textacy Documentation

Release 0.11.0

Burton DeWilde

Apr 12, 2021

CONTENTS

1 features **3**

2 links **5**

3 maintainer **7**

4 contents **9**

4.1 Installation 9
4.2 Quickstart 10
4.3 Tutorials 16
4.4 API Reference 26
4.5 Changes 141

Python Module Index **171**

Index **173**

`textacy` is a Python library for performing a variety of natural language processing (NLP) tasks, built on the high-performance spaCy library. With the fundamentals — tokenization, part-of-speech tagging, dependency parsing, etc. — delegated to another library, `textacy` focuses primarily on the tasks that come before and follow after.

CHAPTER

ONE

FEATURES

- Access and extend spaCy's core functionality for working with one or many documents through convenient methods and custom extensions
- Load prepared datasets with both text content and metadata, from Congressional speeches to historical literature to Reddit comments
- Clean, normalize, and explore raw text before processing it with spaCy
- Extract structured information from processed documents, including n-grams, entities, acronyms, keyterms, and SVO triples
- Compare strings and sequences using a variety of similarity metrics
- Tokenize and vectorize documents then train, interpret, and visualize topic models
- Compute text readability statistics, including Flesch-Kincaid grade level, SMOG index, and multi-lingual Flesch Reading Ease

. . . and much more!

CHAPTER

TWO

LINKS

- Download: https://pypi.org/project/textacy
- Documentation: https://textacy.readthedocs.io
- Source code: https://github.com/chartbeat-labs/textacy
- Bug Tracker: https://github.com/chartbeat-labs/textacy/issues

CHAPTER

THREE

MAINTAINER

Howdy, y'all.

- Burton DeWilde (burtdewilde@gmail.com)

CHAPTER

FOUR

CONTENTS

4.1 Installation

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

```
$ pip install textacy
```

or `conda`:

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

If you prefer — or are obliged — you can download and unzip the source `tar.gz` from PyPi, then install manually:

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

4.1.1 Dependencies

Given the breadth of functionality, `textacy` depends on a number of other Python packages. Most of these are common components in the PyData stack (`numpy`, `scikit-learn`, etc.), but a few are more niche. One heavy dependency has been made optional.

Specifically: To use visualization functionality in `textacy.viz`, you'll need to have `matplotlib` installed. You can do so via `pip install textacy[viz]` or `pip install matplotlib`.

4.1.2 Downloading Data

For most uses of textacy, language-specific model data in spaCy is required. Fortunately, spaCy makes the process of getting this data easy; just follow the instructions in their docs, which also includes a list of currently-supported languages and their models.

Note: In previous versions of spaCy, users were able to link a specific model to a different name (e.g. "en_core_web_sm" => "en"), but this is no longer permitted. As such, `textacy` now requires users to fully specify which model to apply to a text, rather than leveraging automatic language identification to do it for them.

`textacy` itself features convenient access to several datasets comprised of thousands of text + metadata records, as well as a couple linguistic resources. Data can be downloaded via the `.download()` method on corresponding dataset/resource classes (see *Datasets and Resources* for details) or directly from the command line.

```
$ python -m textacy download capitol_words
$ python -m textacy download depeche_mood
$ python -m textacy download lang_identifier --version 2.0
```

These commands download and save a compressed json file with ~11k speeches given by the main protagonists of the 2016 U.S. Presidential election, followed by a set of emotion lexicons in English and Italian with various word representations, and lastly a language identification model that works for 140 languages. For more information about particular datasets/resources use the `info` subcommand:

```
$ python -m textacy info capitol_words
```

4.2 Quickstart

First things first: Import the package. Most functionality is available from this top-level import, but we'll see that some features require their own imports.

```
>>> import textacy
```

4.2.1 Working with Text

Let's start with a single text document:

```
>>> text = (
...     "Since the so-called \"statistical revolution\" in the late 1980s and mid
↪1990s, "
...     "much Natural Language Processing research has relied heavily on machine
↪learning. "
...     "Formerly, many language-processing tasks typically involved the direct hand
↪coding "
...     "of rules, which is not in general robust to natural language variation. "
...     "The machine-learning paradigm calls instead for using statistical inference "
...     "to automatically learn such rules through the analysis of large corpora "
...     "of typical real-world examples."
... )
```

Note: In almost all cases, textacy (as well as spaCy) expects to be working with unicode text data. Throughout the code, this is indicated as `str` to be consistent with Python 3's default string type; users of Python 2, however, must be mindful to use `unicode`, and convert from the default (bytes) string type as needed.

Before (or *in lieu of*) processing this text with spaCy, we can do a few things. First, let's look for keywords-in-context, as a quick way to assess, by eye, how a particular word or phrase is used in a body of text:

```
>>> from textacy import extract
>>> list(extract.keyword_in_context(text, "language", window_width=25, pad_
↪context=True))
[(' mid 1990s, much Natural ', 'Language', ' Processing research has '),
 ('learning. Formerly, many ', 'language', '-processing tasks typical'),
 ('eneral robust to natural ', 'language', ' variation. The machine-l')]
```

Sometimes, "raw" text is messy and must be cleaned up before analysis; other times, an analysis simply benefits from well-standardized text. In either case, the `textacy.preprocessing` sub-package contains a number of functions to normalize (whitespace, quotation marks, etc.), remove (punctuation, accents, etc.), and replace (URLs, emails, numbers, etc.) messy text data. For example:

```
>>> from textacy import preprocessing
>>> preprocessing.normalize.whitespace(preprocessing.remove.punctuation(text))[:80]
'Since the so called statistical revolution in the late 1980s and mid 1990s much '
```

4.2.2 Make a Doc

Usually, though, we want to work with text that's been processed by spaCy: tokenized, part-of-speech tagged, parsed, and so on. Since spaCy's pipelines are language-dependent, we have to load a particular pipeline to match the text; when working with texts from multiple languages, this can be a pain. Fortunately, textacy includes automatic language detection to apply the right pipeline to the text, and it caches the loaded language data to minimize wait time and hassle. Making a `Doc` from text is easy:

```
>>> doc = textacy.make_spacy_doc(text)
>>> doc._.preview
'Doc(85 tokens: "Since the so-called "statistical revolution" in...")'
```

Under the hood, the text has been identified as English, and the default English-language (`"en"`) pipeline has been loaded, cached, and applied to it. If you need to customize the pipeline, you can still easily load and cache it, then specify it yourself when initializing the doc:

```
>>> en = textacy.load_spacy_lang("en_core_web_sm", disable=("parser",))
>>> doc = textacy.make_spacy_doc(text, lang=en)
>>> doc._.preview
'Doc(85 tokens: "Since the so-called "statistical revolution" in...")'
```

Oftentimes, text data comes paired with metadata, such as a title, author, or publication date, and we'd like to keep them together. textacy makes this easy:

```
>>> metadata = {
...     "title": "Natural-language processing",
...     "url": "https://en.wikipedia.org/wiki/Natural-language_processing",
...     "source": "wikipedia",
... }
>>> doc = textacy.make_spacy_doc((text, metadata))
>>> doc._.meta["title"]
'Natural-language processing'
```

`textacy` adds a variety of useful functionality to vanilla spaCy docs, accessible via its `._` "underscore" property. For example: `doc._.preview` gives a convenient preview of the doc's contents, and `doc._.meta` returns any metadata associated with the main text content. Consult the spaCy docs for implementation details.

Note: Older versions of textacy (<0.7.0) used a `textacy.Doc` class as a convenient wrapper around an underlying spaCy `Doc`, with additional functionality available as class attributes and methods. Once spaCy started natively supporting custom extensions on `Doc` objects (as well as custom components in language processing pipelines), that approach was dropped.

4.2.3 Analyze a Doc

There are many ways to understand the content of a `Doc`. For starters, let's extract various elements of interest:

```
>>> list(textacy.extract.ngrams(
...     doc, 3, filter_stops=True, filter_punct=True, filter_nums=False))
[1980s and mid,
 Natural Language Processing,
 Language Processing research,
 research has relied,
 heavily on machine,
 processing tasks typically,
 tasks typically involved,
 involved the direct,
```

(continues on next page)

(continued from previous page)

```
 direct hand coding,
 coding of rules,
 robust to natural,
 natural language variation,
 learning paradigm calls,
 paradigm calls instead,
 inference to automatically,
 learn such rules,
 analysis of large,
 corpora of typical]
>>> list(textacy.extract.ngrams(doc, 2, min_freq=2))
[Natural Language, natural language]
>>> list(textacy.extract.entities(doc, drop_determiners=True))
[late 1980s and mid 1990s]
```

We can also identify key terms in a document by a number of algorithms:

```
>>> from textacy.extract import keyterms as kt
>>> kt.textrank(doc, normalize="lemma", topn=10)
[('Natural Language Processing research', 0.059959246697826624),
 ('natural language variation', 0.04488350959275309),
 ('direct hand coding', 0.037736661821063354),
 ('statistical inference', 0.03432557996664981),
 ('statistical revolution', 0.034007535820683756),
 ('machine learning', 0.03305919655573349),
 ('mid 1990', 0.026993994406706995),
 ('late 1980', 0.026499549123496648),
 ('general robust', 0.024835834233545625),
 ('large corpora', 0.024322049918545637)]
>>> kt.sgrank(doc, ngrams=(1, 2, 3, 4), normalize="lower", topn=0.1)
[('natural language processing research', 0.31279919999041045),
 ('direct hand coding', 0.09373747682969617),
 ('natural language variation', 0.09229056171473927),
 ('mid 1990s', 0.05832421657510258),
 ('machine learning', 0.05536624437146417)]
```

Or we can compute various basic and readability statistics:

```
>>> ts = textacy.TextStats(doc)
>>> ts.n_words, ts.n_syllables, ts.n_chars
(73, 134, 414)
>>> ts.entropy
5.8233192506312115
>>> ts.flesch_kincaid_grade_level, ts.flesch_reading_ease
(15.56027397260274, 26.84351598173518)
>>> ts.lix
65.42922374429223
```

Lastly, we can transform a document into a "bag of terms", with flexible weighting and term inclusion criteria:

```
>>> bot = doc._.to_bag_of_terms(
...     ngrams=(1, 2, 3), entities=True, weighting="count", as_strings=True)
>>> sorted(bot.items(), key=lambda x: x[1], reverse=True)[:15]
[('call', 2),
 ('statistical', 2),
 ('machine', 2),
 ('language', 2),
```

(continues on next page)

(continued from previous page)

```
 ('rule', 2),
 ('learn', 2),
 ('late 1980 and mid 1990', 1),
 ('revolution', 1),
 ('late', 1),
 ('1980', 1),
 ('mid', 1),
 ('1990', 1),
 ('Natural', 1),
 ('Language', 1),
 ('Processing', 1)]
```

4.2.4 Working with Many Texts

Many NLP tasks require datasets comprised of a large number of texts, which are often stored on disk in one or multiple files. textacy makes it easy to efficiently stream text and (text, metadata) pairs from disk, regardless of the format or compression of the data.

Let's start with a single text file, where each line is a new text document:

```
I love Daylight Savings Time: It's a biannual opportunity to find and fix obscure␣
↪date-time bugs in your code. Can't wait for next time!
Somewhere between "this is irritating but meh" and "blergh, why haven't I automated␣
↪this yet?!" Fuzzy decision boundary.
Spent an entire day translating structured data blobs into concise, readable␣
↪sentences. Human language is hard.
...
```

In this case, the texts are tweets from my sporadic presence on Twitter — a fine example of small (and boring) data. Let's stream it from disk so we can analyze it in textacy:

```
>>> texts = textacy.io.read_text('~/Desktop/burton-tweets.txt', lines=True)
>>> for text in texts:
...     doc = textacy.make_spacy_doc(text)
...     print(doc._.preview)
Doc(32 tokens; "I love Daylight Savings Time: It's a biannual o...")
Doc(28 tokens; "Somewhere between "this is irritating but meh" ...")
Doc(20 tokens; "Spent an entire day translating structured data...")
...
```

Okay, let's not *actually* analyze my ramblings on social media...

Instead, let's consider a more complicated dataset: a compressed JSON file in the mostly-standard "lines" format, in which each line is a separate record with both text data and metadata fields. As an example, we can use the "Capitol Words" dataset integrated into textacy (see *Datasets and Resources* for details). The data is downloadable from the textacy-data GitHub repository.

```
>>> records = textacy.io.read_json(
...     "textacy/data/capitol_words/capitol-words-py3.json.gz",
...     mode="rt", lines=True)
>>> for record in records:
...     doc = textacy.make_spacy_doc((record["text"], {"title": record["title"]}))
...     print(doc._.preview)
...     print("meta:", doc._.meta)
...     # do stuff...
```

(continues on next page)

(continued from previous page)

```
...     break
Doc(159 tokens; "Mr. Speaker, 480,000 Federal employees are work...")
meta: {'title': 'JOIN THE SENATE AND PASS A CONTINUING RESOLUTION'}
```

For this and a few other datasets, convenient `Dataset` classes are already implemented in textacy to help users get up and running, faster:

```
>>> import textacy.datasets  # note the import
>>> ds = textacy.datasets.CapitolWords()
>>> ds.download()
>>> records = ds.records(speaker_name={"Hillary Clinton", "Barack Obama"})
>>> next(records)
('I yield myself 15 minutes of the time controlled by the Democrats.',
 {'date': '2001-02-13',
  'congress': 107,
  'speaker_name': 'Hillary Clinton',
  'speaker_party': 'D',
  'title': 'MORNING BUSINESS',
  'chamber': 'Senate'})
```

4.2.5 Make a Corpus

A `textacy.Corpus` is an ordered collection of spaCy `Doc` s, all processed by the same language pipeline. Let's continue with the Capitol Words dataset and make a corpus from a stream of records. (**Note:** This may take a few minutes.)

```
>>> corpus = textacy.Corpus("en", data=records)
>>> corpus
Corpus(1240 docs, 857548 tokens)
```

The language pipeline used to analyze documents in the corpus must be specified on instantiation, but the data added to it may come in the form of one or a stream of texts, records, or (valid) `Doc` s.

```
>>> textacy.Corpus(
...     textacy.load_spacy_lang("en_core_web_sm", disable=("parser", "tagger")),
...     data=ds.texts(speaker_party="R", chamber="House", limit=100))
Corpus(100 docs, 31356 tokens)
```

You can use basic indexing as well as flexible boolean queries to select documents in a corpus:

```
>>> corpus[-1]._.preview
'Doc(2999 tokens: "In the Federalist Papers, we often hear the ref...")'
>>> [doc._.preview for doc in corpus[10:15]]
['Doc(359 tokens: "My good friend from Connecticut raised an issue...")',
 'Doc(83 tokens: "My question would be: In response to the discus...")',
 'Doc(3338 tokens: "Madam President, I come to the floor today to s...")',
 'Doc(221 tokens: "Mr. President, I rise in support of Senator Tho...")',
 'Doc(3061 tokens: "Mr. President, I thank my distinguished colleag...")']
>>> obama_docs = list(corpus.get(lambda doc: doc._.meta["speaker_name"] == "Barack␣
↪Obama"))
>>> len(obama_docs)
411
```

It's important to note that all of the data in a `textacy.Corpus` is stored in-memory, which makes a number of features much easier to implement. Unfortunately, this means that the maximum size of a corpus will be bounded by

RAM.

4.2.6 Analyze a Corpus

There are lots of ways to analyze the data in a corpus. Basic stats are computed on the fly as documents are added (or removed) from a corpus:

```
>>> corpus.n_docs, corpus.n_sents, corpus.n_tokens
(1240, 34530, 857548)
```

You can transform a corpus into a document-term matrix, with flexible tokenization, weighting, and filtering of terms:

```
>>> import textacy.vsm  # note the import
>>> vectorizer = textacy.vsm.Vectorizer(
...     tf_type="linear", apply_idf=True, idf_type="smooth", norm="l2",
...     min_df=2, max_df=0.95)
>>> doc_term_matrix = vectorizer.fit_transform(
...     (doc._.to_terms_list(ngrams=1, entities=True, as_strings=True)
...      for doc in corpus))
>>> print(repr(doc_term_matrix))
<1240x12577 sparse matrix of type '<class 'numpy.float64'>'
    with 217067 stored elements in Compressed Sparse Row format>
```

From a doc-term matrix, you can then train and interpret a topic model:

```
>>> import textacy.tm  # note the import
>>> model = textacy.tm.TopicModel("nmf", n_topics=10)
>>> model.fit(doc_term_matrix)
>>> doc_topic_matrix = model.transform(doc_term_matrix)
>>> doc_topic_matrix.shape
(1240, 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   child   work   need   York   bill   year   school   student
topic 1 : rescind   quorum   order   unanimous   consent   ask   President   Mr.  ␣
↪Madam   objection
topic 2 : dispense   reading   unanimous   consent   amendment   ask   President   Mr.
↪   Madam   OFFICER
topic 3 : motion   table   lay   reconsider   agree   thereto   Madam   preamble  ␣
↪intervene   print
topic 4 : desire   Chamber   vote   Senators   rollcall   voter   amendment   2313  ␣
↪regular   cloture
topic 5 : amendment   pende   aside   set   ask   unanimous   consent   Mr.  ␣
↪President   desk
topic 6 : health   care   patient   Health   mental   quality   child   medical  ␣
↪information   coverage
topic 7 : Iraq   war   troop   iraqi   Iraqis   policy   military   american   U.S.  ␣
↪force
topic 8 : tax   budget   cut   debt   pay   deficit   $   fiscal   billion   spending
topic 9 : Senator   Virginia   yield   West Virginia   West   question   thank  ␣
↪Massachusetts   objection   time
```

And that's just getting started! For now, though, I encourage you to pick a dataset — either your own or one already included in textacy — and start exploring the data. *Most* functionality is well-documented via in-code docstrings; to see that information all together in nicely-formatted HTML, be sure to check out the *API Reference*.

4.2.7 Working with Many Languages

Since a `Corpus` uses the same spaCy language pipeline to process all input texts, it only works in a mono-lingual context. In some cases, though, your collection of texts may contain more than one language; for example, if I occasionally tweeted in Spanish (sí, ¡se habla español!), the `burton-tweets.txt` dataset couldn't be fed in its entirety into a single `Corpus`. This is irritating, but there are some workarounds.

If you haven't already, download spaCy models for the languages you want to analyze — see *Installation* for details. Then, if your use case doesn't require `Corpus` functionality, you can iterate over the texts and only analyze those for which models are available:

```
>>> for text in texts:
...     try:
...         doc = textacy.make_spacy_doc(text)
...     except OSError:
...         continue
...     # do stuff...
```

When the `lang` param is unspecified, textacy tries to auto-detect the text's language and load the corresponding model; if that model is unavailable, spaCy will raise an `OSError`. This try/except also handles the case where language detection fails and returns, say, "un" for "unknown".

It's worth noting that, although spaCy has statistical models for annotating texts in only 10 or so languages, it supports tokenization in dozens of other languages. See https://spacy.io/usage/models#languages for details. You can load such languages in `textacy` via `textacy.load_spacy_lang(langstr, allow_blank=True)`.

If you do need a `Corpus`, you can split the input texts by language into distinct collections, then instantiate monolingual corpora on those collections. For example:

```
>>> en_corpus = textacy.Corpus(
...     "en", data=(
...         text for text in texts
...         if textacy.identify_lang(text) == "en")
... )
>>> es_corpus = textacy.Corpus(
...     "es", data=(
...         text for text in texts
...         if textacy.identify_lang(text) == "es")
... )
```

Both of these options are less convenient than I'd like, but hopefully they get the job done.

4.3 Tutorials

4.3.1 Context and Description of Workers in the U.S. Congress

In this tutorial, we will explore how certain members of the U.S. Congress have spoken about workers, based on a dataset of thousands of speeches sourced from the Congressional Record.

First, let's initialize and download the dataset, which comes built-in with `textacy`:

```
>>> import textacy.datasets
>>> dataset = textacy.datasets.CapitolWords()
>>> dataset.info
{'name': 'capitol_words',
```

(continues on next page)

(continued from previous page)

```
 'site_url': 'http://sunlightlabs.github.io/Capitol-Words/',
 'description': 'Collection of ~11k speeches in the Congressional Record given by␣
↪notable U.S. politicians between Jan 1996 and Jun 2016.'}
>>> dataset.download()
```

Each record in this dataset contains the full text of and basic metadata about the speech. Let's take a peek at the first one, to get our bearings:

```
>>> record = next(dataset.records(limit=1))
>>> record
Record(text='Mr. Speaker, 480,000 Federal employees are working without pay, a form␣
↪of involuntary servitude; 280,000 Federal employees are not working, and they will␣
↪be paid. Virtually all of these workers have mortgages to pay, children to feed,␣
↪and financial obligations to meet.\nMr. Speaker, what is happening to these workers␣
↪is immoral, is wrong, and must be rectified immediately. Newt Gingrich and the␣
↪Republican leadership must not continue to hold the House and the American people␣
↪hostage while they push their disastrous 7-year balanced budget plan. The gentleman␣
↪from Georgia, Mr. Gingrich, and the Republican leadership must join Senator Dole␣
↪and the entire Senate and pass a continuing resolution now, now to reopen␣
↪Government.\nMr. Speaker, that is what the American people want, that is what they␣
↪need, and that is what this body must do.', meta={'date': '1996-01-04', 'congress':␣
↪104, 'speaker_name': 'Bernie Sanders', 'speaker_party': 'I', 'title': 'JOIN THE␣
↪SENATE AND PASS A CONTINUING RESOLUTION', 'chamber': 'House'})
```

This speech was delivered by Bernie Sanders back in 1996, when he was a member of the House of Representatives. By reading the text, we can see that it's about government workers during a shutdown — very relevant to our inquiry! :)

Considering the number of speeches, we'd like to avoid a full read-through and instead extract just the specific parts of interest. As a first step, let's use the `textacy.extract` subpackage to inspect our keywords in context.

```
>>> import textacy.extract
>>> list(textacy.extract.keyword_in_context(record.text, "work(ing|ers?)", window_
↪width=35))
[('ker, 480,000 Federal employees are ', 'working', ' without pay, a form of␣
↪involuntary'),
 (' 280,000 Federal employees are not ', 'working', ', and they will be paid.␣
↪Virtually '),
 ('ll be paid. Virtually all of these ', 'workers', ' have mortgages to pay, children␣
↪to'),
 ('peaker, what is happening to these ', 'workers', ' is immoral, is wrong, and must␣
↪be ')]
```

This is useful for developing our intuitions about how Bernie regards workers, but we'd prefer the information in a more structured form. Processing the text with spaCy will allow us to interrogate the text content in more sophisticated ways.

But first, we should preprocess the text to get rid of potential data quality issues (inconsistent quotation marks, whitespace, unicode characters, etc.) and other distractions that may affect our analysis. For example, maybe it would be better to replace all numbers with a constant placeholder value. For this, we'll use some of the functions available in `textacy.preprocessing`:

```
>>> import textacy.preprocessing
>>> textacy.preprocessing.replace.numbers(record.text)
'Mr. Speaker, _NUMBER_ Federal employees are working without pay, a form of␣
↪involuntary servitude; _NUMBER_ Federal employees are not working, and they will be␣
↪paid. Virtually all of these workers have mortgages to pay, children to feed, and␣
↪financial obligations to meet.\nMr. Speaker, what is happening to these workers is␣
↪immoral, is wrong, and must be rectified immediately. Newt Gingrich and the␣
↪Republican leadership must not continue to hold the House and the American people␣
↪hostage while they push their disastrous _NUMBER_-year balanced budget plan. The␣
↪gentleman from Georgia, Mr. Gingrich, and the Republican leadership must join␣
↪Senator Dole and the entire Senate and pass a continuing resolution now, now to␣
↪reopen Government.\nMr. Speaker, that is what the American people want, that is␣
```

(continues on next page)

(continued from previous page)

Note that these changes are "destructive" — they've changed the data, and we can't reconstruct the original without keeping a copy around or re-loading it from disk. On second thought... let's leave the numbers alone.

However, we should still take care to normalize common text data errors. Let's combine multiple such preprocessors into a lightweight, callable pipeline that applies each sequentially:

```
>>> preproc = textacy.preprocessing.make_pipeline(
...     textacy.preprocessing.normalize.unicode,
...     textacy.preprocessing.normalize.quotation_marks,
...     textacy.preprocessing.normalize.whitespace,
... )
>>> preproc_text = preproc(record.text)
>>> preproc_text[:200]
'Mr. Speaker, 480,000 Federal employees are working without pay, a form of␣
↪involuntary servitude; 280,000 Federal employees are not working, and they will be␣
↪paid. Virtually all of these workers have m'
```

To make a spaCy `Doc`, we need to apply a language-specific model pipeline to the text. (See the installation guide for details on how to download the necessary data!) Assuming most if not all of these speeches were given in English, let's use the "en_core_web_sm" pipeline:

```
>>> doc = textacy.make_spacy_doc((preproc_text, record.meta), lang="en_core_web_sm")
>>> doc._.preview
'Doc(161 tokens: "Mr. Speaker, 480,000 Federal employees are work...")'
>>> doc._.meta
{'date': '1996-01-04',
 'congress': 104,
 'speaker_name': 'Bernie Sanders',
 'speaker_party': 'I',
 'title': 'JOIN THE SENATE AND PASS A CONTINUING RESOLUTION',
 'chamber': 'House'}
```

Now, using the annotated part-of-speech tags, we can extract just the adjectives and determinants immediately preceding our keyword to get a sense of how workers are *described*:

```
>>> patterns = [{"POS": {"IN": ["ADJ", "DET"]}, "OP": "+"}, {"ORTH": {"REGEX":
↪"workers?"}}]
>>> list(textacy.extract.token_matches(doc, patterns))
[these workers, these workers]
```

Well, these particular examples aren't very interesting, but we'd definitely like to see the results aggregated over all speeches: *skilled* workers, *American* workers, *young* workers, and so on.

To accomplish that, let's load many records into a `textacy.Corpus`. *Note:* For the sake of time, we'll limit ourselves to just the first 2000 — this can take a couple minutes!

```
>>> records = dataset.records(limit=2000)
>>> preproc_records = ((preproc(text), meta) for text, meta in records)
>>> corpus = textacy.Corpus("en_core_web_sm", data=preproc_records)
>>> print(corpus)
Corpus(2000 docs, 1049192 tokens)
```

We can leverage the documents' metadata to get a better sense of what's in our corpus:

```
>>> import collections
>>> corpus.agg_metadata("date", min), corpus.agg_metadata("date", max)
('1996-01-04', '1999-10-08')
>>> corpus.agg_metadata("speaker_name", collections.Counter)
Counter({'Bernie Sanders': 421,
         'Lindsey Graham': 98,
         'Rick Santorum': 533,
         'Joseph Biden': 691,
         'John Kasich': 257})
```

We see some familiar politicians, including current president Joe Biden and noted sycophant Lindsey Graham. Now that the documents are processed, let's extract matches from each, lemmatize their texts for consistency, and then inspect the most common descriptions of workers:

```
>>> import itertools
>>> matches = itertools.chain.from_iterable(textacy.extract.token_matches(doc,␣
↪patterns) for doc in corpus)
>>> collections.Counter(match.lemma_ for match in matches).most_common(20)
[('american worker', 95),
 ('average american worker', 21),
 ('the average american worker', 20),
 ('the worker', 15),
 ('social worker', 6),
 ('those worker', 5),
 ('a worker', 5),
 ('these worker', 4),
 ('young worker', 4),
 ('average worker', 4),
 ('an american worker', 4),
 ('the american worker', 4),
 ('federal worker', 3),
 ('that american worker', 3),
 ('that worker', 3),
 ('more worker', 3),
 ('nonunion worker', 3),
 ('the average worker', 3),
 ('young american worker', 2),
 ('every worker', 2)]
```

Apparently, these speakers had a preoccupation with American workers, average workers, and *average American* workers. To better understand the context of these mentions, we can extract keyterms (the most important or "key" terms) from the documents in which they occured.

For example, here are the top 10 keyterms from that first Bernie speech in our dataset, extracted using a variation of the well-known TextRank algorithm:

```
>>> corpus[0]._.extract_keyterms("textrank", normalize="lemma", window_size=10, edge_
↪weighting="count", topn=10)
[('year balanced budget plan', 0.033721812470386026),
 ('Mr. Speaker', 0.032162715590532916),
 ('Mr. Gingrich', 0.031358819981176664),
 ('american people', 0.02612752273629427),
 ('republican leadership', 0.025418705021243045),
 ('federal employee', 0.021731159162187104),
 ('Newt Gingrich', 0.01988327361247088),
 ('pay', 0.018930131314143193),
 ('involuntary servitude', 0.015559235022115406),
 ('entire Senate', 0.015032623278646105)]
```

Now let's select the subset of speeches in which "worker(s)" were mentioned, extract the keyterms from each, then aggregate and rank the results.

```
>>> kt_weights = collections.Counter()
>>> for doc in corpus.get(lambda doc: any(doc._.extract_regex_matches("workers?"))):
...     keyterms = doc._.extract_keyterms(
...         "textrank", normalize="lemma", window_size=10, edge_weighting="count",␣
↪topn=10
...     )
...     kt_weights.update(dict(keyterms))
kt_weights.most_common(20)
[('average american worker', 0.2925480520167547),
 ('american worker', 0.21976899187473325),
 ('american people', 0.2131304787602286),
 ('real wage', 0.20937859927617333),
 ('Mr. Speaker', 0.19605562157627318),
 ('minimum wage today', 0.15268345523692883),
 ('young people', 0.13646481152944478),
 ('Social Security Social Security', 0.1361447369032916),
 ('Social Security Trust Fund', 0.12800826053880315),
 ('wage job', 0.1245701927182434),
 ('minimum wage', 0.1231061204217654),
 ('Mr. Chairman', 0.11731341389089317),
 ('low wage', 0.10747384130103463),
 ('time job', 0.10698519355007824),
 ('Multiple Chemical Sensitivity disorder', 0.09848493865271887),
 ('Mr. President', 0.09740781572099372),
 ('income people', 0.09569570041926843),
 ('Mr. Kucinich', 0.09241855965201626),
 ('violent crime trust fund', 0.08805244819537784),
 ('Social Security system', 0.08688954139546792)]
```

Perhaps unsurprisingly, "average american worker" ranks at the top of the list, but we can see from the rest of the list that they're brought up in discussion of jobs, the minimum wage, and Social Security. Makes sense!

In this tutorial, we've learned how to

- load text+metadata records from a dataset
- inspect and preprocess raw texts
- add a collection of documents processed by spaCy into a corpus
- inspect aggregated corpus metadata
- extract different kinds of structured data from one or many documents

4.3.2 Terms and Topics in the U.S. Congress

In this tutorial, we will explore the broad topics of discussion among certain members of the U.S. Congress, based on a dataset of thousands of their speeches delivered on the floor.

First, let's initialize and download the dataset, which comes built-in with `textacy`:

```
>>> import textacy.datasets
>>> dataset = textacy.datasets.CapitolWords()
>>> dataset.info
{'name': 'capitol_words',
 'site_url': 'http://sunlightlabs.github.io/Capitol-Words/',
```

(continues on next page)

(continued from previous page)

```
 'description': 'Collection of ~11k speeches in the Congressional Record given by␣
↪notable U.S. politicians between Jan 1996 and Jun 2016.'}
>>> dataset.download()
```

Each record in this dataset contains the full text of and basic metadata about the speech. Let's take a peek at the first one:

```
>>> next(dataset.records(limit=1))
Record(text='Mr. Speaker, 480,000 Federal employees are working without pay, a form␣
↪of involuntary servitude; 280,000 Federal employees are not working, and they will␣
↪be paid. Virtually all of these workers have mortgages to pay, children to feed,␣
↪and financial obligations to meet.\nMr. Speaker, what is happening to these workers␣
↪is immoral, is wrong, and must be rectified immediately. Newt Gingrich and the␣
↪Republican leadership must not continue to hold the House and the American people␣
↪hostage while they push their disastrous 7-year balanced budget plan. The gentleman␣
↪from Georgia, Mr. Gingrich, and the Republican leadership must join Senator Dole␣
↪and the entire Senate and pass a continuing resolution now, now to reopen␣
↪Government.\nMr. Speaker, that is what the American people want, that is what they␣
↪need, and that is what this body must do.', meta={'date': '1996-01-04', 'congress':␣
↪104, 'speaker_name': 'Bernie Sanders', 'speaker_party': 'I', 'title': 'JOIN THE␣
↪SENATE AND PASS A CONTINUING RESOLUTION', 'chamber': 'House'})
```

Feel the Bern, circa 1996!

Let's load the first 2000 records into a `textacy.Corpus`. We'll disable the spaCy pipeline's parser for speed (since we won't need dependency annotations), but even still this will take a couple minutes. Hang tight.

```
>>> spacy_lang = textacy.load_spacy_lang("en_core_web_sm", disable=("parser",))
>>> records = dataset.records(limit=2000)
>>> corpus = textacy.Corpus(spacy_lang, data=records)
>>> print(corpus)
Corpus(2000 docs, 1049199 tokens)
```

As we saw in another tutorial, this collection covers speeches given during the late 90s by a handful of politicians, including Bernie Sanders and Joe Biden.

```
>>> corpus.agg_metadata("date", min), corpus.agg_metadata("date", max)
('1996-01-04', '1999-10-08')
```

spaCy's tokenization and annotations provide a flexible base from which we can perform a higher-level splitting of each document into semantically meaningful "terms". For example, let's extract all entities:

```
>>> import textacy.extract
>>> list(textacy.extract.entities(corpus[0]))
[Speaker, 480,000, 280,000, Speaker, Newt Gingrich, Republican, House, American, 7-
↪year, Georgia, Gingrich, Republican, Dole, Senate, Speaker, American]
```

Upon inspection, that seems like a mixed bag, so let's clean it up a bit by including only a subset of entity types, and toss in noun- or adjective-only bigrams as well:

```
>>> from functools import partial
>>> terms = list(textacy.extract.terms(
...     corpus[0],
...     ngs=partial(textacy.extract.ngrams, n=2, include_pos={"NOUN", "ADJ"}),
...     ents=partial(textacy.extract.entities, include_types={"PERSON", "ORG", "GPE",
↪"LOC"}),
```

(continues on next page)

(continued from previous page)

```
...     dedupe=True))
>>> terms
[Federal employees, involuntary servitude, Federal employees, financial obligations,␣
↪Republican leadership, American people, year balanced, balanced budget, budget plan,
↪ Republican leadership, American people, Speaker, Speaker, Newt Gingrich, House,␣
↪Georgia, Gingrich, Dole, Senate, Speaker]
```

Note that "Speaker" (as in *Mr.* Speaker) shows up multiple times: the `dedupe` arg removes exact duplicates based on their positions in the text, but not by their text content.

