stream decisions into a `textacy.Corpus`:

```
>>> text_stream, metadata_stream = textacy.fileio.split_record_fields(
...     sc.records(limit=100), 'text')
>>> c = textacy.Corpus('en', texts=text_stream, metadatas=metadata_stream)
>>> c
Corpus(100 docs; 615135 tokens)
```

Parameters **data_dir** (*str*) – path on disk containing corpus data; if None, textacy's default `data_dir` is used

min_date
str – Earliest date for which decisions are available, as an ISO-formatted string (YYYY-MM-DD).

max_date
str – Latest date for which decisions are available, as an ISO-formatted string (YYYY-MM-DD).

decision_directions
set[str] – all distinct decision directions, e.g. 'liberal'

opinion_author_codes
dict – mapping of majority opinion authors from integer code to (str) full name

issue_area_codes
dict – mapping of high-level issue area of the case's core disagreement from integer code to (str) description

issue_codes
dict – mapping of specific issue of the case's core disagreement from integer code to (str) description

download (*force=False*)
Download a Python version-specific compressed json file from s3, and save it to disk under the `data_dir` directory.

Parameters **force** (*bool*) – Download the file, even if it already exists on disk.

filename
str – Full path on disk for SupremeCourt data as compressed json file. `None` if file is not found, e.g. has not yet been downloaded.

records(*opinion_author=None*, *issue_area=None*, *decision_direction=None*, *date_range=None*, *min_len=None*, *limit=-1*)

Iterate over documents (including text and metadata) in the SupremeCourt corpus, optionally filtering by a variety of metadata and/or text length, in order of decision date.

Parameters

- **opinion_author** (*int or set[int]*) – filter cases by the name(s) of the majority opinion's author, coded as an integer whose mapping is given in *opinion_author_codes*
- **issue_area** (*int or set[int]*) – filter cases by the issue area of the case's subject matter, coded as an integer whose mapping is given in *issue_area_codes*
- **decision_direction** (*str or set[str]*) – filter cases by the ideological direction of the majority decision; see *decision_directions*
- **date_range** (*list[str] or tuple[str]*) – filter cases by the date on which they were decided; both start and end date must be specified, but a null value for either will be replaced by the min/max date available in the corpus
- **min_len** (*int*) – filter cases by the length (number of characters) in their text content
- **limit** (*int*) – return no more than *limit* cases, in order of decision date

Yields *dict* –

full text and metadata of next (by chronological order) court case in corpus passing all filter params

Raises ValueError – If any filtering options are invalid.

texts(*opinion_author=None*, *issue_area=None*, *decision_direction=None*, *date_range=None*, *min_len=None*, *limit=-1*)

Iterate over texts in the SupremeCourt corpus, optionally filtering by a variety of metadata and/or text length, in order of decision date.

Parameters

- **opinion_author** (*int or set[int]*) – filter cases by the name(s) of the majority opinion's author, coded as an integer whose mapping is given in *opinion_author_codes*
- **issue_area** (*int or set[int]*) – filter cases by the issue area of the case's subject matter, coded as an integer whose mapping is given in *issue_area_codes*
- **decision_direction** (*str or set[str]*) – filter cases by the ideological direction of the majority decision; see *decision_directions*
- **date_range** (*list[str] or tuple[str]*) – filter cases by the date on which they were decided; both start and end date must be specified, but a null value for either will be replaced by the min/max date available in the corpus
- **min_len** (*int*) – filter cases by the length (number of characters) in their text content
- **limit** (*int*) – return no more than *limit* cases, in order of decision date

Yields *str* –

full text of next (by chronological order) court case in corpus passing all filter params

Raises ValueError – If any filtering options are invalid.

Wikipedia

All articles for a given language- and version-specific Wikipedia site snapshot, as either texts (str) or records (dict) with both text and metadata.

Records include the following fields:

- `text`: text content of the article, with markup stripped out
- `title`: title of the Wikipedia article
- `page_id`: unique identifier of the page, usable in Wikimedia APIs
- `wiki_links`: a list of other article pages linked to from this page
- `ext_links`: a list of external URLs linked to from this page
- `categories`: a list of Wikipedia categories to which this page belongs

class `textacy.datasets.wikipedia.`**Wikipedia**(*data_dir=u'/home/docs/checkouts/readthedocs.org/user_builds/textacy/en*
packages/textacy-0.4.1-py2.7.egg/textacy/data/wikipedia', *lang=u'en'*, *version=u'latest'*)

Stream Wikipedia articles from versioned, language-specific database dumps, either as texts (str) or records (dict) with both text content and metadata.

Download a database dump for a given language and version:

```
>>> wp = Wikipedia(lang='en', version='latest')
>>> wp.download()
>>> wp.info
{'data_dir': 'path/to/textacy/data/wikipedia',
 'description': 'All articles for a given Wikimedia wiki, including wikitext␣
↪source and metadata, as a single database dump in XML format.',
 'name': 'wikipedia',
 'site_url': 'https://meta.wikimedia.org/wiki/Data_dumps'}
```

Iterate over articles as plain texts or records with both text and metadata:

```
>>> for text in wp.texts(limit=5):
...     print(text)
>>> for record in wp.records(limit=5):
...     print(record['title'], record['text'][:500])
```

Filter articles by text length:

```
>>> for text in wp.texts(min_len=1000, limit=1):
...     print(text)
```

Parameters

- **data_dir** (*str*) – Path to directory on disk under which database dump files are stored. Each file is expected at `{lang}wiki/{version}/{lang}wiki-{version}-pages-articles.xml.bz2` immediately under this directory.
- **lang** (*str*) – Standard two-letter language code, e.g. "en" => "English", "de" => "German". https://en.wikipedia.org/wiki/List_of_ISO_639-1_codes
- **version** (*str*) – Database dump version to use. Either "latest" for the most recently available version or a date formatted as "YYYYMMDD". Dumps are produced intermittently; check for available versions at https://meta.wikimedia.org/wiki/Data_dumps.

lang
str – Standard two-letter language code used in instantiation.

version
str – Database dump version used in instantiation.

filestub
str – The component of `filename` that is unique to this lang- and version-specific database dump.

filename
str – Full path on disk for the lang- and version-specific Wikipedia database dump, found under the `data_dir` directory.

download(*force=False*)
Download the Wikipedia database dump corresponding to the `lang` and `version` used in instantiation, and save it to disk under the `data_dir` directory.

Parameters **force** (*bool*) – Download the file, even if it already exists on disk.

filename
str – Full path on disk for Wikipedia database dump corresponding to the `lang` and `version` used in instantiation. `None` if file not found.

records(*min_len=100*, *limit=-1*, *fast=False*)
Iterate over the pages in a Wikipedia articles database dump (`*articles.xml.bz2`), yielding one page whose structure and content have been parsed, as a dict.

Parameters

- **min_len** (*int*) – minimum length in chars that a page must have for it to be returned; too-short pages are skipped
- **limit** (*int*) – maximum number of pages (passing `min_len`) to yield; if -1, all pages in the db dump are iterated over (optional)
- **fast** (*bool*) – If True, text is extracted using a faster method but which gives lower quality results. Otherwise, a slower but better method is used to extract article text.

Yields *dict* –

the next page's parsed content, including key:value pairs for 'title', 'page_id', 'text', 'categories', 'wiki_links', 'ext_links'

Notes

This function requires mwparserfromhell

texts(*min_len=100*, *limit=-1*)
Iterate over the pages in a Wikipedia articles database dump (`*articles.xml.bz2`), yielding the text of a page, one at a time.

Parameters

- **min_len** (*int*) – minimum length in chars that a page must have for it to be returned; too-short pages are skipped (optional)
- **limit** (*int*) – maximum number of pages (passing *min_len*) to yield; if -1, all pages in the db dump are iterated over (optional)

Yields *str* – plain text for the next page in the wikipedia database dump

Notes

Page and section titles appear immediately before the text content that they label, separated by an empty line.

textacy.datasets.wikipedia.**get_delimited_spans**(*wikitext*, *open_delim=u'[['*, *close_delim=u']]'*)

Parameters

- **wikitext** (*str*) –
- **open_delim** (*str*) –
- **close_delim** (*str*) –

Yields *Tuple[int, int]* –

start and end index of next span delimited by `open_delim` on the left and `close_delim` on the right

textacy.datasets.wikipedia.**remove_templates**(*wikitext*)

Return `wikitext` with all wikimedia markup templates removed, where templates are identified by opening '{{' and closing '}}'.

See also:

http://meta.wikimedia.org/wiki/Help:Template

textacy.datasets.wikipedia.**replace_external_links**(*wikitext*)

Replace external links of the form `[URL text]` with just `text` if present or just `URL` if not.

See also:

https://www.mediawiki.org/wiki/Help:Links#External_links

textacy.datasets.wikipedia.**replace_internal_links**(*wikitext*)

Replace internal links of the form `[[title |...|label]]trail` with just `label`.

textacy.datasets.wikipedia.**strip_markup**(*wikitext*)

Strip Wikimedia markup from the text content of a Wikipedia page and return the page as plain-text.

Parameters **wikitext** (*str*) –

Returns str

Supreme Court Decisions

A collection of ~8.4k (almost all) decisions issued by the U.S. Supreme Court from November 1946 through June 2016 — the "modern" era.

Records include the following fields:

- `text`: full text of the Court's decision
- `case_name`: name of the court case, in all caps
- `argument_date`: date on which the case was argued before the Court, as a string with format 'YYYY-MM-DD'
- `decision_date`: date on which the Court's decision was announced, as a string with format 'YYYY-MM-DD'

- `decision_direction`: ideological direction of the majority decision; either 'conservative', 'liberal', or 'unspecifiable'
- `maj_opinion_author`: name of the majority opinion's author, if available and identifiable, as an integer code whose mapping is given in `SupremeCourt.opinion_author_codes`
- `n_maj_votes`: number of justices voting in the majority
- `n_min_votes`: number of justices voting in the minority
- `issue`: subject matter of the case's core disagreement (e.g. affirmative action) rather than its legal basis (e.g. the equal protection clause), as a string code whose mapping is given in `SupremeCourt.issue_codes`
- `issue_area`: higher-level categorization of the issue (e.g. Civil Rights), as an integer code whose mapping is given in `SupremeCourt.issue_area_codes`
- `us_cite_id`: citation identifier for each case according to the official United States Reports; Note: There are ~300 cases with duplicate ids, and it's not clear if that's "correct" or a data quality problem

The text in this dataset was derived from FindLaw's searchable database of court cases: http://caselaw.findlaw.com/court/us-supreme-court

The metadata was extracted without modification from the Supreme Court Database: Harold J. Spaeth, Lee Epstein, et al. 2016 Supreme Court Database, Version 2016 Release 1. http://supremecourtdatabase.org. Its license is CC BY-NC 3.0 US: https://creativecommons.org/licenses/by-nc/3.0/us/

This corpus' creation was inspired by a blog post by Emily Barry: http://www.emilyinamillion.me/blog/2016/7/13/visualizing-supreme-court-topics-over-time

NOTE: The two datasets were merged through much munging and a carefully trained model using the `dedupe` package. The model's duplicate threshold was set so as to maximize the F-score where precision had twice as much weight as recall. Still, given occasionally baffling inconsistencies in case naming, citation ids, and decision dates, a very small percentage of texts may be incorrectly matched to metadata. Sorry.

class `textacy.datasets.supreme_court.`**`SupremeCourt`**(*data_dir=u'/home/docs/checkouts/readthedocs.org/user_builds/packages/textacy-0.4.1-py2.7.egg/textacy/data/supreme_court'*)

Stream U.S. Supreme Court decisions from a compressed json file on disk, either as texts (str) or records (dict) with both text content and metadata.

Download a Python version-specific file from s3:

```
>>> sc = SupremeCourt()
>>> sc.download()
>>> sc.info
{'data_dir': 'path/to/textacy/data/supreme_court',
 'description': 'Collection of ~8.4k decisions issued by the U.S. Supreme Court
↪between November 1946 and June 2016.',
 'name': 'supreme_court',
 'site_url': 'http://caselaw.findlaw.com/court/us-supreme-court'}
```

Iterate over decisions as plain texts or records with both text and metadata:

```
>>> for text in sc.texts(limit=1):
...     print(text)
>>> for record in sc.records(limit=1):
...     print(record['case_name'], record['decision_date'])
...     print(record['text'])
```

Filter decisions by a variety of metadata fields and text length:

```
>>> for record in sc.records(opinion_author=109, limit=1):  # Notorious RBG!
...     print(record['case_name'], record['decision_direction'], record['n_maj_
↪votes'])
>>> for record in sc.records(decision_direction='liberal',
...                          issue_area={1, 9, 10}, limit=10):
...     print(record['maj_opinion_author'], record['n_maj_votes'])
>>> for record in sc.records(opinion_author=102,
...                          date_range=('1990-01-01', '1999-12-31')):
...     print(record['case_name'], record['decision_date'])
...     print(sc.issue_codes[record['issue']])
>>> for text in sc.texts(min_len=50000):
...     print(len(text))
```

Stream decisions into a `textacy.Corpus`:

```
>>> text_stream, metadata_stream = textacy.fileio.split_record_fields(
...     sc.records(limit=100), 'text')
>>> c = textacy.Corpus('en', texts=text_stream, metadatas=metadata_stream)
>>> c
Corpus(100 docs; 615135 tokens)
```

Parameters **data_dir** (*str*) – path on disk containing corpus data; if None, textacy's default `data_dir` is used

min_date
str – Earliest date for which decisions are available, as an ISO-formatted string (YYYY-MM-DD).

max_date
str – Latest date for which decisions are available, as an ISO-formatted string (YYYY-MM-DD).

decision_directions
set[str] – all distinct decision directions, e.g. 'liberal'

opinion_author_codes
dict – mapping of majority opinion authors from integer code to (str) full name

issue_area_codes
dict – mapping of high-level issue area of the case's core disagreement from integer code to (str) description

issue_codes
dict – mapping of specific issue of the case's core disagreement from integer code to (str) description

download(*force=False*)
Download a Python version-specific compressed json file from s3, and save it to disk under the `data_dir` directory.

Parameters **force** (*bool*) – Download the file, even if it already exists on disk.

filename
str – Full path on disk for SupremeCourt data as compressed json file. `None` if file is not found, e.g. has not yet been downloaded.

records(*opinion_author=None*, *issue_area=None*, *decision_direction=None*, *date_range=None*, *min_len=None*, *limit=-1*)
Iterate over documents (including text and metadata) in the SupremeCourt corpus, optionally filtering by a variety of metadata and/or text length, in order of decision date.