Before building a document-term matrix representation of the corpus, we must first transform the terms' `Span` objects into strings. There are several options to choose from: use the text content as-is, lowercase it, or if available use lemmas (base forms without inflectional suffixes). To reduce sparsity of the matrix, let's lemmatize the terms:

```
>>> list(textacy.extract.terms_to_strings(terms, by="lemma"))
['federal employee', 'involuntary servitude', 'federal employee', 'financial␣
↪obligation', 'republican leadership', 'american people', 'year balanced', 'balanced␣
↪budget', 'budget plan', 'republican leadership', 'american people', 'Speaker',
↪'Speaker', 'Newt Gingrich', 'House', 'Georgia', 'Gingrich', 'Dole', 'Senate',
↪'Speaker']
```

Looks good! Let's apply these steps to all docs in the corpus:

```
>>> docs_terms = (
...     textacy.extract.terms(
...         doc,
...         ngs=partial(textacy.extract.ngrams, n=2, include_pos={"NOUN", "ADJ"}),
...         ents=partial(textacy.extract.entities, include_types={"PERSON", "ORG",
↪"GPE", "LOC"}))
...     for doc in corpus)
>>> tokenized_docs = (
...     textacy.extract.terms_to_strings(doc_terms, by="lemma")
...     for doc_terms in docs_terms)
```

Now we can vectorize the documents. Each row represents a document, each column a unique term, and individual values represent the "weight" of a term in a particular document. These weights may include combinations of local, global, and normalization components; for simplicity, let's use classic TF-IDF weighting, i.e. "Term Frequency" (local) multiplied by "Inverse Doc Frequency" (global).

```
>>> import textacy.representations
>>> doc_term_matrix, vocab = textacy.representations.build_doc_term_matrix(tokenized_
↪docs, tf_type="linear", idf_type="smooth")
>>> doc_term_matrix
<2000x30177 sparse matrix of type '<class 'numpy.float64'>'
            with 58693 stored elements in Compressed Sparse Row format>
```

Let's initialize and fit a topic model to this data. `textacy` provides a common interface to three basic topic models; we'll use an "NMF" model here, and configure it (without any optimization) to use 10 topics.

```
>>> import textacy.tm
>>> model = textacy.tm.TopicModel("nmf", n_topics=10)
>>> model.fit(doc_term_matrix)
```

Using the fit model, we can transform the doc-term matrix into a doc-*topic* matrix, where the columns now correspond to topics and values represent the degree to which a given document is associated with a given topic.

```
>>> doc_topic_matrix = model.transform(doc_term_matrix)
>>> doc_topic_matrix.shape
(2000, 10)
>>> doc_topic_matrix
array([[0.        , 0.        , 0.        , ..., 0.29051627, 0.03107776,
        0.00069874],
       [0.        , 0.        , 0.        , ..., 0.08144687, 0.        ,
        0.        ],
       [0.00210755, 0.        , 0.        , ..., 0.2770269 , 0.        ,
        0.        ],
       ...,
       [0.00559379, 0.00188866, 0.0259026 , ..., 0.01886715, 0.04181629,
        0.00639968],
       [0.        , 0.        , 0.00083651, ..., 0.        , 0.00209634,
        0.        ],
       [0.00407539, 0.00100207, 0.0066426 , ..., 0.05791785, 0.        ,
        0.00239545]])
```

To better understand the topics, we can extract a list of its top terms (those with the highest topic weight), as well as the top documents.

```
>>> id_to_term = {id_: term for term, id_ in vocab.items()}
>>> for topic_idx, terms in model.top_topic_terms(id_to_term, top_n=8):
...     print(f"topic {topic_idx}: {'   '.join(terms)}")
topic 0: NATO   Europe   Russia   Hungary   Poland   Czech Republic   United States␣
↪Madrid
topic 1: raw material   medical device   biomaterial supplier   component part␣
↪civil action   rating system   product liability   DuPont
topic 2: China   great power   United States   international norm   human right␣
↪Pakistan   nuclear weapon   Beijing
topic 3: chemical weapon   Reagan   Bush   Helms   poison gas   Chemical Weapons␣
↪Convention   Saddam   Iraq
topic 4: missile defense   Russia   national missile   nuclear weapon   arm control␣
↪ballistic missile   United States   Soviet Union
topic 5: United Nations   State Department   U.N.   foreign policy   Mexico City␣
↪North Carolina   Helms   U.S.
topic 6: Milosevic   Kosovo   Serbia   Bosnia   NATO   KLA   Belgrade   war criminal
topic 7: Speaker   America   Mexico   health care   middle class   Congress   United␣
↪States   new job
topic 8: birth abortion   Tony   Senate   little baby   partial birth   Donna Joy␣
↪Tony Melendez   Lori
topic 9: CWC   chemical weapon   chemical industry   poison gas   U.S.   Chemical␣
↪Weapons Convention   american chemical   rogue state
>>> for topic_idx, doc_idxs in model.top_topic_docs(doc_topic_matrix, top_n=3):
...     print(f"topic {topic_idx}: {'   '.join(corpus[doc_idx]._.meta['title'] for␣
↪doc_idx in doc_idxs)}")
topic 0: EXECUTIVE SESSION   THE STRATEGIC RATIONALE FOR NATO ENLARGEMENT   NATO␣
↪ENLARGEMENT AFTER PARIS
topic 1: STATEMENTS ON INTRODUCED BILLS AND JOINT RESOLUTIONS   STATEMENTS ON␣
↪INTRODUCED BILLS AND JOINT RESOLUTIONS   DEPARTMENTS OF COMMERCE, JUSTICE, AND␣
↪STATE, THE JUDICIARY, AND RELATED AGENCIES APPROPRIATIONS ACT, 1999
topic 2: THE CHINA SUMMIT: WHAT KIND OF ENGAGEMENT?   THE SEARCH FOR MODERN CHINA:␣
↪THE PRESIDENT'S CHINA TRIP   FOREIGN OPERATIONS, EXPORT FINANCING, AND RELATED␣
↪PROGRAMS APPROPRIATIONS ACT, 1998
topic 3: EXECUTIVE SESSION   FIRST ANNIVERSARY OF THE ENTRY INTO FORCE OF THE␣
↪CHEMICAL WEAPONS CONVENTION   CHEMICAL WEAPONS CONVENTION
topic 4: NATIONAL MISSILE DEFENSE ACT OF 1999   CRISIS IN RUSSIA   AMERICAN MISSILE␣
↪PROTECTION ACT OF 1998--MOTION TO PROCEED
```

(continues on next page)

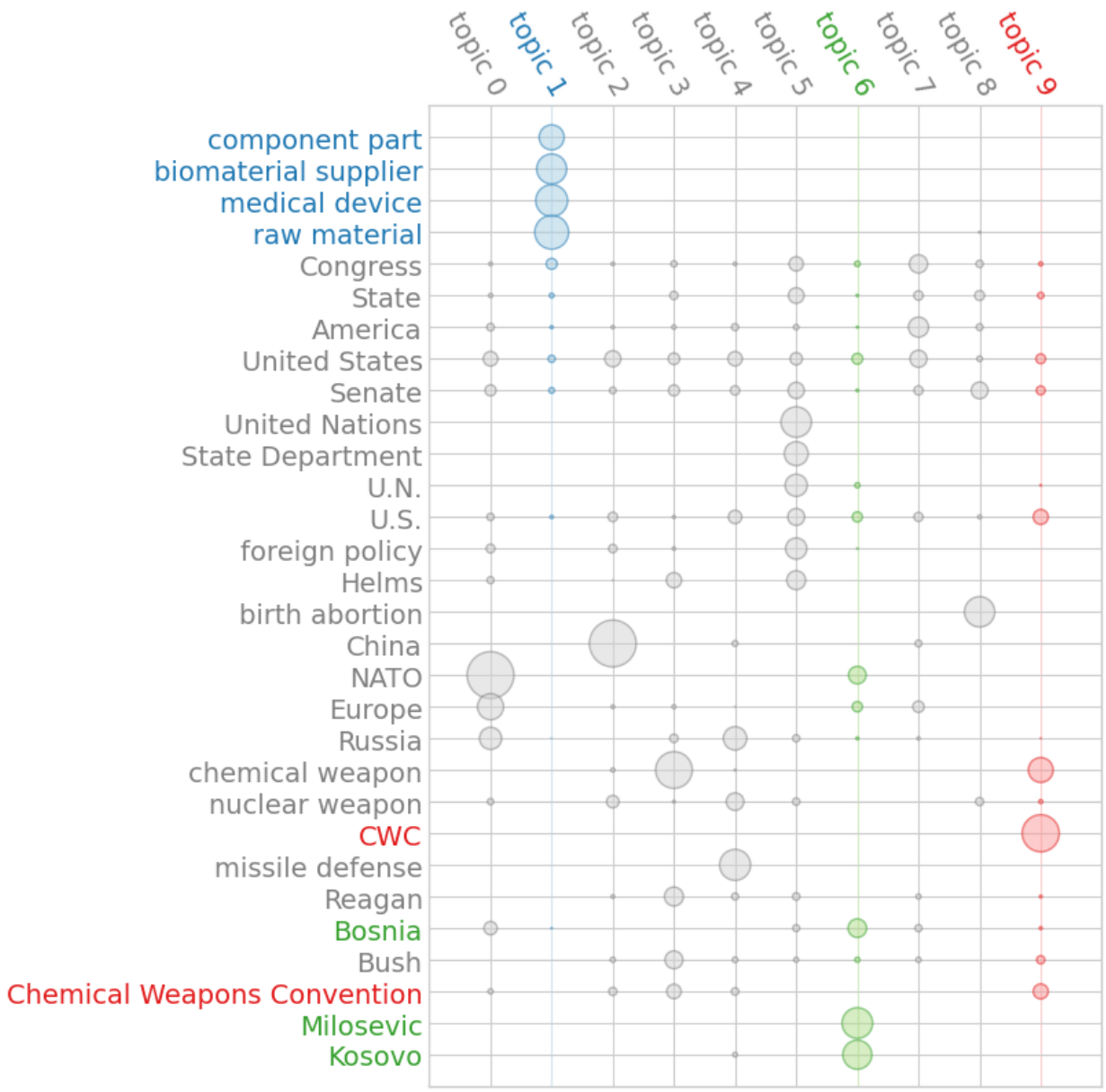
(continued from previous page)

```
topic 5: FOREIGN AFFAIRS REFORM AND RESTRUCTURING ACT OF 1997   FOREIGN AFFAIRS␣
↪REFORM AND RESTRUCTURING ACT--CONFERENCE REPORT   FOREIGN AFFAIRS REFORM AND␣
↪RESTRUCTURING ACT OF 1997
topic 6: THE SITUATION IN KOSOVO   RESOLUTION OF THE KOSOVO PROBLEM   PEACE AGREEMENT
topic 7: THE MOST IMPORTANT ISSUES FACING THE AMERICAN PEOPLE   55TH ANNIVERSARY OF␣
↪THE BATTLE OF CRETE   NARCOTICS CERTIFICATION
topic 8: PARTIAL-BIRTH ABORTION   PARTIAL-BIRTH ABORTION BAN ACT OF 1997   PARTIAL-
↪BIRTH ABORTION BAN ACT of 1997
topic 9: THE URGENT NEED TO OUTLAW POISON GAS   CHEMICAL WEAPONS CONVENTION   ␣
↪EXECUTIVE SESSION
```

At first glance, most of these topics seem relatively interpretable: topic 1 looks to be about medical device manufacturing and liability, topic 6 is focused on the late 90s Kosovo War, topic 9 deals with chemical weapons and related treaties, etc. Seems reasonable!

We can also visualize the relationship between terms and topics using a "termite" plot, where the area of each circle is proportional to a term's weight in a given topic. To aid the eye, we've highlighted those topics called out above:

```
>>> _ = model.termite_plot(doc_term_matrix, id_to_term, n_terms=30, highlight_
↪topics=[1, 6, 9])
```



In this tutorial, we learned how to

- load text+metadata records from a dataset
- add many records to a corpus using a customized spaCy language pipeline
- extract and stringify higher-level "terms" to represent a document
- transform a collection of documents into a doc-term matrix
- fit, inspect, and visualize a topic model

4.4 API Reference

4.4.1 Lang, Doc, Corpus

textacy.spacier.core: Convenient entry point for loading spaCy language pipelines and making spaCy docs.

textacy.spacier.core.**load_spacy_lang**(*name: str | pathlib.Path*, ***kwargs*) → Language

Load a spaCy `Language` — a shared vocabulary and language-specific data for tokenizing text, and (if available) model data and a processing pipeline containing a sequence of components for annotating a document — and cache results, for quick reloading as needed.

Note that as of spaCy v3, for which pipeline aliases are no longer allowed, this function is just a convenient access point to underlying `spacy.load()`.

```
>>> en_nlp = textacy.load_spacy_lang("en_core_web_sm")
>>> en_nlp = textacy.load_spacy_lang("en_core_web_sm", disable=("parser",))
>>> textacy.load_spacy_lang("ar")
...
OSError: [E050] Can't find model 'ar'. It doesn't seem to be a Python package or␣
↪a valid path to a data directory.
```

Parameters

- **name** – Name or path to the spaCy language pipeline to load.
- ****kwargs** –

Note: Although spaCy's API specifies some kwargs as `List[str]`, here we require `Tuple[str, ...]` equivalents. Language pipelines are stored in an LRU cache with unique identifiers generated from the hash of the function name and args — and lists aren't hashable.

Returns Loaded spaCy `Language`.

Raises **OSError** –

See also:

https://spacy.io/api/top-level#spacy.load

textacy.spacier.core.**make_spacy_doc**(*data: Union[str, textacy.types.Record, spacy.tokens.doc.Doc]*, *lang: Union[str, pathlib.Path, spacy.language.Language, Callable[[str], str], Callable[[str], pathlib.Path], Callable[[str], spacy.language.Language]]*, ***, *chunk_size: Optional[int] = None*) → spacy.tokens.doc.Doc

Make a `spacy.tokens.Doc` from valid inputs, and automatically load/validate `spacy.language.Language` pipelines to process `data`.

Make a `Doc` from text:

```
>>> text = "To be, or not to be, that is the question."
>>> doc = make_spacy_doc(text, "en_core_web_sm")
>>> doc._.preview
'Doc(13 tokens: "To be, or not to be, that is the question.")'
```

Make a `Doc` from a (text, metadata) pair, aka a "record":

```
>>> record = (text, {"author": "Shakespeare, William"})
>>> doc = make_spacy_doc(record, "en_core_web_sm")
>>> doc._.preview
'Doc(13 tokens: "To be, or not to be, that is the question.")'
>>> doc._.meta
{'author': 'Shakespeare, William'}
```

Specify the language pipeline used to process the text in a few different ways:

```
>>> make_spacy_doc(text, lang="en_core_web_sm")
>>> make_spacy_doc(text, lang=textacy.load_spacy_lang("en_core_web_sm"))
>>> make_spacy_doc(text, lang=lambda txt: "en_core_web_sm")
```

Ensure that an already-processed `Doc` is compatible with `lang`:

```
>>> spacy_lang = textacy.load_spacy_lang("en_core_web_sm")
>>> doc = spacy_lang(text)
>>> make_spacy_doc(doc, lang="en_core_web_sm")
>>> make_spacy_doc(doc, lang="es_core_news_sm")
...
ValueError: `spacy.Vocab` used to process document must be the same as that used␣
↪by the `lang` pipeline ('es_core_news_sm')
```

Parameters

- **data** – Make a `spacy.tokens.Doc` from a text or (text, metadata) pair. If already a `Doc`, ensure that it's compatible with `lang` to avoid surprises downstream, and return it as-is.
- **lang** – Language with which spaCy processes (or processed) `data`, represented as the full name of a spaCy language pipeline, the path on disk to it, an already instantiated pipeline, or a callable function that takes the text component of `data` and outputs one of the above representations.
- **chunk_size** – Size of chunks in number of characters into which `text` will be split before processing each via spaCy and concatenating the results into a single `Doc`.

  Note: This is intended as a workaround for processing very long texts, for which spaCy is unable to allocate enough RAM. For best performance, chunk size should be somewhere between 1e3 and 1e7 characters, depending on how much RAM you have available.

  Since chunking is done by *character*, chunks' boundaries likely won't respect natural language segmentation, and as a result spaCy's models may make mistakes on sentences/words that cross them.

Returns Processed spaCy Doc.

Raises

- **TypeError** –
- **ValueError** –

textacy.corpus: Class for working with a collection of spaCy `Doc` s. Includes functionality for easily adding, getting, and removing documents; saving to / loading their data from disk; and tracking basic corpus statistics.

class textacy.corpus.**Corpus**(*lang: Union[str, pathlib.Path, spacy.language.Language]*, *data: Optional[Union[str, textacy.types.Record, spacy.tokens.doc.Doc, Iterable[str], Iterable[textacy.types.Record], Iterable[spacy.tokens.doc.Doc]]] = None*)

An ordered collection of `spacy.tokens.Doc`, all of the same language and sharing the same `spacy.language.Language` processing pipeline and vocabulary, with data held *in-memory*.

Initialize from a `Language` name or instance and (optionally) one or a stream of texts or (text, metadata) pairs:

```
>>> ds = textacy.datasets.CapitolWords()
>>> records = ds.records(limit=50)
>>> corpus = textacy.Corpus("en_core_web_sm", data=records)
>>> print(corpus)
Corpus(50 docs, 32175 tokens)
```

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

```
>>> texts = ds.texts(congress=114, limit=25)
>>> corpus.add(texts)
>>> corpus.add("If Burton were a member of Congress, here's what he'd say.")
>>> print(corpus)
Corpus(76 docs, 55906 tokens)
>>> corpus.remove(lambda doc: doc._.meta.get("speaker_name") == "Rick Santorum")
>>> print(corpus)
Corpus(61 docs, 48567 tokens)
```

Get subsets of documents matching your particular use case:

```
>>> match_func = lambda doc: doc._.meta.get("speaker_name") == "Bernie Sanders"
>>> for doc in corpus.get(match_func, limit=3):
...     print(doc._.preview)
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...")
```

Get or remove documents by indexing, too:

```
>>> corpus[0]._.preview
'Doc(159 tokens: "Mr. Speaker, 480,000 Federal employees are work...")'
>>> [doc._.preview for doc in 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...")']
>>> del corpus[:5]
>>> print(corpus)
Corpus(56 docs, 41573 tokens)
```

Compute basic corpus statistics:

```
>>> corpus.n_docs, corpus.n_sents, corpus.n_tokens
(56, 1771, 41573)
>>> word_counts = corpus.word_counts(by="lemma_")
>>> sorted(word_counts.items(), key=lambda x: x[1], reverse=True)[:5]
[('-PRON-', 2553), ('people', 215), ('year', 148), ('Mr.', 139), ('$', 137)]
>>> word_doc_counts = corpus.word_doc_counts(by="lemma_", weighting="freq")
>>> sorted(word_doc_counts.items(), key=lambda x: x[1], reverse=True)[:5]
[('-PRON-', 0.9821428571428571),
 ('Mr.', 0.7678571428571429),
```

(continues on next page)

(continued from previous page)

```
 ('President', 0.5),
 ('people', 0.48214285714285715),
 ('need', 0.44642857142857145)]
```

Save corpus data to and load from disk:

```
>>> corpus.save("./cw_sample.bin.gz")
>>> corpus = textacy.Corpus.load("en_core_web_sm", "./cw_sample.bin.gz")
>>> print(corpus)
Corpus(56 docs, 41573 tokens)
```

Parameters

- **lang** – Language with which spaCy processes (or processed) all documents added to the corpus, whether as `data` now or later.

  Pass the name of a spacy language pipeline (e.g. "en_core_web_sm"), or an already-instantiated `spacy.language.Language` object.

  A given / detected language string is then used to instantiate a corresponding `Language` with all default components enabled.

- **data** – One or a stream of texts, records, or `spacy.tokens.Doc` s to be added to the corpus.

  See also:

  Corpus.add()

lang

Type str

spacy_lang

Type spacy.language.Language

docs

Type List[spacy.tokens.doc.Doc]

n_docs

Type int

n_sents

Type int

n_tokens

Type int

add(*data: Union[str, textacy.types.Record, spacy.tokens.doc.Doc, Iterable[str], Iterable[textacy.types.Record], Iterable[spacy.tokens.doc.Doc]]*, *batch_size: int = 1000*, *n_process: int = 1*)

Add one or a stream of texts, records, or `spacy.tokens.Doc` s to the corpus, ensuring that all processing is or has already been done by the *Corpus.spacy_lang* pipeline.

Parameters

- **data** –
- **batch_size** – Number of texts to buffer when processing with spaCy.

- **n_process** – Number of parallel processors to run when processing. If -1, this is set to `multiprocessing.cpu_count()`.

  Note: This feature is only applies when `data` is a sequence of texts or records.

See also:

- *Corpus.add_text()*
- *Corpus.add_texts()*
- *Corpus.add_record()*
- *Corpus.add_records()*
- *Corpus.add_doc()*
- *Corpus.add_docs()*

add_text(*text: str*) → None

Add one text to the corpus, processing it into a `spacy.tokens.Doc` using the *Corpus.spacy_lang* pipeline.

Parameters **text** (*str*) –

add_texts(*texts: Iterable[str]*, *batch_size: int = 1000*, *n_process: int = 1*) → None

Add a stream of texts to the corpus, efficiently processing them into `spacy.tokens.Doc` s using the *Corpus.spacy_lang* pipeline.

Parameters

- **texts** – Sequence of texts to process and add to corpus.
- **batch_size** – Number of texts to buffer when processing with spaCy.
- **n_process** – Number of parallel processors to run when processing. If -1, this is set to `multiprocessing.cpu_count()`.

  Note: This feature is only available in spaCy 2.2.2+.

add_record(*record:* textacy.types.Record) → None

Add one record to the corpus, processing it into a `spacy.tokens.Doc` using the *Corpus.spacy_lang* pipeline.

Parameters **record** –

add_records(*records: Iterable[*textacy.types.Record*]*, *batch_size: int = 1000*, *n_process: int = 1*) → None

Add a stream of records to the corpus, efficiently processing them into `spacy.tokens.Doc` s using the *Corpus.spacy_lang* pipeline.

Parameters

- **records** – Sequence of records to process and add to corpus.
- **batch_size** – Number of texts to buffer when processing with spaCy.
- **n_process** – Number of parallel processors to run when processing. If -1, this is set to `multiprocessing.cpu_count()`.

Note: This feature is only available in spaCy 2.2.2+.

add_doc(*doc: spacy.tokens.doc.Doc*) → None
Add one `spacy.tokens.Doc` to the corpus, provided it was processed using the *Corpus.spacy_lang* pipeline.

Parameters **doc** –

add_docs(*docs: Iterable[spacy.tokens.doc.Doc]*) → None
Add a stream of `spacy.tokens.Doc` s to the corpus, provided they were processed using the *Corpus.spacy_lang* pipeline.

Parameters **docs** –

get(*match_func: Callable[[spacy.tokens.doc.Doc], bool]*, *limit: Optional[int] = None*) → Iterator[spacy.tokens.doc.Doc]
Get all (or N <= `limit`) docs in *Corpus* for which `match_func(doc)` is True.

Parameters

- **match_func** – Function that takes a `spacy.tokens.Doc` as input and returns a boolean value. For example:

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

gets all docs with at least 100 tokens. And:

```
Corpus.get(lambda doc: doc._.meta["author"] == "Burton DeWilde")
```

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

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

Yields `spacy.tokens.Doc` – Next document passing `match_func`.

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.

remove(*match_func: Callable[[spacy.tokens.doc.Doc], bool]*, *limit: Optional[int] = None*) → None
Remove all (or N <= `limit`) docs in *Corpus* for which `match_func(doc)` is True. Corpus doc/sent/token counts are adjusted accordingly.

Parameters

- **match_func** – Function that takes a `spacy.tokens.Doc` and returns a boolean value. For example:

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

removes docs with at least 100 tokens. And:

```
Corpus.remove(lambda doc: doc._.meta["author"] == "Burton DeWilde
↪")
```

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

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

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.

property vectors
Constituent docs' word vectors stacked in a 2d array.

property vector_norms
Constituent docs' L2-normalized word vectors stacked in a 2d array.

word_counts(*, *by: str = 'lemma'*, *weighting: str = 'count'*, ***kwargs*) → Dict[int, int | float] | Dict[str, int | float]
Map the set of unique words in *Corpus* to their counts as absolute, relative, or binary frequencies of occurence, similar to *Doc._.to_bag_of_words()* but aggregated over all docs.

Parameters

- **by** – Attribute by which spaCy `Token` s are grouped before counting, as given by `getattr(token, by)`. If "lemma", tokens are grouped by their base form w/o inflections; if "lower", by the lowercase form of the token text; if "norm", by the normalized form of the token text; if "orth", by the token text exactly as it appears in documents. To output keys as strings, append an underscore to any of these options; for example, "**lemma_**" groups tokens by their lemmas as strings.
- **weighting** – Type of weighting to assign to unique words given by `by`. If "count", weights are the absolute number of occurrences (i.e. counts); if "freq", weights are counts normalized by the total token count, giving their relative frequency of occurrence.
- ****kwargs** – Passed directly on to `textacy.extract.words()` - filter_stops: If True, stop words are removed before counting. - filter_punct: If True, punctuation tokens are removed before counting. - filter_nums: If True, number-like tokens are removed before counting.

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

See also:

textacy.representations.matrix_utils.get_term_freqs()

word_doc_counts(*, *by: str = 'lemma'*, *weighting: str = 'count'*, *smooth_idf: bool = True*, ***kwargs*) → Dict[int, int | float] | Dict[str, int | float]
Map the set of unique words in *Corpus* to their *document* counts as absolute, relative, or inverse frequencies of occurence.

Parameters

- **by** – Attribute by which spaCy `Token` s are grouped before counting, as given by `getattr(token, by)`. If "lemma", tokens are grouped by their base form w/o inflections; if "lower", by the lowercase form of the token text; if "norm", by the normalized form of the token text; if "orth", by the token text exactly as it appears in documents. To output keys as strings, append an underscore to any of these options; for example, "**lemma_**" groups tokens by their lemmas as strings.
- **weighting** – Type of weighting to assign to unique words given by `by`. If "count", weights are the absolute number of occurrences (i.e. counts); if "freq", weights are counts normalized by the total token count, giving their relative frequency of occurrence; if "idf", weights are the log of the inverse relative frequencies, i.e. `log(n_docs / word_doc_count)` or, if `smooth_idf` is True, `log(1 + (n_docs / word_doc_count))`.

- **smooth_idf** – If True, add 1 to all word doc counts when calculating "idf" weighting, equivalent to adding a single document to the corpus containing every unique word.

Returns Mapping of a unique word id or string (depending on the value of `by`) to the number of documents in which it appears, weighted as absolute, relative, or inverse frequency of occurrence (depending on the value of `weighting`).

See also:

`textacy.vsm.get_doc_freqs()`

agg_metadata(*name: str*, *agg_func: Callable[[Iterable[Any]], Any]*, *default: Optional[Any] = None*) → Any

Aggregate values for a particular metadata field over all documents in *Corpus*.

Parameters

- **name** – Name of metadata field (key) in `Doc._.meta`.
- **agg_func** – Callable that accepts an iterable of field values and outputs a single, aggregated result.
- **default** – Default field value to use if `name` is not found in a given document's metadata.

Returns Aggregated value for metadata field.

save(*filepath: types.PathLike*, *attrs: Optional[str | Iterable[str]] = 'auto'*, *store_user_data: bool = True*)

Save *Corpus* to disk as binary data.

Parameters

- **filepath** – Full path to file on disk where *Corpus* docs data will be saved as a binary file.
- **attrs** – List of token attributes to serialize; if "auto", an appropriate list is inferred from annotations found on the first `Doc`; if None, spaCy's default values are used (https://spacy.io/api/docbin#init)
- **store_user_data** – If True, store user data and values of custom extension attributes along with core spaCy attributes.

See also:

- *Corpus.load()*
- `textacy.io.write_spacy_docs()`
- `spacy.tokens.DocBin`

classmethod load(*lang: Union[str, pathlib.Path, spacy.language.Language]*, *filepath: Union[str, pathlib.Path]*) → *Corpus*

Load previously saved *Corpus* binary data, reproduce the original *:class:`spacy.tokens.Doc`s tokens and annotations, and instantiate a new :class:`Corpus* from them.

Parameters

- **lang** –
- **filepath** – Full path to file on disk where *Corpus* data was previously saved as a binary file.

Returns Initialized corpus.

See also:

- *Corpus.save()*
- textacy.io.read_spacy_docs()
- spacy.tokens.DocBin

Doc Extensions

get_preview	Get a short preview of the Doc, including the number of tokens and an initial snippet.
get_meta	Get custom metadata added to Doc.
set_meta	Add custom metadata to Doc.
to_tokenized_text	Transform doc into an ordered, nested list of token-texts for each sentence.
to_bag_of_words	Transform a Doc or Span into a bag-of-words: the set of unique words therein mapped to their absolute, relative, or binary frequencies of occurrence.
to_bag_of_terms	Transform a Doc or Span into a bag-of-terms: the set of unique terms therein mapped to their absolute, relative, or binary frequencies of occurrence, where "terms" may be a combination of n-grams, entities, and/or noun chunks.

textacy.extensions: Inspect, extend, and transform spaCy's core Doc data structure, either directly via functions that take a Doc as their first arg or as custom attributes / methods on instantiated docs prepended by an underscore:

```
>>> doc = textacy.make_spacy_doc("This is a short text.", "en_core_web_sm")
>>> print(get_preview(doc))
Doc(6 tokens: "This is a short text.")
>>> print(doc._.preview)
Doc(6 tokens: "This is a short text.")
```

textacy.extensions.**get_preview**(*doc: spacy.tokens.doc.Doc*) → str
Get a short preview of the Doc, including the number of tokens and an initial snippet.

textacy.extensions.**get_meta**(*doc: spacy.tokens.doc.Doc*) → dict
Get custom metadata added to Doc.

textacy.extensions.**set_meta**(*doc: spacy.tokens.doc.Doc*, *value: dict*) → None
Add custom metadata to Doc.

textacy.extensions.**to_tokenized_text**(*doc: spacy.tokens.doc.Doc*) → List[List[str]]
Transform doc into an ordered, nested list of token-texts for each sentence.

Parameters **doc** –

Returns A list of tokens' texts for each sentence in doc.

Note: If doc hasn't been segmented into sentences, the entire document is treated as a single sentence.

textacy.extensions.**to_bag_of_words**(*doclike: types.DocLike*, *, *by: str = 'lemma_'*, *weighting: str = 'count'*, ***kwargs*) → Dict[int, int | float] | Dict[str, int | float]

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

Parameters

- **doclike** –
- **by** – Attribute by which spaCy `Token` s are grouped before counting, as given by `getattr(token, by)`. If "lemma", tokens are grouped by their base form w/o inflectional suffixes; if "lower", by the lowercase form of the token text; if "norm", by the normalized form of the token text; if "orth", by the token text exactly as it appears in `doc`. To output keys as strings, simply append an underscore to any of these; for example, "**lemma_**" creates a bag whose keys are token lemmas as strings.
- **weighting** – Type of weighting to assign to unique words given by `by`. If "count", weights are the absolute number of occurrences (i.e. counts); if "freq", weights are counts normalized by the total token count, giving their relative frequency of occurrence; if "binary", weights are set equal to 1.
- ****kwargs** – Passed directly on to `textacy.extract.words()` - filter_stops: If True, stop words are removed before counting. - filter_punct: If True, punctuation tokens are removed before counting. - filter_nums: If True, number-like tokens are removed before counting.

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

Note: For "freq" weighting, the resulting set of frequencies won't (necessarily) sum to 1.0, since all tokens are used when normalizing counts but some (punctuation, stop words, etc.) may be filtered out of the bag afterwards.

See also:

`textacy.extract.words()`

`textacy.extensions.`**`to_bag_of_terms`**(*doclike: types.DocLike*, ***, *by: str = 'lemma_'*, *weighting: str = 'count'*, *ngs: Optional[int | Collection[int] | types.DocLikeToSpans] = None*, *ents: Optional[bool | types.DocLikeToSpans] = None*, *ncs: Optional[bool | types.DocLikeToSpans] = None*, *dedupe: bool = True*) → Dict[str, int] | Dict[str, float]

Transform a `Doc` or `Span` into a bag-of-terms: the set of unique terms therein mapped to their absolute, relative, or binary frequencies of occurrence, where "terms" may be a combination of n-grams, entities, and/or noun chunks.

Parameters

- **doclike** –
- **by** – Attribute by which spaCy `Span` s are grouped before counting, as given by `getattr(token, by)`. If "lemma", tokens are counted by their base form w/o inflectional suffixes; if "lower", by the lowercase form of the token text; if "orth", by the token text exactly as it appears in `doc`. To output keys as strings, simply append an underscore to any of these; for example, "**lemma_**" creates a bag whose keys are token lemmas as strings.
- **weighting** – Type of weighting to assign to unique terms given by `by`. If "count", weights are the absolute number of occurrences (i.e. counts); if "freq", weights are counts normalized by the total token count, giving their relative frequency of occurrence; if "binary", weights are set equal to 1.

- **ngs** – N-gram terms to be extracted. If one or multiple ints, `textacy.extract.ngrams(doclike, n=ngs)` is used to extract terms; if a callable, `ngs(doclike)` is used to extract terms; if None, no n-gram terms are extracted.
- **ents** – Entity terms to be extracted. If True, `textacy.extract.entities(doclike)` is used to extract terms; if a callable, `ents(doclike)` is used to extract terms; if None, no entity terms are extracted.
- **ncs** – Noun chunk terms to be extracted. If True, `textacy.extract.noun_chunks(doclike)` is used to extract terms; if a callable, `ncs(doclike)` is used to extract terms; if None, no noun chunk terms are extracted.
- **dedupe** – If True, deduplicate terms whose spans are extracted by multiple types (e.g. a span that is both an n-gram and an entity), as identified by identical (start, stop) indexes in `doclike`; otherwise, don't.

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

See also:

`textacy.extract.terms()`

`textacy.extensions.`**`get_doc_extensions`**`()` → Dict[str, Dict[str, Any]]

Get textacy's custom property and method doc extensions that can be set on or removed from the global `spacy.tokens.Doc`.

`textacy.extensions.`**`set_doc_extensions`**`()`

Set textacy's custom property and method doc extensions on the global `spacy.tokens.Doc`.

`textacy.extensions.`**`remove_doc_extensions`**`()`

Remove textacy's custom property and method doc extensions from the global `spacy.tokens.Doc`.

4.4.2 Datasets and Resources

capitol_words.CapitolWords	Stream a collection of Congressional speeches from a compressed json file on disk, either as texts or text + metadata pairs.
supreme_court.SupremeCourt	Stream a collection of US Supreme Court decisions from a compressed json file on disk, either as texts or text + metadata pairs.
wikimedia.Wikipedia	Stream a collection of Wikipedia pages from a version- and language-specific database dump, either as texts or text + metadata pairs.
wikimedia.Wikinews	Stream a collection of Wikinews pages from a version- and language-specific database dump, either as texts or text + metadata pairs.
reddit_comments.RedditComments	Stream a collection of Reddit comments from 1 or more compressed files on disk, either as texts or text + metadata pairs.
oxford_text_archive.OxfordTextArchive	Stream a collection of English-language literary works from text files on disk, either as texts or text + metadata pairs.
imdb.IMDB	Stream a collection of IMDB movie reviews from text files on disk, either as texts or text + metadata pairs.

continues on next page

Table 2 – continued from previous page

udhr.UDHR	Stream a collection of UDHR translations from disk, either as texts or text + metadata pairs.

concept_net.ConceptNet	Interface to ConceptNet, a multilingual knowledge base representing common words and phrases and the common-sense relationships between them.
depeche_mood.DepecheMood	Interface to DepecheMood, an emotion lexicon for English and Italian text.

Capitol Words Congressional speeches

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 data:

- `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, "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: Union[str, pathlib.Path] = PosixPath('/home/docs/checkouts/readthedocs.org/user_build packages/textacy/data/capitol_words')*)

Stream a collection of Congressional speeches from a compressed json file on disk, either as texts or text + metadata pairs.

Download the data (one time only!) from the textacy-data repo (https://github.com/bdewilde/textacy-data), and save its contents to disk:

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

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

```
>>> for text in ds.texts(limit=3):
...     print(text, end="\n\n")
>>> for text, meta in ds.records(limit=3):
...     print("\n{} ({})\n{}".format(meta["title"], meta["speaker_name"], text))
```

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

```
>>> for text, meta in ds.records(speaker_name="Bernie Sanders", limit=3):
...     print("\n{}, {}\n{}".format(meta["title"], meta["date"], text))
>>> for text, meta in ds.records(speaker_party="D", congress={110, 111, 112},
...                              chamber="Senate", limit=3):
...     print(meta["title"], meta["speaker_name"], meta["date"])
>>> for text, meta in ds.records(speaker_name={"Barack Obama", "Hillary Clinton"},
...                              date_range=("2005-01-01", "2005-12-31")):
...     print(meta["title"], meta["speaker_name"], meta["date"])
>>> for text in ds.texts(min_len=50000):
...     print(len(text))
```

Stream speeches into a *textacy.Corpus*:

```
>>> textacy.Corpus("en", data=ota.records(limit=100))
Corpus(100 docs; 70496 tokens)
```

Parameters **data_dir** – Path to directory on disk under which dataset is stored, i.e. `/path/to/data_dir/capitol_words`.

full_date_range
First and last dates for which speeches are available, each as an ISO-formatted string (YYYY-MM-DD).

Type Tuple[str, str]

speaker_names
Full names of all speakers included in corpus, e.g. "Bernie Sanders".

Type Set[str]

speaker_parties
All distinct political parties of speakers, e.g. "R".

Type Set[str]

chambers
All distinct chambers in which speeches were given, e.g. "House".

Type Set[str]

congresses
All distinct numbers of the congresses in which speeches were given, e.g. 114.

Type Set[int]

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

download(*, *force: bool = False*) → None
Download the data as a Python version-specific compressed json file and save it to disk under the `data_dir` directory.

Parameters **force** – If True, download the dataset, even if it already exists on disk under `data_dir`.

texts(*, *speaker_name: Optional[Union[str, Set[str]]] = None*, *speaker_party: Optional[Union[str, Set[str]]] = None*, *chamber: Optional[Union[str, Set[str]]] = None*, *congress: Optional[Union[int, Set[int]]] = None*, *date_range: Optional[Tuple[Optional[str], Optional[str]]] = None*, *min_len: Optional[int] = None*, *limit: Optional[int] = None*) → Iterable[str]

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

Parameters

- **speaker_name** – Filter speeches by the speakers' name; see *CapitolWords.speaker_names*.
- **speaker_party** – Filter speeches by the speakers' party; see *CapitolWords.speaker_parties*.
- **chamber** – Filter speeches by the chamber in which they were given; see *CapitolWords.chambers*.
- **congress** – Filter speeches by the congress in which they were given; see *CapitolWords.congresses*.
- **date_range** – Filter speeches 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 for the dataset.
- **min_len** – Filter texts by the length (# characters) of their text content.
- **limit** – Yield no more than `limit` texts that match all specified filters.

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

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

records(*, *speaker_name: Optional[Union[str, Set[str]]] = None*, *speaker_party: Optional[Union[str, Set[str]]] = None*, *chamber: Optional[Union[str, Set[str]]] = None*, *congress: Optional[Union[int, Set[int]]] = None*, *date_range: Optional[Tuple[Optional[str], Optional[str]]] = None*, *min_len: Optional[int] = None*, *limit: Optional[int] = None*) → Iterable[*textacy.types.Record*]

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

Parameters

- **speaker_name** – Filter speeches by the speakers' name; see *CapitolWords.speaker_names*.
- **speaker_party** – Filter speeches by the speakers' party; see *CapitolWords.speaker_parties*.
- **chamber** – Filter speeches by the chamber in which they were given; see *CapitolWords.chambers*.
- **congress** – Filter speeches by the congress in which they were given; see *CapitolWords.congresses*.
- **date_range** – Filter speeches 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 for the dataset.
- **min_len** – Filter speeches by the length (# characters) of their text content.
- **limit** – Yield no more than `limit` speeches that match all specified filters.

Yields Full text of the next (by chronological order) speech in dataset passing all filters, and its corresponding metadata.

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 data:

- `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 an ISO-formatted string ("YYYY-MM-DD").
- `decision_date`: Date on which the Court's decision was announced, as an ISO-formatted string ("YYYY-MM-DD").
- `decision_direction`: Ideological direction of the majority's decision: one of "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 dataset's creation was inspired by a blog post by Emily Barry: http://www.emilyinamillion.me/blog/2016/7/13/visualizing-supreme-court-topics-over-time.

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: Union[str, pathlib.Path] = PosixPath('/home/docs/checkouts/readthedocs.org/user_build packages/textacy/data/supreme_court')*)

Stream a collection of US Supreme Court decisions from a compressed json file on disk, either as texts or text + metadata pairs.

Download the data (one time only!) from the textacy-data repo (https://github.com/bdewilde/textacy-data), and save its contents to disk:

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

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

```
>>> for text in ds.texts(limit=3):
...     print(text[:500], end="\n\n")
>>> for text, meta in ds.records(limit=3):
...     print("\n{} ({})\n{}".format(meta["case_name"], meta["decision_date"],␣
↪text[:500]))
```

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

```
>>> for text, meta in ds.records(opinion_author=109, limit=3):  # Notorious RBG!
...     print(meta["case_name"], meta["decision_direction"], meta["n_maj_votes"])
>>> for text, meta in ds.records(decision_direction="liberal",
...                              issue_area={1, 9, 10}, limit=3):
...     print(meta["case_name"], meta["maj_opinion_author"], meta["n_maj_votes"])
>>> for text, meta in ds.records(opinion_author=102, date_range=('1985-02-11',
↪'1986-02-11')):
...     print("\n{} ({})".format(meta["case_name"], meta["decision_date"]))
...     print(ds.issue_codes[meta["issue"]], "=>", meta["decision_direction"])
>>> for text in ds.texts(min_len=250000):
...     print(len(text))
```

Stream decisions into a *textacy.Corpus*:

```
>>> textacy.Corpus("en", data=ds.records(limit=25))
Corpus(25 docs; 136696 tokens)
```

Parameters **data_dir** (str or pathlib.Path) – Path to directory on disk under which the data is stored, i.e. /path/to/data_dir/supreme_court.

full_date_range
First and last dates for which decisions are available, each as an ISO-formatted string (YYYY-MM-DD).

Type Tuple[str, str]

decision_directions
All distinct decision directions, e.g. "liberal".

Type Set[str]

opinion_author_codes
Mapping of majority opinion authors, from id code to full name.

Type Dict[int, Optional[str]]

issue_area_codes
Mapping of high-level issue area of the case's core disagreement, from id code to description.

Type Dict[int, Optional[str]]

issue_codes
Mapping of the specific issue of the case's core disagreement, from id code to description.

Type Dict[str, str]

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

download(*, *force: bool = False*) → None
Download the data as a Python version-specific compressed json file and save it to disk under the `data_dir` directory.

Parameters **force** – If True, download the dataset, even if it already exists on disk under `data_dir`.

texts(*, *opinion_author: Optional[Union[int, Set[int]]] = None*, *decision_direction: Optional[Union[str, Set[str]]] = None*, *issue_area: Optional[Union[int, Set[int]]] = None*, *date_range: Optional[Tuple[Optional[str], Optional[str]]] = None*, *min_len: Optional[int] = None*, *limit: Optional[int] = None*) → Iterable[str]
Iterate over decisions in this dataset, optionally filtering by a variety of metadata and/or text length, and yield texts only, in chronological order by decision date.

Parameters

- **opinion_author** – Filter decisions by the name(s) of the majority opinion's author, coded as an integer whose mapping is given in *SupremeCourt.opinion_author_codes*.
- **decision_direction** – Filter decisions by the ideological direction of the majority's decision; see *SupremeCourt.decision_directions*.
- **issue_area** – Filter decisions by the issue area of the case's subject matter, coded as an integer whose mapping is given in *SupremeCourt.issue_area_codes*.
- **date_range** – Filter decisions 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 for the dataset.
- **min_len** – Filter decisions by the length (# characters) of their text content.
- **limit** – Yield no more than `limit` decisions that match all specified filters.