Parameters

- **opinion_author** (*int or set[int]*) – filter cases by the name(s) of the majority opinion's author, coded as an integer whose mapping is given in *opinion_author_codes*
- **issue_area** (*int or set[int]*) – filter cases by the issue area of the case's subject matter, coded as an integer whose mapping is given in *issue_area_codes*
- **decision_direction** (*str or set[str]*) – filter cases by the ideological direction of the majority decision; see *decision_directions*
- **date_range** (*list[str] or tuple[str]*) – filter cases by the date on which they were decided; both start and end date must be specified, but a null value for either will be replaced by the min/max date available in the corpus
- **min_len** (*int*) – filter cases by the length (number of characters) in their text content
- **limit** (*int*) – return no more than *limit* cases, in order of decision date

Yields *dict* –

full text and metadata of next (by chronological order) court case in corpus passing all filter params

Raises `ValueError` – If any filtering options are invalid.

texts(*opinion_author=None*, *issue_area=None*, *decision_direction=None*, *date_range=None*, *min_len=None*, *limit=-1*)

Iterate over texts in the SupremeCourt corpus, optionally filtering by a variety of metadata and/or text length, in order of decision date.

Parameters

- **opinion_author** (*int or set[int]*) – filter cases by the name(s) of the majority opinion's author, coded as an integer whose mapping is given in *opinion_author_codes*
- **issue_area** (*int or set[int]*) – filter cases by the issue area of the case's subject matter, coded as an integer whose mapping is given in *issue_area_codes*
- **decision_direction** (*str or set[str]*) – filter cases by the ideological direction of the majority decision; see *decision_directions*
- **date_range** (*list[str] or tuple[str]*) – filter cases by the date on which they were decided; both start and end date must be specified, but a null value for either will be replaced by the min/max date available in the corpus
- **min_len** (*int*) – filter cases by the length (number of characters) in their text content
- **limit** (*int*) – return no more than *limit* cases, in order of decision date

Yields *str* –

full text of next (by chronological order) court case in corpus passing all filter params

Raises `ValueError` – If any filtering options are invalid.

Oxford Text Archive

A collection of ~2.7k Creative Commons texts from the Oxford Text Archive, containing primarily English-language 16th-20th century literature and history.

Record include the following fields:

- `text`: full text of the literary work

- `title`: title of the literary work
- `author`: author(s) of the literary work
- `year`: year that the literary work was published
- `url`: url at which literary work can be found online via the OTA

This dataset was compiled by [DAVID?] Mimno from the Oxford Text Archive and stored in his GitHub repo to avoid unnecessary scraping of the OTA site. It is downloaded from that repo, and excluding some light cleaning of its metadata, is reproduced exactly here.

class `textacy.datasets.oxford_text_archive.`**`OxfordTextArchive`**(*data_dir=u'/home/docs/checkouts/readthedocs.org/user_builds/textacy/envs/0.4.1/lib/python2.7/site-packages/textacy-0.4.1-py2.7.egg/textacy/data/oxford_text_archive'*)

Stream literary works from a zip file on disk, either as texts (str) or records (dict) with both text content and metadata.

Download the zip file from its GitHub repository:

```
>>> ota = OxfordTextArchive()
>>> ota.download()
>>> ota.info
{'data_dir': 'path/to/textacy/data/oxford_text_archive',
 'description': 'Collection of ~2.7k Creative Commons texts from the Oxford Text␣
↪Archive, containing primarily English-language 16th-20th century literature and␣
↪history.',
 'name': 'oxford_text_archive',
 'site_url': 'https://ota.ox.ac.uk/'}
```

Iterate over literary works as plain texts or records with both text and metadata:

```
>>> for text in ota.texts(limit=5):
...     print(text[:400])
>>> for record in ota.records(limit=5):
...     print(record['title'], record['year'])
...     print(record['text'][:400])
```

Filter literary works by a variety of metadata fields and text length:

```
>>> for record in ota.records(author='Shakespeare, William', limit=1):
...     print(record['year'], record['text'])
>>> for record in ota.records(date_range=('1900-01-01', '2000-01-01'), limit=5):
...     print(record['year'], record['author'])
>>> for text in ota.texts(min_len=4000000):
...     print(len(text))
...     print(text[:200], '...')
```

Stream literary works into a `textacy.Corpus`:

```
>>> text_stream, metadata_stream = textacy.fileio.split_record_fields(
...     ota.records(limit=10), 'text')
>>> c = textacy.Corpus('en', texts=text_stream, metadatas=metadata_stream)
>>> c
Corpus(10 docs; 686881 tokens)
```

Parameters **`data_dir`** (*`str`*) – Path to directory on disk under which dataset's file is stored.

min_date
str – Earliest date for which speeches are available, as an ISO-formatted string (YYYY-MM-DD).

max_date
str – Latest date for which speeches are available, as an ISO-formatted string (YYYY-MM-DD).

authors
Set[str] – Full names of all distinct authors included in this dataset, e.g. `'Shakespeare, William'`.

download(*force=False*)
Download dataset from `DOWNLOAD_ROOT` and save it to disk under the `OxfordTextArchive.data_dir` directory.

Parameters **force** (*bool*) – If True, download the file, even if it already exists on disk.

filename
str – Full path on disk for OxfordTextArchive data as a zip archive file. `None` if file is not found, e.g. has not yet been downloaded.

records(*author=None*, *date_range=None*, *min_len=None*, *limit=-1*)
Iterate over records (including text and metadata) in this dataset, optionally filtering by a variety of metadata and/or text length.

Parameters

- **author** (*str or Set[str]*) – Filter records by the authors' name; see *authors*.
- **date_range** (*List[str] or Tuple[str]*) – Filter records by the date on which it was published; both start and end date must be specified, but a null value for either will be replaced by the min/max date available in the dataset.
- **min_len** (*int*) – Filter records by the length (number of characters) of their text content.
- **limit** (*int*) – Return no more than `limit` records.

Yields *dict* –

Text and metadata of next document in dataset passing all filter params.

Raises `ValueError` – If any filtering options are invalid.

texts(*author=None*, *date_range=None*, *min_len=None*, *limit=-1*)
Iterate over texts in the dataset, optionally filtering by a variety of metadata and/or text length.

Parameters

- **author** (*str or Set[str]*) – Filter texts by the authors' name; see *authors*.
- **date_range** (*List[str] or Tuple[str]*) – Filter texts by the date on which it was published; both start and end date must be specified, but a null value for either will be replaced by the min/max date available in the dataset.
- **min_len** (*int*) – Filter texts by the length (number of characters) of their text content.
- **limit** (*int*) – Return no more than `limit` texts.

Yields *str* – Full text of next document in dataset passing all filter params.

Raises `ValueError` – If any filtering options are invalid.

File IO

Module with functions for reading content from disk in common formats.

textacy.fileio.read.**read_csv**(*filepath*, *encoding=None*, *dialect=u'excel'*, *delimiter=u','*)
Iterate over a stream of rows, where each row is an iterable of strings and/or numbers with individual values separated by `delimiter`.

Parameters

- **filepath** (*str*) – /path/to/file on disk from which rows will be streamed
- **encoding** (*str*) –
- **dialect** (*str*) – a grouping of formatting parameters that determine how the tabular data is parsed when reading/writing; if 'infer', the first 1024 bytes of the file is analyzed, producing a best guess for the correct dialect
- **delimiter** (*str*) – 1-character string used to separate fields in a row

Yields *List[obj]* – next row, whose elements are strings and/or numbers

See also:

https://docs.python.org/3/library/csv.html#csv.reader

textacy.fileio.read.**read_file**(*filepath*, *mode=u'rt'*, *encoding=None*)
Read the full contents of a file. Files compressed with gzip, bz2, or lzma are handled automatically.

textacy.fileio.read.**read_file_lines**(*filepath*, *mode=u'rt'*, *encoding=None*)
Read the contents of a file, line by line. Files compressed with gzip, bz2, or lzma are handled automatically.

textacy.fileio.read.**read_json**(*filepath*, *mode=u'rt'*, *encoding=None*, *prefix=u''*)
Iterate over JSON objects matching the field given by `prefix`. Useful for reading a large JSON array one item (with `prefix='item'`) or sub-item (`prefix='item.fieldname'`) at a time.

Parameters

- **filepath** (*str*) – /path/to/file on disk from which json items will be streamed, such as items in a JSON array; for example:

```
[
    {"title": "Harrison Bergeron", "text": "The year was 2081, and
↪everybody was finally equal."},
    {"title": "2BR02B", "text": "Everything was perfectly swell."}
]
```

- **mode** (*str, optional*) –
- **encoding** (*str, optional*) –
- **prefix** (*str, optional*) – if '', the entire JSON object will be read in at once; if 'item', each item in a top-level array will be read in successively; if 'item.text', each array item's 'text' value will be read in successively

Yields

next matching JSON object; could be a dict, list, int, float, str, depending on the value of `prefix`

Notes

Refer to `ijson` at https://pypi.python.org/pypi/ijson/ for usage details.

textacy.fileio.read.**read_json_lines**(*filepath*, *mode=u'rt'*, *encoding=None*)
Iterate over a stream of JSON objects, where each line of file `filepath` is a valid JSON object but no JSON object (e.g. array) exists at the top level.

Parameters

- **filepath** (*str*) – /path/to/file on disk from which json objects will be streamed, where each line in the file must be its own json object; for example:

```
{"title": "Harrison Bergeron", "text": "The year was 2081, and
↪everybody was finally equal."}

{"title": "2BR02B", "text": "Everything was perfectly swell."}
```

- **mode** (*str, optional*) –
- **encoding** (*str, optional*) –

Yields *dict* – next valid JSON object, converted to native Python equivalent

textacy.fileio.read.**read_json_mash**(*filepath*, *mode='rt'*, *encoding=None*, *buffersize=2048*)

Iterate over a stream of JSON objects, all of them mashed together, end-to-end, on a single line of a file. Bad form, but still manageable.

Parameters

- **filepath** (*str*) – /path/to/file on disk from which json objects will be streamed, where all json objects are mashed together, end-to-end, on a single line,; for example:

```
{"title": "Harrison Bergeron", "text": "The year was 2081, and
↪everybody was finally equal."}{"title": "2BR02B", "text":
↪"Everything was perfectly swell."}
```

- **mode** (*str, optional*) –
- **encoding** (*str, optional*) –
- **buffersize** (*int, optional*) – number of bytes to read in as a chunk

Yields *dict* – next valid JSON object, converted to native Python equivalent

textacy.fileio.read.**read_spacy_docs**(*spacy_vocab*, *filepath*)

Stream `spacy.Doc` s from disk at `filepath` where they were serialized using Spacy's `spacy.Doc.to_bytes()` functionality.

Parameters

- **spacy_vocab** (`spacy.Vocab`) – the spacy vocab object used to serialize the docs in `filepath`
- **filepath** (*str*) – /path/to/file on disk from which spacy docs will be streamed

Yields the next deserialized `spacy.Doc`

textacy.fileio.read.**read_sparse_csc_matrix**(*filepath*)

Read the data, indices, indptr, and shape arrays from a `.npz` file on disk at `filepath`, and return an instantiated `scipy.sparse.csc_matrix`.

textacy.fileio.read.**read_sparse_csr_matrix**(*filepath*)

Read the data, indices, indptr, and shape arrays from a `.npz` file on disk at `filepath`, and return an instantiated `scipy.sparse.csr_matrix`.

Functions for writing content to disk in common formats.

textacy.fileio.write.**write_csv**(*rows*, *filepath*, *encoding=None*, *auto_make_dirs=False*, *dialect='excel'*, *delimiter=','*)

Iterate over a sequence of rows, where each row is an iterable of strings and/or numbers, writing each to a separate line in file `filepath` with individual values separated by `delimiter`.

Parameters

- **rows** (*Iterable[Iterable]*) – iterable of iterables of strings and/or numbers to write to disk; for example:

```
[['That was a great movie!', 0.9],
 ['The movie was okay, I guess.', 0.2],
 ['Worst. Movie. Ever.', -1.0]]
```

- **filepath** (*str*) – /path/to/file on disk where rows will be written
- **encoding** (*str*) –
- **auto_make_dirs** (*bool*) –
- **dialect** (*str*) – a grouping of formatting parameters that determine how the tabular data is parsed when reading/writing
- **delimiter** (*str*) – 1-character string used to separate fields in a row

See also:

https://docs.python.org/3/library/csv.html#csv.writer

Note: Here, CSV is used as a catch-all term for *any* delimited file format, and `delimiter=','` is merely the function's default value. Other common delimited formats are TSV (tab-separated-value, with `delimiter='\t'`) and PSV (pipe-separated-value, with `delimiter='|'`.

---

textacy.fileio.write.**write_file**(*content*, *filepath*, *mode=u'wt'*, *encoding=None*, *auto_make_dirs=False*)

Write `content` to disk at `filepath`. Files with appropriate extensions are compressed with gzip or bz2 automatically. Any intermediate folders not found on disk may automatically be created.

See also:

open_sesame()

textacy.fileio.write.**write_file_lines**(*lines*, *filepath*, *mode=u'wt'*, *encoding=None*, *auto_make_dirs=False*)

Write the content in `lines` to disk at `filepath`, line by line. Files with appropriate extensions are compressed with gzip or bz2 automatically. Any intermediate folders not found on disk may automatically be created.

textacy.fileio.write.**write_json**(*json_object*, *filepath*, *mode=u'wt'*, *encoding=None*, *auto_make_dirs=False*, *ensure_ascii=False*, *indent=None*, *separators=(u', ', u':')*, *sort_keys=False*)

Write JSON object all at once to disk at `filepath`.

Parameters

- **json_object** (*json*) – valid JSON object to be written
- **filepath** (*str*) – /path/to/file on disk to which json object will be written, such as a JSON array; for example:

```
[
    {"title": "Harrison Bergeron", "text": "The year was 2081, and
↪everybody was finally equal."},
    {"title": "2BR02B", "text": "Everything was perfectly swell."}
]
```

- **mode** (*str*) –

- **encoding** (*str*) –
- **auto_make_dirs** (*bool*) –
- **indent** (*int or str*) –
- **ensure_ascii** (*bool*) –
- **separators** (*tuple[str]*) –
- **sort_keys** (*bool*) –

See also:

https://docs.python.org/3/library/json.html#json.dump

textacy.fileio.write.**write_json_lines**(*json_objects*, *filepath*, *mode='wt'*, *encoding=None*, *auto_make_dirs=False*, *ensure_ascii=False*, *separators=(', ', ':')*, *sort_keys=False*)

Iterate over a stream of JSON objects, writing each to a separate line in file `filepath` but without a top-level JSON object (e.g. array).

Parameters

- **json_objects** (*iterable[json]*) – iterable of valid JSON objects to be written
- **filepath** (*str*) – /path/to/file on disk to which JSON objects will be written, where each line in the file is its own json object; for example:

```
{"title": "Harrison Bergeron", "text": "The year was 2081, and
↪everybody was finally equal."}

{"title": "2BR02B", "text": "Everything was perfectly swell."}
```

- **mode** (*str*) –
- **encoding** (*str*) –
- **auto_make_dirs** (*bool*) –
- **ensure_ascii** (*bool*) –
- **separators** (*tuple[str]*) –
- **sort_keys** (*bool*) –

See also:

https://docs.python.org/3/library/json.html#json.dump

textacy.fileio.write.**write_spacy_docs**(*spacy_docs*, *filepath*, *auto_make_dirs=False*)

Serialize a sequence of `spacy.Doc` s to disk at `filepath` using Spacy's `spacy.Doc.to_bytes()` functionality.