Yields Text of the next decision in dataset passing all filters.

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

records(*, *opinion_author: Optional[Union[int, Set[int]]] = None*, *decision_direction: Optional[Union[str, Set[str]]] = None*, *issue_area: Optional[Union[int, Set[int]]] = None*, *date_range: Optional[Tuple[Optional[str], Optional[str]]] = None*, *min_len: Optional[int] = None*, *limit: Optional[int] = None*) → Iterable[*textacy.types.Record*]
Iterate over decisions in this dataset, optionally filtering by a variety of metadata and/or text length, and yield text + metadata pairs, in chronological order by decision date.

Parameters

- **opinion_author** – Filter decisions by the name(s) of the majority opinion's author, coded as an integer whose mapping is given in *SupremeCourt.opinion_author_codes*.
- **decision_direction** – Filter decisions by the ideological direction of the majority's decision; see *SupremeCourt.decision_directions*.

- **issue_area** – Filter decisions by the issue area of the case's subject matter, coded as an integer whose mapping is given in *SupremeCourt.issue_area_codes*.
- **date_range** – Filter decisions 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 for the dataset.
- **min_len** – Filter decisions by the length (# characters) of their text content.
- **limit** – Yield no more than `limit` decisions that match all specified filters.

Yields Text of the next decision in dataset passing all filters, and its corresponding metadata.

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

Wikimedia articles

All articles for a given Wikimedia project, specified by language and version.

Records include the following key fields (plus a few others):

- `text`: Plain text content of the wiki page – no wiki markup!
- `title`: Title of the wiki page.
- `wiki_links`: A list of other wiki pages linked to from this page.
- `ext_links`: A list of external URLs linked to from this page.
- `categories`: A list of categories to which this wiki page belongs.
- `dt_created`: Date on which the wiki page was first created.
- `page_id`: Unique identifier of the wiki page, usable in Wikimedia APIs.

Datasets are generated by the Wikimedia Foundation for a variety of projects, such as Wikipedia and Wikinews. The source files are meant for search indexes, so they're dumped in Elasticsearch bulk insert format – basically, a compressed JSON file with one record per line. For more information, refer to https://meta.wikimedia.org/wiki/Data_dumps.

class textacy.datasets.wikimedia.**Wikimedia**(*name*, *meta*, *project*, *data_dir*, *lang='en'*, *version='current'*, *namespace=0*)

Base class for project-specific Wikimedia datasets. See:

- *Wikipedia*
- *Wikinews*

property filepath

Full path on disk for the Wikimedia CirrusSearch db dump corresponding to the `project`, `lang`, and `version`.

Type str

download(*, *force: bool = False*) → None

Download the Wikimedia CirrusSearch db dump corresponding to the given `project`, `lang`, and `version` as a compressed JSON file, and save it to disk under the `data_dir` directory.

Parameters **force** – If True, download the dataset, even if it already exists on disk under `data_dir`.

Note: Some datasets are quite large (e.g. English Wikipedia is ~28GB) and can take hours to fully download.

texts(*, *category: Optional[Union[str, Set[str]]] = None*, *wiki_link: Optional[Union[str, Set[str]]] = None*, *min_len: Optional[int] = None*, *limit: Optional[int] = None*) → Iterable[str]

Iterate over wiki pages in this dataset, optionally filtering by a variety of metadata and/or text length, and yield texts only, in order of appearance in the db dump file.

Parameters

- **category** – Filter wiki pages by the categories to which they've been assigned. For multiple values (Set[str]), ANY rather than ALL of the values must be found among a given page's categories.
- **wiki_link** – Filter wiki pages by the other wiki pages to which they've been linked. For multiple values (Set[str]), ANY rather than ALL of the values must be found among a given page's wiki links.
- **min_len** – Filter wiki pages by the length (# characters) of their text content.
- **limit** – Yield no more than `limit` wiki pages that match all specified filters.

Yields Text of the next wiki page in dataset passing all filters.

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

records(*, *category: Optional[Union[str, Set[str]]] = None*, *wiki_link: Optional[Union[str, Set[str]]] = None*, *min_len: Optional[int] = None*, *limit: Optional[int] = None*) → Iterable[*textacy.types.Record*]

Iterate over wiki pages in this dataset, optionally filtering by a variety of metadata and/or text length, and yield text + metadata pairs, in order of appearance in the db dump file.

Parameters

- **category** – Filter wiki pages by the categories to which they've been assigned. For multiple values (Set[str]), ANY rather than ALL of the values must be found among a given page's categories.
- **wiki_link** – Filter wiki pages by the other wiki pages to which they've been linked. For multiple values (Set[str]), ANY rather than ALL of the values must be found among a given page's wiki links.
- **min_len** – Filter wiki pages by the length (# characters) of their text content.
- **limit** – Yield no more than `limit` wiki pages that match all specified filters.

Yields Text of the next wiki page in dataset passing all filters, and its corresponding metadata.

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

class textacy.datasets.wikimedia.**Wikipedia**(*data_dir: Union[str, pathlib.Path] = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/textacy/...packages/textacy/data/wikipedia')*, *lang: str = 'en'*, *version: str = 'current'*, *namespace: int = 0*)

Stream a collection of Wikipedia pages from a version- and language-specific database dump, either as texts or text + metadata pairs.

Download a database dump (one time only!) and save its contents to disk:

```
>>> import textacy.datasets
>>> ds = textacy.datasets.Wikipedia(lang="en", version="current")
>>> ds.download()
>>> ds.info
{'name': 'wikipedia',
 'site_url': 'https://en.wikipedia.org/wiki/Main_Page',
 'description': 'All pages for a given language- and version-specific Wikipedia␣
↪site snapshot.'}
```

Iterate over wiki pages as texts or records with both text and metadata:

```
>>> for text in ds.texts(limit=5):
...     print(text[:500])
>>> for text, meta in ds.records(limit=5):
...     print(meta["page_id"], meta["title"])
```

Filter wiki pages by a variety of metadata fields and text length:

```
>>> for text, meta in ds.records(category="Living people", limit=5):
...     print(meta["title"], meta["categories"])
>>> for text, meta in ds.records(wiki_link="United_States", limit=5):
...     print(meta["title"], meta["wiki_links"])
>>> for text in ds.texts(min_len=10000, limit=5):
...     print(len(text))
```

Stream wiki pages into a *textacy.Corpus*:

```
>>> textacy.Corpus("en", data=ds.records(min_len=2000, limit=50))
Corpus(50 docs; 72368 tokens)
```

Parameters

- **data_dir** – Path to directory on disk under which database dump files are stored. Each file is expected as `{lang}{project}/{version}/{lang}{project}-{version}-cirrussearch-content.json.gz` immediately under this directory.
- **lang** – Standard two-letter language code, e.g. "en" => "English", "de" => "German". https://en.wikipedia.org/wiki/List_of_ISO_639-1_codes
- **version** – Database dump version to use. Either "current" for the most recently available version or a date formatted as "YYYYMMDD". Dumps are produced weekly; check for available versions at https://dumps.wikimedia.org/other/cirrussearch/.
- **namespace** – Namespace of the wiki pages to include. Typical, public- facing content is in the 0 (default) namespace.

class textacy.datasets.wikimedia.**Wikinews**(*data_dir: Union[str, pathlib.Path] = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/textacy/en packages/textacy/data/wikinews')*, *lang: str = 'en'*, *version: str = 'current'*, *namespace: int = 0*)

Stream a collection of Wikinews pages from a version- and language-specific database dump, either as texts or text + metadata pairs.

Download a database dump (one time only!) and save its contents to disk:

```
>>> import textacy.datasets
>>> ds = textacy.datasets.Wikinews(lang="en", version="current")
>>> ds.download()
>>> ds.info
{'name': 'wikinews',
 'site_url': 'https://en.wikinews.org/wiki/Main_Page',
 'description': 'All pages for a given language- and version-specific Wikinews␣
↪site snapshot.'}
```

Iterate over wiki pages as texts or records with both text and metadata:

```
>>> for text in ds.texts(limit=5):
...     print(text[:500])
>>> for text, meta in ds.records(limit=5):
...     print(meta["page_id"], meta["title"])
```

Filter wiki pages by a variety of metadata fields and text length:

```
>>> for text, meta in ds.records(category="Politics and conflicts", limit=5):
...     print(meta["title"], meta["categories"])
>>> for text, meta in ds.records(wiki_link="Reuters", limit=5):
...     print(meta["title"], meta["wiki_links"])
>>> for text in ds.texts(min_len=5000, limit=5):
...     print(len(text))
```

Stream wiki pages into a *textacy.Corpus*:

```
>>> textacy.Corpus("en", data=ds.records(limit=100))
Corpus(100 docs; 33092 tokens)
```

Parameters

- **data_dir** – Path to directory on disk under which database dump files are stored. Each file is expected as `{lang}{project}/{version}/{lang}{project}-{version}-cirrussearch-content.json.gz` immediately under this directory.
- **lang** – Standard two-letter language code, e.g. "en" => "English", "de" => "German". https://en.wikipedia.org/wiki/List_of_ISO_639-1_codes
- **version** – Database dump version to use. Either "current" for the most recently available version or a date formatted as "YYYYMMDD". Dumps are produced weekly; check for available versions at https://dumps.wikimedia.org/other/cirrussearch/.
- **namespace** – Namespace of the wiki pages to include. Typical, public- facing content is in the 0 (default) namespace.

Reddit comments

A collection of up to ~1.5 billion Reddit comments posted from October 2007 through May 2015.

Records include the following key fields (plus a few others):

- `body`: Full text of the comment.
- `created_utc`: Date on which the comment was posted.
- `subreddit`: Sub-reddit in which the comment was posted, excluding the familiar "/r/" prefix.
- `score`: Net score (upvotes - downvotes) on the comment.
- `gilded`: Number of times this comment received reddit gold.

The raw data was originally collected by /u/Stuck_In_the_Matrix via Reddit's APIS, and stored for posterity by the Internet Archive. For more details, refer to https://archive.org/details/2015_reddit_comments_corpus.

class textacy.datasets.reddit_comments.**RedditComments**(*data_dir: Union[str, pathlib.Path] = PosixPath('/home/docs/checkouts/readthedocs.org/user packages/textacy/data/reddit_comments')*)

Stream a collection of Reddit comments from 1 or more compressed files on disk, either as texts or text + metadata pairs.

Download the data (one time only!) or subsets thereof by specifying a date range:

```
>>> import textacy.datasets
>>> ds = textacy.datasets.RedditComments()
>>> ds.download(date_range=("2007-10", "2008-03"))
>>> ds.info
{'name': 'reddit_comments',
 'site_url': 'https://archive.org/details/2015_reddit_comments_corpus',
 'description': 'Collection of ~1.5 billion publicly available Reddit comments␣
↪from October 2007 through May 2015.'}
```

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

```
>>> for text in ds.texts(limit=5):
...     print(text)
>>> for text, meta in ds.records(limit=5):
...     print("\n{} {}\n{}".format(meta["author"], meta["created_utc"], text))
```

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

```
>>> for text, meta in ds.records(subreddit="politics", limit=5):
...     print(meta["score"], ":", text)
>>> for text, meta in ds.records(date_range=("2008-01", "2008-03"), limit=5):
...     print(meta["created_utc"])
>>> for text, meta in ds.records(score_range=(10, None), limit=5):
...     print(meta["score"], ":", text)
>>> for text in ds.texts(min_len=2000, limit=5):
...     print(len(text))
```

Stream comments into a *textacy.Corpus*:

```
>>> textacy.Corpus("en", data=ds.records(limit=1000))
Corpus(1000 docs; 27582 tokens)
```

Parameters **data_dir** – Path to directory on disk under which the data is stored, i.e. `/path/to/data_dir/reddit_comments`. Each file covers a given month, as indicated in the filename like "YYYY/RC_YYYY-MM.bz2".

full_date_range
First and last dates for which comments are available, each as an ISO-formatted string (YYYY-MM-DD).

Type Tuple[str, str]

property filepaths
Full paths on disk for all Reddit comments files found under `RedditComments.data_dir` directory, sorted in chronological order.

download(*, *date_range: Tuple[Optional[str], Optional[str]] = (None, None)*, *force: bool = False*) → None
Download 1 or more monthly Reddit comments files from archive.org and save them to disk under the `data_dir` directory.

Parameters

- **date_range** – Interval specifying the [start, end) dates for which comments files will be downloaded. Each item must be a str formatted as YYYY-MM or YYYY-MM-DD (the latter is converted to the corresponding YYYY-MM value). Both start and end values must be specified, but a null value for either is automatically replaced by the minimum or maximum valid values, respectively.
- **force** – If True, download the dataset, even if it already exists on disk under `data_dir`.

texts(*, *subreddit: Optional[Union[str, Set[str]]] = None*, *date_range: Optional[Tuple[Optional[str], Optional[str]]] = None*, *score_range: Optional[Tuple[Optional[int], Optional[int]]] = None*, *min_len: Optional[int] = None*, *limit: Optional[int] = None*) → Iterable[str]
Iterate over comments (text-only) in 1 or more files of this dataset, optionally filtering by a variety of metadata and/or text length, in chronological order.

Parameters

- **subreddit** – Filter comments for those which were posted in the specified subreddit(s).
- **date_range** – Filter comments for those which were posted within the interval [start, end). Each item must be a str in ISO-standard format, i.e. some amount of YYYY-MM-DDTHH:mm:ss. Both start and end values must be specified, but a null value for either is automatically replaced by the minimum or maximum valid values, respectively.
- **score_range** – Filter comments for those whose score (# upvotes minus # downvotes) is within the interval [low, high). Both start and end values must be specified, but a null value for either is automatically replaced by the minimum or maximum valid values, respectively.
- **min_len** – Filter comments for those whose body length in chars is at least this long.
- **limit** – Maximum number of comments passing all filters to yield. If None, all comments are iterated over.

Yields Text of the next comment in dataset passing all filters.

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

records(*, *subreddit: Optional[Union[str, Set[str]]] = None*, *date_range: Optional[Tuple[Optional[str], Optional[str]]] = None*, *score_range: Optional[Tuple[Optional[int], Optional[int]]] = None*, *min_len: Optional[int] = None*, *limit: Optional[int] = None*) → Iterable[*textacy.types.Record*]
Iterate over comments (including text and metadata) in 1 or more files of this dataset, optionally filtering by a variety of metadata and/or text length, in chronological order.

Parameters

- **subreddit** – Filter comments for those which were posted in the specified subreddit(s).
- **date_range** – Filter comments for those which were posted within the interval [start, end). Each item must be a str in ISO-standard format, i.e. some amount of YYYY-MM-DDTHH:mm:ss. Both start and end values must be specified, but a null value for either is automatically replaced by the minimum or maximum valid values, respectively.
- **score_range** – Filter comments for those whose score (# upvotes minus # downvotes) is within the interval [low, high). Both start and end values must be specified, but a null value for either is automatically replaced by the minimum or maximum valid values, respectively.
- **min_len** – Filter comments for those whose body length in chars is at least this long.
- **limit** – Maximum number of comments passing all filters to yield. If None, all comments are iterated over.

Yields Text of the next comment in dataset passing all filters, and its corresponding metadata.

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

Oxford Text Archive literary works

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

Records include the following data:

- `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.
- `id`: Unique identifier of the literary work within 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: Union[str, pathlib.Path] = PosixPath('/home/docs/checkouts/readthedo packages/textacy/data/oxford_text_archive')*)

Stream a collection of English-language literary works from text files on disk, either as texts or text + metadata pairs.

Download the data (one time only!), saving and extracting its contents to disk:

```
>>> import textacy.datasets
>>> ds = textacy.datasets.OxfordTextArchive()
>>> ds.download()
>>> ds.info
{'name': 'oxford_text_archive',
```

(continues on next page)

(continued from previous page)

```
 'site_url': 'https://ota.ox.ac.uk/',
 'description': 'Collection of ~2.7k Creative Commons texts from the Oxford Text␣
↪Archive, containing primarily English-language 16th-20th century literature and␣
↪history.'}
```

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

```
>>> for text in ds.texts(limit=3):
...     print(text[:200])
>>> for text, meta in ds.records(limit=3):
...     print("\n{}, {}".format(meta["title"], meta["year"]))
...     print(text[:300])
```

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

```
>>> for text, meta in ds.records(author="Shakespeare, William", limit=1):
...     print("{}\n{}".format(meta["title"], text[:500]))
>>> for text, meta in ds.records(date_range=("1900-01-01", "1990-01-01"),␣
↪limit=5):
...     print(meta["year"], meta["author"])
>>> for text in ds.texts(min_len=4000000):
...     print(len(text))
```

Stream literary works into a *textacy.Corpus*:

```
>>> textacy.Corpus("en", data=ds.records(limit=5))
Corpus(5 docs; 182289 tokens)
```

Parameters **data_dir** (str or pathlib.Path) – Path to directory on disk under which dataset is stored, i.e. `/path/to/data_dir/oxford_text_archive`.

full_date_range
First and last dates for which works are available, each as an ISO-formatted string (YYYY-MM-DD).

Type Tuple[str, str]

authors
Full names of all distinct authors included in this dataset, e.g. "Shakespeare, William".

Type Set[str]

download(*, *force: bool = False*) → None
Download the data as a zip archive file, then save it to disk and extract its contents under the `OxfordTextArchive.data_dir` directory.

Parameters **force** – If True, download the dataset, even if it already exists on disk under `data_dir`.

property metadata
Dict[str, dict]

texts(*, *author: Optional[Union[str, Set[str]]] = None*, *date_range: Optional[Tuple[Optional[str], Optional[str]]] = None*, *min_len: Optional[int] = None*, *limit: Optional[int] = None*) → Iterable[str]
Iterate over works in this dataset, optionally filtering by a variety of metadata and/or text length, and yield texts only.

Parameters

- **author** – Filter texts by the authors' name. For multiple values (Set[str]), ANY rather than ALL of the authors must be found among a given works's authors.
- **date_range** – 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** – Filter texts by the length (# characters) of their text content.
- **limit** – Yield no more than `limit` texts that match all specified filters.

Yields Text of the next work in dataset passing all filters.

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

records(*, *author: Optional[Union[str, Set[str]]] = None*, *date_range: Optional[Tuple[Optional[str], Optional[str]]] = None*, *min_len: Optional[int] = None*, *limit: Optional[int] = None*) → Iterable[*textacy.types.Record*]

Iterate over works in this dataset, optionally filtering by a variety of metadata and/or text length, and yield text + metadata pairs.

Parameters

- **author** – Filter texts by the authors' name. For multiple values (Set[str]), ANY rather than ALL of the authors must be found among a given works's authors.
- **date_range** – 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** – Filter texts by the length (# characters) of their text content.
- **limit** – Yield no more than `limit` texts that match all specified filters.

Yields Text of the next work in dataset passing all filters, and its corresponding metadata.

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

IMDB movie reviews

A collection of 50k highly polar movie reviews posted to IMDB, split evenly into training and testing sets, with 25k positive and 25k negative sentiment labels, as well as some unlabeled reviews.

Records include the following key fields (plus a few others):

- `text`: Full text of the review.
- `subset`: Subset of the dataset ("train" or "test") into which the review has been split.
- `label`: Sentiment label ("pos" or "neg") assigned to the review.
- `rating`: Numeric rating assigned by the original reviewer, ranging from 1 to 10. Reviews with a rating <= 5 are "neg"; the rest are "pos".
- `movie_id`: Unique identifier for the movie under review within IMDB, useful for grouping reviews or joining with an external movie dataset.

Reference: Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. (2011). Learning Word Vectors for Sentiment Analysis. The 49th Annual Meeting of the Association for Computational Linguistics (ACL 2011).

class textacy.datasets.imdb.**IMDB**(*data_dir=PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/textacy/envs/packages/textacy/data/imdb')*)

Stream a collection of IMDB movie reviews from text files on disk, either as texts or text + metadata pairs.

Download the data (one time only!), saving and extracting its contents to disk:

```
>>> import textacy.datasets
>>> ds = textacy.datasets.IMDB()
>>> ds.download()
>>> ds.info
{'name': 'imdb',
 'site_url': 'http://ai.stanford.edu/~amaas/data/sentiment',
 'description': 'Collection of 50k highly polar movie reviews split evenly into␣
↪train and test sets, with 25k positive and 25k negative labels. Also includes␣
↪some unlabeled reviews.'}
```

Iterate over movie reviews as texts or records with both text and metadata:

```
>>> for text in ds.texts(limit=5):
...     print(text)
>>> for text, meta in ds.records(limit=5):
...     print("\n{} {}\n{}".format(meta["label"], meta["rating"], text))
```

Filter movie reviews by a variety of metadata fields and text length:

```
>>> for text, meta in ds.records(label="pos", limit=5):
...     print(meta["rating"], ":", text)
>>> for text, meta in ds.records(rating_range=(9, 11), limit=5):
...     print(meta["rating"], text)
>>> for text in ds.texts(min_len=1000, limit=5):
...     print(len(text))
```

Stream movie reviews into a *textacy.Corpus*:

```
>>> textacy.Corpus("en", data=ds.records(limit=100))
Corpus(100 docs; 24340 tokens)
```

Parameters **data_dir** – Path to directory on disk under which the data is stored, i.e. `/path/to/data_dir/imdb`.

full_rating_range

Lowest and highest ratings for which movie reviews are available.

Type Tuple[int, int]

download(*, *force: bool = False*) → None

Download the data as a compressed tar archive file, then save it to disk and extract its contents under the `data_dir` directory.

Parameters **force** – If True, always download the dataset even if it already exists on disk under `data_dir`.

texts(*, *subset: Optional[str] = None*, *label: Optional[str] = None*, *rating_range: Optional[Tuple[Optional[int], Optional[int]]] = None*, *min_len: Optional[int] = None*, *limit: Optional[int] = None*) → Iterable[str]

Iterate over movie reviews in this dataset, optionally filtering by a variety of metadata and/or text length, and yield texts only.

Parameters

- **subset** (*{"train", "test"}*) – Filter movie reviews by the dataset subset into which they've already been split.

- **label** ({"pos", "neg", "unsup"}) – Filter movie reviews by the assigned sentiment label (or lack thereof, for "unsup").
- **rating_range** – Filter movie reviews by the rating assigned by the reviewer. Only those with ratings in the interval [low, high) are included. Both low and high values must be specified, but a null value for either is automatically replaced by the minimum or maximum valid values, respectively.
- **min_len** – Filter reviews by the length (# characters) of their text content.
- **limit** – Yield no more than `limit` reviews that match all specified filters.

Yields Text of the next movie review in dataset passing all filters.

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

records(*, *subset: Optional[str] = None*, *label: Optional[str] = None*, *rating_range: Optional[Tuple[Optional[int], Optional[int]]] = None*, *min_len: Optional[int] = None*, *limit: Optional[int] = None*) → Iterable[*textacy.types.Record*]

Iterate over movie reviews in this dataset, optionally filtering by a variety of metadata and/or text length, and yield text + metadata pairs.

Parameters

- **subset** ({"train", "test"}) – Filter movie reviews by the dataset subset into which they've already been split.
- **label** ({"pos", "neg", "unsup"}) – Filter movie reviews by the assigned sentiment label (or lack thereof, for "unsup").
- **rating_range** – Filter movie reviews by the rating assigned by the reviewer. Only those with ratings in the interval [low, high) are included. Both low and high values must be specified, but a null value for either is automatically replaced by the minimum or maximum valid values, respectively.
- **min_len** – Filter reviews by the length (# characters) of their text content.
- **limit** – Yield no more than `limit` reviews that match all specified filters.

Yields Text of the next movie review in dataset passing all filters, and its corresponding metadata.

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

UDHR translations

A collection of translations of the Universal Declaration of Human Rights (UDHR), a milestone document in the history of human rights that first, formally established fundamental human rights to be universally protected.

Records include the following fields:

- `text`: Full text of the translated UDHR document.
- `lang`: ISO-639-1 language code of the text.
- `lang_name`: Ethnologue entry for the language (see https://www.ethnologue.com).

The source dataset was compiled and is updated by the Unicode Consortium as a way to demonstrate the use of unicode in representing a wide variety of languages. In fact, the UDHR was chosen because it's been translated into more languages than any other document! However, this dataset only provides access to records translated into ISO-639-1 languages — that is, major living languages *only*, rather than every language, major or minor, that has ever existed. If you need access to texts in those other languages, you can find them at `UDHR._texts_dirpath`.

For more details, go to https://unicode.org/udhr.

class textacy.datasets.udhr.**UDHR**(*data_dir: Union[str, pathlib.Path] = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/textacy/envs/latest/lib/python3.8/site-packages/textacy/data/udhr')*)

Stream a collection of UDHR translations from disk, either as texts or text + metadata pairs.

Download the data (one time only!), saving and extracting its contents to disk:

```
>>> import textacy.datasets
>>> ds = textacy.datasets.UDHR()
>>> ds.download()
>>> ds.info
{'name': 'udhr',
 'site_url': 'http://www.ohchr.org/EN/UDHR',
 'description': 'A collection of translations of the Universal Declaration of␣
↪Human Rights (UDHR), a milestone document in the history of human rights that␣
↪first, formally established fundamental human rights to be universally␣
↪protected.'}
```

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

```
>>> for text in ds.texts(limit=5):
...     print(text[:500])
>>> for text, meta in ds.records(limit=5):
...     print("\n{} ({})\n{}".format(meta["lang_name"], meta["lang"], text[:500]))
```

Filter translations by language, and note that some languages have multiple translations:

```
>>> for text, meta in ds.records(lang="en"):
...     print("\n{} ({})\n{}".format(meta["lang_name"], meta["lang"], text[:500]))
>>> for text, meta in ds.records(lang="zh"):
...     print("\n{} ({})\n{}".format(meta["lang_name"], meta["lang"], text[:500]))
```

Note: Streaming translations into a *textacy.Corpus* doesn't work as for other available datasets, since this dataset is multilingual.

Parameters **data_dir** (str or pathlib.Path) – Path to directory on disk under which the data is stored, i.e. /path/to/data_dir/udhr.

langs

All distinct language codes with texts in this dataset, e.g. "en" for English.

Type Set[str]

download(*, *force: bool = False*) → None

Download the data as a zipped archive of language-specific text files, then save it to disk and extract its contents under the data_dir directory.

Parameters **force** – If True, download the dataset, even if it already exists on disk under data_dir.

texts(*, *lang: Optional[Union[str, Set[str]]] = None*, *limit: Optional[int] = None*) → Iterable[str]

Iterate over records in this dataset, optionally filtering by language, and yield texts only.

Parameters

- **lang** – Filter records by the language in which they're written; see *UDHR.langs*.
- **limit** – Yield no more than limit texts that match specified filter.

Yields Text of the next record in dataset passing filters.

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

records(*, *lang: Optional[Union[str, Set[str]]] = None*, *limit: Optional[int] = None*) → Iterable[*textacy.types.Record*]

Iterate over reocrds in this dataset, optionally filtering by a language, and yield text + metadata pairs.

Parameters

- **lang** – Filter records by the language in which they're written; see *UDHR.langs*.
- **limit** – Yield no more than `limit` texts that match specified filter.

Yields Text of the next record in dataset passing filters, and its corresponding metadata.

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

ConceptNet

ConceptNet is a multilingual knowledge base, representing common words and phrases and the common-sense relationships between them. This information is collected from a variety of sources, including crowd-sourced resources (e.g. Wiktionary, Open Mind Common Sense), games with a purpose (e.g. Verbosity, nadya.jp), and expert-created resources (e.g. WordNet, JMDict).

The interface in textacy gives access to several key relationships between terms that are useful in a variety of NLP tasks:

- antonyms: terms that are opposites of each other in some relevant way
- hyponyms: terms that are subtypes or specific instances of other terms
- meronyms: terms that are parts of other terms
- synonyms: terms that are sufficiently similar that they may be used interchangeably

class textacy.resources.concept_net.**ConceptNet**(*data_dir=PosixPath('/home/docs/checkouts/readthedocs.org/user_packages/textacy/data/concept_net')*, *version='5.7.0'*)

Interface to ConceptNet, a multilingual knowledge base representing common words and phrases and the common-sense relationships between them.

Download the data (one time only!), and save its contents to disk:

```
>>> import textacy.resources
>>> rs = textacy.resources.ConceptNet()
>>> rs.download()
>>> rs.info
{'name': 'concept_net',
 'site_url': 'http://conceptnet.io',
 'publication_url': 'https://arxiv.org/abs/1612.03975',
 'description': 'An open, multilingual semantic network of general knowledge,␣
↪designed to help computers understand the meanings of words.'}
```

Access other same-language terms related to a given term in a variety of ways:

```
>>> rs.get_synonyms("spouse", lang="en", sense="n")
['mate', 'married person', 'better half', 'partner']
>>> rs.get_antonyms("love", lang="en", sense="v")
['detest', 'hate', 'loathe']
>>> rs.get_hyponyms("marriage", lang="en", sense="n")
['cohabitation situation', 'union', 'legal agreement', 'ritual', 'family',
↪'marital status']
```

Note: The very first time a given relationship is accessed, the full ConceptNet db must be parsed and split for fast future access. This can take a couple minutes; be patient.

When passing a spaCy `Token` or `Span`, the corresponding `lang` and `sense` are inferred automatically from the object:

```
>>> text = "The quick brown fox jumps over the lazy dog."
>>> doc = textacy.make_spacy_doc(text, lang="en")
>>> rs.get_synonyms(doc[1])  # quick
['flying', 'fast', 'rapid', 'ready', 'straightaway', 'nimble', 'speedy', 'warm']
>>> rs.get_synonyms(doc[4:5])  # jumps over
['leap', 'startle', 'hump', 'flinch', 'jump off', 'skydive', 'jumpstart', ...]
```

Many terms won't have entries, for actual linguistic reasons or because the db's coverage of a given language's vocabulary isn't comprehensive:

```
>>> rs.get_meronyms(doc[3])  # fox
[]
>>> rs.get_antonyms(doc[7])  # lazy
[]
```

Parameters

- **data_dir** (str or `pathlib.Path`) – Path to directory on disk under which resource data is stored, i.e. `/path/to/data_dir/concept_net`.
- **version** (*{"5.7.0", "5.6.0", "5.5.5"}*) – Version string of the ConceptNet db to use. Since newer versions typically represent improvements over earlier versions, you'll probably want "5.7.0" (the default value).

download(*, *force=False*)

Download resource data as a gzipped csv file, then save it to disk under the `ConceptNet.data_dir` directory.

Parameters **force** (*bool*) – If True, download resource data, even if it already exists on disk; otherwise, don't re-download the data.

property filepath

Full path on disk for the ConceptNet gzipped csv file corresponding to the given `ConceptNet.data_dir`.

Type str

property antonyms

Mapping of language code to term to sense to set of term's antonyms – opposites of the term in some relevant way, like being at opposite ends of a scale or fundamentally similar but with a key difference between them – such as black <=> white or hot <=> cold. Note that this relationship is symmetric.

Based on the "/r/Antonym" relation in ConceptNet.

Type Dict[str, Dict[str, Dict[str, List[str]]]]

get_antonyms(*term*, *, *lang=None*, *sense=None*)

Parameters

- **term** (str or `spacy.tokens.Token` or `spacy.tokens.Span`) –
- **lang** (*str*) – Standard code for the language of `term`.
- **sense** (*str*) – Sense in which `term` is used in context, which in practice is just its part of speech. Valid values: "n" or "NOUN", "v" or "VERB", "a" or "ADJ", "r" or "ADV".

Returns List[str]

property hyponyms

Mapping of language code to term to sense to set of term's hyponyms – subtypes or specific instances of the term – such as car => vehicle or Chicago => city. Every A is a B.

Based on the "/r/IsA" relation in ConceptNet.

Type Dict[str, Dict[str, Dict[str, List[str]]]]

get_hyponyms(*term*, *, *lang=None*, *sense=None*)

Parameters

- **term** (str or `spacy.tokens.Token` or `spacy.tokens.Span`) –
- **lang** (*str*) – Standard code for the language of `term`.
- **sense** (*str*) – Sense in which `term` is used in context, which in practice is just its part of speech. Valid values: "n" or "NOUN", "v" or "VERB", "a" or "ADJ", "r" or "ADV".

Returns List[str]

property meronyms

Mapping of language code to term to sense to set of term's meronyms – parts of the term – such as gearshift => car.

Based on the "/r/PartOf" relation in ConceptNet.

Type Dict[str, Dict[str, Dict[str, List[str]]]]

get_meronyms(*term*, *, *lang=None*, *sense=None*)

Parameters

- **term** (str or `spacy.tokens.Token` or `spacy.tokens.Span`) –
- **lang** (*str*) – Standard code for the language of `term`.
- **sense** (*str*) – Sense in which `term` is used in context, which in practice is just its part of speech. Valid values: "n" or "NOUN", "v" or "VERB", "a" or "ADJ", "r" or "ADV".

Returns List[str]

property synonyms

Mapping of language code to term to sense to set of term's synonyms – sufficiently similar concepts that they may be used interchangeably – such as sunlight <=> sunshine. Note that this relationship is symmetric.

Based on the "/r/Synonym" relation in ConceptNet.

Type Dict[str, Dict[str, Dict[str, List[str]]]]

get_synonyms(*term*, *, *lang=None*, *sense=None*)

Parameters

- **term** (str or `spacy.tokens.Token` or `spacy.tokens.Span`) –
- **lang** (*str*) – Standard code for the language of `term`.
- **sense** (*str*) – Sense in which `term` is used in context, which in practice is just its part of speech. Valid values: "n" or "NOUN", "v" or "VERB", "a" or "ADJ", "r" or "ADV".

Returns List[str]

DepecheMood

DepecheMood is a high-quality and high-coverage emotion lexicon for English and Italian text, mapping individual terms to their emotional valences. These word-emotion weights are inferred from crowd-sourced datasets of emotionally tagged news articles (rappler.com for English, corriere.it for Italian).

English terms are assigned weights to eight emotions:

- AFRAID
- AMUSED
- ANGRY
- ANNOYED
- DONT_CARE
- HAPPY
- INSPIRED
- SAD

Italian terms are assigned weights to five emotions:

- DIVERTITO (~amused)
- INDIGNATO (~annoyed)
- PREOCCUPATO (~afraid)
- SODDISFATTO (~happy)
- TRISTE (~sad)

class textacy.resources.depeche_mood.**DepecheMood**(*data_dir=PosixPath('/home/docs/checkouts/readthedocs.org/u packages/textacy/data/depeche_mood')*, *lang='en'*, *word_rep='lemmapos'*, *min_freq=3*)

Interface to DepecheMood, an emotion lexicon for English and Italian text.

Download the data (one time only!), and save its contents to disk:

```
>>> import textacy.resources
>>> rs = textacy.resources.DepecheMood(lang="en", word_rep="lemmapos")
>>> rs.download()
>>> rs.info
{'name': 'depeche_mood',
 'site_url': 'http://www.depechemood.eu',
 'publication_url': 'https://arxiv.org/abs/1810.03660',
 'description': 'A simple tool to analyze the emotions evoked by a text.'}
```

Access emotional valences for individual terms:

```
>>> rs.get_emotional_valence("disease#n")
{'AFRAID': 0.37093526222120465,
 'AMUSED': 0.06953745082761113,
 'ANGRY': 0.06979683067736414,
 'ANNOYED': 0.06465401081252636,
 'DONT_CARE': 0.07080580707440012,
 'HAPPY': 0.07537324330608403,
 'INSPIRED': 0.13394731320662606,
 'SAD': 0.14495008187418348}
```

(continues on next page)

(continued from previous page)

```
>>> rs.get_emotional_valence("heal#v")
{'AFRAID': 0.060450319886187334,
 'AMUSED': 0.09284046387491741,
 'ANGRY': 0.06207816933776029,
 'ANNOYED': 0.10027622719958346,
 'DONT_CARE': 0.11259594401785,
 'HAPPY': 0.09946106491457314,
 'INSPIRED': 0.37794768332634626,
 'SAD': 0.09435012744278205}
```

When passing multiple terms in the form of a List[str] or `Span` or `Doc`, emotion weights are averaged over all terms for which weights are available:

```
>>> rs.get_emotional_valence(["disease#n", "heal#v"])
{'AFRAID': 0.215692791053696,
 'AMUSED': 0.08118895735126427,
 'ANGRY': 0.06593750000756221,
 'ANNOYED': 0.08246511900605491,
 'DONT_CARE': 0.09170087554612506,
 'HAPPY': 0.08741715411032858,
 'INSPIRED': 0.25594749826648616,
 'SAD': 0.11965010465848278}
>>> text = "The acting was sweet and amazing, but the plot was dumb and terrible."
>>> doc = textacy.make_spacy_doc(text, lang="en")
>>> rs.get_emotional_valence(doc)
{'AFRAID': 0.05272350876803627,
 'AMUSED': 0.13725054992595098,
 'ANGRY': 0.15787016147081184,
 'ANNOYED': 0.1398733360688608,
 'DONT_CARE': 0.14356943460620503,
 'HAPPY': 0.11923217912716871,
 'INSPIRED': 0.17880214720077342,
 'SAD': 0.07067868283219296}
>>> rs.get_emotional_valence(doc[0:6])  # the acting was sweet and amazing
{'AFRAID': 0.039790959333750785,
 'AMUSED': 0.1346884072825313,
 'ANGRY': 0.1373596223131593,
 'ANNOYED': 0.11391999698695347,
 'DONT_CARE': 0.1574819173485831,
 'HAPPY': 0.1552521762333925,
 'INSPIRED': 0.21232264216449326,
 'SAD': 0.049184278337136296}
```

For good measure, here's how Italian w/o POS-tagged words looks:

```
>>> rs = textacy.resources.DepecheMood(lang="it", word_rep="lemma")
>>> rs.get_emotional_valence("amore")
{'INDIGNATO': 0.11451408951814121,
 'PREOCCUPATO': 0.1323655108545536,
 'TRISTE': 0.18249663560400609,
 'DIVERTITO': 0.33558928569110086,
 'SODDISFATTO': 0.23503447833219815}
```

Parameters

- **data_dir** (str or `pathlib.Path`) – Path to directory on disk under which resource data is stored, i.e. `/path/to/data_dir/depeche_mood`.

- **lang** (*{"en", "it"}*) – Standard two-letter code for the language of terms for which emotional valences are to be retrieved.
- **word_rep** (*{"token", "lemma", "lemmapos"}*) – Level of text processing used in computing terms' emotion weights. "token" => tokenization only; "lemma" => tokenization and lemmatization; "lemmapos" => tokenization, lemmatization, and part-of-speech tagging.
- **min_freq** (*int*) – Minimum number of times that a given term must have appeared in the source dataset for it to be included in the emotion weights dict. This can be used to remove noisy terms at the expense of reducing coverage. Researchers observed peak performance at 10, but anywhere between 1 and 20 is reasonable.

property filepath
Full path on disk for the DepecheMood tsv file corresponding to the `lang` and `word_rep`.

Type str

property weights
Mapping of term string (or term#POS, if `DepecheMood.word_rep` is "lemmapos") to the terms' normalized weights on a fixed set of affective dimensions (aka "emotions").

Type Dict[str, Dict[str, float]]

download(*, *force=False*)
Download resource data as a zip archive file, then save it to disk and extract its contents under the `data_dir` directory.

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

get_emotional_valence(*terms*)
Get average emotional valence over all terms in `terms` for which emotion weights are available.

Parameters **terms** (str or Sequence[str], `Token` or Sequence[`Token`]) – One or more terms over which to average emotional valences. Note that only nouns, adjectives, adverbs, and verbs are included.