Parameters

- **spacy_docs** (`spacy.Doc` or iterable(`spacy.Doc`)) – a single spacy doc or a sequence of spacy docs to serialize to disk at `filepath`
- **filepath** (*str*) – /path/to/file on disk to which spacy docs will be streamed
- **auto_make_dirs** (*bool*) –

textacy.fileio.write.**write_sparse_matrix**(*matrix*, *filepath*, *compressed=True*)

Write a `scipy.sparse.csr_matrix` or `scipy.sparse.csc_matrix` to disk at `filepath`, optionally compressed.

Parameters

- **matrix** (scipy.sparse.csr_matrix or scipy.sparse.csr_matrix) –
- **filepath** (*str*) – /path/to/file on disk to which matrix objects will be written; if filepath does not end in .npz, that extension is automatically appended to the name
- **compressed** (*bool*) – if True, save arrays into a single file in compressed .npz format

See also:

http://docs.scipy.org/doc/numpy-1.10.0/reference/generated/numpy.savez.html

See also:

http://docs.scipy.org/doc/numpy-1.10.0/reference/generated/numpy.savez_compressed.html

textacy.fileio.write.**write_streaming_download_file**(*url*, *filepath*, *mode=u'wt'*, *encoding=None*, *auto_make_dirs=False*, *chunk_size=1024*)

Download content from url in a stream; write successive chunks of size chunk_size bytes to disk at filepath. Files with appropriate extensions are compressed with gzip or bz2 automatically. Any intermediate folders not found on disk may automatically be created.

See also:

open_sesame()

textacy.fileio.utils.**coerce_content_type**(*content*, *file_mode*)

If the *content* to be written to file and the *file_mode* used to open it are incompatible (either bytes with text mode or unicode with bytes mode), try to coerce the content type so it can be written.

textacy.fileio.utils.**get_filenames**(*dirname*, *match_substr=None*, *ignore_substr=None*, *match_regex=None*, *ignore_regex=None*, *extension=None*, *ignore_invisible=True*, *recursive=False*)

Yield full paths of files on disk under directory dirname, optionally filtering for or against particular substrings or file extensions and crawling all subdirectories.

Parameters

- **dirname** (*str*) – /path/to/dir on disk where files to read are saved
- **match_substr** (*str*) – match only files with given substring (DEPRECATED; use match_regex)
- **ignore_substr** (*str*) – match only files *without* given substring (DEPRECATED; use ignore_regex)
- **match_regex** (*str*) – include files whose names match this regex pattern
- **ignore_regex** (*str*) – include files whose names do *not* match this regex pattern
- **extension** (*str*) – if files only of a certain type are wanted, specify the file extension (e.g. ".txt")
- **ignore_invisible** (*bool*) – if True, ignore invisible files, i.e. those that begin with a period
- **recursive** (*bool*) – if True, iterate recursively through all files in subdirectories; otherwise, only return files directly under dirname

Yields *str* – next file's name, including the full path on disk

Raises OSError – if dirname is not found on disk

textacy.fileio.utils.**make_dirs**(*filepath*, *mode*)

If writing `filepath` to a directory that doesn't exist, all intermediate directories will be created as needed.

textacy.fileio.utils.**open_sesame**(*filepath*, *mode=u'rt'*, *encoding=None*, *auto_make_dirs=False*, *errors=None*, *newline=None*)

Open file `filepath`. Compression (if any) is inferred from the file extension ('.gz', '.bz2', or '.xz') and handled automatically; '~', '.', and/or '..' in paths are automatically expanded; if writing to a directory that doesn't exist, all intermediate directories can be created automatically, as needed.

open_sesame may be used as a drop-in replacement for the built-in *open*.

Parameters

- **filepath** (*str*) – path on disk (absolute or relative) of the file to open
- **mode** (*str*) – optional string specifying the mode in which `filepath` is opened
- **encoding** (*str*) – optional name of the encoding used to decode or encode `filepath`; only applicable in text mode
- **errors** (*str*) – optional string specifying how encoding/decoding errors are handled; only applicable in text mode
- **newline** (*str*) – optional string specifying how universal newlines mode works; only applicable in text mode
- **auto_make_dirs** (*bool*) – if True, automatically create (sub)directories if not already present in order to write *filepath*

Returns file object

textacy.fileio.utils.**split_record_fields**(*items*, *content_field*, *itemwise=False*)

Split records' content (text) field from associated metadata fields, but keep them paired together for convenient loading into a `textacy.Doc <textacy.doc.Doc>` (with `itemwise = True`) or `textacy.Corpus <textacy.corpus.Corpus>` (with `itemwise = False`). Output format depends on the form of the input items (dicts vs. lists) and the value for `itemwise`.

Parameters

- **items** (*Iterable[dict] or Iterable[list]*) – an iterable of dicts, e.g. as read from disk by *read_json_lines()*, or an iterable of lists, e.g. as read from disk by *read_csv()*
- **content_field** (*str or int*) – if str, key in each dict item whose value is the item's content (text); if int, index of the value in each list item corresponding to the item's content (text)
- **itemwise** (*bool*) – if True, content + metadata are paired item-wise as an iterable of (content, metadata) 2-tuples; if False, content + metadata are paired by position in two parallel iterables in the form of a (iterable(content), iterable(metadata)) 2-tuple

Returns

if itemwise is True and items is an iterable of dicts; the first element in each tuple is the item's content, the second element is its metadata as a dictionary

generator(Tuple[str, list]): if itemwise is True and items is an iterable of lists; the first element in each tuple is the item's content, the second element is its metadata as a list

Tuple[Iterable[str], Iterable[dict]]: if itemwise is False and items is an iterable of dicts; the first element of the tuple is an iterable of items' contents, the second is an iterable of their metadata dicts

Tuple[Iterable[str], Iterable[list]]: if `itemwise` is False and `items` is an iterable of lists; the first element of the tuple is an iterable of items' contents, the second is an iterable of their metadata lists

Return type generator(Tuple[str, dict])

`textacy.fileio.utils.`**`unzip`**(*seq*)

Borrowed from `toolz.sandbox.core.unzip`, but using cytoolz instead of toolz to avoid the additional dependency.

Visualization

`textacy.viz.termite.`**`draw_termite_plot`**(*values_mat*, *col_labels*, *row_labels*, *highlight_cols=None*, *highlight_colors=None*, *save=False*)

Make a "termite" plot, typically used for assessing topic models with a tabular layout that promotes comparison of terms both within and across topics.

Parameters

- **`values_mat`** (`np.ndarray` or matrix) – matrix of values with shape (# row labels, # col labels) used to size the dots on the grid
- **`col_labels`** (*`seq[str]`*) – labels used to identify x-axis ticks on the grid
- **`row_labels`** (*`seq[str]`*) – labels used to identify y-axis ticks on the grid
- **`highlight_cols`** (*`int or seq[int], optional`*) – indices for columns to visually highlight in the plot with contrasting colors
- **`highlight_colors`** (*`tuple of 2-tuples`*) – each 2-tuple corresponds to a pair of (light/dark) matplotlib-friendly colors used to highlight a single column; if not specified (default), a good set of 6 pairs are used
- **`save`** (*`str, optional`*) – give the full /path/to/fname on disk to save figure

Returns axis on which termite plot is plotted

Return type `matplotlib.axes.Axes.axis`

Raises `ValueError` – if more columns are selected for highlighting than colors or if any of the inputs' dimensions don't match

References

See also:

`TopicModel.termite_plot`

`textacy.viz.network.`**`draw_semantic_network`**(*graph*, *node_weights=None*, *spread=3.0*, *draw_nodes=False*, *base_node_size=300*, *node_alpha=0.25*, *line_width=0.5*, *line_alpha=0.1*, *base_font_size=12*, *save=False*)

Draw a semantic network with nodes representing either terms or sentences, edges representing coocurrence or similarity, and positions given by a force- directed layout.

Parameters

- **`graph`** (`networkx.Graph`) –

- **node_weights** (*dict*) – mapping of node: weight, used to size node labels (and, optionally, node circles) according to their weight
- **spread** (*float*) – number that drives the spread of the network; higher values give more spread-out networks
- **draw_nodes** (*bool*) – if True, circles are drawn under the node labels
- **base_node_size** (*int*) – if *node_weights* not given and *draw_nodes* is True, this is the size of all nodes in the network; if *node_weights* _is_ given, node sizes will be scaled against this value based on their weights compared to the max weight
- **node_alpha** (*float*) – alpha of the circular nodes drawn behind labels if *draw_nodes* is True
- **line_width** (*float*) – width of the lines (edges) drawn between nodes
- **line_alpha** (*float*) – alpha of the lines (edges) drawn between nodes
- **base_font_size** (*int*) – if *node_weights* not given, this is the font size used to draw all labels; otherwise, font sizes will be scaled against this value based on the corresponding node weights compared to the max
- **save** (*str*) – give the full /path/to/fname on disk to save figure (optional)

Returns axis on which network plot is drawn

Return type `matplotlib.axes.Axes.axis`

Utilities

Set of small utility functions that take text strings as input.

`textacy.text_utils.`**KWIC**(*text*, *keyword*, *ignore_case=True*, *window_width=50*, *print_only=True*)
Alias of *keyword_in_context*.

`textacy.text_utils.`**clean_terms**(*terms*)
Clean up a sequence of single- or multi-word strings: strip leading/trailing junk chars, handle dangling parens and odd hyphenation, etc.

Parameters **terms** (*Iterable[str]*) – sequence of terms such as "presidency", "epic failure", or "George W. Bush" that may be _unclean_ for whatever reason

Yields *str* –

next term in *terms* but with the cruft cleaned up, excluding terms that were _entirely_ cruft

> **Warning:** Terms with (intentionally) unusual punctuation may get "cleaned" into a form that changes or obscures the original meaning of the term.

`textacy.text_utils.`**detect_language**(*text*)
Detect the most likely language of a text and return its 2-letter code (see https://cloud.google.com/translate/v2/using_rest#language-params). Uses the cld2-cffi package; to take advantage of optional params, call `cld2.detect()` directly.

Parameters **text** (*str*) –

Returns str

`textacy.text_utils.`**is_acronym**(*token*, *exclude=None*)
Pass single token as a string, return True/False if is/is not valid acronym.

Parameters

- **token** (*str*) – single word to check for acronym-ness
- **exclude** (*Set[str]*) – if technically valid but not actually good acronyms are known in advance, pass them in as a set of strings; matching tokens will return False

Returns bool

textacy.text_utils.**keyword_in_context**(*text*, *keyword*, *ignore_case=True*, *window_width=50*, *print_only=True*)

Search for `keyword` in `text` via regular expression, return or print strings spanning `window_width` characters before and after each occurrence of keyword.

Parameters

- **text** (*str*) – text in which to search for `keyword`
- **keyword** (*str*) – technically, any valid regular expression string should work, but usually this is a single word or short phrase: "spam", "spam and eggs"; to account for variations, use regex: "[Ss]pam (and|&) [Ee]ggs?"

  N.B. If keyword contains special characters, be sure to escape them!!!
- **ignore_case** (*bool*) – if True, ignore letter case in *keyword* matching
- **window_width** (*int*) – number of characters on either side of *keyword* to include as "context"
- **print_only** (*bool*) – if True, print out all results with nice formatting; if False, return all (pre, kw, post) matches as generator of raw strings

Returns generator(Tuple[str, str, str]), or None

Set of small utility functions that take Spacy objects as input.

textacy.spacy_utils.**get_main_verbs_of_sent**(*sent*)

Return the main (non-auxiliary) verbs in a sentence.

textacy.spacy_utils.**get_objects_of_verb**(*verb*)

Return all objects of a verb according to the dependency parse, including open clausal complements.

textacy.spacy_utils.**get_span_for_compound_noun**(*noun*)

Return document indexes spanning all (adjacent) tokens in a compound noun.

textacy.spacy_utils.**get_span_for_verb_auxiliaries**(*verb*)

Return document indexes spanning all (adjacent) tokens around a verb that are auxiliary verbs or negations.

textacy.spacy_utils.**get_subjects_of_verb**(*verb*)

Return all subjects of a verb according to the dependency parse.

textacy.spacy_utils.**is_negated_verb**(*token*)

Returns True if verb is negated by one of its (dependency parse) children, False otherwise.

Parameters **token** (`spacy.Token`) – parent document must have parse information

Returns bool

TODO: generalize to other parts of speech; rule-based is pretty lacking, so will probably require training a model; this is an unsolved research problem

textacy.spacy_utils.**is_plural_noun**(*token*)

Returns True if token is a plural noun, False otherwise.

Parameters **token** (`spacy.Token`) – parent document must have POS information

Returns bool

textacy.spacy_utils.**merge_spans**(*spans*)
Merge spans *in-place* within parent doc so that each takes up a single token.

Parameters **spans** (Iterable[spacy.Span]) –

textacy.spacy_utils.**normalized_str**(*token*)
Return as-is text for tokens that are proper nouns or acronyms, lemmatized text for everything else.

Parameters **token** (spacy.Token or spacy.Span) –

Returns str

textacy.spacy_utils.**preserve_case**(*token*)
Returns True if *token* is a proper noun or acronym, False otherwise.

Parameters **token** (spacy.Token) – parent document must have POS information

Returns bool

Set of small utility functions that do mathy stuff.

textacy.math_utils.**cosine_similarity**(*vec1*, *vec2*)
Return the cosine similarity between two vectors.

Parameters

- **vec1** (numpy.array) –
- **vec2** (numpy.array) –

Returns float

Other Stuff!

Collection of semantic similarity metrics.

textacy.similarity.**hamming**(*str1*, *str2*)
Measure the similarity between two strings using Hamming distance, which simply gives the number of characters in the strings that are different i.e. the number of substitution edits needed to change one string into the other.

Parameters

- **str1** (*str*) –
- **str2** (*str*) –

Returns

similarity between *str1* and *str2* in the interval [0.0, 1.0], where larger values correspond to more similar strings

Return type float

Note: This uses a *modified* Hamming distance in that it permits strings of different lengths to be compared.

textacy.similarity.**jaccard**(*obj1*, *obj2*, *fuzzy_match=False*, *match_threshold=0.8*)
Measure the semantic similarity between two strings or sequences of strings using Jaccard distance, with optional fuzzy matching of not-identical pairs when *obj1* and *obj2* are sequences of strings.