Note: If the resource was initialized with `word_rep="lemmapos"`, then string terms must have matching parts-of-speech appended to them like TERM#POS. Only "n" => noun, "v" => verb, "a" => adjective, and "r" => adverb are included in the data.

Returns Mapping of emotion to average weight.

Return type Dict[str, float]

4.4.3 Text Preprocessing

pipeline.make_pipeline	Make a callable pipeline that takes a text as input, passes it through one or more functions in sequential order, then outputs a single (preprocessed) text string.
normalize.bullet_points	Normalize all "fancy" bullet point symbols in `text` to just the basic ASCII "-", provided they are the first non-whitespace characters on a new line (like a list of items).

continues on next page

Table 4 – continued from previous page

normalize.hyphenated_words	Normalize words in `text` that have been split across lines by a hyphen for visual consistency (aka hyphenated) by joining the pieces back together, sans hyphen and whitespace.
normalize.quotation_marks	Normalize all "fancy" single- and double-quotation marks in `text` to just the basic ASCII equivalents.
normalize.repeating_chars	Normalize repeating characters in `text` by truncating their number of consecutive repetitions to `maxn`.
normalize.unicode	Normalize unicode characters in `text` into canonical forms.
normalize.whitespace	Replace all contiguous zero-width spaces with an empty string, line-breaking spaces with a single newline, and non-breaking spaces with a single space, then strip any leading/trailing whitespace.
remove.accents	Remove accents from any accented unicode characters in `text`, either by replacing them with ASCII equivalents or removing them entirely.
remove.brackets	Remove text within curly {}, square [], and/or round () brackets, as well as the brackets themselves.
remove.html_tags	Remove HTML tags from `text`, returning just the text found between tags and other non-data elements.
remove.punctuation	Remove punctuation from `text` by replacing all instances of punctuation (or a subset thereof specified by `only`) with whitespace.
replace.currency_symbols	Replace all currency symbols in `text` with `repl`.
replace.emails	Replace all email addresses in `text` with `repl`.
replace.emojis	Replace all emoji and pictographs in `text` with `repl`.
replace.hashtags	Replace all hashtags in `text` with `repl`.
replace.numbers	Replace all numbers in `text` with `repl`.
replace.phone_numbers	Replace all phone numbers in `text` with `repl`.
replace.urls	Replace all URLs in `text` with `repl`.
replace.user_handles	Replace all (Twitter-style) user handles in `text` with `repl`.

Pipeline

textacy.preprocessing.pipeline: Basic functionality for composing multiple preprocessing steps into a single callable pipeline.

textacy.preprocessing.pipeline.**make_pipeline**(**funcs: Callable[[str], str]*) → Callable[[str], str]

Make a callable pipeline that takes a text as input, passes it through one or more functions in sequential order, then outputs a single (preprocessed) text string.

This function is intended as a lightweight convenience for users, allowing them to flexibly specify which (and in which order) preprocessing functions are to be applied to raw texts, then treating the whole thing as a single callable.

```
>>> from textacy import preprocessing
>>> preproc = preprocessing.make_pipeline(
...     preprocessing.replace.hashtags,
...     preprocessing.replace.user_handles,
```

(continues on next page)

(continued from previous page)

```
...     preprocessing.replace.emojis,
... )
>>> preproc("@spacy_io is OSS for industrial-strength NLP in Python developed by␣
↪@explosion_ai ")
'_USER_ is OSS for industrial-strength NLP in Python developed by _USER_ _EMOJI_'
>>> preproc("hacking with my buddy Isaac Mewton  #PawProgramming")
'hacking with my buddy Isaac Mewton _EMOJI_ _TAG_'
```

Parameters ***funcs** –

Returns Pipeline composed of `*funcs` that applies each in sequential order.

Normalize

textacy.preprocessing.normalize: Normalize aspects of raw text that may vary in problematic ways.

textacy.preprocessing.normalize.**bullet_points**(*text: str*) → str
Normalize all "fancy" bullet point symbols in `text` to just the basic ASCII "-", provided they are the first non-whitespace characters on a new line (like a list of items).

textacy.preprocessing.normalize.**hyphenated_words**(*text: str*) → str
Normalize words in `text` that have been split across lines by a hyphen for visual consistency (aka hyphenated) by joining the pieces back together, sans hyphen and whitespace.

textacy.preprocessing.normalize.**quotation_marks**(*text: str*) → str
Normalize all "fancy" single- and double-quotation marks in `text` to just the basic ASCII equivalents. Note that this will also normalize fancy apostrophes, which are typically represented as single quotation marks.

textacy.preprocessing.normalize.**repeating_chars**(*text: str*, *, *chars: str*, *maxn: int = 1*) → str
Normalize repeating characters in `text` by truncating their number of consecutive repetitions to `maxn`.

Parameters

- **text** –
- **chars** – One or more characters whose consecutive repetitions are to be normalized, e.g. "." or "?!".
- **maxn** – Maximum number of consecutive repetitions of `chars` to which longer repetitions will be truncated.

Returns str

textacy.preprocessing.normalize.**unicode**(*text: str*, *, *form: str = 'NFC'*) → str
Normalize unicode characters in `text` into canonical forms.

Parameters

- **text** –
- **form** (*{"NFC", "NFD", "NFKC", "NFKD"}*) – Form of normalization applied to unicode characters. For example, an "e" with accute accent "´" can be written as "e´" (canonical decomposition, "NFD") or "é" (canonical composition, "NFC"). Unicode can be normalized to NFC form without any change in meaning, so it's usually a safe bet. If "NFKC", additional normalizations are applied that can change characters' meanings, e.g. ellipsis characters are replaced with three periods.

See also:

https://docs.python.org/3/library/unicodedata.html#unicodedata.normalize

textacy.preprocessing.normalize.**whitespace**(*text: str*) → str
Replace all contiguous zero-width spaces with an empty string, line-breaking spaces with a single newline, and non-breaking spaces with a single space, then strip any leading/trailing whitespace.

Remove

textacy.preprocessing.remove: Remove aspects of raw text that may be unwanted for certain use cases.

textacy.preprocessing.remove.**accents**(*text: str*, *, *fast: bool = False*) → str
Remove accents from any accented unicode characters in `text`, either by replacing them with ASCII equivalents or removing them entirely.

Parameters

- **text** –
- **fast** – If False, accents are removed from any unicode symbol with a direct ASCII equivalent; if True, accented chars for all unicode symbols are removed, regardless.

Note: `fast=True` can be significantly faster than `fast=False`, but its transformation of `text` is less "safe" and more likely to result in changes of meaning, spelling errors, etc.

Returns str

See also:

For a more powerful (but slower) alternative, check out `unidecode`: https://github.com/avian2/unidecode

textacy.preprocessing.remove.**brackets**(*text: str*, *, *only: Optional[str | Collection[str]] = None*) → str
Remove text within curly {}, square [], and/or round () brackets, as well as the brackets themselves.

Parameters

- **text** –
- **only** – Remove only those bracketed contents as specified here: "curly", "square", and/or "round". For example, `"square"` removes only those contents found between square brackets, while `["round", "square"]` removes those contents found between square or round brackets, but not curly.

Returns str

Note: This function relies on regular expressions, applied sequentially for curly, square, then round brackets; as such, it doesn't handle nested brackets of the same type and may behave unexpectedly on text with "wild" use of brackets. It should be fine removing structured bracketed contents, as is often used, for instance, to denote in-text citations.

---

textacy.preprocessing.remove.**html_tags**(*text: str*) → str
Remove HTML tags from `text`, returning just the text found between tags and other non-data elements.

Parameters **text** –

Returns str

Note: This function relies on the stdlib `html.parser.HTMLParser` and doesn't do anything fancy. For a better and potentially faster solution, consider using `lxml` and/or `beautifulsoup4`.

textacy.preprocessing.remove.**punctuation**(*text: str*, *, *only: Optional[str | Collection[str]] = None*) → str

Remove punctuation from `text` by replacing all instances of punctuation (or a subset thereof specified by `only`) with whitespace.

Parameters

- **text** –
- **only** – Remove only those punctuation marks specified here. For example, `"."` removes only periods, while `[",", ";", ":"]` removes commas, semicolons, and colons; if None, all unicode punctuation marks are removed.

Returns str

Note: When `only=None`, Python's built-in `str.translate()` is used to remove punctuation; otherwise, a regular expression is used. The former's performance can be up to an order of magnitude faster.

Replace

textacy.preprocessing.replace: Replace parts of raw text that are semantically important as members of a group but not so much in the individual instances. Can also be used to remove such parts by specifying `repl=""` in function calls.

textacy.preprocessing.replace.**currency_symbols**(*text: str*, *repl: str = '_CUR_'*) → str
Replace all currency symbols in `text` with `repl`.

textacy.preprocessing.replace.**emails**(*text: str*, *repl: str = '_EMAIL_'*) → str
Replace all email addresses in `text` with `repl`.

textacy.preprocessing.replace.**emojis**(*text: str*, *repl: str = '_EMOJI_'*) → str
Replace all emoji and pictographs in `text` with `repl`.

Note: If your Python has a narrow unicode build ("USC-2"), only dingbats and miscellaneous symbols are replaced because Python isn't able to represent the unicode data for things like emoticons. Sorry!

textacy.preprocessing.replace.**hashtags**(*text: str*, *repl: str = '_TAG_'*) → str
Replace all hashtags in `text` with `repl`.

textacy.preprocessing.replace.**numbers**(*text: str*, *repl: str = '_NUMBER_'*) → str
Replace all numbers in `text` with `repl`.

textacy.preprocessing.replace.**phone_numbers**(*text: str*, *repl: str = '_PHONE_'*) → str
Replace all phone numbers in `text` with `repl`.

textacy.preprocessing.replace.**urls**(*text: str*, *repl: str = '_URL_'*) → str
Replace all URLs in `text` with `repl`.

textacy.preprocessing.replace.**user_handles**(*text: str*, *repl: str = '_USER_'*) → str
Replace all (Twitter-style) user handles in `text` with `repl`.

4.4.4 Information Extraction

basics.words	Extract an ordered sequence of words from a document processed by spaCy, optionally filtering words by part-of-speech tag and frequency.
basics.ngrams	Extract an ordered sequence of n-grams (`n` consecutive tokens) from a spaCy `Doc` or `Span`, for one or multiple `n` values, optionally filtering n-grams by the types and parts-of-speech of the constituent tokens.
basics.entities	Extract an ordered sequence of named entities (PERSON, ORG, LOC, etc.) from a `Doc`, optionally filtering by entity types and frequencies.
basics.noun_chunks	Extract an ordered sequence of noun chunks from a spacy-parsed doc, optionally filtering by frequency and dropping leading determiners.
basics.terms	Extract one or multiple types of terms – ngrams, entities, and/or noun chunks – from `doclike` as a single, concatenated collection, with optional deduplication of spans extracted by more than one type.
matches.token_matches	Extract `Span` s from a document or sentence matching one or more patterns of per-token attr:value pairs, with optional quantity qualifiers.
matches.regex_matches	Extract `Span` s from a document or sentence whose full texts match against a regular expression `pattern`.
triples.subject_verb_object_triples	Extract an ordered sequence of subject-verb-object triples from a document or sentence.
triples.semistructured_statements	Extract "semi-structured statements" from a document as a sequence of (entity, cue, fragment) triples.
triples.direct_quotations	Extract direct quotations with an attributable speaker from a document using simple rules and patterns.
acros.acronyms	Extract tokens whose text is "acronym-like" from a document or sentence, in order of appearance.
acros.acronyms_and_definitions	Extract a collection of acronyms and their most likely definitions, if available, from a spacy-parsed doc.
kwic.keyword_in_context	Search for `keyword` matches in `doc` via regular expression and yield matches along with `window_width` characters of context before and after occurrence.
keyterms.textrank	Extract key terms from a document using the TextRank algorithm, or a variation thereof.
keyterms.yake	Extract key terms from a document using the YAKE algorithm.
keyterms.scake	Extract key terms from a document using the sCAKE algorithm.
keyterms.sgrank	Extract key terms from a document using the SGRank algorithm.

Basics

textacy.extract.basics: Extract basic components from a document or sentence via spaCy, with bells and whistles for filtering the results.

textacy.extract.basics.**words**(*doclike: types.DocLike*, *, *filter_stops: bool = True*, *filter_punct: bool = True*, *filter_nums: bool = False*, *include_pos: Optional[str | Collection[str]] = None*, *exclude_pos: Optional[str | Collection[str]] = None*, *min_freq: int = 1*) → Iterable[Token]

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

Parameters

- **doclike** –
- **filter_stops** – If True, remove stop words from word list.
- **filter_punct** – If True, remove punctuation from word list.
- **filter_nums** – If True, remove number-like words (e.g. 10, "ten") from word list.
- **include_pos** – Remove words whose part-of-speech tag IS NOT in the specified tags.
- **exclude_pos** – Remove words whose part-of-speech tag IS in the specified tags.
- **min_freq** – Remove words that occur in `doclike` fewer than `min_freq` times.

Yields Next token from `doclike` 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; for details, check spaCy's docs: https://spacy.io/api/annotation#pos-tagging

---

textacy.extract.basics.**ngrams**(*doclike: types.DocLike*, *n: int | Collection[int]*, *, *filter_stops: bool = True*, *filter_punct: bool = True*, *filter_nums: bool = False*, *include_pos: Optional[str | Collection[str]] = None*, *exclude_pos: Optional[str | Collection[str]] = None*, *min_freq: int = 1*) → Iterable[Span]

Extract an ordered sequence of n-grams (`n` consecutive tokens) from a spaCy `Doc` or `Span`, for one or multiple `n` values, optionally filtering n-grams by the types and parts-of-speech of the constituent tokens.

Parameters

- **doclike** –
- **n** – Number of tokens included per n-gram; for example, `2` yields bigrams and `3` yields trigrams. If multiple values are specified, then the collections of n-grams are concatenated together; for example, `(2, 3)` yields bigrams and then trigrams.
- **filter_stops** – If True, remove ngrams that start or end with a stop word.
- **filter_punct** – If True, remove ngrams that contain any punctuation-only tokens.
- **filter_nums** – If True, remove ngrams that contain any numbers or number-like tokens (e.g. 10, 'ten').
- **include_pos** – Remove ngrams if any constituent tokens' part-of-speech tags ARE NOT included in this param.
- **exclude_pos** – Remove ngrams if any constituent tokens' part-of-speech tags ARE included in this param.

- **min_freq** – Remove ngrams that occur in `doclike` fewer than `min_freq` times

Yields Next ngram from `doclike` passing all specified filters, in order of appearance in the document.

Raises

- **ValueError** – if any `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; for details, check spaCy's docs: https://spacy.io/api/annotation#pos-tagging

---

`textacy.extract.basics.`**entities**(*doclike: types.DocLike*, ***, *include_types: Optional[str | Collection[str]] = None*, *exclude_types: Optional[str | Collection[str]] = None*, *drop_determiners: bool = True*, *min_freq: int = 1*) → Iterable[Span]

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

Parameters

- **doclike** –
- **include_types** – Remove entities whose type IS NOT in this param; if "NUMERIC", all numeric entity types ("DATE", "MONEY", "ORDINAL", etc.) are included
- **exclude_types** – Remove entities whose type IS in this param; if "NUMERIC", all numeric entity types ("DATE", "MONEY", "ORDINAL", etc.) are excluded
- **drop_determiners** – Remove leading determiners (e.g. "the") from entities (e.g. "the United States" => "United States").

  Note: Entities from which a leading determiner has been removed are, effectively, *new* entities, and not saved to the `Doc` from which they came. This is irritating but unavoidable, since this function is not meant to have side-effects on document state. If you're only using the text of the returned spans, this is no big deal, but watch out if you're counting on determiner-less entities associated with the doc downstream.

- **min_freq** – Remove entities that occur in `doclike` fewer than `min_freq` times

Yields Next entity from `doclike` passing all specified filters in order of appearance in the document

Raises **TypeError** – if `include_types` or `exclude_types` is not a str, a set of str, or a falsy value

`textacy.extract.basics.`**noun_chunks**(*doclike: Union[spacy.tokens.doc.Doc, spacy.tokens.span.Span]*, ***, *drop_determiners: bool = True*, *min_freq: int = 1*) → Iterable[spacy.tokens.span.Span]

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

Parameters

- **doclike** –

- **drop_determiners** – Remove leading determiners (e.g. "the") from phrases (e.g. "the quick brown fox" => "quick brown fox")
- **min_freq** – Remove chunks that occur in `doclike` fewer than `min_freq` times

Yields Next noun chunk from `doclike` in order of appearance in the document

`textacy.extract.basics.`**terms**(*doclike: types.DocLike*, ***, *ngs: Optional[int | Collection[int] | types.DocLikeToSpans] = None*, *ents: Optional[bool | types.DocLikeToSpans] = None*, *ncs: Optional[bool | types.DocLikeToSpans] = None*, *dedupe: bool = True*) → Iterable[Span]

Extract one or multiple types of terms – ngrams, entities, and/or noun chunks – from `doclike` as a single, concatenated collection, with optional deduplication of spans extracted by more than one type.

```
>>> extract.terms(doc, ngs=2, ents=True, ncs=True)
>>> extract.terms(doc, ngs=lambda doc: extract.ngrams(doc, n=2))
>>> extract.terms(doc, ents=extract.entities)
>>> extract.terms(doc, ents=partial(extract.entities, include_types="PERSON"))
```

Parameters

- **doclike** –
- **ngs** – N-gram terms to be extracted. If one or multiple ints, `textacy.extract.ngrams(doclike, n=ngs)()` is used to extract terms; if a callable, `ngs(doclike)` is used to extract terms; if None, no n-gram terms are extracted.
- **ents** – Entity terms to be extracted. If True, `textacy.extract.entities(doclike)()` is used to extract terms; if a callable, `ents(doclike)` is used to extract terms; if None, no entity terms are extracted.
- **ncs** – Noun chunk terms to be extracted. If True, `textacy.extract.noun_chunks(doclike)()` is used to extract terms; if a callable, `ncs(doclike)` is used to extract terms; if None, no noun chunk terms are extracted.
- **dedupe** – If True, deduplicate terms whose spans are extracted by multiple types (e.g. a span that is both an n-gram and an entity), as identified by identical (start, stop) indexes in `doclike`; otherwise, don't.

Returns Next term from `doclike`, in order of n-grams then entities then noun chunks, with each collection's terms given in order of appearance.

Note: This function is *not* to be confused with keyterm extraction, which leverages statistics and algorithms to quantify the "key"-ness of terms before returning the top-ranking terms. There is no such scoring or ranking here.

See also:

- `textacy.extact.ngrams()`
- `textacy.extact.entities()`
- `textacy.extact.noun_chunks()`
- `textacy.extact.keyterms`

Matches

textacy.extract.matches: Extract matching spans from a document or sentence using spaCy's built-in matcher or regular expressions.

textacy.extract.matches.**token_matches**(*doclike: types.DocLike*, *patterns: str | List[str] | List[Dict[str, str]] | List[List[Dict[str, str]]]*, ***, *on_match: Optional[Callable] = None*) → Iterable[Span]

Extract `Span` s from a document or sentence matching one or more patterns of per-token attr:value pairs, with optional quantity qualifiers.

Parameters

- **doclike** –
- **patterns** – One or multiple patterns to match against `doclike` using a `spacy.matcher.Matcher`.

  If List[dict] or List[List[dict]], each pattern is specified as attr: value pairs per token, with optional quantity qualifiers:

  - `[{"POS": "NOUN"}]` matches singular or plural nouns, like "friend" or "enemies"
  - `[{"POS": "PREP"}, {"POS": "DET", "OP": "?"}, {"POS": "ADJ", "OP": "?"}, {"POS": "NOUN", "OP": "+"}]` matches prepositional phrases, like "in the future" or "from the distant past"
  - `[{"IS_DIGIT": True}, {"TAG": "NNS"}]` matches numbered plural nouns, like "60 seconds" or "2 beers"
  - `[{"POS": "PROPN", "OP": "+"}, {}]` matches proper nouns and whatever word follows them, like "Burton DeWilde yaaasss"

  If str or List[str], each pattern is specified as one or more per-token patterns separated by whitespace where attribute, value, and optional quantity qualifiers are delimited by colons. Note that boolean and integer values have special syntax — "bool(val)" and "int(val)", respectively — and that wildcard tokens still need a colon between the (empty) attribute and value strings.

  - `"POS:NOUN"` matches singular or plural nouns
  - `"POS:PREP POS:DET:? POS:ADJ:? POS:NOUN:+"` matches prepositional phrases
  - `"IS_DIGIT:bool(True) TAG:NNS"` matches numbered plural nouns
  - `"POS:PROPN:+ :"` matches proper nouns and whatever word follows them

  Also note that these pattern strings don't support spaCy v2.1's "extended" pattern syntax; if you need such complex patterns, it's probably better to use a List[dict] or List[List[dict]], anyway.
- **on_match** – Callback function to act on matches. Takes the arguments `matcher`, `doclike`, `i` and `matches`.

Yields Next matching `Span` in `doclike`, in order of appearance

Raises

- **TypeError** –
- **ValueError** –

See also:

- https://spacy.io/usage/rule-based-matching
- https://spacy.io/api/matcher

textacy.extract.matches.**regex_matches**(*doclike: types.DocLike*, *pattern: str | Pattern*, *, *alignment_mode: str = 'strict'*) → Iterable[Span]

Extract `Span` s from a document or sentence whose full texts match against a regular expression `pattern`.

Parameters

- **doclike** –
- **pattern** – Valid regular expression against which to match document text, either as a string or compiled pattern object.
- **alignment_mode** – How character indices of regex matches snap to spaCy token boundaries. If "strict", only exact alignments are included (no snapping); if "contract", tokens completely within the character span are included; if "expand", tokens at least partially covered by the character span are included.

Yields Next matching `Span`.

Triples

textacy.extract.triples: Extract structured triples from a document or sentence through rule-based pattern-matching of the annotated tokens.

class textacy.extract.triples.**SVOTriple**(*subject*, *verb*, *object*)

object
Alias for field number 2

subject
Alias for field number 0

verb
Alias for field number 1

class textacy.extract.triples.**SSSTriple**(*entity*, *cue*, *fragment*)

cue
Alias for field number 1

entity
Alias for field number 0

fragment
Alias for field number 2

class textacy.extract.triples.**DQTriple**(*speaker*, *cue*, *content*)

content
Alias for field number 2

cue
Alias for field number 1

speaker
Alias for field number 0

textacy.extract.triples.**subject_verb_object_triples**(*doclike: Union[spacy.tokens.doc.Doc, spacy.tokens.span.Span]*) → Iterable[*textacy.extract.triples.SVOTriple*]

Extract an ordered sequence of subject-verb-object triples from a document or sentence.

Parameters **doclike** –

Yields Next SVO triple as (subject, verb, object), in approximate order of appearance.

textacy.extract.triples.**semistructured_statements**(*doclike: types.DocLike*, ***, *entity: str | Pattern*, *cue: str*, *fragment_len_range: Optional[Tuple[Optional[int], Optional[int]]] = None*) → Iterable[*SSSTriple*]

Extract "semi-structured statements" from a document as a sequence of (entity, cue, fragment) triples.

Parameters

- **doclike** –
- **entity** – Noun or noun phrase of interest expressed as a regular expression pattern string (e.g. `"[Gg]lobal [Ww]arming"`) or compiled object (e.g. `re.compile("global warming", re.IGNORECASE)`).
- **cue** – Verb lemma with which `entity` is associated (e.g. "be", "have", "say").
- **fragment_len_range** – Filter statements to those whose fragment length in tokens is within the specified [low, high) interval. Both low and high values must be specified, but a null value for either is automatically replaced by safe default values. None (default) skips filtering by fragment length.

Yields Next matching triple, consisting of (entity, cue, fragment), in order of appearance.

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.triples.**direct_quotations**(*doc: spacy.tokens.doc.Doc*) → Iterable[*textacy.extract.triples.DQTriple*]

Extract direct quotations with an attributable speaker from a document using simple rules and patterns. Does not extract indirect or mixed quotations!

Parameters **doc** –

Yields Next direct quotation in `doc` as a (speaker, cue, content) triple.

Notes

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

textacy.extract.triples.**expand_noun**(*tok: spacy.tokens.token.Token*) → List[spacy.tokens.token.Token]

Expand a noun token to include all associated conjunct and compound nouns.

textacy.extract.triples.**expand_verb**(*tok: spacy.tokens.token.Token*) → List[spacy.tokens.token.Token]

Expand a verb token to include all associated auxiliary and negation tokens.

Acronyms

`textacy.extract.acronyms`: Extract acronyms and their definitions from a document or sentence through rule-based pattern-matching of the annotated tokens.

textacy.extract.acros.**acronyms**(*doclike: Union[spacy.tokens.doc.Doc, spacy.tokens.span.Span]*) → Iterable[spacy.tokens.token.Token]

Extract tokens whose text is "acronym-like" from a document or sentence, in order of appearance.

Parameters **doclike** –

Yields Next acronym-like `Token`.

textacy.extract.acros.**acronyms_and_definitions**(*doclike: Union[spacy.tokens.doc.Doc, spacy.tokens.span.Span]*, *known_acro_defs: Optional[Dict[str, str]] = None*) → Dict[str, List[str]]

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

- **doclike** –
- **known_acro_defs** – 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)

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.acros.**is_acronym**(*token: str*, *exclude: Optional[Set[str]] = None*) → bool

Pass single token as a string, return True/False if is/is not valid acronym.

Parameters

- **token** – Single word to check for acronym-ness
- **exclude** – If technically valid but not actual acronyms are known in advance, pass them in as a set of strings; matching tokens will return False.

Returns Whether or not `token` is an acronym.

KWIC

textacy.extract.kwic: Extract keywords with their surrounding contexts from a text document using regular expressions.

textacy.extract.kwic.**keyword_in_context**(*doc: Doc | str*, *keyword: str | Pattern*, ***, *ignore_case: bool = True*, *window_width: int = 50*, *pad_context: bool = False*) → Iterable[Tuple[str, str, str]]

Search for `keyword` matches in `doc` via regular expression and yield matches along with `window_width` characters of context before and after occurrence.

Parameters

- **doc** – spaCy `Doc` or raw text in which to search for `keyword`. If a `Doc`, constituent text is grabbed via `spacy.tokens.Doc.text`. Note that spaCy annotations aren't used at all here, they're just a convenient owner of document text.
- **keyword** – String or regular expression pattern defining the keyword(s) to match. Typically, this is a single word or short phrase ("spam", "spam and eggs"), but to account for variations, use regex (`r"[Ss]pam (and|&) [Ee]ggs?"`), optionally compiled (`re.compile(r"[Ss]pam (and|&) [Ee]ggs?")`).
- **ignore_case** – If True, ignore letter case in `keyword` matching; otherwise, use case-sensitive matching. Note that this argument is only used if `keyword` is a string; for pre-compiled regular expressions, the `re.IGNORECASE` flag is left as-is.
- **window_width** – Number of characters on either side of `keyword` to include as "context".
- **pad_context** – If True, pad pre- and post-context strings to `window_width` chars in length; otherwise, us as many chars as are found in the text, up to the specified width.

Yields Next matching triple of (pre-context, keyword match, post-context).

Keyterms

textacy.extract.keyterms: Extract keyterms from documents using a variety of rule-based algorithms.

textacy.extract.keyterms.textrank.**textrank**(*doc: Doc*, ***, *normalize: Optional[str | Callable[[Token], str]] = 'lemma'*, *include_pos: Optional[str | Collection[str]] = ('NOUN', 'PROPN', 'ADJ')*, *window_size: int = 2*, *edge_weighting: str = 'binary'*, *position_bias: bool = False*, *topn: int | float = 10*) → List[Tuple[str, float]]

Extract key terms from a document using the TextRank algorithm, or a variation thereof. For example:

- TextRank: `window_size=2, edge_weighting="binary", position_bias=False`
- SingleRank: `window_size=10, edge_weighting="count", position_bias=False`
- PositionRank: `window_size=10, edge_weighting="count", position_bias=True`

Parameters

- **doc** – spaCy `Doc` from which to extract keyterms.
- **normalize** – 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 `Token` and return a str, e.g. *textacy.spacier.utils.get_normalized_text()*.

- **include_pos** – One or more POS tags with which to filter for good candidate keyterms. If None, include tokens of all POS tags (which also allows keyterm extraction from docs without POS-tagging.)
- **window_size** – Size of sliding window in which term co-occurrences are determined.
- **edge_weighting** (*{"count", "binary"}*) – : If "count", the nodes for all co-occurring terms are connected by edges with weight equal to the number of times they co-occurred within a sliding window; if "binary", all such edges have weight = 1.
- **position_bias** – If True, bias the PageRank algorithm for weighting nodes in the word graph, such that words appearing earlier and more frequently in `doc` tend to get larger weights.
- **topn** – Number of top-ranked terms to return as key terms. If an integer, represents the absolute number; if a float, value must be in the interval (0.0, 1.0], which is converted to an int by `int(round(len(set(candidates)) * topn))`.

Returns Sorted list of top `topn` key terms and their corresponding TextRank ranking scores.

References

- Mihalcea, R., & Tarau, P. (2004, July). TextRank: Bringing order into texts. Association for Computational Linguistics.
- Wan, Xiaojun and Jianguo Xiao. 2008. Single document keyphrase extraction using neighborhood knowledge. In Proceedings of the 23rd AAAI Conference on Artificial Intelligence, pages 855–860.
- Florescu, C. and Cornelia, C. (2017). PositionRank: An Unsupervised Approach to Keyphrase Extraction from Scholarly Documents. In proceedings of ACL*, pages 1105-1115.

`textacy.extract.keyterms.yake.`**yake**(*doc: Doc*, ***, *normalize: Optional[str] = 'lemma'*, *ngrams: int | Collection[int] = (1, 2, 3)*, *include_pos: Optional[str | Collection[str]] = ('NOUN', 'PROPN', 'ADJ')*, *window_size: int = 2*, *topn: int | float = 10*) → List[Tuple[str, float]]

Extract key terms from a document using the YAKE algorithm.

Parameters

- **doc** – spaCy `Doc` from which to extract keyterms. Must be sentence-segmented; optionally POS-tagged.
- **normalize** – If "lemma", lemmatize terms; if "lower", lowercase terms; if None, use the form of terms as they appeared in `doc`.

  Note: Unlike the other keyterm extraction functions, this one doesn't accept a callable for `normalize`.

- **ngrams** – n of which n-grams to consider as keyterm candidates. For example, *(1, 2, 3)*` includes all unigrams, bigrams, and trigrams, while `2` includes bigrams only.
- **include_pos** – One or more POS tags with which to filter for good candidate keyterms. If None, include tokens of all POS tags (which also allows keyterm extraction from docs without POS-tagging.)

- **window_size** – Number of words to the right and left of a given word to use as context when computing the "relatedness to context" component of its score. Note that the resulting sliding window's full width is `1 + (2 * window_size)`.
- **topn** – Number of top-ranked terms to return as key terms. If an integer, represents the absolute number; if a float, value must be in the interval (0.0, 1.0], which is converted to an int by `int(round(len(candidates) * topn))`

Returns Sorted list of top `topn` key terms and their corresponding YAKE scores.

References

Campos, Mangaravite, Pasquali, Jorge, Nunes, and Jatowt. (2018). A Text Feature Based Automatic Keyword Extraction Method for Single Documents. Advances in Information Retrieval. ECIR 2018. Lecture Notes in Computer Science, vol 10772, pp. 684-691.

`textacy.extract.keyterms.scake.`**scake**(*doc: Doc*, ***, *normalize: Optional[str | Callable[[Token], str]] = 'lemma'*, *include_pos: Optional[str | Collection[str]] = ('NOUN', 'PROPN', 'ADJ')*, *topn: int | float = 10*) → List[Tuple[str, float]]

Extract key terms from a document using the sCAKE algorithm.

Parameters

- **doc** – spaCy `Doc` from which to extract keyterms. Must be sentence-segmented; optionally POS-tagged.
- **normalize** – 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 `Token` and return a str, e.g. *`textacy.spacier.utils.get_normalized_text()`*.
- **include_pos** – One or more POS tags with which to filter for good candidate keyterms. If None, include tokens of all POS tags (which also allows keyterm extraction from docs without POS-tagging.)
- **topn** – Number of top-ranked terms to return as key terms. If an integer, represents the absolute number; if a float, value must be in the interval (0.0, 1.0], which is converted to an int by `int(round(len(candidates) * topn))`

Returns Sorted list of top `topn` key terms and their corresponding scores.

References

Duari, Swagata & Bhatnagar, Vasudha. (2018). sCAKE: Semantic Connectivity Aware Keyword Extraction. Information Sciences. 477. https://arxiv.org/abs/1811.10831v1

class `textacy.extract.keyterms.sgrank.`**Candidate**(*text*, *idx*, *length*, *count*)

count
Alias for field number 3

idx
Alias for field number 1

length
Alias for field number 2

text
Alias for field number 0

textacy.extract.keyterms.sgrank.**sgrank**(*doc: Doc, *, normalize: Optional[str | Callable[[Span], str]] = 'lemma', ngrams: int | Collection[int] = (1, 2, 3, 4, 5, 6), include_pos: Optional[str | Collection[str]] = ('NOUN', 'PROPN', 'ADJ'), window_size: int = 1500, topn: int | float = 10, idf: Dict[str, float] = None*) → List[Tuple[str, float]]

Extract key terms from a document using the SGRank algorithm.

Parameters

- **doc** – spaCy `Doc` from which to extract keyterms.
- **normalize** – 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 `Span` and return a str, e.g. *textacy.spacier.utils.get_normalized_text()*
- **ngrams** – n of which n-grams to include. For example, `(1, 2, 3, 4, 5, 6)` (default) includes all ngrams from 1 to 6; *2* if only bigrams are wanted
- **include_pos** – One or more POS tags with which to filter for good candidate keyterms. If None, include tokens of all POS tags (which also allows keyterm extraction from docs without POS-tagging.)
- **window_size** – Size 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.
- **topn** – 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(candidates) * topn))`
- **idf** – Mapping of `normalize(term)` to inverse document frequency for re-weighting of unigrams (n-grams with n > 1 have df assumed = 1). Results are typically better with idf information.

Returns Sorted list of top `topn` key terms and their corresponding SGRank scores

Raises **ValueError** – if `topn` is a float but not in (0.0, 1.0] or `window_size` < 2

References

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.

Utils

textacy.extract.utils: Functions for working with extraction results.

textacy.extract.utils.**terms_to_strings**(*terms: Iterable[types.SpanLike], by: str | Callable[[types.SpanLike], str]*) → Iterable[str]

Transform a sequence of terms as spaCy `Token` s or `Span` s into strings.

Parameters

- **terms** –

- **by** – Method by which terms are transformed into strings. If "orth", terms are represented by their text exactly as written; if "lower", by the lowercased form of their text; if "lemma", by their base form w/o inflectional suffixes; if a callable, must accept a `Token` or `Span` and return a string.

Yields Next term in `terms`, as a string.

`textacy.extract.utils.`**`clean_term_strings`**(*terms: Iterable[str]*) → Iterable[str]

Clean up a sequence of single- or multi-word terms as strings: strip leading/trailing junk chars, handle dangling parens and odd hyphenation, and normalize whitespace.

Parameters **terms** – Sequence of terms such as "environment", "plastic pollution", or "fossil fuel industry" that may be _unclean_ for whatever reason.

Yields 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.extract.utils.`**`aggregate_term_variants`**(*terms: Set[str]*, ***, *acro_defs: Optional[Dict[str, str]] = None*, *fuzzy_dedupe: bool = True*) → List[Set[str]]

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 of unique terms with potential duplicates
- **acro_defs** – 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** – 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.

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.extract.utils.`**`get_longest_subsequence_candidates`**(*doc: spacy.tokens.doc.Doc*, *match_func: Callable[[spacy.tokens.token.Token], bool]*) → Iterable[Tuple[spacy.tokens.token.Token, ...]]

Get candidate keyterms from `doc`, where candidates are longest consecutive subsequences of tokens for which all `match_func(token)` is True.

Parameters

- **doc** –

- **match_func** – Function applied sequentially to each `Token` in `doc` that returns True for matching ("good") tokens, False otherwise.

Yields Next longest consecutive subsequence candidate, as a tuple of constituent tokens.

`textacy.extract.utils.`**`get_ngram_candidates`**(*doc: Doc*, *ns: int | Collection[int]*, ***, *include_pos: Optional[str | Collection[str]] = ('NOUN', 'PROPN', 'ADJ')*) → Iterable[Tuple[Token, ...]]

Get candidate keyterms from `doc`, where candidates are n-length sequences of tokens (for all n in `ns`) that don't start/end with a stop word or contain punctuation tokens, and whose constituent tokens are filtered by POS tag.

Parameters

- **doc** –
- **ns** – One or more n values for which to generate n-grams. For example, `2` gets bigrams; `(2, 3)` gets bigrams and trigrams.
- **include_pos** – One or more POS tags with which to filter ngrams. If None, include tokens of all POS tags.

Yields Next ngram candidate, as a tuple of constituent Tokens.

See also:

`textacy.extract.ngrams()`

`textacy.extract.utils.`**`get_pattern_matching_candidates`**(*doc: Doc*, *patterns: str | List[str] | List[dict] | List[List[dict]]*) → Iterable[Tuple[Token, ...]]

Get candidate keyterms from `doc`, where candidates are sequences of tokens that match any pattern in `patterns`

Parameters

- **doc** –
- **patterns** – One or multiple patterns to match against `doc` using a `spacy.matcher.Matcher`.

Yields Tuple[`spacy.tokens.Token`] – Next pattern-matching candidate, as a tuple of constituent Tokens.

See also:

`textacy.extract.token_matches()`

`textacy.extract.utils.`**`get_filtered_topn_terms`**(*term_scores: Iterable[Tuple[str, float]]*, *topn: int*, ***, *match_threshold: Optional[float] = None*) → List[Tuple[str, float]]

Build up a list of the `topn` terms, filtering out any that are substrings of better-scoring terms and optionally filtering out any that are sufficiently similar to better-scoring terms.

Parameters

- **term_scores** – Iterable of (term, score) pairs, sorted in order of score from best to worst. Note that this may be from high to low value or low to high, depending on the scoring algorithm.
- **topn** – Maximum number of top-scoring terms to get.

- **match_threshold** – Minimal edit distance between a term and previously seen terms, used to filter out terms that are sufficiently similar to higher-scoring terms. Uses `textacy.similarity.token_sort_ratio()`.