Parameters

- **obj1** (*str or Sequence[str]*) –

- **obj2** (*str or Sequence[str]*) – if str, both inputs are treated as sequences of *characters*, in which case fuzzy matching is not permitted
- **fuzzy_match** (*bool*) – if True, allow for fuzzy matching in addition to the usual identical matching of pairs between input vectors
- **match_threshold** (*float*) – value in the interval [0.0, 1.0]; fuzzy comparisons with a score >= this value will be considered matches

Returns

similarity between *obj1* and *obj2* in the interval [0.0, 1.0], where larger values correspond to more similar strings or sequences of strings

Return type float

Raises ValueError – if *fuzzy_match* is True but *obj1* and *obj2* are strings

textacy.similarity.**jaro_winkler**(*str1*, *str2*, *prefix_weight=0.1*)

Measure the similarity between two strings using Jaro-Winkler similarity metric, a modification of Jaro metric giving more weight to a shared prefix.

Parameters

- **str1** (*str*) –
- **str2** (*str*) –
- **prefix_weight** (*float*) – the inverse value of common prefix length needed to consider the strings identical

Returns

similarity between *str1* and *str2* in the interval [0.0, 1.0], where larger values correspond to more similar strings

Return type float

textacy.similarity.**levenshtein**(*str1*, *str2*)

Measure the similarity between two strings using Levenshtein distance, which gives the minimum number of character insertions, deletions, and substitutions needed to change one string into the other.

Parameters

- **str1** (*str*) –
- **str2** (*str*) –
- **normalize** (*bool*) – if True, divide Levenshtein distance by the total number of characters in the longest string; otherwise leave the distance as-is

Returns

similarity between *str1* and *str2* in the interval [0.0, 1.0], where larger values correspond to more similar strings

Return type float

textacy.similarity.**token_sort_ratio**(*str1*, *str2*)

Measure of similarity between two strings based on minimal edit distance, where ordering of words in each string is normalized before comparing.

Parameters

- **str1** (*str*) –
- **str2** (*str*) –

Returns

similarity between `str1` and `str2` in the interval [0.0, 1.0], where larger values correspond to more similar strings.

Return type float

`textacy.similarity.`**`word2vec`**(*obj1*, *obj2*)

Measure the semantic similarity between one Doc or spacy Doc, Span, Token, or Lexeme and another like object using the cosine distance between the objects' (average) word2vec vectors.

Parameters

- **`obj1`** (`textacy.Doc`, `spacy.Doc`, `spacy.Span`, `spacy.Token`, or `spacy.Lexeme`) –
- **`obj2`** (`textacy.Doc`, `spacy.Doc`, `spacy.Span`, `spacy.Token`, or `spacy.Lexeme`) –

Returns

float: similarity between *obj1* and *obj2* in the interval [0.0, 1.0], where larger values correspond to more similar objects

`textacy.similarity.`**`word_movers`**(*doc1*, *doc2*, *metric=u'cosine'*)

Measure the semantic similarity between two documents using Word Movers Distance.

Parameters

- **`doc1`** (`textacy.Doc` or `spacy.Doc`) –
- **`doc2`** (`textacy.Doc` or `spacy.Doc`) –
- **`metric`**(*{'cosine', 'euclidean', 'l1', 'l2', 'manhattan'}*) –

Returns

similarity between *doc1* and *doc2* in the interval [0.0, 1.0], where larger values correspond to more similar documents

Return type float

References

Ofir Pele and Michael Werman, "A linear time histogram metric for improved SIFT matching," in Computer Vision - ECCV 2008, Marseille, France, 2008.

Ofir Pele and Michael Werman, "Fast and robust earth mover's distances," in Proc. 2009 IEEE 12th Int. Conf. on Computer Vision, Kyoto, Japan, 2009.

Kusner, Matt J., et al. "From word embeddings to document distances." Proceedings of the 32nd International Conference on Machine Learning (ICML 2015). 2015. http://jmlr.org/proceedings/papers/v37/kusnerb15.pdf

Calculations of basic counts and readability statistics for text documents.

class `textacy.text_stats.`**`TextStats`**(*doc*)

Compute a variety of basic counts and readability statistics for a given text document. For example:

```
>>> text = list(textacy.datasets.CapitolWords().texts(limit=1))[0]
>>> doc = textacy.Doc(text)
>>> ts = TextStats(doc)
>>> ts.n_words
136
>>> ts.flesch_kincaid_grade_level
11.817647058823532
>>> ts.basic_counts
{'n_chars': 685,
 'n_long_words': 43,
 'n_monosyllable_words': 90,
 'n_polysyllable_words': 24,
 'n_sents': 6,
 'n_syllables': 214,
 'n_unique_words': 80,
 'n_words': 136}
>>> ts.readability_stats
{'automated_readability_index': 13.626495098039214,
 'coleman_liau_index': 12.509300816176474,
 'flesch_kincaid_grade_level': 11.817647058823532,
 'flesch_readability_ease': 50.707745098039254,
 'gulpease_index': 51.86764705882353,
 'gunning_fog_index': 16.12549019607843,
 'lix': 54.28431372549019,
 'smog_index': 14.554592549557764,
 'wiener_sachtextformel': 8.266410784313727}
```

Parameters **doc** (`textacy.Doc` or `SpacyDoc`) – A text document processed by spacy. Need only be tokenized.

n_sents
int – Number of sentences in `doc`.

n_words
int – Number of words in `doc`, including numbers + stop words but excluding punctuation.

n_chars
int – Number of characters for all words in `doc`.

n_syllables
int – Number of syllables for all words in `doc`.

n_unique_words
int – Number of unique (lower-cased) words in `doc`.

n_long_words
int – Number of words in `doc` with 7 or more characters.

n_monosyllable_words
int – Number of words in `doc` with 1 syllable only.

n_polysyllable_words
int – Number of words in `doc` with 3 or more syllables. Note: Since this excludes words with exactly 2 syllables, it's likely that `n_monosyllable_words + n_polysyllable_words != n_words`.

flesch_kincaid_grade_level
float – https://en.wikipedia.org/wiki/Flesch%E2%80%93Kincaid_readability_tests#Flesch.E2.80.93Kincaid_grade_level

flesch_readability_ease
float – https://en.wikipedia.org/wiki/Flesch%E2%80%93Kincaid_readability_tests#Flesch_reading_ease

smog_index
float – https://en.wikipedia.org/wiki/SMOG

gunning_fog_index
float – https://en.wikipedia.org/wiki/Gunning_fog_index

coleman_liau_index
float – https://en.wikipedia.org/wiki/Coleman%E2%80%93Liau_index

automated_readability_index
float – https://en.wikipedia.org/wiki/Automated_readability_index

lix
float – https://en.wikipedia.org/wiki/LIX

gulpease_index
float – https://it.wikipedia.org/wiki/Indice_Gulpease

wiener_sachtextformel
float – https://de.wikipedia.org/wiki/Lesbarkeitsindex#Wiener_Sachtextformel Note: This always returns variant #1.

basic_counts
Dict[str, int] – Mapping of basic count names to values, where basic counts are the attributes listed above between `n_sents` and `n_polysyllable_words`.

readability_stats
Dict[str, float] – Mapping of readability statistic names to values, where readability stats are the attributes listed above between `flesch_kincaid_grade_level` and `wiener_sachtextformel`.

Raises `ValueError` – If `doc` is not a `textacy.Doc` or `SpacyDoc`.