4.4.5 Text Statistics

api.TextStats	Class to compute a variety of basic and readability statistics for a given doc, where each stat is a lazily-computed attribute.
basics.n_sents	Compute the number of sentences in a document.
basics.n_words	Compute the number of words in a document.
basics.n_unique_words	Compute the number of *unique* words in a document.
basics.n_chars_per_word	Compute the number of characters for each word in a document.
basics.n_chars	Compute the total number of characters in a document.
basics.n_long_words	Compute the number of long words in a document.
basics.n_syllables_per_word	Compute the number of syllables for each word in a document.
basics.n_syllables	Compute the total number of syllables in a document.
basics.n_monosyllable_words	Compute the number of monosyllobic words in a document.
basics.n_polysyllable_words	Compute the number of polysyllobic words in a document.
basics.entropy	Compute the entropy of words in a document.
readability.automated_readability_index	Readability test for English-language texts, particularly for technical writing, whose value estimates the U.S.
readability.automatic_arabic_readability_index	Readability test for Arabic-language texts based on number of characters and average word and sentence lengths.
readability.coleman_liau_index	Readability test whose value estimates the number of years of education required to understand a text, similar to `flesch_kincaid_grade_level()` and `smog_index()`, but using characters per word instead of syllables.
readability.flesch_kincaid_grade_level	Readability test used widely in education, whose value estimates the U.S.
readability.flesch_reading_ease	Readability test used as a general-purpose standard in several languages, based on a weighted combination of avg.
readability.gulpease_index	Readability test for Italian-language texts, whose value is in the range [0, 100] similar to `flesch_reading_ease()`.
readability.gunning_fog_index	Readability test whose value estimates the number of years of education required to understand a text, similar to `flesch_kincaid_grade_level()` and `smog_index()`.
readability.lix	Readability test commonly used in Sweden on both English- and non-English-language texts, whose value estimates the difficulty of reading a foreign text.

continues on next page

Table 6 – continued from previous page

readability.mu_legibility_index	Readability test for Spanish-language texts based on number of words and the mean and variance of their lengths in characters, whose value is in the range [0, 100].
readability.perspicuity_index	Readability test for Spanish-language texts, whose value is in the range [0, 100]; very similar to the Spanish-specific formulation of `flesch_reading_ease()`, but included additionally since it's become a common readability standard.
readability.smog_index	Readability test commonly used in medical writing and the healthcare industry, whose value estimates the number of years of education required to understand a text similar to `flesch_kincaid_grade_level()` and intended as a substitute for `gunning_fog_index()`.
readability.wiener_sachtextformel	Readability test for German-language texts, whose value estimates the grade level required to understand a text.

textacy.text_stats.api: Compute basic and readability statistics of documents.

class textacy.text_stats.api.**TextStats**(*doc: spacy.tokens.doc.Doc*)

Class to compute a variety of basic and readability statistics for a given doc, where each stat is a lazily-computed attribute.

```
>>> text = next(textacy.datasets.CapitolWords().texts(limit=1))
>>> doc = textacy.make_spacy_doc(text)
>>> ts = textacy.text_stats.TextStats(doc)
>>> ts.n_words
136
>>> ts.n_unique_words
80
>>> ts.entropy
6.00420319027642
>>> ts.flesch_kincaid_grade_level
11.817647058823532
>>> ts.flesch_reading_ease
50.707745098039254
```

Some stats vary by language or are designed for use with specific languages:

```
>>> text = (
...     "Muchos años después, frente al pelotón de fusilamiento, "
...     "el coronel Aureliano Buendía había de recordar aquella tarde remota "
...     "en que su padre lo llevó a conocer el hielo."
... )
>>> doc = textacy.make_spacy_doc(text, lang="es")
>>> ts = textacy.text_stats.TextStats(doc)
>>> ts.n_words
28
>>> ts.perspicuity_index
56.46000000000002
>>> ts.mu_legibility_index
71.18644067796609
```

Each of these stats have stand-alone functions in *textacy.text_stats.basics* and *textacy.text_stats.readability* with more detailed info and links in the docstrings – when in doubt, read the docs!

Parameters **doc** – A text document tokenized and (optionally) sentence-segmented by spaCy.

property n_sents
Number of sentences in document.

See also:

textacy.text_stats.basics.n_sents()

property n_words
Number of words in document.

See also:

textacy.text_stats.basics.n_words()

property n_unique_words
Number of *unique* words in document.

See also:

textacy.text_stats.basics.n_unique_words()

property n_long_words
Number of long words in document.

See also:

textacy.text_stats.basics.n_long_words()

property n_chars_per_word
Number of characters for each word in document.

See also:

textacy.text_stats.basics.n_chars_per_word()

property n_chars
Total number of characters in document.

See also:

textacy.text_stats.basics.n_chars()

property n_syllables_per_word
Number of syllables for each word in document.

See also:

textacy.text_stats.basics.n_syllables_per_word()

property n_syllables
Total number of syllables in document.

See also:

textacy.text_stats.basics.n_syllables()

property n_monosyllable_words
Number of monosyllobic words in document.

See also:

textacy.text_stats.basics.n_monosyllable_words()

property n_polysyllable_words
Number of polysyllobic words in document.

See also:

textacy.text_stats.basics.n_polysyllable_words()

property entropy
Entropy of words in document.

See also:

textacy.text_stats.basics.entropy()

property automated_readability_index
Readability test for English-language texts. Higher value => more difficult text.

See also:

textacy.text_stats.readability.automated_readability_index()

property automatic_arabic_readability_index
Readability test for Arabic-language texts. Higher value => more difficult text.

See also:

textacy.text_stats.readability.automatic_arabic_readability_index()

property coleman_liau_index
Readability test, not language-specific. Higher value => more difficult text.

See also:

textacy.text_stats.readability.coleman_liau_index()

property flesch_kincaid_grade_level
Readability test, not language-specific. Higher value => more difficult text.

See also:

textacy.text_stats.readability.flesch_kincaid_grade_level()

property flesch_reading_ease
Readability test with several language-specific formulations. Higher value => easier text.

See also:

textacy.text_stats.readability.flesch_reading_ease()

property gulpease_index
Readability test for Italian-language texts. Higher value => easier text.

See also:

textacy.text_stats.readability.gulpease_index()

property gunning_fog_index
Readability test, not language-specific. Higher value => more difficult text.

See also:

textacy.text_stats.readability.gunning_fog_index()

property lix
Readability test for both English- and non-English-language texts. Higher value => more difficult text.

See also:

textacy.text_stats.readability.lix()

property mu_legibility_index
Readability test for Spanish-language texts. Higher value => easier text.

See also:

textacy.text_stats.readability.mu_legibility_index()

property perspicuity_index
Readability test for Spanish-language texts. Higher value => easier text.

See also:

textacy.text_stats.readability.perspicuity_index()

property smog_index
Readability test, not language-specific. Higher value => more difficult text.

See also:

textacy.text_stats.readability.smog_index()

property wiener_sachtextformel
Readability test for German-language texts. Higher value => more difficult text.

See also:

textacy.text_stats.readability.wiener_sachtextformel()

textacy.text_stats.api.**load_hyphenator**(*lang: str*)
Load an object that hyphenates words at valid points, as used in LaTex typesetting.

Parameters **lang** – Standard 2-letter language abbreviation. To get a list of valid values:

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

Returns pyphen.Pyphen()

Basic Stats

textacy.text_stats.basics: Low-level functions for computing basic text statistics, typically accessed via textacy.text_stats.TextStats.

textacy.text_stats.basics.**n_words**(*doc_or_words: Union[spacy.tokens.doc.Doc, Iterable[spacy.tokens.token.Token]]*) → int
Compute the number of words in a document.

Parameters **doc_or_words** – If a spaCy Doc, non-punctuation tokens (words) are extracted; if an iterable of spaCy Token s, all are included as-is.

textacy.text_stats.basics.**n_unique_words**(*doc_or_words: Union[spacy.tokens.doc.Doc, Iterable[spacy.tokens.token.Token]]*) → int
Compute the number of *unique* words in a document.

Parameters **doc_or_words** – If a spaCy Doc, non-punctuation tokens (words) are extracted; if an iterable of spaCy Token s, all are included as-is.

textacy.text_stats.basics.**n_chars_per_word**(*doc_or_words: Union[spacy.tokens.doc.Doc, Iterable[spacy.tokens.token.Token]]*) → Tuple[int, ...]
Compute the number of characters for each word in a document.

Parameters **doc_or_words** – If a spaCy `Doc`, non-punctuation tokens (words) are extracted; if an iterable of spaCy `Token` s, all are included as-is.

textacy.text_stats.basics.**n_chars**(*n_chars_per_word: Tuple[int, ...]*) → int

Compute the total number of characters in a document.

Parameters **n_chars_per_word** – Number of characters per word in a given document, as computed by *n_chars_per_word()*.

textacy.text_stats.basics.**n_long_words**(*n_chars_per_word: Tuple[int, ...]*, *min_n_chars: int = 7*) → int

Compute the number of long words in a document.

Parameters

- **n_chars_per_word** – Number of characters per word in a given document, as computed by *n_chars_per_word()*.
- **min_n_chars** – Minimum number of characters required for a word to be considered "long".

textacy.text_stats.basics.**n_syllables_per_word**(*doc_or_words: Union[spacy.tokens.doc.Doc, Iterable[spacy.tokens.token.Token]]*, *lang: str*) → Tuple[int, ...]

Compute the number of syllables for each word in a document.

Parameters **doc_or_words** – If a spaCy `Doc`, non-punctuation tokens (words) are extracted; if an iterable of spaCy `Token` s, all are included as-is.

Note: Identifying syllables is _tricky_; this method relies on hyphenation, which is more straightforward but doesn't always give the correct number of syllables. While all hyphenation points fall on syllable divisions, not all syllable divisions are valid hyphenation points.

---

textacy.text_stats.basics.**n_syllables**(*n_syllables_per_word: Tuple[int, ...]*) → int

Compute the total number of syllables in a document.

Parameters **n_syllables_per_word** – Number of syllables per word in a given document, as computed by *n_syllables_per_word()*.

textacy.text_stats.basics.**n_monosyllable_words**(*n_syllables_per_word: Tuple[int, ...]*) → int

Compute the number of monosyllobic words in a document.

Parameters **n_syllables_per_word** – Number of syllables per word in a given document, as computed by *n_syllables_per_word()*.

textacy.text_stats.basics.**n_polysyllable_words**(*n_syllables_per_word: Tuple[int, ...]*, *min_n_syllables: int = 3*) → int

Compute the number of polysyllobic words in a document.

Parameters

- **n_syllables_per_word** – Number of syllables per word in a given document, as computed by *n_syllables_per_word()*.
- **min_n_syllables** – Minimum number of syllables required for a word to be considered "polysyllobic".

textacy.text_stats.basics.**n_sents**(*doc: spacy.tokens.doc.Doc*) → int

Compute the number of sentences in a document.

> **Warning:** If `doc` has not been segmented into sentences, it will be modified in-place using spaCy's rule-based `Sentencizer` pipeline component before counting.

`textacy.text_stats.basics.`**`entropy`**(*doc_or_words: Union[spacy.tokens.doc.Doc, Iterable[spacy.tokens.token.Token]]*) → float

Compute the entropy of words in a document.

Parameters **`doc_or_words`** – If a spaCy `Doc`, non-punctuation tokens (words) are extracted; if an iterable of spaCy `Token` s, all are included as-is.

Readability Stats

`textacy.text_stats.readability`: Low-level functions for computing various measures of text "readability", typically accessed via `textacy.text_stats.TextStats`.

`textacy.text_stats.readability.`**`automated_readability_index`**(*n_chars: int*, *n_words: int*, *n_sents: int*) → float

Readability test for English-language texts, particularly for technical writing, whose value estimates the U.S. grade level required to understand a text. Similar to several other tests (e.g. *`flesch_kincaid_grade_level()`*), but uses characters per word instead of syllables like *`coleman_liau_index()`*. Higher value => more difficult text.

References

https://en.wikipedia.org/wiki/Automated_readability_index

`textacy.text_stats.readability.`**`automatic_arabic_readability_index`**(*n_chars: int*, *n_words: int*, *n_sents: int*) → float

Readability test for Arabic-language texts based on number of characters and average word and sentence lengths. Higher value => more difficult text.

References

Al Tamimi, Abdel Karim, et al. "AARI: automatic arabic readability index." Int. Arab J. Inf. Technol. 11.4 (2014): 370-378.

`textacy.text_stats.readability.`**`coleman_liau_index`**(*n_chars: int*, *n_words: int*, *n_sents: int*) → float

Readability test whose value estimates the number of years of education required to understand a text, similar to *`flesch_kincaid_grade_level()`* and *`smog_index()`*, but using characters per word instead of syllables. Higher value => more difficult text.

References

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

textacy.text_stats.readability.**flesch_kincaid_grade_level**(*n_syllables: int*, *n_words: int*, *n_sents: int*) → float

Readability test used widely in education, whose value estimates the U.S. grade level / number of years of education required to understand a text. Higher value => more difficult text.

References

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

textacy.text_stats.readability.**flesch_reading_ease**(*n_syllables: int*, *n_words: int*, *n_sents: int*, *, *lang: Optional[str] = None*) → float

Readability test used as a general-purpose standard in several languages, based on a weighted combination of avg. sentence length and avg. word length. Values usually fall in the range [0, 100], but may be arbitrarily negative in extreme cases. Higher value => easier text.

Note: Coefficients in this formula are language-dependent; if `lang` is null, the English-language formulation is used.

References

English: https://en.wikipedia.org/wiki/Flesch%E2%80%93Kincaid_readability_tests#Flesch_reading_ease German: https://de.wikipedia.org/wiki/Lesbarkeitsindex#Flesch-Reading-Ease Spanish: Fernández-Huerta formulation French: ? Italian: https://it.wikipedia.org/wiki/Formula_di_Flesch Dutch: ? Portuguese: https://pt.wikipedia.org/wiki/Legibilidade_de_Flesch Turkish: Atesman formulation Russian: https://ru.wikipedia.org/wiki/%D0%98%D0%BD%D0%B4%D0%B5%D0%BA%D1%81_%D1%83%D0%B4%D0%BE%D0%B1%D0%BE%D1%87%D0%B8%D1%82%D0%B0%D0%B5%D0%BC%D0%BE%D1%81%D1%82%D0%B8

textacy.text_stats.readability.**gulpease_index**(*n_chars: int*, *n_words: int*, *n_sents: int*) → float

Readability test for Italian-language texts, whose value is in the range [0, 100] similar to *flesch_reading_ease()*. Higher value => easier text.

References

https://it.wikipedia.org/wiki/Indice_Gulpease

textacy.text_stats.readability.**gunning_fog_index**(*n_words: int*, *n_polysyllable_words: int*, *n_sents: int*) → float

Readability test whose value estimates the number of years of education required to understand a text, similar to *flesch_kincaid_grade_level()* and *smog_index()*. Higher value => more difficult text.

References

https://en.wikipedia.org/wiki/Gunning_fog_index

textacy.text_stats.readability.**lix**(*n_words: int*, *n_long_words: int*, *n_sents: int*) → float

Readability test commonly used in Sweden on both English- and non-English-language texts, whose value estimates the difficulty of reading a foreign text. Higher value => more difficult text.

References

https://en.wikipedia.org/wiki/Lix_(readability_test)

textacy.text_stats.readability.**mu_legibility_index**(*n_chars_per_word: Collection[int]*) → float

Readability test for Spanish-language texts based on number of words and the mean and variance of their lengths in characters, whose value is in the range [0, 100]. Higher value => easier text.

References

Muñoz, M., and J. Muñoz. "Legibilidad Mμ." Viña del Mar: CHL (2006).

textacy.text_stats.readability.**perspicuity_index**(*n_syllables: int*, *n_words: int*, *n_sents: int*) → float

Readability test for Spanish-language texts, whose value is in the range [0, 100]; very similar to the Spanish-specific formulation of *flesch_reading_ease()*, but included additionally since it's become a common readability standard. Higher value => easier text.

References

Pazos, Francisco Szigriszt. Sistemas predictivos de legibilidad del mensaje escrito: fórmula de perspicuidad. Universidad Complutense de Madrid, Servicio de Reprografía, 1993.

textacy.text_stats.readability.**smog_index**(*n_polysyllable_words: int*, *n_sents: int*) → float

Readability test commonly used in medical writing and the healthcare industry, whose value estimates the number of years of education required to understand a text similar to *flesch_kincaid_grade_level()* and intended as a substitute for *gunning_fog_index()*. Higher value => more difficult text.

References

https://en.wikipedia.org/wiki/SMOG

textacy.text_stats.readability.**wiener_sachtextformel**(*n_words: int*, *n_polysyllable_words: int*, *n_monosyllable_words: int*, *n_long_words: int*, *n_sents: int*, *, *variant: int = 1*) → float

Readability test for German-language texts, whose value estimates the grade level required to understand a text. Higher value => more difficult text.

References

https://de.wikipedia.org/wiki/Lesbarkeitsindex#Wiener_Sachtextformel

Pipeline Components

textacy.text_stats.components: Custom components to add to a spaCy language pipeline.

class textacy.text_stats.components.**TextStatsComponent**(*attrs: Optional[Union[str, Collection[str]]] = None*)

A custom component to be added to a spaCy language pipeline that computes one, some, or all text stats for a parsed doc and sets the values as custom attributes on a `spacy.tokens.Doc`.

Add the component to a pipeline, *after* the parser and any subsequent components that modify the tokens/sentences of the doc (to be safe, just put it last):

```
>>> en = spacy.load("en_core_web_sm")
>>> en.add_pipe("textacy_text_stats", last=True)
```

Process a text with the pipeline and access the custom attributes via spaCy's underscore syntax:

```
>>> doc = en(u"This is a test test someverylongword.")
>>> doc._.n_words
6
>>> doc._.flesch_reading_ease
73.84500000000001
```

Specify which attributes of the `textacy.text_stats.TextStats()` to add to processed documents:

```
>>> en = spacy.load("en_core_web_sm")
>>> en.add_pipe("textacy_text_stats", last=True, config={"attrs": "n_words"})
>>> doc = en(u"This is a test test someverylongword.")
>>> doc._.n_words
6
>>> doc._.flesch_reading_ease
AttributeError: [E046] Can't retrieve unregistered extension attribute 'flesch_
↪reading_ease'. Did you forget to call the `set_extension` method?
```

Parameters **attr** – If str, a single text stat to compute and set on a `Doc`; if Iterable[str], set multiple text stats; if None, *all* text stats are computed and set as extensions.

See also:

`textacy.text_stats.TextStats`

4.4.6 Document Similarity

edits.hamming	Compute the similarity between two strings using Hamming distance, which gives the number of characters at corresponding string indices that differ, including chars in the longer string that have no correspondents in the shorter.

continues on next page

Table 7 – continued from previous page

edits.levenshtein	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.
edits.jaro	Measure the similarity between two strings using Jaro (*not* Jaro-Winkler) distance, which searches for common characters while taking transpositions and string lengths into account.
edits.character_ngrams	Measure the similarity between two strings using a character ngrams similarity metric, in which strings are transformed into trigrams of alnum-only characters, vectorized and weighted by tf-idf, then compared by cosine similarity.
tokens.jaccard	Measure the similarity between two sequences of strings as sets using the Jaccard index.
tokens.sorensen_dice	Measure the similarity between two sequences of strings as sets using the Sørensen-Dice index, which is similar to the Jaccard index.
tokens.tversky	Measure the similarity between two sequences of strings as sets using the (symmetric) Tversky index, which is a generalization of Jaccard (`alpha=0.5, beta=2.0`) and Sørensen-Dice (`alpha=0.5, beta=1.0`).
tokens.cosine	Measure the similarity between two sequences of strings as sets using the Otsuka-Ochiai variation of cosine similarity (which is equivalent to the usual formulation when values are binary).
tokens.bag	Measure the similarity between two sequences of strings (*not* as sets) using the "bag distance" measure, which can be considered an approximation of edit distance.
sequences.matching_subsequences_ratio	Measure the similarity between two sequences of strings by finding contiguous matching subsequences without any "junk" elements and normalizing by the total number of elements.
hybrid.token_sort_ratio	Measure the similarity between two strings or sequences of strings using Levenshtein distance, only with non-alphanumeric characters removed and the ordering of tokens in each sorted before comparison.
hybrid.monge_elkan	Measure the similarity between two sequences of strings using the (symmetric) Monge-Elkan method, which takes the average of the maximum pairwise similarity between the tokens in each sequence as compared to those in the other sequence.

Edit-based Metrics

textacy.similarity.edits: Normalized similarity metrics built on edit-based algorithms that compute the number of operations (additions, subtractions, ...) needed to transform one string into another.

textacy.similarity.edits.**hamming**(*str1: str*, *str2: str*) → float

Compute the similarity between two strings using Hamming distance, which gives the number of characters at corresponding string indices that differ, including chars in the longer string that have no correspondents in the shorter.

Parameters

- **str1** –
- **str2** –

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

textacy.similarity.edits.**levenshtein**(*str1: str*, *str2: str*) → float

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** –
- **str2** –

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

textacy.similarity.edits.**jaro**(*str1: str*, *str2: str*) → float

Measure the similarity between two strings using Jaro (*not* Jaro-Winkler) distance, which searches for common characters while taking transpositions and string lengths into account.

Parameters

- **str1** –
- **str2** –

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

textacy.similarity.edits.**character_ngrams**(*str1: str*, *str2: str*) → float

Measure the similarity between two strings using a character ngrams similarity metric, in which strings are transformed into trigrams of alnum-only characters, vectorized and weighted by tf-idf, then compared by cosine similarity.

Parameters

- **str1** –
- **str2** –

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

---

Note: This method has been used in cross-lingual plagiarism detection and authorship attribution, and seems to work better on longer texts. At the very least, it is *slow* on shorter texts relative to the other similarity measures.

Token-based Metrics

textacy.similarity.edits: Normalized similarity metrics built on token-based algorithms that identify and count similar tokens between one sequence and another, and don't rely on the *ordering* of those tokens.

textacy.similarity.tokens.**jaccard**(*seq1: Iterable[str]*, *seq2: Iterable[str]*) → float

Measure the similarity between two sequences of strings as sets using the Jaccard index.

Parameters

- **seq1** –
- **seq2** –

Returns Similarity between `seq1` and `seq2` in the interval [0.0, 1.0], where larger values correspond to more similar sequences of strings

Reference: https://en.wikipedia.org/wiki/Jaccard_index

textacy.similarity.tokens.**sorensen_dice**(*seq1: Iterable[str]*, *seq2: Iterable[str]*) → float

Measure the similarity between two sequences of strings as sets using the Sørensen-Dice index, which is similar to the Jaccard index.

Parameters

- **seq1** –
- **seq2** –

Returns Similarity between `seq1` and `seq2` in the interval [0.0, 1.0], where larger values correspond to more similar sequences

Reference: https://en.wikipedia.org/wiki/S%C3%B8rensen%E2%80%93Dice_coefficient

textacy.similarity.tokens.**tversky**(*seq1: Iterable[str]*, *seq2: Iterable[str]*, *, *alpha: float = 1.0*, *beta: float = 1.0*) → float

Measure the similarity between two sequences of strings as sets using the (symmetric) Tversky index, which is a generalization of Jaccard (`alpha=0.5, beta=2.0`) and Sørensen-Dice (`alpha=0.5, beta=1.0`).

Parameters

- **seq1** –
- **seq2** –
- **alpha** –
- **beta** –

Returns Similarity between `seq1` and `seq2` in the interval [0.0, 1.0], where larger values correspond to more similar sequences

Reference: https://en.wikipedia.org/wiki/Tversky_index

textacy.similarity.tokens.**cosine**(*seq1: Iterable[str]*, *seq2: Iterable[str]*) → float

Measure the similarity between two sequences of strings as sets using the Otsuka-Ochiai variation of cosine similarity (which is equivalent to the usual formulation when values are binary).

Parameters

- **seq1** –
- **seq2** –

Returns Similarity between `seq1` and `seq2` in the interval [0.0, 1.0], where larger values correspond to more similar sequences

Reference: https://en.wikipedia.org/wiki/Cosine_similarity#Otsuka-Ochiai_coefficient

`textacy.similarity.tokens.`**`bag`**(*seq1: Iterable[str]*, *seq2: Iterable[str]*) → float

Measure the similarity between two sequences of strings (*not* as sets) using the "bag distance" measure, which can be considered an approximation of edit distance.

Parameters

- **seq1** –
- **seq2** –

Returns Similarity between `seq1` and `seq2` in the interval [0.0, 1.0], where larger values correspond to more similar sequences

Reference: Bartolini, Ilaria, Paolo Ciaccia, and Marco Patella. "String matching with metric trees using an approximate distance." International Symposium on String Processing and Information Retrieval. Springer, Berlin, Heidelberg, 2002.

Sequence-based Metrics

textacy.similarity.sequences: Normalized similarity metrics built on sequence-based algorithms that identify and measure the subsequences common to each.

`textacy.similarity.sequences.`**`matching_subsequences_ratio`**(*seq1: Sequence[str]*, *seq2: Sequence[str]*, ***kwargs*) → float

Measure the similarity between two sequences of strings by finding contiguous matching subsequences without any "junk" elements and normalizing by the total number of elements.

Parameters

- **seq1** –
- **seq2** –
- ****kwargs** – isjunk: Optional[Callable[str], bool] = None autojunk: bool = True

Returns Similarity between `seq1` and `seq2` in the interval [0.0, 1.0], where larger values correspond to more similar sequences of strings

Reference: https://docs.python.org/3/library/difflib.html#difflib.SequenceMatcher.ratio

Hybrid Metrics

textacy.similarity.hybrid: Normalized similarity metrics that combine edit-, token-, and/or sequence-based algorithms.

`textacy.similarity.hybrid.`**`token_sort_ratio`**(*s1: str | Sequence[str]*, *s2: str | Sequence[str]*) → float

Measure the similarity between two strings or sequences of strings using Levenshtein distance, only with non-alphanumeric characters removed and the ordering of tokens in each sorted before comparison.

Parameters

- **s1** –
- **s2** –

Returns Similarity between `s1` and `s2` in the interval [0.0, 1.0], where larger values correspond to more similar strings.

See also:

textacy.similarity.edits.levenshtein()

textacy.similarity.hybrid.**monge_elkan**(*seq1: Sequence[str], seq2: Sequence[str], sim_func: Callable[[str, str], float] = <function levenshtein>*) → float

Measure the similarity between two sequences of strings using the (symmetric) Monge-Elkan method, which takes the average of the maximum pairwise similarity between the tokens in each sequence as compared to those in the other sequence.

Parameters

- **seq1** –
- **seq2** –
- **sim_func** – Callable that computes a string-to-string similarity metric; by default, Levenshtein edit distance.

Returns Similarity between `seq1` and `seq2` in the interval [0.0, 1.0], where larger values correspond to more similar strings.

See also:

textacy.similarity.edits.levenshtein()

4.4.7 Document Representations

network.build_cooccurrence_network	Transform an ordered sequence of strings (or a sequence of such sequences) into a graph, where each string is represented by a node with weighted edges linking it to other strings that co-occur within `window_size` elements of itself.
network.build_similarity_network	Transform a sequence of strings (or a sequence of such sequences) into a graph, where each element of the top-level sequence is represented by a node with edges linking it to all other elements weighted by their pairwise similarity.
sparse_vec.build_doc_term_matrix	Transform one or more tokenized documents into a document-term matrix of shape (# docs, # unique terms), with flexible weighting/normalization of values.
sparse_vec.build_grp_term_matrix	Transform one or more tokenized documents into a group-term matrix of shape (# unique groups, # unique terms), with flexible weighting/normalization of values.
vectorizers.Vectorizer	Transform one or more tokenized documents into a sparse document-term matrix of shape (# docs, # unique terms), with flexible weighting/normalization of values.

continues on next page

Table 8 – continued from previous page

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

Networks

textacy.representations.network: Represent document data as networks, where nodes are terms, sentences, or even full documents and edges between them are weighted by the strength of their co-occurrence or similarity.

textacy.representations.network.**build_cooccurrence_network**(*data: Sequence[str] | Sequence[Sequence[str]], *, window_size: int = 2, edge_weighting: str = 'count'*) → nx.Graph

Transform an ordered sequence of strings (or a sequence of such sequences) into a graph, where each string is represented by a node with weighted edges linking it to other strings that co-occur within `window_size` elements of itself.

Input `data` can take a variety of forms. For example, as a `Sequence[str]` where elements are token or term strings from a single document:

```
>>> texts = [
...     "Mary had a little lamb. Its fleece was white as snow.",
...     "Everywhere that Mary went the lamb was sure to go.",
... ]
>>> docs = [make_spacy_doc(text, lang="en_core_web_sm") for text in texts]
>>> data = [tok.text for tok in docs[0]]
>>> graph = build_cooccurrence_network(data, window_size=2)
>>> sorted(graph.adjacency())[0]
('.', {'lamb': {'weight': 1}, 'Its': {'weight': 1}, 'snow': {'weight': 1}})
```

Or as a `Sequence[Sequence[str]]`, where elements are token or term strings per sentence from a single document:

```
>>> data = [[tok.text for tok in sent] for sent in docs[0].sents]
>>> graph = build_cooccurrence_network(data, window_size=2)
>>> sorted(graph.adjacency())[0]
('.', {'lamb': {'weight': 1}, 'snow': {'weight': 1}})
```

Or as a `Sequence[Sequence[str]]`, where elements are token or term strings per document from multiple documents:

```
>>> data = [[tok.text for tok in doc] for doc in docs]
>>> graph = build_cooccurrence_network(data, window_size=2)
>>> sorted(graph.adjacency())[0]
('.',
 {'lamb': {'weight': 1},
  'Its': {'weight': 1},
  'snow': {'weight': 1},
  'go': {'weight': 1}})
```

Note how the "." token's connections to other nodes change for each case. (Note that in real usage, you'll probably want to remove stopwords, punctuation, etc. so that nodes in the graph represent meaningful concepts.)

Parameters

- **data** –
- **window_size** – Size of sliding window over `data` that determines which strings are said to co-occur. For example, a value of 2 means that only immediately adjacent strings will have edges in the network; larger values loosen the definition of co-occurrence and typically lead to a more densely-connected network.

  Note: Co-occurrence windows are not permitted to cross sequences. So, if `data` is a `Sequence[Sequence[str]]`, then co-occ counts are computed separately for each sub-sequence, then summed together.

- **edge_weighting** – Method by which edges between nodes are weighted. If "count", nodes are connected by edges with weights equal to the number of times they co-occurred within a sliding window; if "binary", all such edges have weight set equal to 1.

Returns Graph whose nodes correspond to individual strings from `data`; those that co-occur are connected by edges with weights determined by `edge_weighting`.

Reference: https://en.wikipedia.org/wiki/Co-occurrence_network

`textacy.representations.network.`**build_similarity_network**(*data: Sequence[str] | Sequence[Sequence[str]]*, *edge_weighting: str*) → nx.Graph

Transform a sequence of strings (or a sequence of such sequences) into a graph, where each element of the top-level sequence is represented by a node with edges linking it to all other elements weighted by their pairwise similarity.

Input `data` can take a variety of forms. For example, as a `Sequence[str]` where elements are sentence texts from a single document:

```
>>> texts = [
...     "Mary had a little lamb. Its fleece was white as snow.",
...     "Everywhere that Mary went the lamb was sure to go.",
... ]
>>> docs = [make_spacy_doc(text, lang="en_core_web_sm") for text in texts]
>>> data = [sent.text.lower() for sent in docs[0].sents]
>>> graph = build_similarity_network(data, "levenshtein")
>>> sorted(graph.adjacency())[0]
('its fleece was white as snow.',
 {'mary had a little lamb.': {'weight': 0.24137931034482762}})
```

Or as a `Sequence[str]` where elements are full texts from multiple documents:

```
>>> data = [doc.text.lower() for doc in docs]
>>> graph = build_similarity_network(data, "jaro")
>>> sorted(graph.adjacency())[0]
('everywhere that mary went the lamb was sure to go.',
 {'mary had a little lamb. its fleece was white as snow.': {'weight': 0.
↪6516002795248078}})
```

Or as a `Sequence[Sequence[str]]` where elements are tokenized texts from multiple documents:

```
>>> data = [[tok.lower_ for tok in doc] for doc in docs]
>>> graph = build_similarity_network(data, "jaccard")
>>> sorted(graph.adjacency())[0]
(('everywhere', 'that', 'mary', 'went', 'the', 'lamb', 'was', 'sure', 'to', 'go',
↪'.'),
 {('mary', 'had', 'a', 'little', 'lamb', '.', 'its', 'fleece', 'was', 'white', 'as
↪', 'snow', '.'): {'weight': 0.21052631578947367}})
```

Parameters

- **data** –
- **edge_weighting** – Similarity metric to use for weighting edges between elements in `data`, represented as the name of a function available in `textacy.similarity`.

Note: Different metrics are suited for different forms and contexts of `data`. You'll have to decide which method makes sense. For example, when comparing a sequence of short strings, "levenshtein" is often a reasonable bet; when comparing a sequence of sequences of somewhat noisy strings (e.g. includes punctuation, cruft tokens), you might try "matching_subsequences_ratio" to help filter out the noise.

Returns Graph whose nodes correspond to top-level sequence elements in `data`, connected by edges to all other nodes with weights determined by their pairwise similarity.

Reference: https://en.wikipedia.org/wiki/Semantic_similarity_network – this is *not* the same as what's implemented here, but they're similar in spirit.

`textacy.representations.network.`**rank_nodes_by_pagerank**(*graph: networkx.classes.graph.Graph*, *weight: str = 'weight'*, ***kwargs*) → Dict[Any, float]

Rank nodes in `graph` using the Pagegrank algorithm.

Parameters

- **graph** –
- **weight** – Key in edge data that holds weights.
- ****kwargs** –

Returns Mapping of node object to Pagerank score.

`textacy.representations.network.`**rank_nodes_by_bestcoverage**(*graph: networkx.classes.graph.Graph*, *k: int*, *c: int = 1*, *alpha: float = 1.0*, *weight: str = 'weight'*) → Dict[Any, float]

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

Parameters

- **graph** –

- **k** – Number of results to return for top-k search.
- **c** – *l* parameter for *l*-step expansion; best if 1 or 2
- **alpha** – 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
- **weight** – Key in edge data that holds weights.

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

References

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

`textacy.representations.network.`**`rank_nodes_by_divrank`**(*graph: networkx.classes.graph.Graph*, *r: Optional[numpy.ndarray] = None*, *lambda_: float = 0.5*, *alpha: float = 0.5*) → Dict[str, float]

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

Parameters

- **graph** –
- **r** – The "personalization vector"; by default, `r = ones(1, n)/n`
- **lambda** – Float in [0.0, 1.0]
- **alpha** – Float in [0.0, 1.0] that controls the strength of self-links.

Returns Mapping of node to score ordered by descending divrank score

References

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

Sparse Vectors

textacy.representations.sparse_vec: Transform a collection of tokenized docs into a doc-term matrix of shape (# docs, # unique terms) or a group-term matrix of shape (# unique groups, # unique terms), with various ways to filter/limit included terms and flexible weighting/normalization schemes for their values.

Intended primarily as a simpler- and higher-level API for sparse vectorization of docs.

textacy.representations.sparse_vec.**build_doc_term_matrix**(*tokenized_docs: Iterable[Iterable[str]]*, ***, *tf_type: str = 'linear'*, *idf_type: Optional[str] = None*, *dl_type: Optional[str] = None*, ***kwargs*) → Tuple[scipy.sparse.csr.csr_matrix, Dict[str, int]]

Transform one or more tokenized documents into a document-term matrix of shape (# docs, # unique terms), with flexible weighting/normalization of values.

Parameters

- **tokenized_docs** – A sequence of tokenized documents, where each is a sequence of term strings. For example:

```
>>> ([tok.lemma_ for tok in spacy_doc]
...  for spacy_doc in spacy_docs)
>>> ((ne.text for ne in extract.entities(doc))
...  for doc in corpus)
```

- **tf_type** – Type of term frequency (tf) to use for weights' local component:
  - "linear": tf (tfs are already linear, so left as-is)
  - "sqrt": tf => sqrt(tf)
  - "log": tf => log(tf) + 1
  - "binary": tf => 1
- **idf_type** – Type of inverse document frequency (idf) to use for weights' global component:
  - "standard": idf = log(n_docs / df) + 1.0
  - "smooth": idf = log(n_docs + 1 / df + 1) + 1.0, i.e. 1 is added to all document frequencies, as if a single document containing every unique term was added to the corpus.
  - "bm25": idf = log((n_docs - df + 0.5) / (df + 0.5)), which is a form commonly used in information retrieval that allows for very common terms to receive negative weights.
  - None: no global weighting is applied to local term weights.
- **dl_type** – Type of document-length scaling to use for weights' normalization component:
  - "linear": dl (dls are already linear, so left as-is)
  - "sqrt": dl => sqrt(dl)
  - "log": dl => log(dl)
  - None: no normalization is applied to local (+global?) weights
- ****kwargs** – Passed directly into vectorizer class

Returns Document-term matrix as a sparse row matrix, and the corresponding mapping of term strings to integer ids (column indexes).

Note: If you need to transform other sequences of tokenized documents in the same way, or if you

need more access to the underlying vectorization process, consider using *textacy.representations.vectorizers.Vectorizer* directly.

See also:

- *textacy.representations.vectorizers.Vectorizer*
- scipy.sparse.csr_matrix

Reference: https://en.wikipedia.org/wiki/Document-term_matrix

textacy.representations.sparse_vec.**build_grp_term_matrix**(*tokenized_docs: Iterable[Iterable[str]]*, *grps: Iterable[str]*, ***, *tf_type: str = 'linear'*, *idf_type: Optional[str] = None*, *dl_type: Optional[str] = None*, ***kwargs*) → scipy.sparse.csr.csr_matrix

Transform one or more tokenized documents into a group-term matrix of shape (# unique groups, # unique terms), with flexible weighting/normalization of values.

This is an extension of typical document-term matrix vectorization, where terms are grouped by the documents in which they co-occur. It allows for customized grouping, such as by a shared author or publication year, that may span multiple documents, without forcing users to merge those documents themselves.

Parameters

- **tokenized_docs** – A sequence of tokenized documents, where each is a sequence of term strings. For example:

```
>>> ([tok.lemma_ for tok in spacy_doc]
...  for spacy_doc in spacy_docs)
>>> ((ne.text for ne in extract.entities(doc))
...  for doc in corpus)
```

- **grps** – Sequence of group names by which the terms in `tokenized_docs` are aggregated, where the first item in `grps` corresponds to the first item in `tokenized_docs`, and so on.
- **tf_type** – Type of term frequency (tf) to use for weights' local component:
  - "linear": tf (tfs are already linear, so left as-is)
  - "sqrt": tf => sqrt(tf)
  - "log": tf => log(tf) + 1
  - "binary": tf => 1
- **idf_type** – Type of inverse document frequency (idf) to use for weights' global component:
  - "standard": idf = log(n_docs / df) + 1.0
  - "smooth": idf = log(n_docs + 1 / df + 1) + 1.0, i.e. 1 is added to all document frequencies, as if a single document containing every unique term was added to the corpus.
  - "bm25": idf = log((n_docs - df + 0.5) / (df + 0.5)), which is a form commonly used in information retrieval that allows for very common terms to receive negative weights.

  - None: no global weighting is applied to local term weights.
- **dl_type** – Type of document-length scaling to use for weights' normalization component:
  - "linear": dl (dls are already linear, so left as-is)
  - "sqrt": dl => sqrt(dl)
  - "log": dl => log(dl)
  - None: no normalization is applied to local (+global?) weights
- ****kwargs** – Passed directly into vectorizer class

Returns Group-term matrix as a sparse row matrix, and the corresponding mapping of term strings to integer ids (column indexes), and the corresponding mapping of group strings to integer ids (row indexes).

Note: If you need to transform other sequences of tokenized documents in the same way, or if you need more access to the underlying vectorization process, consider using *textacy.representations.vectorizers.GroupVectorizer* directly.

See also:

- *textacy.representations.vectorizers.GroupVectorizer*
- scipy.sparse.csr_matrix

Reference: https://en.wikipedia.org/wiki/Document-term_matrix

Vectorizers

textacy.representations.vectorizers: Transform a collection of tokenized docs into a doc-term matrix of shape (# docs, # unique terms), with various ways to filter or limit included terms and flexible weighting schemes for their values.

A second option aggregates terms in tokenized documents by provided group labels, resulting in a "group-term-matrix" of shape (# unique groups, # unique terms), with filtering and weighting functionality as described above.