`textacy.text_stats.`**automated_readability_index**(*n_chars*, *n_words*, *n_sents*)
https://en.wikipedia.org/wiki/Automated_readability_index

`textacy.text_stats.`**coleman_liau_index**(*n_chars*, *n_words*, *n_sents*)
https://en.wikipedia.org/wiki/Coleman%E2%80%93Liau_index

`textacy.text_stats.`**flesch_kincaid_grade_level**(*n_syllables*, *n_words*, *n_sents*)
https://en.wikipedia.org/wiki/Flesch%E2%80%93Kincaid_readability_tests#Flesch.E2.80.93Kincaid_grade_level

`textacy.text_stats.`**flesch_readability_ease**(*n_syllables*, *n_words*, *n_sents*)
https://en.wikipedia.org/wiki/Flesch%E2%80%93Kincaid_readability_tests#Flesch_reading_ease

`textacy.text_stats.`**gulpease_index**(*n_chars*, *n_words*, *n_sents*)
https://it.wikipedia.org/wiki/Indice_Gulpease

`textacy.text_stats.`**gunning_fog_index**(*n_words*, *n_polysyllable_words*, *n_sents*)
https://en.wikipedia.org/wiki/Gunning_fog_index

`textacy.text_stats.`**lix**(*n_words*, *n_long_words*, *n_sents*)
https://en.wikipedia.org/wiki/LIX

`textacy.text_stats.`**readability_stats**(*doc*)
Get calculated values for a variety of statistics related to the "readability" of a text: Flesch-Kincaid Grade Level, Flesch Reading Ease, SMOG Index, Gunning-Fog Index, Coleman-Liau Index, and Automated Readability Index.

Also includes constituent values needed to compute the stats, e.g. word count.

DEPRECATED

Parameters **doc** (textacy.Doc) –

Returns mapping of readability statistic name (str) to value (int or float)

Return type dict

Raises NotImplementedError – if doc is not English language. sorry.

textacy.text_stats.**smog_index**(*n_polysyllable_words*, *n_sents*)
https://en.wikipedia.org/wiki/SMOG

textacy.text_stats.**wiener_sachtextformel**(*n_words*, *n_polysyllable_words*, *n_monosyllable_words*, *n_long_words*, *n_sents*, *variant=1*)
https://de.wikipedia.org/wiki/Lesbarkeitsindex#Wiener_Sachtextformel

Functions to load and cache language data and other NLP resources.

textacy.data.**load_depechemood**(**args*, ***kwargs*)
Load DepecheMood lexicon text file from disk, munge into nested dictionary for convenient lookup by lemma#POS. NB: English only!

Each version of DepecheMood is built starting from word-by-document matrices either using raw frequencies (DepecheMood_freq.txt), normalized frequencies (DepecheMood_normfreq.txt) or tf-idf (DepecheMood_tfidf.txt). The files are tab-separated; each row contains one Lemma#PoS followed by the scores for the following emotions: AFRAID, AMUSED, ANGRY, ANNOYED, DONT_CARE, HAPPY, INSPIRED, SAD.

Parameters

- **data_dir** (*str, optional*) – directory on disk where DepecheMood lexicon text files are stored, i.e. the location of the 'DepecheMood_V1.0' directory created when unzipping the DM dataset
- **download_if_missing** (*bool, optional*) – if True and data not found on disk, it will be automatically downloaded and saved to disk
- **weighting** (*str {'freq', 'normfreq', 'tfidf'}, optional*) – type of word weighting used in building DepecheMood matrix

Returns

top-level keys are Lemma#POS strings, values are nested dicts with emotion names as keys and weights as floats

Return type dict[dict]

References

Staiano, J., & Guerini, M. (2014). "DepecheMood: a Lexicon for Emotion Analysis from Crowd-Annotated News". Proceedings of ACL-2014. (arXiv:1405.1605) Data available at https://github.com/marcoguerini/DepecheMood/releases .

textacy.data.**load_hyphenator**(**args*, ***kwargs*)
Load an object that hyphenates words at valid points, as used in LaTex typesetting.

Note that while hyphenation points always fall on syllable divisions, not all syllable divisions are valid hyphenation points. But it's decent.

Parameters **lang** (*str, optional*) – standard 2-letter language abbreviation; to get list of valid values:

```
>>> import pyphen; pyphen.LANGUAGES
```

Returns pyphen.Pyphen()

textacy.data.**load_spacy**(**args*, ***kwargs*)

Load a language-specific spaCy pipeline (collection of data, models, and resources) for tokenizing, tagging, parsing, etc. text. The most recent result is cached.

Parameters

- **name** (*str*) – Standard 2-letter language abbreviation for a language. Currently, spaCy supports English ('en') and German ('de').
- **path** (*str*) – path/to/directory on disk where spaCy models are saved. If None, spaCy's default data path is used.
- **create_pipeline** (*func*) – Callable that takes a spaCy Language instance as its argument and returns a sequence of callables. Each callable takes a SpacyDoc as its sole positional argument and modifies the document in place.
- ****kwargs** – Keyword arguments passed to spacy.load().
  - vocab
  - tokenizer
  - tagger
  - parser
  - matcher
  - entity
  - add_vectors
  - create_make_doc

Returns spacy.

Raises RuntimeError – if package can't be loaded

See also:

https://spacy.io/docs/#language

Collection of lexicon-based methods for characterizing texts by sentiment, emotional valence, etc.

textacy.lexicon_methods.**emotional_valence**(*words*, *threshold=0.0*, *dm_data_dir=None*, *dm_weighting='normfreq'*)

Get average emotional valence over all words for the following emotions: AFRAID, AMUSED, ANGRY, ANNOYED, DONT_CARE, HAPPY, INSPIRED, SAD.

Parameters

- **words** (List[spacy.Token]) – list of words for which to get average emotional valence; note that only nouns, adjectives, adverbs, and verbs will be counted
- **threshold** (*float*) – minimum emotional valence score for which to count a given word for a given emotion; value must be in [0.0, 1.0)
- **dm_data_dir** (*str*) – full path to directory where DepecheMood data is saved on disk

- **dm_weighting** (*{'freq', 'normfreq', 'tfidf'}*) – type of word weighting used in building DepecheMood matrix

Returns mapping of emotion (str) to average valence score (float)

Return type dict

References

Bibliography

[BestCoverage] Küçüktunç, O., Saule, E., Kaya, K., & Çatalyürek, Ü. V. (2013, May). Diversified recommendation on graphs: pitfalls, measures, and algorithms. In Proceedings of the 22nd international conference on World Wide Web (pp. 715-726). International World Wide Web Conferences Steering Committee. http://www2013.wwwconference.org/proceedings/p715.pdf

[DivRank] Mei, Q., Guo, J., & Radev, D. (2010, July). Divrank: the interplay of prestige and diversity in information networks. In Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining (pp. 1009-1018). ACM. http://clair.si.umich.edu/~radev/papers/SIGKDD2010.pdf

[SGRank] Danesh, Sumner, and Martin. "SGRank: Combining Statistical and Graphical Methods to Improve the State of the Art in Unsupervised Keyphrase Extraction". Lexical and Computational Semantics (* SEM 2015) (2015): 117.

[SingleRank] Hasan, K. S., & Ng, V. (2010, August). Conundrums in unsupervised keyphrase extraction: making sense of the state-of-the-art. In Proceedings of the 23rd International Conference on Computational Linguistics: Posters (pp. 365-373). Association for Computational Linguistics.

[TextRank] Mihalcea, R., & Tarau, P. (2004, July). TextRank: Bringing order into texts. Association for Computational Linguistics.

[DepecheMood] Staiano and Guerini. DepecheMood: a Lexicon for Emotion Analysis from Crowd-Annotated News. 2014. Data available at https://github.com/marcoguerini/DepecheMood/releases

Python Module Index

t

Index

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W