See the *Vectorizer* and *GroupVectorizer* docstrings for usage examples and explanations of the various weighting schemes.

class textacy.representations.vectorizers.**Vectorizer**(*, *tf_type: str = 'linear'*, *idf_type: Optional[str] = None*, *dl_type: Optional[str] = None*, *norm: Optional[str] = None*, *min_df: int | float = 1*, *max_df: int | float = 1.0*, *max_n_terms: Optional[int] = None*, *vocabulary_terms: Optional[Dict[str, int] | Iterable[str]] = None*)

Transform one or more tokenized documents into a sparse document-term matrix of shape (# docs, # unique terms), with flexible weighting/normalization of values.

Stream a corpus with metadata from disk:

```
>>> ds = textacy.datasets.CapitolWords()
>>> records = ds.records(limit=1000)
>>> corpus = textacy.Corpus("en_core_web_sm", data=records)
>>> print(corpus)
Corpus(1000 docs, 538397 tokens)
```

Tokenize and vectorize the first 600 documents of this corpus:

```
>>> tokenized_docs = (
...     (term.lemma_ for term in textacy.extract.terms(doc, ngs=1, ents=True))
...     for doc in corpus[:600])
>>> vectorizer = Vectorizer(
...     tf_type="linear", idf_type="smooth", norm="l2",
...     min_df=3, max_df=0.95)
>>> doc_term_matrix = vectorizer.fit_transform(tokenized_docs)
>>> doc_term_matrix
<600x4412 sparse matrix of type '<class 'numpy.float64'>'
        with 65210 stored elements in Compressed Sparse Row format>
```

Tokenize and vectorize the remaining 400 documents of the corpus, using only the groups, terms, and weights learned in the previous step:

```
>>> tokenized_docs = (
...     (term.lemma_ for term in textacy.extract.terms(doc, ngs=1, ents=True))
...     for doc in corpus[600:])
>>> doc_term_matrix = vectorizer.transform(tokenized_docs)
>>> doc_term_matrix
<400x4412 sparse matrix of type '<class 'numpy.float64'>'
        with 36212 stored elements in Compressed Sparse Row format>
```

Inspect the terms associated with columns; they're sorted alphabetically:

```
>>> vectorizer.terms_list[:5]
['', '$', '$ 1 million', '$ 1.2 billion', '$ 10 billion']
```

(Btw: That empty string shouldn't be there. Somehow, spaCy is labeling it as a named entity... ?)

If known in advance, limit the terms included in vectorized outputs to a particular set of values:

```
>>> tokenized_docs = (
...     (term.lemma_ for term in textacy.extract.terms(doc, ngs=1, ents=True))
...     for doc in corpus[:600])
>>> vectorizer = Vectorizer(
...     idf_type="smooth", norm="l2",
...     min_df=3, max_df=0.95,
...     vocabulary_terms=["president", "bill", "unanimous", "distinguished",
↪"american"])
>>> doc_term_matrix = vectorizer.fit_transform(tokenized_docs)
>>> doc_term_matrix
<600x5 sparse matrix of type '<class 'numpy.float64'>'
        with 516 stored elements in Compressed Sparse Row format>
>>> vectorizer.terms_list
['american', 'bill', 'distinguished', 'president', 'unanimous']
```

Specify different weighting schemes to determine values in the matrix, adding or customizing individual components, as desired:

```
>>> tokenized_docs = [
    [term.lemma_ for term in textacy.extract.terms(doc, ngs=1, ents=True)]
    for doc in corpus[:600]]
>>> doc_term_matrix = Vectorizer(
...     tf_type="linear", norm=None, min_df=3, max_df=0.95
... ).fit_transform(tokenized_docs)
>>> print(doc_term_matrix[:8, vectorizer.vocabulary_terms["$"]].toarray())
[[0]
 [0]
 [1]
 [4]
 [0]
 [0]
 [2]
 [4]]
>>> doc_term_matrix = Vectorizer(
...     tf_type="sqrt", dl_type="sqrt", norm=None, min_df=3, max_df=0.95
... ).fit_transform(tokenized_docs)
>>> print(doc_term_matrix[:8, vectorizer.vocabulary_terms["$"]].toarray())
[[0.        ]
 [0.        ]
 [0.10660036]
 [0.2773501 ]
 [0.        ]
 [0.        ]
 [0.11704115]
 [0.24806947]]
>>> doc_term_matrix = Vectorizer(
...     tf_type="bm25", idf_type="smooth", norm=None, min_df=3, max_df=0.95
... ).fit_transform(tokenized_docs)
>>> print(doc_term_matrix[:8, vectorizer.vocabulary_terms["$"]].toarray())
[[0.        ]
 [0.        ]
 [2.68009606]
 [4.97732126]
 [0.        ]
 [0.        ]
 [3.87124987]
 [4.97732126]]
```

If you're not sure what's going on mathematically, *Vectorizer.weighting* gives the formula being used to calculate weights, based on the parameters set when initializing the vectorizer:

```
>>> vectorizer.weighting
'(tf * (k + 1)) / (k + tf) * log((n_docs + 1) / (df + 1)) + 1'
```

In general, weights may consist of a local component (term frequency), a global component (inverse document frequency), and a normalization component (document length). Individual components may be modified: they may have different scaling (e.g. tf vs. sqrt(tf)) or different behaviors (e.g. "standard" idf vs bm25's version). There are *many* possible weightings, and some may be better for particular use cases than others. When in doubt, though, just go with something standard.

- "tf": Weights are simply the absolute per-document term frequencies (tfs), i.e. value (i, j) in an output doc-term matrix corresponds to the number of occurrences of term j in doc i. Terms appearing many times in a given doc receive higher weights than less common terms. Params: `tf_type="linear", apply_idf=False, apply_dl=False`
- "tfidf": Doc-specific, *local* tfs are multiplied by their corpus-wide, *global* inverse document frequen-

cies (idfs). Terms appearing in many docs have higher document frequencies (dfs), correspondingly smaller idfs, and in turn, lower weights. Params: `tf_type="linear", apply_idf=True, idf_type="smooth", apply_dl=False`

- "bm25": This scheme includes a local tf component that increases asymptotically, so higher tfs have diminishing effects on the overall weight; a global idf component that can go *negative* for terms that appear in a sufficiently high proportion of docs; as well as a row-wise normalization that accounts for document length, such that terms in shorter docs hit the tf asymptote sooner than those in longer docs. Params: `tf_type="bm25", apply_idf=True, idf_type="bm25", apply_dl=True`
- "binary": This weighting scheme simply replaces all non-zero tfs with 1, indicating the presence or absence of a term in a particular doc. That's it. Params: `tf_type="binary", apply_idf=False, apply_dl=False`

Slightly altered versions of these "standard" weighting schemes are common, and may have better behavior in general use cases:

- "lucene-style tfidf": Adds a doc-length normalization to the usual local and global components. Params: `tf_type="linear", apply_idf=True, idf_type="smooth", apply_dl=True, dl_type="sqrt"`
- "lucene-style bm25": Uses a smoothed idf instead of the classic bm25 variant to prevent weights on terms from going negative. Params: `tf_type="bm25", apply_idf=True, idf_type="smooth", apply_dl=True, dl_type="linear"`

Parameters

- **`tf_type`** – Type of term frequency (tf) to use for weights' local component:
  - "linear": tf (tfs are already linear, so left as-is)
  - "sqrt": tf => sqrt(tf)
  - "log": tf => log(tf) + 1
  - "binary": tf => 1
- **`idf_type`** – Type of inverse document frequency (idf) to use for weights' global component:
  - "standard": idf = log(n_docs / df) + 1.0
  - "smooth": idf = log(n_docs + 1 / df + 1) + 1.0, i.e. 1 is added to all document frequencies, as if a single document containing every unique term was added to the corpus.
  - "bm25": idf = log((n_docs - df + 0.5) / (df + 0.5)), which is a form commonly used in information retrieval that allows for very common terms to receive negative weights.
  - None: no global weighting is applied to local term weights.
- **`dl_type`** – Type of document-length scaling to use for weights' normalization component:
  - "linear": dl (dls are already linear, so left as-is)
  - "sqrt": dl => sqrt(dl)
  - "log": dl => log(dl)
  - None: no normalization is applied to local (+global?) weights
- **`norm`** – If "l1" or "l2", normalize weights by the L1 or L2 norms, respectively, of row-wise vectors; otherwise, don't.

- **min_df** – Minimum number of documents in which a term must appear for it to be included in the vocabulary and as a column in a transformed doc-term matrix. If float, value is the fractional proportion of the total number of docs, which must be in [0.0, 1.0]; if int, value is the absolute number.
- **max_df** – Maximum number of documents in which a term may appear for it to be included in the vocabulary and as a column in a transformed doc-term matrix. If float, value is the fractional proportion of the total number of docs, which must be in [0.0, 1.0]; if int, value is the absolute number.
- **max_n_terms** – If specified, only include terms whose document frequency is within the top `max_n_terms`.
- **vocabulary_terms** – Mapping of unique term string to unique term id, or an iterable of term strings that gets converted into such a mapping. Note that, if specified, vectorized outputs will include *only* these terms.

vocabulary_terms
Mapping of unique term string to unique term id, either provided on instantiation or generated by calling *Vectorizer.fit()* on a collection of tokenized documents.

Type Dict[str, int]

property id_to_term
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.

property terms_list
List of term strings in column order of vectorized outputs. For example, `terms_list[0]` gives the term assigned to the first column in an output doc-term-matrix, `doc_term_matrix[:, 0]`.

fit(*tokenized_docs: Iterable[Iterable[str]]*) → *Vectorizer*
Count terms in `tokenized_docs` and, if not already provided, build up a vocabulary based those terms. Fit and store global weights (IDFs) and, if needed for term weighting, the average document length.

Parameters **tokenized_docs** – A sequence of tokenized documents, where each is a sequence of term strings. For example:

```
>>> ([tok.lemma_ for tok in spacy_doc]
...  for spacy_doc in spacy_docs)
>>> ((ne.text for ne in extract.entities(doc))
...  for doc in corpus)
```

Returns Vectorizer instance that has just been fit.

fit_transform(*tokenized_docs: Iterable[Iterable[str]]*) → scipy.sparse.csr.csr_matrix
Count terms in `tokenized_docs` and, if not already provided, build up a vocabulary based those terms. Fit and store global weights (IDFs) and, if needed for term weighting, the average document length. Transform `tokenized_docs` into a document-term matrix with values weighted according to the parameters in *Vectorizer* initialization.

Parameters **tokenized_docs** – A sequence of tokenized documents, where each is a sequence of term strings. For example:

```
>>> ([tok.lemma_ for tok in spacy_doc]
...  for spacy_doc in spacy_docs)
>>> ((ne.text for ne in extract.entities(doc))
...  for doc in corpus)
```

Returns The transformed document-term matrix, where rows correspond to documents and columns correspond to terms, as a sparse row matrix.

transform(*tokenized_docs: Iterable[Iterable[str]]*) → scipy.sparse.csr.csr_matrix

Transform `tokenized_docs` into a document-term matrix with values weighted according to the parameters in *Vectorizer* initialization and the global weights computed by calling *Vectorizer.fit()*.

Parameters **tokenized_docs** – A sequence of tokenized documents, where each is a sequence of term strings. For example:

```
>>> ([tok.lemma_ for tok in spacy_doc]
...  for spacy_doc in spacy_docs)
>>> ((ne.text for ne in extract.entities(doc))
...  for doc in corpus)
```

Returns The transformed document-term matrix, where rows correspond to documents and columns correspond to terms, as a sparse row matrix.

Note: For best results, the tokenization used to produce `tokenized_docs` should be the same as was applied to the docs used in fitting this vectorizer or in generating a fixed input vocabulary.

Consider an extreme case where the docs used in fitting consist of lowercased (non-numeric) terms, while the docs to be transformed are all uppercased: The output doc-term-matrix will be empty.

---

property weighting

A mathematical representation of the overall weighting scheme used to determine values in the vectorized matrix, depending on the params used to initialize the *Vectorizer*.

class textacy.representations.vectorizers.**GroupVectorizer**(*, *tf_type: str = 'linear'*, *idf_type: Optional[str] = None*, *dl_type: Optional[str] = None*, *norm: Optional[str] = None*, *min_df: int | float = 1*, *max_df: int | float = 1.0*, *max_n_terms: Optional[int] = None*, *vocabulary_terms: Optional[Dict[str, int] | Iterable[str]] = None*, *vocabulary_grps: Optional[Dict[str, int] | Iterable[str]] = None*)

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

This is an extension of typical document-term matrix vectorization, where terms are grouped by the documents in which they co-occur. It allows for customized grouping, such as by a shared author or publication year, that may span multiple documents, without forcing users to merge those documents themselves.

Stream a corpus with metadata from disk:

```
>>> ds = textacy.datasets.CapitolWords()
>>> records = ds.records(limit=1000)
>>> corpus = textacy.Corpus("en_core_web_sm", data=records)
```

(continues on next page)

(continued from previous page)

```
>>> corpus
Corpus(1000 docs, 538397 tokens)
```

Tokenize and vectorize the first 600 documents of this corpus, where terms are grouped not by documents but by a categorical value in the docs' metadata:

```
>>> tokenized_docs, groups = textacy.io.unzip(
...     ((term.lemma_ for term in textacy.extract.terms(doc, ngs=1, ents=True)),
...      doc._.meta["speaker_name"])
...     for doc in corpus[:600])
>>> vectorizer = GroupVectorizer(
...     tf_type="linear", idf_type="smooth", norm="l2",
...     min_df=3, max_df=0.95)
>>> grp_term_matrix = vectorizer.fit_transform(tokenized_docs, groups)
>>> grp_term_matrix
<5x1822 sparse matrix of type '<class 'numpy.float64'>'
        with 6139 stored elements in Compressed Sparse Row format>
```

Tokenize and vectorize the remaining 400 documents of the corpus, using only the groups, terms, and weights learned in the previous step:

```
>>> tokenized_docs, groups = textacy.io.unzip(
...     ((term.lemma_ for term in textacy.extract.terms(doc, ngs=1, ents=True)),
...      doc._.meta["speaker_name"])
...     for doc in corpus[600:])
>>> grp_term_matrix = vectorizer.transform(tokenized_docs, groups)
>>> grp_term_matrix
<5x1822 sparse matrix of type '<class 'numpy.float64'>'
        with 4414 stored elements in Compressed Sparse Row format>
```

Inspect the terms associated with columns and groups associated with rows; they're sorted alphabetically:

```
>>> vectorizer.terms_list[:5]
['', '$ 1 million', '$ 160 million', '$ 5 billion', '$ 7 billion']
>>> vectorizer.grps_list
['Bernie Sanders', 'John Kasich', 'Joseph Biden', 'Lindsey Graham', 'Rick Santorum
↪']
```

If known in advance, limit the terms and/or groups included in vectorized outputs to a particular set of values:

```
>>> tokenized_docs, groups = textacy.io.unzip(
...     ((term.lemma_ for term in textacy.extract.terms(doc, ngs=1, ents=True)),
...      doc._.meta["speaker_name"])
...     for doc in corpus[:600])
>>> vectorizer = GroupVectorizer(
...     tf_type="linear", idf_type="smooth", norm="l2",
...     min_df=3, max_df=0.95,
...     vocabulary_terms=["legislation", "federal government", "house",
↪"constitutional"],
...     vocabulary_grps=["Bernie Sanders", "Lindsey Graham", "Rick Santorum"])
>>> grp_term_matrix = vectorizer.fit_transform(tokenized_docs, groups)
>>> grp_term_matrix
<3x4 sparse matrix of type '<class 'numpy.float64'>'
        with 9 stored elements in Compressed Sparse Row format>
>>> vectorizer.terms_list
['constitutional', 'federal government', 'house', 'legislation']
```

(continues on next page)

(continued from previous page)

```
>>> vectorizer.grps_list
['Bernie Sanders', 'Lindsey Graham', 'Rick Santorum']
```

For a discussion of the various weighting schemes that can be applied, check out the *Vectorizer* docstring.

Parameters

- **tf_type** – Type of term frequency (tf) to use for weights' local component:
  - "linear": tf (tfs are already linear, so left as-is)
  - "sqrt": tf => sqrt(tf)
  - "log": tf => log(tf) + 1
  - "binary": tf => 1
- **idf_type** – Type of inverse document frequency (idf) to use for weights' global component:
  - "standard": idf = log(n_docs / df) + 1.0
  - "smooth": idf = log(n_docs + 1 / df + 1) + 1.0, i.e. 1 is added to all document frequencies, as if a single document containing every unique term was added to the corpus.
  - "bm25": idf = log((n_docs - df + 0.5) / (df + 0.5)), which is a form commonly used in information retrieval that allows for very common terms to receive negative weights.
  - None: no global weighting is applied to local term weights.
- **dl_type** – Type of document-length scaling to use for weights' normalization component:
  - "linear": dl (dls are already linear, so left as-is)
  - "sqrt": dl => sqrt(dl)
  - "log": dl => log(dl)
  - None: no normalization is applied to local (+global?) weights
- **norm** – If "l1" or "l2", normalize weights by the L1 or L2 norms, respectively, of row-wise vectors; otherwise, don't.
- **min_df** – Minimum number of documents in which a term must appear for it to be included in the vocabulary and as a column in a transformed doc-term matrix. If float, value is the fractional proportion of the total number of docs, which must be in [0.0, 1.0]; if int, value is the absolute number.
- **max_df** – Maximum number of documents in which a term may appear for it to be included in the vocabulary and as a column in a transformed doc-term matrix. If float, value is the fractional proportion of the total number of docs, which must be in [0.0, 1.0]; if int, value is the absolute number.
- **max_n_terms** – If specified, only include terms whose document frequency is within the top `max_n_terms`.
- **vocabulary_terms** – Mapping of unique term string to unique term id, or an iterable of term strings that gets converted into such a mapping. Note that, if specified, vectorized output will include *only* these terms.
- **vocabulary_grps** – Mapping of unique group string to unique group id, or an iterable of group strings that gets converted into such a mapping. Note that, if specified, vectorized output will include *only* these groups.

vocabulary_terms
Mapping of unique term string to unique term id, either provided on instantiation or generated by calling *GroupVectorizer.fit()* on a collection of tokenized documents.

Type Dict[str, int]

vocabulary_grps
Mapping of unique group string to unique group id, either provided on instantiation or generated by calling *GroupVectorizer.fit()* on a collection of tokenized documents.

Type Dict[str, int]

See also:

Vectorizer

property id_to_grp
Mapping of unique group id (int) to unique group string (str), i.e. the inverse of *GroupVectorizer.vocabulary_grps*. This attribute is only generated if needed, and it is automatically kept in sync with the corresponding vocabulary.

property grps_list
List of group strings in row order of vectorized outputs. For example, `grps_list[0]` gives the group assigned to the first row in an output group-term-matrix, `grp_term_matrix[0, :]`.

fit(*tokenized_docs: Iterable[Iterable[str]]*, *grps: Iterable[str]*) → *GroupVectorizer*
Count terms in `tokenized_docs` and, if not already provided, build up a vocabulary based those terms; do the same for the groups in `grps`. Fit and store global weights (IDFs) and, if needed for term weighting, the average document length.

Parameters

- **tokenized_docs** – A sequence of tokenized documents, where each is a sequence of term strings. For example:

```
>>> ([tok.lemma_ for tok in spacy_doc]
...  for spacy_doc in spacy_docs)
>>> ((ne.text for ne in extract.entities(doc))
...  for doc in corpus)
```

- **grps** – Sequence of group names by which the terms in `tokenized_docs` are aggregated, where the first item in `grps` corresponds to the first item in `tokenized_docs`, and so on.

Returns GroupVectorizer instance that has just been fit.

fit_transform(*tokenized_docs: Iterable[Iterable[str]]*, *grps: Iterable[str]*) → scipy.sparse.csr.csr_matrix
Count terms in `tokenized_docs` and, if not already provided, build up a vocabulary based those terms; do the same for the groups in `grps`. Fit and store global weights (IDFs) and, if needed for term weighting, the average document length. Transform `tokenized_docs` into a group-term matrix with values weighted according to the parameters in *GroupVectorizer* initialization.

Parameters

- **tokenized_docs** – A sequence of tokenized documents, where each is a sequence of term strings. For example:

```
>>> ([tok.lemma_ for tok in spacy_doc]
...  for spacy_doc in spacy_docs)
>>> ((ne.text for ne in extract.entities(doc))
...  for doc in corpus)
```

- **grps** – Sequence of group names by which the terms in `tokenized_docs` are aggregated, where the first item in `grps` corresponds to the first item in `tokenized_docs`, and so on.

Returns The transformed group-term matrix, where rows correspond to groups and columns correspond to terms, as a sparse row matrix.

transform(*tokenized_docs: Iterable[Iterable[str]]*, *grps: Iterable[str]*) → scipy.sparse.csr.csr_matrix

Transform `tokenized_docs` and `grps` into a group-term matrix with values weighted according to the parameters in *GroupVectorizer* initialization and the global weights computed by calling *GroupVectorizer.fit()*.

Parameters

- **tokenized_docs** – A sequence of tokenized documents, where each is a sequence of term strings. For example:

```
>>> ([tok.lemma_ for tok in spacy_doc]
...  for spacy_doc in spacy_docs)
>>> ((ne.text for ne in extract.entities(doc))
...  for doc in corpus)
```

- **grps** – Sequence of group names by which the terms in `tokenized_docs` are aggregated, where the first item in `grps` corresponds to the first item in `tokenized_docs`, and so on.

Returns The transformed group-term matrix, where rows correspond to groups and columns correspond to terms, as a sparse row matrix.

Note: For best results, the tokenization used to produce `tokenized_docs` should be the same as was applied to the docs used in fitting this vectorizer or in generating a fixed input vocabulary.

Consider an extreme case where the docs used in fitting consist of lowercased (non-numeric) terms, while the docs to be transformed are all uppercased: The output group-term-matrix will be empty.

`textacy.vsm.matrix_utils`: Functions for computing corpus-wide term- or document-based values, like term frequency, document frequency, and document length, and filtering terms from a matrix by their document frequency.

`textacy.representations.matrix_utils.`**get_term_freqs**(*doc_term_matrix*, *, *type_='linear'*)

Compute frequencies for all terms in a document-term matrix, with optional sub-linear scaling.

Parameters

- **doc_term_matrix** (`scipy.sparse.csr_matrix`) – M x N sparse matrix, where M is the # of docs and N is the # of unique terms. Values must be the linear, un-scaled counts of term n per doc m.
- **type** (*{'linear', 'sqrt', 'log'}*) – Scaling applied to absolute term counts. If 'linear', term counts are left as-is, since the sums are already linear; if 'sqrt', tf => sqrt(tf); if 'log', tf => log(tf) + 1.

Returns Array of term frequencies, with length equal to the # of unique terms (# of columns) in `doc_term_matrix`.

Return type `numpy.ndarray`

Raises **ValueError** – if `doc_term_matrix` doesn't have any non-zero entries, or if `type_` isn't one of {"linear", "sqrt", "log"}.

textacy.representations.matrix_utils.**get_doc_freqs**(*doc_term_matrix*)

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

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

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

Returns Array of document frequencies, with length equal to the # of unique terms (# of columns) in doc_term_matrix.

Return type numpy.ndarray

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

textacy.representations.matrix_utils.**get_inverse_doc_freqs**(*doc_term_matrix*, *, *type_='smooth'*)

Compute inverse document frequencies for all terms in a document-term matrix, using one of several IDF formulations.

Parameters

- **doc_term_matrix** (scipy.sparse.csr_matrix) – M x N sparse matrix, where M is the # of docs and N is the # of unique terms. The particular weighting of matrix values doesn't matter.
- **type** (*{'standard', 'smooth', 'bm25'}*) – Type of IDF formulation to use. If 'standard', idfs => log(n_docs / dfs) + 1.0; if 'smooth', idfs => log(n_docs + 1 / dfs + 1) + 1.0, i.e. 1 is added to all document frequencies, equivalent to adding a single document to the corpus containing every unique term; if 'bm25', idfs => log((n_docs - dfs + 0.5) / (dfs + 0.5)), which is a form commonly used in BM25 ranking that allows for extremely common terms to have negative idf weights.

Returns Array of inverse document frequencies, with length equal to the # of unique terms (# of columns) in doc_term_matrix.

Return type numpy.ndarray

Raises **ValueError** – if type_ isn't one of {"standard", "smooth", "bm25"}.

textacy.representations.matrix_utils.**get_doc_lengths**(*doc_term_matrix*, *, *type_='linear'*)

Compute the lengths (i.e. number of terms) for all documents in a document-term matrix.

Parameters

- **doc_term_matrix** (scipy.sparse.csr_matrix) – M x N sparse matrix, where M is the # of docs, N is the # of unique terms, and values are the absolute counts of term n per doc m.
- **type** (*{'linear', 'sqrt', 'log'}*) – Scaling applied to absolute doc lengths. If 'linear', lengths are left as-is, since the sums are already linear; if 'sqrt', dl => sqrt(dl); if 'log', dl => log(dl) + 1.

Returns Array of document lengths, with length equal to the # of documents (# of rows) in doc_term_matrix.

Return type numpy.ndarray

Raises **ValueError** – if type_ isn't one of {"linear", "sqrt", "log"}.

textacy.representations.matrix_utils.**get_information_content**(*doc_term_matrix*)

Compute information content for all terms in a document-term 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) – M x N sparse matrix, where M is the # of docs and N is the # of unique terms.

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

Returns Array of term information content values, with length equal to the # of unique terms (# of columns) in `doc_term_matrix`.

Return type numpy.ndarray

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

textacy.representations.matrix_utils.**apply_idf_weighting**(*doc_term_matrix*, *, *type_='smooth'*)

Apply inverse document frequency (idf) weighting to a term-frequency (tf) weighted document-term matrix, using one of several IDF formulations.

Parameters

- **doc_term_matrix** (scipy.sparse.csr_matrix) – M x N sparse matrix, where M is the # of docs and N is the # of unique terms.
- **type** (*{'standard', 'smooth', 'bm25'}*) – Type of IDF formulation to use.

Returns Sparse matrix of shape M x N, where value (i, j) is the tfidf weight of term j in doc i.

Return type scipy.sparse.csr_matrix

See also:

get_inverse_doc_freqs()

textacy.representations.matrix_utils.**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_terms`.
- **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[str, int]: Term to id mapping, where keys are unique *filtered* terms as strings and values are their corresponding integer ids.

Return type scipy.sparse.csr_matrix

Raises **ValueError** – if max_df or min_df or max_n_terms < 0.

textacy.representations.matrix_utils.**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 sparse 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_terms.
- **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[str, int]: Term to id mapping, where keys are unique *filtered* terms as strings and values are their corresponding integer ids.

Return type scipy.sparse.csr_matrix

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

4.4.8 Topic Modeling

textacy.tm.topic_model: 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. Also any other topic model implementations that have *component_*, *n_topics* and *transform* attributes. 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.representations.vectorizers.Vectorizer* for details. In short:

```
>>> vectorizer = Vectorizer(
...     tf_type="linear", idf_type="smooth", norm="l2",
...     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]._.meta["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]._.meta["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). 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`

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)))]
```

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 *TopicModel.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_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 *TopicModel.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)))]
```

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 *`TopicModel.get_doc_topic_matrix()`*

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

Return type `numpy.ndarray`

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

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** (`numpy.ndarray` 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 rc_params (dict, optional): allow passing parameters to rc_context in matplotlib.plyplot, details in https://matplotlib.org/3.1.0/api/_as_gen/matplotlib.pyplot.rc_context.html

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

- Chuang, Jason, Christopher D. Manning, and Jeffrey Heer. "Termite: Visualization techniques for assessing textual topic models." Proceedings of the International Working Conference on Advanced Visual Interfaces. ACM, 2012.
- for sorting by "seriation", see https://arxiv.org/abs/1406.5370

See also:

`viz.termite_plot`

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

4.4.9 File I/O

text.read_text	Read the contents of a text file at `filepath`, either all at once or streaming line-by-line.
text.write_text	Write text `data` to disk at `filepath`, either all at once or streaming line-by-line.
json.read_json	Read the contents of a JSON file at `filepath`, either all at once or streaming item-by-item.
json.write_json	Write JSON `data` to disk at `filepath`, either all at once or streaming item-by-item.
csv.read_csv	Read the contents of a CSV file at `filepath`, streaming line-by-line, where each line is a list of strings and/or floats whose values are separated by `delimiter`.
csv.write_csv	Write rows of `data` to disk at `filepath`, where each row is an iterable or a dictionary of strings and/or numbers, written to one line with values separated by `delimiter`.
matrix.read_sparse_matrix	Read the data, indices, indptr, and shape arrays from a `.npz` file on disk at `filepath`, and return an instantiated sparse matrix.
matrix.write_sparse_matrix	Write sparse matrix `data` to disk at `filepath`, optionally compressed, into a single `.npz` file.
spacy.read_spacy_docs	Read the contents of a file at `filepath`, written in binary or pickle format.
spacy.write_spacy_docs	Write one or more `Doc` s to disk at `filepath` in binary or pickle format.
http.read_http_stream	Read data from `url` in a stream, either all at once or line-by-line.
http.write_http_stream	Download data from `url` in a stream, and write successive chunks to disk at `filepath`.
utils.open_sesame	Open file `filepath`.
utils.split_records	Split records' content (text) from associated metadata, but keep them paired together.

continues on next page

Table 9 – continued from previous page

utils.unzip	Borrowed from `toolz.sandbox.core.unzip`, but using cytoolz instead of toolz to avoid the additional dependency.
utils.get_filepaths	Yield full paths of files on disk under directory `dirpath`, optionally filtering for or against particular patterns or file extensions and crawling all subdirectories.
utils.download_file	Download a file from `url` and save it to disk.
utils.unpack_archive	Extract data from a zip or tar archive file into a directory (or do nothing if the file isn't an archive).

textacy.io.text: Functions for reading from and writing to disk records in plain text format, either as one text per file or one text per *line* in a file.

`textacy.io.text.`**`read_text`**(*filepath: Union[str, pathlib.Path]*, ***, *mode: str = 'rt'*, *encoding: Optional[str] = None*, *lines: bool = False*) → Iterable[str]

Read the contents of a text file at `filepath`, either all at once or streaming line-by-line.

Parameters

- **`filepath`** – Path to file on disk from which data will be read.
- **`mode`** – Mode with which `filepath` is opened.
- **`encoding`** – Name of the encoding used to decode or encode the data in `filepath`. Only applicable in text mode.
- **`lines`** – If False, all data is read in at once; otherwise, data is read in one line at a time.

Yields Next line of text to read in.

If `lines` is False, wrap this output in `next()` to conveniently access the full text.

`textacy.io.text.`**`write_text`**(*data: str | Iterable[str]*, *filepath: types.PathLike*, ***, *mode: str = 'wt'*, *encoding: Optional[str] = None*, *make_dirs: bool = False*, *lines: bool = False*) → None

Write text `data` to disk at `filepath`, either all at once or streaming line-by-line.

Parameters

- **`If lines is False`** (*data*) – "isnt rick and morty that thing you get when you die and your body gets all stiff"

  If `lines` is True, an iterable of strings to write to disk, one item per line; for example:

```
["isnt rick and morty that thing you get when you die and your␣
↪body gets all stiff",
 "You're thinking of rigor mortis. Rick and morty is when you get␣
↪trolled into watching "never gonna give you up"",
 "That's rickrolling. Rick and morty is a type of pasta"]
```

- **`single string to write to disk; for example::`** (*a*) – "isnt rick and morty that thing you get when you die and your body gets all stiff"

  If `lines` is True, an iterable of strings to write to disk, one item per line; for example:

```
["isnt rick and morty that thing you get when you die and your␣
↪body gets all stiff",
 "You're thinking of rigor mortis. Rick and morty is when you get␣
↪trolled into watching "never gonna give you up"",
```

(continues on next page)

(continued from previous page)

```
"That's rickrolling. Rick and morty is a type of pasta"]
```

- **filepath** – Path to file on disk to which data will be written.
- **mode** – Mode with which `filepath` is opened.
- **encoding** – Name of the encoding used to decode or encode the data in `filepath`. Only applicable in text mode.
- **make_dirs** – If True, automatically create (sub)directories if not already present in order to write `filepath`.
- **lines** – If False, all data is written at once; otherwise, data is written to disk one line at a time.

textacy.io.json: Functions for reading from and writing to disk records in JSON format, as one record per file or one record per *line* in a file.

textacy.io.json.**read_json**(*filepath: Union[str, pathlib.Path]*, *, *mode: str = 'rt'*, *encoding: Optional[str] = None*, *lines: bool = False*) → Iterable

Read the contents of a JSON file at `filepath`, either all at once or streaming item-by-item.

Parameters

- **filepath** – Path to file on disk from which data will be read.
- **mode** – Mode with which `filepath` is opened.
- **encoding** – Name of the encoding used to decode or encode the data in `filepath`. Only applicable in text mode.
- **lines** – If False, all data is read in at once; otherwise, data is read in one line at a time.

Yields Next JSON item; could be a dict, list, int, float, str, depending on the data and the value of `lines`.

textacy.io.json.**read_json_mash**(*filepath: Union[str, pathlib.Path]*, *, *mode: str = 'rt'*, *encoding: Optional[str] = None*, *buffer_size: int = 2048*) → Iterable

Read the contents of a JSON file at `filepath` one item at a time, where all of the items have been mashed together, end-to-end, on a single line.

Parameters

- **filepath** – Path to file on disk to which data will be written.
- **mode** – Mode with which `filepath` is opened.
- **encoding** – Name of the encoding used to decode or encode the data in `filepath`. Only applicable in text mode.
- **buffer_size** – Number of bytes to read in as a chunk.

Yields Next valid JSON object, converted to native Python equivalent.

Note: Storing JSON data in this format is Not Good. Reading it is doable, so this function is included for users' convenience, but note that there is no analogous `write_json_mash()` function. Don't do it.

---

textacy.io.json.**write_json**(*data: Any*, *filepath: types.PathLike*, *, *mode: str = 'wt'*, *encoding: Optional[str] = None*, *make_dirs: bool = False*, *lines: bool = False*, *ensure_ascii: bool = False*, *separators: Tuple[str, str] = (',', ':')*, *sort_keys: bool = False*, *indent: Optional[int | str] = None*) → None

Write JSON `data` to disk at `filepath`, either all at once or streaming item-by-item.

Parameters

- **data** – JSON data to write to disk, including any Python objects encodable by default in `json`, as well as dates and datetimes. For example:

```
[
    {"title": "Harrison Bergeron", "text": "The year was 2081, and␣
↪everybody was finally equal."},
    {"title": "2BR02B", "text": "Everything was perfectly swell."},
    {"title": "Slaughterhouse-Five", "text": "All this happened,␣
↪more or less."},
]
```

  If `lines` is False, all of `data` is written as a single object; if True, each item is written to a separate line in `filepath`.

- **filepath** – Path to file on disk to which data will be written.
- **mode** – Mode with which `filepath` is opened.
- **encoding** – Name of the encoding used to decode or encode the data in `filepath`. Only applicable in text mode.
- **make_dirs** – If True, automatically create (sub)directories if not already present in order to write `filepath`.
- **lines** – If False, all data is written at once; otherwise, data is written to disk one item at a time.
- **ensure_ascii** – If True, all non-ASCII characters are escaped; otherwise, non-ASCII characters are output as-is.
- **separators** – An (item_separator, key_separator) pair specifying how items and keys are separated in output.
- **sort_keys** – If True, each output dictionary is sorted by key; otherwise, dictionary ordering is taken as-is.
- **indent** – If a non-negative integer or string, items are pretty-printed with the specified indent level; if 0, negative, or "", items are separated by newlines; if None, the most compact representation is used when storing `data`.

See also:

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

class `textacy.io.json.`**ExtendedJSONEncoder**(*, *skipkeys=False*, *ensure_ascii=True*, *check_circular=True*, *allow_nan=True*, *sort_keys=False*, *indent=None*, *separators=None*, *default=None*)

Sub-class of `json.JSONEncoder`, used to write JSON data to disk in *`write_json()`* while handling a broader range of Python objects.

- `datetime.datetime` => ISO-formatted string
- `datetime.date` => ISO-formatted string

default(*obj*)

Implement this method in a subclass such that it returns a serializable object for `o`, or calls the base implementation (to raise a `TypeError`).

For example, to support arbitrary iterators, you could implement default like this:

```
def default(self, o):
    try:
        iterable = iter(o)
    except TypeError:
        pass
    else:
        return list(iterable)
    # Let the base class default method raise the TypeError
    return JSONEncoder.default(self, o)
```

textacy.io.csv: Functions for reading from and writing to disk records in CSV format, where CSVs may be delimited not only by commas (the default) but tabs, pipes, and other valid one-char delimiters.

textacy.io.csv.**read_csv**(*filepath: types.PathLike*, ***, *encoding: Optional[str] = None*, *fieldnames: Optional[str | Sequence[str]] = None*, *dialect: str | Type[csv.Dialect] = 'excel'*, *delimiter: str = ','*, *quoting: int = 2*) → Iterable[list] | Iterable[dict]

Read the contents of a CSV file at `filepath`, streaming line-by-line, where each line is a list of strings and/or floats whose values are separated by `delimiter`.

Parameters

- **filepath** – Path to file on disk from which data will be read.
- **encoding** – Name of the encoding used to decode or encode the data in `filepath`.
- **fieldnames** – If specified, gives names for columns of values, which are used as keys in an ordered dictionary representation of each line's data. If 'infer', the first kB of data is analyzed to make a guess about whether the first row is a header of column names, and if so, those names are used as keys. If None, no column names are used, and each line is returned as a list of strings/floats.
- **dialect** – Grouping of formatting parameters that determine how the data is parsed when reading/writing. If 'infer', the first kB of data is analyzed to get a best guess for the correct dialect.
- **delimiter** – 1-character string used to separate fields in a row.
- **quoting** – Type of quoting to apply to field values. See: https://docs.python.org/3/library/csv.html#csv.QUOTE_NONNUMERIC

Yields *List[obj]* – Next row, whose elements are strings and/or floats. If `fieldnames` is None or 'infer' doesn't detect a header row.

or

Dict[str, obj]: Next row, as an ordered dictionary of (key, value) pairs, where keys are column names and values are the corresponding strings and/or floats. If `fieldnames` is a list of column names or 'infer' detects a header row.

See also:

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

textacy.io.csv.**write_csv**(*data: Iterable[Dict[str, Any]] | Iterable[Iterable]*, *filepath: types.PathLike*, ***, *encoding: Optional[str] = None*, *make_dirs: bool = False*, *fieldnames: Optional[Sequence[str]] = None*, *dialect: str = 'excel'*, *delimiter: str = ','*, *quoting: int = 2*) → None

Write rows of `data` to disk at `filepath`, where each row is an iterable or a dictionary of strings and/or numbers, written to one line with values separated by `delimiter`.

Parameters

- **data** – If `fieldnames` is None, an 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]]
```

  If `fieldnames` is specified, an iterable of dictionaries with string and/or number values to write to disk; for example:

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

- **filepath** – Path to file on disk to which data will be written.
- **encoding** – Name of the encoding used to decode or encode the data in `filepath`.
- **make_dirs** – If True, automatically create (sub)directories if not already present in order to write `filepath`.
- **fieldnames** – Sequence of keys that identify the order in which values in each rows' dictionary is written to `filepath`. These are included in `filepath` as a header row of column names.

  Note: Only specify this if `data` is an iterable of dictionaries.

- **dialect** – Grouping of formatting parameters that determine how the data is parsed when reading/writing.
- **delimiter** – 1-character string used to separate fields in a row.
- **quoting** – Type of quoting to apply to field values. See: https://docs.python.org/3/library/csv.html#csv.QUOTE_NONNUMERIC

See also:

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

textacy.io.matrix: Functions for reading from and writing to disk CSC and CSR sparse matrices in numpy binary format.

textacy.io.matrix.**read_sparse_matrix**(*filepath: types.PathLike*, *, *kind: str = 'csc'*) → sp.csc_matrix | sp.csr_matrix

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

Parameters

- **filepath** – Path to file on disk from which data will be read.
- **kind** (*{'csc', 'csr'}*) – Kind of sparse matrix to instantiate.

Returns An instantiated sparse matrix, whose type depends on the value of `kind`.

See also:

https://docs.scipy.org/doc/numpy-1.13.0/reference/routines.io.html#numpy-binary-files-npy-npz

textacy.io.matrix.**write_sparse_matrix**(*data: sp.csc_matrix | sp.csr_matrix*, *filepath: types.PathLike*, ***, *compressed: bool = True*, *make_dirs: bool = False*) → None

Write sparse matrix `data` to disk at `filepath`, optionally compressed, into a single `.npz` file.

Parameters

- **data** –
- **filepath** – Path to file on disk to which data will be written. If `filepath` does not end in `.npz`, that extension is automatically appended to the name.
- **compressed** – If True, save arrays into a single file in compressed numpy binary format.
- **make_dirs** – If True, automatically create (sub)directories if not already present in order to write `filepath`.

See also:

https://docs.scipy.org/doc/numpy-1.13.0/reference/routines.io.html#numpy-binary-files-npy-npz

textacy.io.spacy: Functions for reading from and writing to disk spacy documents in either pickle or binary format. Be warned: Both formats have pros and cons.

textacy.io.spacy.**read_spacy_docs**(*filepath: Union[str, pathlib.Path]*, ***, *format: str = 'binary'*, *lang: Optional[Union[str, pathlib.Path, spacy.language.Language]] = None*) → Iterable[spacy.tokens.doc.Doc]

Read the contents of a file at `filepath`, written in binary or pickle format.

Parameters

- **filepath** – Path to file on disk from which data will be read.
- **format** (*{"binary", "pickle"}*) – Format of the data that was written to disk. If "binary", uses `spacy.tokens.DocBin` to deserialie data; if "pickle", uses python's stdlib `pickle`.

  Warning: Docs written in pickle format were saved all together as a list, which means they're all loaded into memory at once before streaming one by one. Mind your RAM usage, especially when reading many docs!

- **lang** – Language with which spaCy originally processed docs, represented as the full name of or path on disk to the pipeline, or an already instantiated pipeline instance. Note that this is only required when `format` is "binary".

Yields Next deserialized document.

Raises **ValueError** – if format is not "binary" or "pickle", or if `lang` is None when `format="binary"`

textacy.io.spacy.**write_spacy_docs**(*data: Doc | Iterable[Doc]*, *filepath: types.PathLike*, ***, *make_dirs: bool = False*, *format: str = 'binary'*, *attrs: Optional[Iterable[str]] = None*, *store_user_data: bool = False*) → None

Write one or more `Doc` s to disk at `filepath` in binary or pickle format.

Parameters

- **data** – A single `Doc` or a sequence of `Doc` s to write to disk.
- **filepath** – Path to file on disk to which data will be written.

- **make_dirs** – If True, automatically create (sub)directories if not already present in order to write `filepath`.
- **format** (*{"pickle", "binary"}*) – Format of the data written to disk. If "binary", uses `spacy.tokens.DocBin` to serialie data; if "pickle", uses python's stdlib `pickle`.

  Warning: When writing docs in pickle format, all the docs in `data` must be saved as a list, which means they're all loaded into memory. Mind your RAM usage, especially when writing many docs!

- **attrs** – List of attributes to serialize if `format` is "binary". If None, spaCy's default values are used; see here: https://spacy.io/api/docbin#init
- **store_user_data** – If True, write :attr`Doc.user_data` and the values of custom extension attributes to disk; otherwise, don't.

Raises **ValueError** – if format is not "binary" or "pickle"

textacy.io.http: Functions for reading data from URLs via streaming HTTP requests and either reading it into memory or writing it directly to disk.

textacy.io.http.**read_http_stream**(*url: str*, ***, *lines: bool = False*, *decode_unicode: bool = False*, *chunk_size: int = 1024*, *auth: Optional[Tuple[str, str]] = None*) → Iterable[str] | Iterable[bytes]

Read data from `url` in a stream, either all at once or line-by-line.

Parameters

- **url** – URL to which a GET request is made for data.
- **lines** – If False, yield all of the data at once; otherwise, yield data line-by-line.
- **decode_unicode** – If True, yield data as unicode, where the encoding is taken from the HTTP response headers; otherwise, yield bytes.
- **chunk_size** – Number of bytes read into memory per chunk. Because decoding may occur, this is not necessarily the length of each chunk.
- **auth** – (username, password) pair for simple HTTP authentication required (if at all) to access the data at `url`.

  See also:

  http://docs.python-requests.org/en/master/user/authentication/

Yields If `lines` is True, the next line in the response data, which is bytes if `decode_unicode` is False or unicode otherwise. If `lines` is False, yields the full response content, either as bytes or unicode.

textacy.io.http.**write_http_stream**(*url: str*, *filepath: Union[str, pathlib.Path]*, ***, *mode: str = 'wt'*, *encoding: Optional[str] = None*, *make_dirs: bool = False*, *chunk_size: int = 1024*, *auth: Optional[Tuple[str, str]] = None*) → None

Download data from `url` in a stream, and write successive chunks to disk at `filepath`.

Parameters

- **url** – URL to which a GET request is made for data.
- **filepath** – Path to file on disk to which data will be written.
- **mode** – Mode with which `filepath` is opened.

- **encoding** – Name of the encoding used to decode or encode the data in `filepath`. Only applicable in text mode.

  Note: The encoding on the HTTP response is inferred from its headers, or set to 'utf-8' as a fall-back in the case that no encoding is detected. It is *not* set by `encoding`.

- **make_dirs** – If True, automatically create (sub)directories if not already present in order to write `filepath`.
- **chunk_size** – Number of bytes read into memory per chunk. Because decoding may occur, this is not necessarily the length of each chunk.
- **auth** – (username, password) pair for simple HTTP authentication required (if at all) to access the data at `url`.

  See also:

  http://docs.python-requests.org/en/master/user/authentication/

I/O Utils

textacy.io.utils: Functions to help read and write data to disk in a variety of formats.

textacy.io.utils.**open_sesame**(*filepath: Union[str, pathlib.Path]*, ***, *mode: str = 'rt'*, *encoding: Optional[str] = None*, *errors: Optional[str] = None*, *newline: Optional[str] = None*, *compression: str = 'infer'*, *make_dirs: bool = False*) → IO

Open file `filepath`. Automatically handle file compression, relative paths and symlinks, and missing intermediate directory creation, as needed.

`open_sesame` may be used as a drop-in replacement for `io.open()`.

Parameters

- **filepath** – Path on disk (absolute or relative) of the file to open.
- **mode** – The mode in which `filepath` is opened.
- **encoding** – Name of the encoding used to decode or encode `filepath`. Only applicable in text mode.
- **errors** – String specifying how encoding/decoding errors are handled. Only applicable in text mode.
- **newline** – String specifying how universal newlines mode works. Only applicable in text mode.
- **compression** – Type of compression, if any, with which `filepath` is read from or written to disk. If None, no compression is used; if 'infer', compression is inferred from the extension on `filepath`.
- **make_dirs** – If True, automatically create (sub)directories if not already present in order to write `filepath`.

Returns file object

Raises

- **TypeError** – if `filepath` is not a string
- **ValueError** – if `encoding` is specified but `mode` is binary

- **OSError** – if `filepath` doesn't exist but `mode` is read

`textacy.io.utils.`**`coerce_content_type`**(*content: str | bytes*, *file_mode: str*) → str | bytes

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.io.utils.`**`split_records`**(*items: Iterable*, *content_field: str | int*, *itemwise: bool = False*) → Iterable

Split records' content (text) from associated metadata, but keep them paired together.

Parameters

- **items** – An iterable of dicts, e.g. as read from disk by *`read_json(lines=True)`*, or an iterable of lists, e.g. as read from disk by *`read_csv()`*.
- **content_field** – 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** – 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 Iterable[dict]; 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 Iterable[list]; 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 Iterable[dict]; 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 Iterable[list]; 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.io.utils.`**`unzip`**(*seq: Iterable*) → Tuple

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

`textacy.io.utils.`**`get_filepaths`**(*dirpath: Union[str, pathlib.Path]*, ***, *match_regex: Optional[str] = None*, *ignore_regex: Optional[str] = None*, *extension: Optional[str] = None*, *ignore_invisible: bool = True*, *recursive: bool = False*) → Iterable[str]

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

Parameters

- **dirpath** – Path to directory on disk where files are stored.
- **match_regex** – Regular expression pattern. Only files whose names match this pattern are included.
- **ignore_regex** – Regular expression pattern. Only files whose names *do not* match this pattern are included.
- **extension** – File extension, e.g. ".txt" or ".json". Only files whose extensions match are included.

- **ignore_invisible** – If True, ignore invisible files, i.e. those that begin with a period.; otherwise, include them.
- **recursive** – If True, iterate recursively through subdirectories in search of files to include; otherwise, only return files located directly under `dirpath`.

Yields Next file's name, including the full path on disk.

Raises **OSError** – if `dirpath` is not found on disk

textacy.io.utils.**download_file**(*url: str*, ***, *filename: Optional[str] = None*, *dirpath: Union[str, pathlib.Path] = PosixPath('/home/docs/checkouts/readthedocs.org/user_builds/textacy/envs/latest/lib/python3.7/site-packages/textacy/data')*, *force: bool = False*) → Optional[str]

Download a file from `url` and save it to disk.

Parameters

- **url** – Web address from which to download data.
- **filename** – Name of the file to which downloaded data is saved. If None, a filename will be inferred from the `url`.
- **dirpath** – Full path to the directory on disk under which downloaded data will be saved as `filename`.
- **force** – If True, download the data even if it already exists at `dirpath/filename`; otherwise, only download if the data doesn't already exist on disk.

Returns Full path of file saved to disk.

textacy.io.utils.**get_filename_from_url**(*url: str*) → str

Derive a filename from a URL's path.

Parameters **url** – URL from which to extract a filename.

Returns Filename in URL.

textacy.io.utils.**unpack_archive**(*filepath: Union[str, pathlib.Path]*, ***, *extract_dir: Optional[Union[str, pathlib.Path]] = None*) → Union[str, pathlib.Path]

Extract data from a zip or tar archive file into a directory (or do nothing if the file isn't an archive).

Parameters

- **filepath** – Full path to file on disk from which archived contents will be extracted.
- **extract_dir** – Full path of the directory into which contents will be extracted. If not provided, the same directory as `filepath` is used.

Returns Path to directory of extracted contents.

4.4.10 Visualization

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

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
- **rc_params** (*dict, optional*) – allow passing parameters to rc_context in matplotlib.plyplot, details in https://matplotlib.org/3.1.0/api/_as_gen/matplotlib.pyplot.rc_context.html

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

Chuang, Jason, Christopher D. Manning, and Jeffrey Heer. "Termite: Visualization techniques for assessing textual topic models." Proceedings of the International Working Conference on Advanced Visual Interfaces. ACM, 2012.

See also:

TopicModel.termite_plot()

textacy.viz.termite.**termite_df_plot**(*components*, ***, *highlight_topics=None*, *n_terms=25*, *rank_terms_by='max'*, *sort_terms_by='seriation'*, *save=False*, *rc_params=None*)

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

Parameters

- **components** (pandas.DataFrame or sparse matrix) – corpus represented as a term-topic matrix with shape (n_terms, n_topics); should have terms as index and topics as column names
- **topics** (*int or Sequence[int]*) – topic(s) to include in termite plot; if -1, all topics are included
- **highlight_topics** (*str or Sequence[str]*) – names 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** (*{'max', 'mean', 'var'}*) – argument to dataframe *agg* function, 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 rc_params (dict, optional): allow passing parameters to rc_context in matplotlib.plyplot, details in https://matplotlib.org/3.1.0/api/_as_gen/matplotlib.pyplot.rc_context.html

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

- **Chuang, Jason, Christopher D. Manning, and Jeffrey Heer. "Termite:** Visualization techniques for assessing textual topic models." Proceedings of the International Working Conference on Advanced Visual Interfaces. ACM, 2012.
- **Fajwel Fogel, Alexandre d'Aspremont, and Milan Vojnovic. 2016.** Spectral ranking using seriation. J. Mach. Learn. Res. 17, 1 (January 2016), 3013–3057.

See also:

`viz.termite_plot`

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

`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

Note: This function requires matplotlib.

4.4.11 Data Augmentation

augmenter.Augmenter	Randomly apply one or many data augmentation transforms to spaCy Doc s to produce new docs with additional variety and/or noise in the data.
transforms.substitute_word_synonyms	Randomly substitute words for which synonyms are available with a randomly selected synonym, up to num times or with a probability of num.
transforms.insert_word_synonyms	Randomly insert random synonyms of tokens for which synonyms are available, up to num times or with a probability of num.
transforms.swap_words	Randomly swap the positions of two *adjacent* words, up to num times or with a probability of num.
transforms.delete_words	Randomly delete words, up to num times or with a probability of num.
transforms.substitute_chars	Randomly substitute a single character in randomly-selected words with another, up to num times or with a probability of num.
transforms.insert_chars	Randomly insert a character into randomly-selected words, up to num times or with a probability of num.
transforms.swap_chars	Randomly swap two *adjacent* characters in randomly-selected words, up to num times or with a probability of num.
transforms.delete_chars	Randomly delete a character in randomly-selected words, up to num times or with a probability of num.
utils.to_aug_toks	Transform a spaCy Doc or Span into a list of AugTok objects, suitable for use in data augmentation transform functions.
utils.get_char_weights	Get lang-specific character weights for use in certain data augmentation transforms, based on texts in textacy.datasets.UDHR.

class textacy.augmentation.augmenter.**Augmenter**(*transforms: Sequence[AugTransform]*, **, num: Optional[int | float | Sequence[float]] = None*)

Randomly apply one or many data augmentation transforms to spaCy Doc s to produce new docs with additional variety and/or noise in the data.

Initialize an Augmenter with multiple transforms, and customize the randomization of their selection when applying to a document:

```
>>> tfs = [transforms.delete_words, transforms.swap_chars, transforms.delete_
↪chars]
>>> Augmenter(tfs, num=None)  # all tfs applied each time
>>> Augmenter(tfs, num=1)  # one randomly-selected tf applied each time
```

(continues on next page)

(continued from previous page)

```
>>> Augmenter(tfs, num=0.5)  # tfs randomly selected with 50% prob each time
>>> augmenter = Augmenter(tfs, num=[0.4, 0.8, 0.6])  # tfs randomly selected with␣
↪40%, 80%, 60% probs, respectively, each time
```

Apply transforms to a given `Doc` to produce new documents:

```
>>> text = "The quick brown fox jumps over the lazy dog."
>>> doc = textacy.make_spacy_doc(text, lang="en_core_web_sm")
>>> augmenter.apply_transforms(doc, lang="en_core_web_sm")
The quick brown ox jupms over the lazy dog.
>>> augmenter.apply_transforms(doc, lang="en_core_web_sm")
The quikc brown fox over the lazy dog.
>>> augmenter.apply_transforms(doc, lang="en_core_web_sm")
quick brown fox jumps over teh lazy dog.
```

Parameters for individual transforms may be specified when initializing `Augmenter` or, if necessary, when applying to individual documents:

```
>>> from functools import partial
>>> tfs = [partial(transforms.delete_words, num=3), transforms.swap_chars]
>>> augmenter = Augmenter(tfs)
>>> augmenter.apply_transforms(doc, lang="en_core_web_sm")
brown fox jumps over layz dog.
>>> augmenter.apply_transforms(doc, lang="en_core_web_sm", pos={"NOUN", "ADJ"})
The jumps over the lazy odg.
```

Parameters

- **transforms** – Ordered sequence of callables that must take List[`AugTok`] as their first positional argument and return another List[`AugTok`].

  Note: Although the particular transforms applied may vary doc-by-doc, they are applied *in order* as listed here. Since some transforms may clobber text in a way that makes other transforms less effective, a stable ordering can improve the quality of augmented data.

- **num** – If int, number of transforms to randomly select from `transforms` each time `Augmenter.apply_tranforms()` is called. If float, probability that any given transform will be selected. If Sequence[float], the probability that the corresponding transform in `transforms` will be selected (these must be the same length). If None (default), num is set to `len(transforms)`, which means that every transform is applied each time.

See also:

A collection of general-purpose transforms are implemented in *textacy.augmentation.transforms*.

apply_transforms(*doc: spacy.tokens.doc.Doc*, *lang: Union[str, pathlib.Path, spacy.language.Language]*, ***kwargs*) → spacy.tokens.doc.Doc

Sequentially apply some subset of data augmentation transforms to `doc`, then return a new `Doc` created from the augmented text using `lang`.

Parameters

- **doc** –
- **lang** –

- ****kwargs** – If, for whatever reason, you have to pass keyword argument values into transforms that vary or depend on characteristics of `doc`, specify them here. The transforms' call signatures will be inspected, and values will be passed along, as needed.

Returns `spacy.tokens.Doc`

`textacy.augmentation.transforms.`**`substitute_word_synonyms`**(*aug_toks: List[aug_utils.AugTok]*, ***, *num: int | float = 1*, *pos: Optional[str | Set[str]] = None*) → List[aug_utils.AugTok]

Randomly substitute words for which synonyms are available with a randomly selected synonym, up to `num` times or with a probability of `num`.

Parameters

- **aug_toks** – Sequence of tokens to augment through synonym substitution.
- **num** – If int, maximum number of words with available synonyms to substitute with a randomly selected synonym; if float, probability that a given word with synonyms will be substituted.
- **pos** – Part of speech tag(s) of words to be considered for augmentation. If None, all words with synonyms are considered.

Returns New, augmented sequence of tokens.

Note: This transform requires `textacy.resources.ConceptNet` to be downloaded to work properly, since this is the data source for word synonyms to be substituted.

`textacy.augmentation.transforms.`**`insert_word_synonyms`**(*aug_toks: List[aug_utils.AugTok]*, ***, *num: int | float = 1*, *pos: Optional[str | Set[str]] = None*) → List[aug_utils.AugTok]

Randomly insert random synonyms of tokens for which synonyms are available, up to `num` times or with a probability of `num`.

Parameters

- **aug_toks** – Sequence of tokens to augment through synonym insertion.
- **num** – If int, maximum number of words with available synonyms from which a random synonym is selected and randomly inserted; if float, probability that a given word with synonyms will provide a synonym to be inserted.
- **pos** – Part of speech tag(s) of words to be considered for augmentation. If None, all words with synonyms are considered.

Returns New, augmented sequence of tokens.

Note: This transform requires `textacy.resources.ConceptNet` to be downloaded to work properly, since this is the data source for word synonyms to be inserted.

`textacy.augmentation.transforms.`**`swap_words`**(*aug_toks: List[aug_utils.AugTok]*, ***, *num: int | float = 1*, *pos: Optional[str | Set[str]] = None*) → List[aug_utils.AugTok]

Randomly swap the positions of two *adjacent* words, up to `num` times or with a probability of `num`.

Parameters

- **aug_toks** – Sequence of tokens to augment through position swapping.
- **num** – If int, maximum number of adjacent word pairs to swap; if float, probability that a given word pair will be swapped.
- **pos** – Part of speech tag(s) of words to be considered for augmentation. If None, all words are considered.

Returns New, augmented sequence of tokens.

`textacy.augmentation.transforms.`**delete_words**(*aug_toks: List[aug_utils.AugTok]*, ***, *num: int | float = 1*, *pos: Optional[str | Set[str]] = None*) → List[aug_utils.AugTok]

Randomly delete words, up to `num` times or with a probability of `num`.

Parameters

- **aug_toks** – Sequence of tokens to augment through word deletion.
- **num** – If int, maximum number of words to delete; if float, probability that a given word will be deleted.
- **pos** – Part of speech tag(s) of words to be considered for augmentation. If None, all words are considered.

Returns New, augmented sequence of tokens.

`textacy.augmentation.transforms.`**substitute_chars**(*aug_toks: List[aug_utils.AugTok]*, ***, *num: int | float = 1*, *lang: Optional[str] = None*) → List[aug_utils.AugTok]

Randomly substitute a single character in randomly-selected words with another, up to `num` times or with a probability of `num`.

Parameters

- **aug_toks** – Sequence of tokens to augment through character substitution.
- **num** – If int, maximum number of words to modify with a random character substitution; if float, probability that a given word will be modified.
- **lang** – Standard, two-letter language code corresponding to `aug_toks`. Used to load a weighted distribution of language-appropriate characters that are randomly selected for substitution. More common characters are more likely to be substituted. If not specified, ascii letters and digits are randomly selected with equal probability.

Returns New, augmented sequence of tokens.

Note: This transform requires `textacy.datasets.UDHR` to be downloaded to work properly, since this is the data source for character weights when deciding which char(s) to insert.

`textacy.augmentation.transforms.`**insert_chars**(*aug_toks: List[aug_utils.AugTok]*, ***, *num: int | float = 1*, *lang: Optional[str] = None*) → List[aug_utils.AugTok]

Randomly insert a character into randomly-selected words, up to `num` times or with a probability of `num`.

Parameters

- **aug_toks** – Sequence of tokens to augment through character insertion.

- **num** – If int, maximum number of words to modify with a random character insertion; if float, probability that a given word will be modified.
- **lang** – Standard, two-letter language code corresponding to `aug_toks`. Used to load a weighted distribution of language-appropriate characters that are randomly selected for substitution. More common characters are more likely to be substituted. If not specified, ascii letters and digits are randomly selected with equal probability.

Returns New, augmented sequence of tokens.

Note: This transform requires `textacy.datasets.UDHR` to be downloaded to work properly, since this is the data source for character weights when deciding which char(s) to insert.

`textacy.augmentation.transforms.`**`swap_chars`**(*aug_toks: List[aug_utils.AugTok]*, ***, *num: int | float = 1*) → List[aug_utils.AugTok]

Randomly swap two *adjacent* characters in randomly-selected words, up to `num` times or with a probability of `num`.

Parameters

- **aug_toks** – Sequence of tokens to augment through character swapping.
- **num** – If int, maximum number of words to modify with a random character swap; if float, probability that a given word will be modified.

Returns New, augmented sequence of tokens.

`textacy.augmentation.transforms.`**`delete_chars`**(*aug_toks: List[aug_utils.AugTok]*, ***, *num: int | float = 1*) → List[aug_utils.AugTok]

Randomly delete a character in randomly-selected words, up to `num` times or with a probability of `num`.

Parameters

- **aug_toks** – Sequence of tokens to augment through character deletion.
- **num** – If int, maximum number of words to modify with a random character deletion; if float, probability that a given word will be modified.

Returns New, augmented sequence of tokens.

class `textacy.augmentation.utils.`**`AugTok`**(*text*, *ws*, *pos*, *is_word*, *syns*)

tuple: Minimal token data required for data augmentation transforms.

is_word
Alias for field number 3

pos
Alias for field number 2

syns
Alias for field number 4

text
Alias for field number 0

ws
Alias for field number 1

`textacy.augmentation.utils.`**`to_aug_toks`**(*doclike: Union[spacy.tokens.doc.Doc, spacy.tokens.span.Span]*) → List[*textacy.augmentation.utils.AugTok*]

Transform a spaCy `Doc` or `Span` into a list of `AugTok` objects, suitable for use in data augmentation transform

functions.

textacy.augmentation.utils.**get_char_weights**(*lang: str*) → List[Tuple[str, int]]

Get lang-specific character weights for use in certain data augmentation transforms, based on texts in `textacy.datasets.UDHR`.

Parameters **lang** – Standard two-letter language code.

Returns Collection of (character, weight) pairs, based on the distribution of characters found in the source text.

4.4.12 Miscellany

lang_id.lang_identifier.identify_lang	Identify the most probable language identified in `text`, with or without the corresponding probability.
lang_id.lang_identifier.identify_topn_langs	Identify the `topn` most probable languages identified in `text`, with or without the corresponding probabilities.
utils.get_config	Get key configuration info about dev environment: OS, python, spacy, and textacy.
utils.print_markdown	Print `items` as a markdown-formatted list.
utils.is_record	Check whether `obj` is a "record" – that is, a (text, metadata) 2-tuple.
utils.to_collection	Validate and cast a value or values to a collection.
utils.to_bytes	Coerce string `s` to bytes.
utils.to_unicode	Coerce string `s` to unicode.
utils.to_path	Coerce `path` to a `pathlib.Path`.
utils.validate_set_members	Validate values that must be of a certain type and (optionally) found among a set of known valid values.
utils.validate_and_clip_range	Validate and clip range values.

Language Identification

`textacy.lang_id`: Interface for de/serializing a language identification model, and using it to identify the most probable language(s) of a given text. Inspired by Google's Compact Language Detector v3 (https://github.com/google/cld3) and implemented with `thinc` v8.0.

Model

Character unigrams, bigrams, and trigrams are extracted separately from the first 1000 characters of lower-cased input text. Each collection of ngrams is hash-embedded into a 100-dimensional space, then averaged. The resulting feature vectors are concatenated into a single embedding layer, then passed on to a dense layer with ReLu activation and finally a Softmax output layer. The model's predictions give the probabilities for a text to be written in ~140 ISO 639-1 languages.

Dataset

The model was trained on a randomized, stratified subset of ~375k texts drawn from several sources:

- **WiLi:** A public dataset of short text extracts from Wikipedias in over 230 languages. Style is relatively formal; subject matter is "encyclopedic". Source: https://zenodo.org/record/841984
- **Tatoeba:** A crowd-sourced collection of sentences and their translations into many languages. Style is relatively informal; subject matter is a variety of everyday things and goings-on. Source: https://tatoeba.org/eng/downloads.
- **UDHR:** The UN's Universal Declaration of Human Rights document, translated into hundreds of languages and split into paragraphs. Style is formal; subject matter is fundamental human rights to be universally protected. Source: https://unicode.org/udhr/index.html
- **DSLCC**: Two collections of short excerpts of journalistic texts in a handful of language groups that are highly similar to each other. Style is relatively formal; subject matter is current events. Source: http://ttg.uni-saarland.de/resources/DSLCC/

Performance

The trained model achieved F1 = 0.97 when averaged over all languages.

A few languages have worse performance; for example, the two Norwegians ("nb" and "no"), as well as Bosnian ("bs"), Serbian ("sr"), and Croatian ("hr"), which are extremely similar to each other. See the textacy-data releases for more details: https://github.com/bdewilde/textacy-data/releases/tag/lang-identifier-v2.0

class textacy.lang_id.lang_identifier.**LangIdentifier**(*version: float | str*, *data_dir: str | pathlib.Path = PosixPath('/home/docs/checkouts/readthedocs.org/user_packages/textacy/data/lang_identifier')*, *model_base: Model = <thinc.model.Model object>*)

Parameters

- **version** –
- **data_dir** –
- **model_base** –

model

classes

save_model()

Save trained *LangIdentifier.model* to disk, as bytes.

load_model() → thinc.model.Model
Load trained model from bytes on disk, using `LangIdentifier.model_base` as the framework into which the data is fit.

download(*force: bool = False*)
Download version-specific model data as a binary file and save it to disk at `LangIdentifier.model_fpath`.

Parameters **force** – If True, download the model data, even if it already exists on disk under `self.data_dir`; otherwise, don't.

identify_lang(*text: str*, *with_probs: bool = False*) → str | Tuple[str, float]
Identify the most probable language identified in `text`, with or without the corresponding probability.

Parameters

- **text** –
- **with_probs** –

Returns ISO 639-1 standard language code of the most probable language, optionally with its probability.

identify_topn_langs(*text: str*, *topn: int = 3*, *with_probs: bool = False*) → List[str] | List[Tuple[str, float]]
Identify the `topn` most probable languages identified in `text`, with or without the corresponding probabilities.

Parameters

- **text** –
- **topn** –
- **with_probs** –

Returns ISO 639-1 standard language code and optionally with its probability of the `topn` most probable languages.

textacy.lang_id.lang_identifier.**identify_lang**(*text: str*, *with_probs: bool = False*) → str | Tuple[str, float]
Identify the most probable language identified in `text`, with or without the corresponding probability.

Parameters

- **text** –
- **with_probs** –

Returns ISO 639-1 standard language code of the most probable language, optionally with its probability.

textacy.lang_id.lang_identifier.**identify_topn_langs**(*text: str*, *topn: int = 3*, *with_probs: bool = False*) → List[str] | List[Tuple[str, float]]
Identify the `topn` most probable languages identified in `text`, with or without the corresponding probabilities.

Parameters

- **text** –
- **topn** –
- **with_probs** –

Returns ISO 639-1 standard language code and optionally with its probability of the `topn` most probable languages.

Utilities

textacy.utils: Variety of general-purpose utility functions for inspecting / validating / transforming args and facilitating meta package tasks.

textacy.utils.**deprecated**(*message: str*, *, *action: str = 'always'*)
Show a deprecation warning, optionally filtered.

Parameters

- **message** – Message to display with `DeprecationWarning`.
- **action** – Filter controlling whether warning is ignored, displayed, or turned into an error. For reference:

See also:

https://docs.python.org/3/library/warnings.html#the-warnings-filter

textacy.utils.**get_config**() → Dict[str, Any]
Get key configuration info about dev environment: OS, python, spacy, and textacy.

Returns dict

textacy.utils.**print_markdown**(*items: Union[Dict[Any, Any], Iterable[Tuple[Any, Any]]]*)
Print `items` as a markdown-formatted list. Specifically useful when submitting config info on GitHub issues.

Parameters **items** –

textacy.utils.**is_record**(*obj: Any*) → bool
Check whether `obj` is a "record" – that is, a (text, metadata) 2-tuple.

textacy.utils.**to_collection**(*val: Any*, *val_type: Union[Type[Any], Tuple[Type[Any], ...]]*, *col_type: Type[Any]*) → Optional[Collection[Any]]
Validate and cast a value or values to a collection.

Parameters

- **val** (*object*) – Value or values to validate and cast.
- **val_type** (*type*) – Type of each value in collection, e.g. `int` or `str`.
- **col_type** (*type*) – Type of collection to return, e.g. `tuple` or `set`.

Returns Collection of type `col_type` with values all of type `val_type`.

Raises **TypeError** –

textacy.utils.**to_bytes**(*s: Union[str, bytes]*, *, *encoding: str = 'utf-8'*, *errors: str = 'strict'*) → bytes
Coerce string `s` to bytes.

textacy.utils.**to_unicode**(*s: Union[str, bytes]*, *, *encoding: str = 'utf-8'*, *errors: str = 'strict'*) → str
Coerce string `s` to unicode.

textacy.utils.**to_path**(*path: Union[str, pathlib.Path]*) → pathlib.Path
Coerce `path` to a `pathlib.Path`.

Parameters **path** –

Returns `pathlib.Path`

textacy.utils.**validate_set_members**(*vals: Union[Any, Set[Any]]*, *val_type: Union[Type[Any], Tuple[Type[Any], ...]]*, *valid_vals: Optional[Set[Any]] = None*) → Set[Any]

Validate values that must be of a certain type and (optionally) found among a set of known valid values.

Parameters

- **vals** – Value or values to validate.
- **val_type** – Type(s) of which all `vals` must be instances.
- **valid_vals** – Set of valid values in which all `vals` must be found.

Returns Validated values.

Return type Set[obj]

Raises

- **TypeError** –
- **ValueError** –

textacy.utils.**validate_and_clip_range**(*range_vals: Tuple[Any, Any]*, *full_range: Tuple[Any, Any]*, *val_type: Optional[Union[Type[Any], Tuple[Type[Any], ...]]] = None*) → Tuple[Any, Any]

Validate and clip range values.

Parameters

- **range_vals** – Range values, i.e. [start_val, end_val), to validate and, if necessary, clip. If None, the value is set to the corresponding value in `full_range`.
- **full_range** – Full range of values, i.e. [min_val, max_val), within which `range_vals` must lie.
- **val_type** – Type(s) of which all `range_vals` must be instances, unless val is None.

Returns Range for which null or too-small/large values have been clipped to the min/max valid values.

Raises

- **TypeError** –
- **ValueError** –

textacy.utils.**get_kwargs_for_func**(*func: Callable*, *kwargs: Dict[str, Any]*) → Dict[str, Any]

Get the set of keyword arguments from `kwargs` that are used by `func`. Useful when calling a func from another func and inferring its signature from provided `**kwargs`.

textacy.utils.**text_to_char_ngrams**(*text: str*, *n: int*, *, *pad: bool = False*) → Tuple[str, ...]

Convert a text string into an ordered sequence of character ngrams.

Parameters

- **text** –
- **n** – Number of characters to concatenate in each `n`-gram.
- **pad** – If True, pad `text` by adding `n - 1` "_" characters on either side; if False, leave `text` as-is.

Returns Ordered sequence of character ngrams.

textacy.types: Definitions for common object types used throughout the package.

class textacy.types.**Record**(*text*, *meta*)

meta
Alias for field number 1

text
Alias for field number 0

textacy.errors: Helper functions for making consistent errors.

textacy.cache: Functionality for caching language data and other NLP resources. Loading data from disk can be slow; let's just do it once and forget about it. :)

textacy.cache.**LRU_CACHE = LRUCache([], maxsize=2147483648, currsize=0)**
Least Recently Used (LRU) cache for loaded data.

The max cache size may be set by the *TEXTACY_MAX_CACHE_SIZE* environment variable, where the value must be an integer (in bytes). Otherwise, the max size is 2GB.

Type cachetools.LRUCache

textacy.cache.**clear**()
Clear textacy's cache of loaded data.

spaCy Utils

textacy.spacier.utils: Helper functions for working with / extending spaCy's core functionality.

textacy.spacier.utils.**make_doc_from_text_chunks**(*text: str*, *lang: Union[str, pathlib.Path, spacy.language.Language]*, *chunk_size: int = 100000*) → spacy.tokens.doc.Doc
Make a single spaCy-processed document from 1 or more chunks of `text`. This is a workaround for processing very long texts, for which spaCy is unable to allocate enough RAM.

Parameters

- **text** – Text document to be chunked and processed by spaCy.
- **lang** – Language with which spaCy processes `text`, represented as the full name of or path on disk to the pipeline, or an already instantiated pipeline instance.
- **chunk_size** – Number of characters comprising each text chunk (excluding the last chunk, which is probably smaller). For best performance, value should be somewhere between 1e3 and 1e7, depending on how much RAM you have available.

Note: Since chunking is done by character, chunks edges' probably won't respect natural language segmentation, which means that every `chunk_size` characters, spaCy's models may make mistakes.

Returns A single processed document, built from concatenated text chunks.

textacy.spacier.utils.**merge_spans**(*spans: Iterable[spacy.tokens.span.Span]*, *doc: spacy.tokens.doc.Doc*) → None
Merge spans into single tokens in `doc`, *in-place*.

Parameters

- **spans** (Iterable[`spacy.tokens.Span`]) –

- **doc** (spacy.tokens.Doc) –

textacy.spacier.utils.**preserve_case**(*token: spacy.tokens.token.Token*) → bool
Return True if token is a proper noun or acronym; otherwise, False.

Raises **ValueError** – If parent document has not been POS-tagged.

textacy.spacier.utils.**get_normalized_text**(*span_or_token: Span | Token*) → str
Get the text of a spaCy span or token, normalized depending on its characteristics. For proper nouns and acronyms, text is returned as-is; for everything else, text is lemmatized.

textacy.spacier.utils.**get_main_verbs_of_sent**(*sent: spacy.tokens.span.Span*) → List[spacy.tokens.token.Token]
Return the main (non-auxiliary) verbs in a sentence.

textacy.spacier.utils.**get_subjects_of_verb**(*verb: spacy.tokens.token.Token*) → List[spacy.tokens.token.Token]
Return all subjects of a verb according to the dependency parse.

textacy.spacier.utils.**get_objects_of_verb**(*verb: spacy.tokens.token.Token*) → List[spacy.tokens.token.Token]
Return all objects of a verb according to the dependency parse, including open clausal complements.

textacy.spacier.utils.**get_span_for_compound_noun**(*noun: spacy.tokens.token.Token*) → Tuple[int, int]
Return document indexes spanning all (adjacent) tokens in a compound noun.

textacy.spacier.utils.**get_span_for_verb_auxiliaries**(*verb: spacy.tokens.token.Token*) → Tuple[int, int]
Return document indexes spanning all (adjacent) tokens around a verb that are auxiliary verbs or negations.

4.5 Changes

4.5.1 0.11.0 (2021-04-12)

- **Refactored, standardized, and extended several areas of functionality**
  - text preprocessing (`textacy.preprocessing`)
    - Added functions for normalizing bullet points in lists (`normalize.bullet_points()`), removing HTML tags (`remove.html_tags()`), and removing bracketed contents such as in-line citations (`remove.brackets()`).
    - Added `make_pipeline()` function for combining multiple preprocessors applied sequentially to input text into a single callable.
    - Renamed functions for flexibility and clarity of use; in most cases, this entails replacing an underscore with a period, e.g. `preprocessing.normalize_whitespace()` => `preprocessing.normalize.whitespace()`.
    - Renamed and standardized some funcs' args; for example, all "replace" functions had their (optional) second argument renamed from `replace_with` => `repl`, and `remove.punctuation(text, marks=".?!")` => `remove.punctuation(text, only=[".", "?", "!"])`.
  - structured information extraction (`textacy.extract`)
    - Consolidated and restructured functionality previously spread across the `extract.py` and `text_utils.py` modules and `ke` subpackage. For the latter two, imports have changed:
      - `from textacy import ke; ke.textrank()` => `from textacy import extract; extract.keyterms.textrank()`

      - `from textacy import text_utils; text_utils.keywords_in_context()` => `from textacy import extract; extract.keywords_in_context()`

    * Added new extraction functions:

      - `extract.regex_matches()`: For matching regex patterns in a document's text that cross spaCy token boundaries, with various options for aligning matches back to tokens.
      - `extract.acronyms()`: For extracting acronym-like tokens, without looking around for related definitions.
      - `extract.terms()`: For flexibly combining n-grams, entities, and noun chunks into a single collection, with optional deduplication.

    * Improved the generality and quality of extracted "triples" such as Subject-Verb-Objects, and changed the structure of returned objects accordingly. Previously, only contiguous spans were permitted for each element, but this was overly restrictive: A sentence like "I did not really like the movie." would produce an SVO of `("I", "like", "movie")` which is... misleading. The new approach uses lists of tokens that need not be adjacent; in this case, it produces `(["I"], ["did", "not", "like"], ["movie"])`. For convenience, triple results are all named tuples, so elements may be accessed by name or index (e.g. `svo.subject` == `svo[0]`).

    * Changed `extract.keywords_in_context()` to always yield results, with optional padding of contexts, leaving printing of contexts up to users; also extended it to accept `Doc` or `str` objects as input.

    * Removed deprecated `extract.pos_regex_matches()` function, which is superseded by the more powerful `extract.token_matches()`.

  - string and sequence similarity metrics (`textacy.similarity`)

    * Refactored top-level `similarity.py` module into a subpackage, with metrics split out into categories: edit-, token-, and sequence-based approaches, as well as hybrid metrics.

    * Added several similarity metrics:

      - edit-based Jaro (`similarity.jaro()`)
      - token-based Cosine (`similarity.cosine()`), Bag (`similarity.bag()`), and Tversky (`similarity.tvserky()`)
      - sequence-based Matching Subsequences Ratio (`similarity.matching_subsequences_ratio()`)
      - hybrid Monge-Elkan (`similarity.monge_elkan()`)

    * Removed a couple similarity metrics: Word Movers Distance relied on a troublesome external dependency, and Word2Vec+Cosine is available in spaCy via `Doc.similarity`.

  - network- and vector-based document representations (`textacy.representations`)

    * Consolidated and reworked networks functionality in `representations.network` module

      - Added `build_cooccurrence_network()` function to represent a sequence of strings (or a sequence of such sequences) as a graph with nodes for each unique string and edges to other strings that co-occurred.
      - Added `build_similarity_network()` function to represent a sequence of strings (or a sequence of such sequences) as a graph with nodes as top-level elements and edges to all others weighted by pairwise similarity.
      - Removed obsolete `network.py` module and duplicative `extract.keyterms.graph_base.py` module.

    - Refined vectorizer initialization, and moved from `vsm.vectorizers` to `representations.vectorizers` module.
        - For both `Vectorizer` and `GroupVectorizer`, applying global inverse document frequency weights is now handled by a single arg: `idf_type: Optional[str]`, rather than a combination of `apply_idf: bool, idf_type: str`; similarly, applying document-length weight normalizations is handled by `dl_type: Optional[str]` instead of `apply_dl: bool, dl_type: str`
    - Added `representations.sparse_vec` module for higher-level access to document vectorization via `build_doc_term_matrix()` and `build_grp_term_matrix()` functions, for cases when a single fit+transform is all you need.
  - automatic language identification (`textacy.lang_id`)
    - Moved functionality from `lang_utils.py` module into a subpackage, and added the primary user interface (`identify_lang()` and `identify_topn_langs()`) as package-level imports.
    - Implemented and trained a more accurate `thinc`-based language identification model that's closer to the original CLD3 inspiration, replacing the simpler `sklearn`-based pipeline.
- **Updated interface with spaCy for v3, and better leveraged the new functionality**
  - Restricted `textacy.load_spacy_lang()` to only accept full spaCy language pipeline names or paths, in accordance with v3's removal of pipeline aliases and general tightening-up on this front. Unfortunately, `textacy` can no longer play fast and loose with automatic language identification => pipeline loading...
  - Extended `textacy.make_spacy_doc()` to accept a `chunk_size` arg that splits input text into chunks, processes each individually, then joins them into a single `Doc`; supersedes `spacier.utils.make_doc_from_text_chunks()`, which is now deprecated.
  - Moved core `Doc` extensions into a top-level `extensions.py` module, and improved/streamlined the collection
    - Refactored and improved performance of `Doc._.to_bag_of_words()` and `Doc._.to_bag_of_terms()`, leveraging related functionality in `extract.words()` and `extract.terms()`
    - Removed redundant/awkward extensions:
      - `Doc._.lang` => use `Doc.lang_`
      - `Doc._.tokens` => use `iter(Doc)`
      - `Doc._.n_tokens` => `len(Doc)`
      - `Doc._.to_terms_list()` => `extract.terms(doc)` or `Doc._.extract_terms()`
      - `Doc._.to_tagged_text()` => NA, this was an old holdover that's not used in practice anymore
      - `Doc._.to_semantic_network()` => NA, use a function in `textacy.representations.networks`
  - Added `Doc` extensions for `textacy.extract` functions (see above for details), with most functions having direct analogues; for example, to extract acronyms, use either `textacy.extract.acronyms(doc)` or `doc._.extract_acronyms()`. Keyterm extraction functions share a single extension: `textacy.extract.keyterms.textrank(doc)` <> `doc._.extract_keyterms(method="textrank")`

  - Leveraged spaCy's new `DocBin` for efficiently saving/loading `Doc`s in binary format, with corresponding arg changes in `io.write_spacy_docs()` and `Corpus.save()+.load()`

- **Improved package documentation, tests, dependencies, and type annotations**
  - Added two beginner-oriented tutorials to documentation, showing how to use various aspects of the package in the context of specific tasks.
  - Reorganized API reference docs to put like functionality together and more consistently provide summary tables up top
  - Updated dependencies list and package versions
    - Removed: `pyemd` and `srsly`
    - Un-capped max versions: `numpy` and `scikit-learn`
    - Bumped min versions: `cytoolz`, `jellyfish`, `matplotlib`, `pyphen`, and `spacy` (v3.0+ only!)
  - Bumped min Python version from 3.6 => 3.7, and added PY3.9 support
  - Removed `textacy.export` module, which had functions for exporting spaCy docs into other external formats; this was a soft dependency on `gensim` and CONLL-U that wasn't enforced or guaranteed, so better to remove.
  - Added `types.py` module for shared types, and used them everywhere. Also added/fixed type annotations throughout the code base.
  - Improved, added, and parametrized literally hundreds of tests.

Contributors

Many thanks to @timgates42, @datanizing, @8W9aG, @0x2b3bfa0, and @gryBox for submitting PRs, either merged or used as inspiration for my own rework-in-progress.

4.5.2 0.10.1 (2020-08-29)

New and Changed:

- **Expanded text statistics and refactored into a sub-package (PR #307)**
  - Refactored `text_stats` module into a sub-package with the same name and top-level API, but restructured under the hood for better consistency
  - Improved performance, API, and documentation on the main `TextStats` class, and improved documentation on many of the individual stats functions
  - Added new readability tests for texts in Arabic (Automated Arabic Readability Index), Spanish (µ-legibility and perspecuity index), and Turkish (a lang-specific formulation of Flesch Reading Ease)
  - *Breaking change:* Removed `TextStats.basic_counts` and `TextStats.readability_stats` attributes, since typically only one or a couple needed for a given use case; also, some of the readability tests are language-specific, which meant bad results could get mixed in with good ones
- **Improved and standardized some code quality and performance (PR #305, #306)**
  - Standardized error messages via top-level `errors.py` module
  - Replaced `str.format()` with f-strings (almost) everywhere, for performance and readability
  - Fixed a whole mess of linting errors, significantly improving code quality and consistency

- **Improved package configuration, and maintenance (PRs #298, #305, #306)**
  - Added automated GitHub workflows for building and testing the package, linting and formatting, publishing new releases to PyPi, and building documentation (and ripped out Travis CI)
  - Added a makefile with common commands for dev work, plus instructions
  - Adopted the new `pyproject.toml` package configuration standard; updated and streamlined `setup.py` and `setup.cfg` accordingly; and removed `requirements.txt`
  - Moved all source code into a `/src` directory, for technical reasons
  - Added `mypy`-specific config file to reduce output noisiness when type-checking
- **Improved and moved package documentation (PR #309)**
  - Moved the docs site back to ReadTheDocs (https://textacy.readthedocs.io)! Pardon the years-long detour into GitHub Pages...
  - Enabled markdown-based documentation using `recommonmark` instead of `m2r`, and migrated all "narrative" docs from `.rst` to equivalent `.md` files
  - Added auto-generated summary tables to many sections of the API Reference, to help users get an overview of functionality and better find what they're looking for; also added auto-generated section heading references
  - Tidied up and further standardized docstrings throughout the code
- **Kept up with the Python ecosystem**
  - Trained a v1.1 language identifier model using `scikit-learn==0.23.0`, and bumped the upper bound on that dependency's version accordingly
  - Updated and parametrized many tests using modern `pytest` functionality (PR #306)
  - Got `textacy` versions 0.9.1 and 0.10.0 up on `conda-forge` (Issue #294)
  - Added spectral seriation as a term-ordering technique when making a "Termite" visualization by taking advantage of `pandas.DataFrame` functionality, and otherwise tidied up the default for nice-looking plots (PR #295)

Fixed:

- Corrected an incorrect and misleading reference in the quickstart docs (Issue #300, PR #302)
- Fixed a bug in the `delete_words()` augmentation transform (Issue #308)

Contributors:

Special thanks to @tbsexton, @marius-mather, and @rmax for their contributions!

4.5.3 0.10.0 (2020-03-01)

New:

- Added a logo to textacy's documentation and social preview :page_with_curl:
- Added type hints throughout the code base, for more expressive type indicators in docstrings and for static type checkers used by developers to code more effectively (PR #289)
- Added a preprocessing function to normalize sequences of repeating characters (Issue #275)

Changed:

- Improved core `Corpus` functionality using recent additions to spacy (PR #285)
  - Re-implemented `Corpus.save()` and `Corpus.load()` using spacy's new `DocBin` class, which resolved a few bugs/issues (Issue #254)
  - Added `n_process` arg to `Corpus.add()` to set the number of parallel processes used when adding many items to a corpus, following spacy's updates to `nlp.pipe()` (Issue #277)
  - Bumped minimum spaCy version from 2.0.12 => 2.2.0, accordingly
- Added handling for zero-width whitespaces into `normalize_whitespace()` function (Issue #278)
- Improved a couple rough spots in package administration:
  - Moved package setup information into a declarative configuration file, in an attempt to keep up with evolving best practices for Python packaging
  - Simplified the configuration and interoperability of sphinx + github pages for generating package documentation

Fixed:

- Fixed typo in ConceptNet docstring (Issue #280)
- Trained and distributed a `LangIdentifier` model using `scikit-learn==0.22`, to prevent ambiguous errors when trying to load a file that didn't exist (Issues #291, #292)

4.5.4 0.9.1 (2019-09-03)

Changed:

- Tweaked `TopicModel` class to work with newer versions of `scikit-learn`, and updated version requirements accordingly from `>=0.18.0,<0.21.0` to `>=0.19`

Fixed:

- Fixed residual bugs in the script for training language identification pipelines, then trained and released one using `scikit-learn==0.19` to prevent errors for users on that version

4.5.5 0.9.0 (2019-09-03)

Note: `textacy` is now PY3-only! Specifically, support for PY2.7 has been dropped, and the minimum PY3 version has been bumped to 3.6 (PR #261). See below for related changes.

New:

- **Added `augmentation` subpackage for basic text data augmentation** (PR #268, #269)
  - implemented several transformer functions for substituting, inserting, swapping, and deleting elements of text at both the word- and character-level
  - implemented an `Augmenter` class for combining multiple transforms and applying them to spaCy `Docs` in a randomized but configurable manner
  - **Note:** This API is provisional, and subject to change in future releases.
- **Added `resources` subpackage for standardized access to linguistic resources** (PR #265)
  - **DepecheMood++:** high-coverage emotion lexicons for understanding the emotions evoked by a text. Updated from a previous version, and now features better English data and Italian data with expanded, consistent functionality.
    * removed `lexicon_methods.py` module with previous implementation
  - **ConceptNet:** multilingual knowledge base for representing relationships between words, similar to WordNet. Currently supports getting word antonyms, hyponyms, meronyms, and synonyms in dozens of languages.
- **Added `UDHR` dataset, a collection of translations of the Universal Declaration of Human Rights** (PR #271)

Changed:

- Updated and extended functionality previously blocked by PY2 compatibility while reducing code bloat / complexity
  - made many args keyword-only, to prevent user error
  - args accepting strings for directory / file paths now also accept `pathlib.Path` objects, with `pathlib` adopted widely under the hood
  - increased minimum versions and/or uncapped maximum versions of several dependencies, including `jellyfish`, `networkx`, and `numpy`
- Added a Portuguese-specific formulation of Flesch Reading Ease score to `text_stats` (PR #263)
- Reorganized and grouped together some like functionality
  - moved core functionality for loading spaCy langs and making spaCy docs into `spacier.core`, out of `cache.py` and `doc.py`
  - moved some general-purpose functionality from `dataset.utils` to `io.utils` and `utils.py`
  - moved function for loading "hyphenator" out of `cache.py` and into `text_stats.py`, where it's used

- Re-trained and released language identification pipelines using a better mix of training data, for slightly improved performance; also added the script used to train the pipeline
- Changed API Reference docs to show items in source code rather than alphabetical order, which should make the ordering more human-friendly
- Updated repo README and PyPi metadata to be more consistent and representative of current functionality
- Removed previously deprecated `textacy.io.split_record_fields()` function

Fixed:

- Fixed a regex for cleaning up crufty terms to prevent catastrophic backtracking in certain edge cases (true story: this bug was encountered in *production code*, and ruined my day)
- Fixed bad handling of edge cases in sCAKE keyterm extraction (Issue #270)
- Changed order in which URL regexes are applied in `preprocessing.replace_urls()` to properly handle certain edge case URLs (Issue #267)

Contributors:

Thanks much to @hugoabonizio for the contribution.

4.5.6 0.8.0 (2019-07-14)

New and Changed:

- **Refactored and expanded text preprocessing functionality** (PR #253)
  - Moved code from a top-level `preprocess` module into a `preprocessing` sub-package, and reorganized it in the process
  - Added new functions:
    * `replace_hashtags()` to replace hashtags like `#FollowFriday` or `#spacyIRL2019` with `_TAG_`
    * `replace_user_handles()` to replace user handles like `@bjdewilde` or `@spacy_io` with `_USER_`
    * `replace_emojis()` to replace emoji symbols like or with `_EMOJI_`
    * `normalize_hyphenated_words()` to join hyphenated words back together, like `anticipation` => `anticipation`
    * `normalize_quotation_marks()` to replace "fancy" quotation marks with simple ascii equivalents, like `"the god particle"` => `"the god particle"`
  - Changed a couple functions for clarity and consistency:
    * `replace_currency_symbols()` now replaces *all* dedicated ascii and unicode currency symbols with `_CUR_`, rather than just a subset thereof, and no longer provides for replacement with the corresponding currency code (like € => `EUR`)
    * `remove_punct()` now has a `fast (bool)` kwarg rather than `method (str)`
  - Removed `normalize_contractions()`, `preprocess_text()`, and `fix_bad_unicode()` functions, since they were bad/awkward and more trouble than they were worth

- **Refactored and expanded keyterm extraction functionality** (PR #257)
  - Moved code from a top-level `keyterms` module into a `ke` sub-package, and cleaned it up / standardized arg names / better shared functionality in the process
  - Added new unsupervised keyterm extraction algorithms: YAKE (`ke.yake()`), sCAKE (`ke.scake()`), and PositionRank (`ke.textrank()`, with non-default parameter values)
  - Added new methods for selecting candidate keyterms: longest matching subsequence candidates (`ke.utils.get_longest_subsequence_candidates()`) and pattern-matching candidates (`ke.utils.get_pattern_matching_candidates()`)
  - Improved speed of SGRank implementation, and generally optimized much of the code
- **Improved document similarity functionality** (PR #256)
  - Added a character ngram-based similarity measure (`similarity.character_ngrams()`), for something that's useful in different contexts than the other measures
  - Removed Jaro-Winkler string similarity measure (`similarity.jaro_winkler()`), since it didn't add much beyond other measures
  - Improved speed of Token Sort Ratio implementation
  - Replaced `python-levenshtein` dependency with `jellyfish`, for its active development, better documentation, and *actually-compliant* license
- **Added customizability to certain functionality**
  - Added options to `Doc._.to_bag_of_words()` and `Corpus.word_counts()` for filtering out stop words, punctuation, and/or numbers (PR #249)
  - Allowed for objects that *look like* `sklearn`-style topic modeling classes to be passed into `tm.TopicModel()` (PR #248)
  - Added options to customize rc params used by `matplotlib` when drawing a "termite" plot in `viz.draw_termite_plot()` (PR #248)
- Removed deprecated functions with direct replacements: `io.utils.get_filenames()` and `spacier.components.merge_entities()`

Contributors:

Huge thanks to @kjoshi and @zf109 for the PRs!

4.5.7 0.7.1 (2019-06-25)

New:

- Added a default, built-in language identification classifier that's moderately fast, moderately accurate, and covers a relatively large number of languages [PR #247]
  - Implemented a Google CLD3-inspired model in `scikit-learn` and trained it on ~1.5M texts in ~130 different languages spanning a wide variety of subject matter and stylistic formality; overall, speed and performance compare favorably to other open-source options (`langid`, `langdetect`, `cld2-cffi`, and `cld3`)
  - Dropped `cld2-cffi` dependency [Issue #246]

- Added `extract.matches()` function to extract spans from a document matching one or more pattern of per-token (attribute, value) pairs, with optional quantity qualifiers; this is a convenient interface to spaCy's rule-based `Matcher` and a more powerful replacement for textacy's existing (now deprecated) `extract.pos_regex_matches()`
- Added `preprocess.normalize_unicode()` function to transform unicode characters into their canonical forms; this is a less-intensive consolation prize for the previously-removed `fix_unicode()` function

Changed:

- Enabled loading blank spaCy `Language` pipelines (tokenization only – no model-based tagging, parsing, etc.) via `load_spacy_lang(name, allow_blank=True)` for use cases that don't rely on annotations; disabled by default to avoid unwelcome surprises
- Changed inclusion/exclusion and de-duplication of entities and ngrams in `to_terms_list()` [Issues #169, #179]
  - `entities = True` => include entities, and drop exact duplicate ngrams
  - `entities = False` => don't include entities, and also drop exact duplicate ngrams
  - `entities = None` => use ngrams as-is without checking against entities
- Moved `to_collection()` function from the `datasets.utils` module to the top-level `utils` module, for use throughout the code base
- Added `quoting` option to `io.read_csv()` and `io.write_csv()`, for problematic cases
- Deprecated the `spacier.components.merge_entities()` pipeline component, an implementation of which has since been added into spaCy itself
- Updated documentation for developer convenience and reader clarity
  - Split API reference docs into related chunks, rather than having them all together in one long page, and tidied up headers
  - Fixed errors / inconsistencies in various docstrings (a never-ending struggle. . .)
  - Ported package readme and changelog from `.rst` to `.md` format

Fixed:

- The `NotImplementedError` previously added to `preprocess.fix_unicode()` is now *raised* rather than returned [Issue #243]

4.5.8 0.7.0 (2019-05-13)

New and Changed:

- **Removed textacy.Doc, and split its functionality into two parts**
  - **New:** Added `textacy.make_spacy_doc()` as a convenient and flexible entry point for making spaCy `Doc` s from text or (text, metadata) pairs, with optional spaCy language pipeline specification. It's similar to `textacy.Doc.__init__`, with the exception that text and metadata are passed in together as a 2-tuple.

  - **New:** Added a variety of custom doc property and method extensions to the global `spacy.tokens.Doc` class, accessible via its `Doc._` "underscore" property. These are similar to the properties/methods on `textacy.Doc`, they just require an interstitial underscore. For example, `textacy.Doc.to_bag_of_words()` => `spacy.tokens.Doc._.to_bag_of_words()`.
  - **New:** Added functions for setting, getting, and removing these extensions. Note that they are set automatically when textacy is imported.

- **Simplified and improved performance of textacy.Corpus**
  - Documents are now added through a simpler API, either in `Corpus.__init__` or `Corpus.add()`; they may be one or a stream of texts, (text, metadata) pairs, or existing spaCy `Doc` s. When adding many documents, the spaCy language processing pipeline is used in a faster and more efficient way.
  - Saving / loading corpus data to disk is now more efficient and robust.
  - Note: `Corpus` is now a collection of spaCy `Doc` s rather than `textacy.Doc` s.

- **Simplified, standardized, and added Dataset functionality**
  - **New:** Added an `IMDB` dataset, built on the classic 2011 dataset commonly used to train sentiment analysis models.
  - **New:** Added a base `Wikimedia` dataset, from which a reworked `Wikipedia` dataset and a separate `Wikinews` dataset inherit. The underlying data source has changed, from XML db dumps of raw wiki markup to JSON db dumps of (relatively) clean text and metadata; now, the code is simpler, faster, and totally language-agnostic.
  - `Dataset.records()` now streams (text, metadata) pairs rather than a dict containing both text and metadata, so users don't need to know field names and split them into separate streams before creating `Doc` or `Corpus` objects from the data.
  - Filtering and limiting the number of texts/records produced is now clearer and more consistent between `.texts()` and `.records()` methods on a given `Dataset` — and more performant!
  - Downloading datasets now always shows progress bars and saves to the same file names. When appropriate, downloaded archive files' contents are automatically extracted for easy inspection.
  - Common functionality (such as validating filter values) is now standardized and consolidated in the `datasets.utils` module.

- **Quality of life improvements**
  - Reduced load time for `import textacy` from ~2-3 seconds to ~1 second, by lazy-loading expensive variables, deferring a couple heavy imports, and dropping a couple dependencies. Specifically:
    * `ftfy` was dropped, and a `NotImplementedError` is now raised in textacy's wrapper function, `textacy.preprocess.fix_bad_unicode()`. Users with bad unicode should now directly call `ftfy.fix_text()`.
    * `ijson` was dropped, and the behavior of `textacy.read_json()` is now simpler and consistent with other functions for line-delimited data.
    * `mwparserfromhell` was dropped, since the reworked `Wikipedia` dataset no longer requires complicated and slow parsing of wiki markup.
  - Renamed certain functions and variables for clarity, and for consistency with existing conventions:
    * `textacy.load_spacy()` => `textacy.load_spacy_lang()`
    * `textacy.extract.named_entities()` => `textacy.extract.entities()`
    * `textacy.data_dir` => `textacy.DEFAULT_DATA_DIR`

    * `filename` => `filepath` and `dirname` => `dirpath` when specifying full paths to files/dirs on disk, and `textacy.io.utils.get_filenames()` => `textacy.io.utils.get_filepaths()` accordingly
    * compiled regular expressions now consistently start with `RE_`
    * `SpacyDoc` => `Doc`, `SpacySpan` => `Span`, `SpacyToken` => `Token`, `SpacyLang` => `Language` as variables and in docs
  - Removed deprecated functionality
    * top-level `spacy_utils.py` and `spacy_pipelines.py` are gone; use equivalent functionality in the `spacier` subpackage instead
    * `math_utils.py` is gone; it was long neglected, and never actually used
  - Replaced `textacy.compat.bytes_to_unicode()` and `textacy.compat.unicode_to_bytes()` with `textacy.compat.to_unicode()` and `textacy.compat.to_bytes()`, which are safer and accept either binary or text strings as input.
  - Moved and renamed language detection functionality, `textacy.text_utils.detect_language()` => `textacy.lang_utils.detect_lang()`. The idea is to add more/better lang-related functionality here in the future.
  - Updated and cleaned up documentation throughout the code base.
  - Added and refactored *many* tests, for both new and old functionality, significantly increasing test coverage while significantly reducing run-time. Also, added a proper coverage report to CI builds. This should help prevent future errors and inspire better test-writing.
  - Bumped the minimum required spaCy version: `v2.0.0` => `v2.0.12`, for access to their full set of custom extension functionality.

Fixed:

- The progress bar during an HTTP download now always closes, preventing weird nesting issues if another bar is subsequently displayed.
- Filtering datasets by multiple values performed either a logical AND or OR over the values, which was confusing; now, a logical OR is always performed.
- The existence of files/directories on disk is now checked *properly* via `os.path.isfile()` or `os.path.isdir()`, rather than `os.path.exists()`.
- Fixed a variety of formatting errors raised by sphinx when generating HTML docs.

4.5.9 0.6.3 (2019-03-23)

New:

- Added a proper contributing guide and code of conduct, as well as separate GitHub issue templates for different user situations. This should help folks contribute to the project more effectively, and make maintaining it a bit easier, too. [Issue #212]
- Gave the documentation a new look, using a template popularized by `requests`. Added documentation on dealing with multi-lingual datasets. [Issue #233]
- Made some minor adjustments to package dependencies, the way they're specified, and the Travis CI setup, making for a faster and better development experience.

- Confirmed and enabled compatibility with v2.1+ of `spacy`. :dizzy:

Changed:

- Improved the `Wikipedia` dataset class in a variety of ways: it can now read Wikinews db dumps; access records in namespaces other than the usual "0" (such as category pages in namespace "14"); parse and extract category pages in several languages, including in the case of bad wiki markup; and filter out section headings from the accompanying text via an `include_headings` kwarg. [PR #219, #220, #223, #224, #231]
- Removed the `transliterate_unicode()` preprocessing function that transliterated non-ascii text into a reasonable ascii approximation, for technical and philosophical reasons. Also removed its GPL-licensed `unidecode` dependency, for legal-ish reasons. [Issue #203]
- Added convention-abiding `exclude` argument to the function that writes `spacy` docs to disk, to limit which pipeline annotations are serialized. Replaced the existing but non-standard `include_tensor` arg.
- Deprecated the `n_threads` argument in `Corpus.add_texts()`, which had not been working in `spacy.pipe` for some time and, as of v2.1, is defunct.
- Made many tests model- and python-version agnostic and thus less likely to break when `spacy` releases new and improved models.
- Auto-formatted the entire code base using `black`; the results aren't always more readable, but they are pleasingly consistent.

Fixed:

- Fixed bad behavior of `key_terms_from_semantic_network()`, where an error would be raised if no suitable key terms could be found; now, an empty list is returned instead. [Issue #211]
- Fixed variable name typo so `GroupVectorizer.fit()` actually works. [Issue #215]
- Fixed a minor typo in the quick-start docs. [PR #217]
- Check for and filter out any named entities that are entirely whitespace, seemingly caused by an issue in `spacy`.
- Fixed an undefined variable error when merging spans. [Issue #225]
- Fixed a unicode/bytes issue in experimental function for deserializing `spacy` docs in "binary" format. [Issue #228, PR #229]

Contributors:

Many thanks to @abevieiramota, @ckot, @Jude188, and @digest0r for their help!

4.5.10 0.6.2 (2018-07-19)

Changed:

- Add a `spacier.util` module, and add / reorganize relevant functionality
  - move (most) `spacy_util` functions here, and add a deprecation warning to the `spacy_util` module
  - rename `normalized_str()` => `get_normalized_text()`, for consistency and clarity
  - add a function to split long texts up into chunks but combine them into a single `Doc`. This is a workaround for a current limitation of spaCy's neural models, whose RAM usage scales with the length of input text.

- Add experimental support for reading and writing spaCy docs in binary format, where multiple docs are contained in a single file. This functionality was supported by spaCy v1, but is not in spaCy v2; I've implemented a workaround that should work well in most situations, but YMMV.
- Package documentation is now "officially" hosted on GitHub pages. The docs are automatically built on and deployed from Travis via `doctr`, so they stay up-to-date with the master branch on GitHub. Maybe someday I'll get ReadTheDocs to successfully build `textacy` once again...
- Minor improvements/updates to documentation

Fixed:

- Add missing return statement in deprecated `text_stats.flesch_readability_ease()` function (Issue #191)
- Catch an empty graph error in bestcoverage-style keyterm ranking (Issue #196)
- Fix mishandling when specifying a single named entity type to in/exclude in `extract.named_entities` (Issue #202)
- Make `networkx` usage in keyterms module compatible with v1.11+ (Issue #199)

4.5.11 0.6.1 (2018-04-11)

New:

- **Add a new spacier sub-package for spaCy-oriented functionality** (#168, #187)
  - Thus far, this includes a `components` module with two custom spaCy pipeline components: one to compute text stats on parsed documents, and another to merge named entities into single tokens in an efficient manner. More to come!
  - Similar functionality in the top-level `spacy_pipelines` module has been deprecated; it will be removed in v0.7.0.

Changed:

- Update the readme, usage, and API reference docs to be clearer and (I hope) more useful. (#186)
- Removing punctuation from a text via the `preprocessing` module now replaces punctuation marks with a single space rather than an empty string. This gives better behavior in many situations; for example, "won't" => "won t" rather than "wont", the latter of which is a valid word with a different meaning.
- Categories are now correctly extracted from non-English language Wikipedia datasets, starting with French and German and extendable to others. (#175)
- Log progress when adding documents to a corpus. At the debug level, every doc's addition is logged; at the info level, only one message per batch of documents is logged. (#183)

Fixed:

- Fix two breaking typos in `extract.direct_quotations()`. (issue #177)
- Prevent crashes when adding non-parsed documents to a `Corpus`. (#180)
- Fix bugs in `keyterms.most_discriminating_terms()` that used `vsm` functionality as it was *before* the changes in v0.6.0. (#189)
- Fix a breaking typo in `vsm.matrix_utils.apply_idf_weighting()`, and rename the problematic kwarg for consistency with related functions. (#190)

Contributors:

Big thanks to @sammous, @dixiekong (nice name!), and @SandyRogers for the pull requests, and many more for pointing out various bugs and the rougher edges / unsupported use cases of this package.

4.5.12 0.6.0 (2018-02-25)

Changed:

- **Rename, refactor, and extend I/O functionality** (PR #151)
  - Related read/write functions were moved from `read.py` and `write.py` into format-specific modules, and similar functions were consolidated into one with the addition of an arg. For example, `write.write_json()` and `write.write_json_lines()` => `json.write_json(lines=True|False)`.
  - Useful functionality was added to a few readers/writers. For example, `write_json()` now automatically handles python dates/datetimes, writing them to disk as ISO-formatted strings rather than raising a TypeError ("datetime is not JSON serializable", ugh). CSVs can now be written to / read from disk when each row is a dict rather than a list. Reading/writing HTTP streams now allows for basic authentication.
  - Several things were renamed to improve clarity and consistency from a user's perspective, most notably the subpackage name: `fileio` => `io`. Others: `read_file()` and `write_file()` => `read_text()` and `write_text()`; `split_record_fields()` => `split_records()`, although I kept an alias to the old function for folks; `auto_make_dirs` boolean kwarg => `make_dirs`.
  - `io.open_sesame()` now handles zip files (provided they contain only 1 file) as it already does for gzip, bz2, and lzma files. On a related note, Python 2 users can now open lzma (`.xz`) files if they've installed `backports.lzma`.
- **Improve, refactor, and extend vector space model functionality** (PRs #156 and #167)
  - BM25 term weighting and document-length normalization were implemented, and and users can now flexibly add and customize individual components of an overall weighting scheme (local scaling + global scaling + doc-wise normalization). For API sanity, several additions and changes to the `Vectorizer` init params were required — sorry bout it!
  - Given all the new weighting possibilities, a `Vectorizer.weighting` attribute was added for curious users, to give a mathematical representation of how values in a doc-term matrix are being calculated. Here's a simple and a not-so-simple case:

```
>>> Vectorizer(apply_idf=True, idf_type='smooth').weighting
'tf * log((n_docs + 1) / (df + 1)) + 1'
```

(continues on next page)

(continued from previous page)

```
>>> Vectorizer(tf_type='bm25', apply_idf=True, idf_type='smooth', apply_
↪dl=True).weighting
'(tf * (k + 1)) / (tf + k * (1 - b + b * (length / avg(lengths))) * log((n_
↪docs - df + 0.5) / (df + 0.5))'
```

  - Terms are now sorted alphabetically after fitting, so you'll have a consistent and interpretable ordering in your vocabulary and doc-term-matrix.
  - A `GroupVectorizer` class was added, as a child of `Vectorizer` and an extension of typical document-term matrix vectorization, in which each row vector corresponds to the weighted terms co-occurring in a single document. This allows for customized grouping, such as by a shared author or publication year, that may span multiple documents, without forcing users to merge /concatenate those documents themselves.
  - Lastly, the `vsm.py` module was refactored into a `vsm` subpackage with two modules. Imports should stay the same, but the code structure is now more amenable to future additions.

- **Miscellaneous additions and improvements**
  - Flesch Reading Ease in the `textstats` module is now multi-lingual! Language- specific formulations for German, Spanish, French, Italian, Dutch, and Russian were added, in addition to (the default) English. (PR #158, prompted by Issue #155)
  - Runtime performance, as well as docs and error messages, of functions for generating semantic networks from lists of terms or sentences were improved. (PR #163)
  - Labels on named entities from which determiners have been dropped are now preserved. There's still a minor gotcha, but it's explained in the docs.
  - The size of `textacy`'s data cache can now be set via an environment variable, `TEXTACY_MAX_CACHE_SIZE`, in case the default 2GB cache doesn't meet your needs.
  - Docstrings were improved in many ways, large and small, throughout the code. May they guide you even more effectively than before!
  - The package version is now set from a single source. This isn't for you so much as me, but it does prevent confusing version mismatches b/w code, pypi, and docs.
  - All tests have been converted from `unittest` to `pytest` style. They run faster, they're more informative in failure, and they're easier to extend.

Fixed:

- Fixed an issue where existing metadata associated with a spacy Doc was being overwritten with an empty dict when using it to initialize a textacy Doc. Users can still overwrite existing metadata, but only if they pass in new data.
- Added a missing import to the README's usage example. (#149)
- The intersphinx mapping to `numpy` got fixed (and items for `scipy` and `matplotlib` were added, too). Taking advantage of that, a bunch of broken object links scattered throughout the docs got fixed.
- Fixed broken formatting of old entries in the changelog, for your reading pleasure.

4.5.13 0.5.0 (2017-12-04)

Changed:

- **Bumped version requirement for spaCy from < 2.0 to >= 2.0** — textacy no longer works with spaCy 1.x! It's worth the upgrade, though. v2.0's new features and API enabled (or required) a few changes on textacy's end
  - `textacy.load_spacy()` takes the same inputs as the new `spacy.load()`, i.e. a package `name` string and an optional list of pipes to `disable`
  - textacy's `Doc` metadata and language string are now stored in `user_data` directly on the spaCy `Doc` object; although the API from a user's perspective is unchanged, this made the next change possible
  - `Doc` and `Corpus` classes are now de/serialized via pickle into a single file — no more side-car JSON files for metadata! Accordingly, the `.save()` and `.load()` methods on both classes have a simpler API: they take a single string specifying the file on disk where data is stored.
- **Cleaned up docs, imports, and tests throughout the entire code base.**
  - docstrings and https://textacy.readthedocs.io 's API reference are easier to read, with better cross-referencing and far fewer broken web links
  - namespaces are less cluttered, and textacy's source code is easier to follow
  - `import textacy` takes less than half the time from before
  - the full test suite also runs about twice as fast, and most tests are now more robust to changes in the performance of spaCy's models
  - consistent adherence to conventions eases users' cognitive load :)
- **The module responsible for caching loaded data in memory was cleaned up and improved**, as well as renamed: from `data.py` to `cache.py`, which is more descriptive of its purpose. Otherwise, you shouldn't notice much of a difference besides *things working correctly*.
  - All loaded data (e.g. spacy language pipelines) is now cached together in a single LRU cache whose max size is set to 2GB, and the size of each element in the cache is now accurately computed. (tl;dr: `sys.getsizeof` does not work on non-built-in objects like, say, a `spacy.tokens.Doc`.)
  - Loading and downloading of the DepecheMood resource is now less hacky and weird, and much closer to how users already deal with textacy's various `Dataset` s, In fact, it can be downloaded in exactly the same way as the datasets via textacy's new CLI: `$ python -m textacy download depechemood`. P.S. A brief guide for using the CLI got added to the README.
- **Several function/method arguments marked for deprecation have been removed.** If you've been ignoring the warnings that print out when you use `lemmatize=True` instead of `normalize='lemma'` (etc.), now is the time to update your calls!
  - Of particular note: The `readability_stats()` function has been removed; use `TextStats(doc).readability_stats` instead.

Fixed:

- In certain situations, the text of a spaCy span was being returned without whitespace between tokens; that has been avoided in textacy, and the source bug in spaCy got fixed (by yours truly! https://github.com/explosion/spaCy/pull/1621).
- When adding already-parsed `Doc`s to a `Corpus`, including `metadata` now correctly overwrites any existing metadata on those docs.
- Fixed a couple related issues involving the assignment of a 2-letter language string to the `.lang` attribute of `Doc` and `Corpus` objects.
- textacy's CLI wasn't correctly handling certain dataset kwargs in all cases; now, all kwargs get to their intended destinations.

4.5.14 0.4.2 (2017-11-28)

New:

- Added a CLI for downloading `textacy`-related data, inspired by the `spaCy` equivalent. It's *temporarily* undocumented, but to see available commands and options, just pass the usual flag: `$ python -m textacy --help`. Expect more functionality (and docs!) to be added soonish. (#144)
  - Note: The existing `Dataset.download()` methods work as before, and in fact, they are being called under the hood from the command line.

Changed:

- Made usage of `networkx` v2.0-compatible, and therefore dropped the <2.0 version requirement on that dependency. Upgrade as you please! (#131)
- Improved the regex for identifying phone numbers so that it's easier to view and interpret its matches. (#128)

Fixed:

- Fixed caching of counts on `textacy.Doc` instance-specific, rather than shared by all instances of the class. Oops.
- Fixed currency symbols regex, so as not to replace all instances of the letter "z" when a custom string is passed into `replace_currency_symbols()`. (#137)
- Fixed README usage example, which skipped downloading of dataset data. Btw, see above for another way! (#124)
- Fixed typo in the API reference, which included the SupremeCourt dataset twice and omitted the RedditComments dataset. (#129)
- Fixed typo in `RedditComments.download()` that prevented it from downloading any data. (#143)

Contributors:

Many thanks to @asifm, @harryhoch, and @mdlynch37 for submitting PRs!

4.5.15 0.4.1 (2017-07-27)

Changed:

- Added key classes to the top-level `textacy` imports, for convenience:
  - `textacy.text_stats.TextStats` => `textacy.TextStats`
  - `textacy.vsm.Vectorizer` => `textacy.Vectorizer`
  - `textacy.tm.TopicModel` => `textacy.TopicModel`
- Added tests for `textacy.Doc` and updated the README's usage example

Fixed:

- Added explicit encoding when opening Wikipedia database files in text mode to resolve an issue when doing so without encoding on Windows (PR #118)
- Fixed `keyterms.most_discriminating_terms` to use the `vsm.Vectorizer` class rather than the `vsm.doc_term_matrix` function that it replaced (PR #120)
- Fixed mishandling of a couple optional args in `Doc.to_terms_list`

Contributors:

Thanks to @minketeer and @Gregory-Howard for the fixes!

4.5.16 0.4.0 (2017-06-21)

New and Changed:

- Refactored and expanded built-in `corpora`, now called `datasets` (PR #112)
  - The various classes in the old `corpora` subpackage had a similar but frustratingly not-identical API. Also, some fetched the corresponding dataset automatically, while others required users to do it themselves. Ugh.
  - These classes have been ported over to a new `datasets` subpackage; they now have a consistent API, consistent features, and consistent documentation. They also have some new functionality, including pain-free downloading of the data and saving it to disk in a stream (so as not to use all your RAM).
  - Also, there's a new dataset: A collection of 2.7k Creative Commons texts from the Oxford Text Archive, which rounds out the included datasets with English-language, 16th-20th century *literary* works. (h/t @JonathanReeve)
- A `Vectorizer` class to convert tokenized texts into variously weighted document-term matrices (Issue #69, PR #113)
  - This class uses the familiar `scikit-learn` API (which is also consistent with the `textacy.tm.TopicModel` class) to convert one or more documents in the form of "term lists" into weighted vectors. An initial set of documents is used to build up the matrix vocabulary (via `.fit()`), which can then be applied to new documents (via `.transform()`).

  - It's similar in concept and usage to sklearn's `CountVectorizer` or `TfidfVectorizer`, but doesn't convolve the tokenization task as they do. This means users have more flexibility in deciding which terms to vectorize. This class outright replaces the `textacy.vsm.doc_term_matrix()` function.

- Customizable automatic language detection for `Doc` s
  - Although `cld2-cffi` is fast and accurate, its installation is problematic for some users. Since other language detection libraries are available (e.g. `langdetect` and `langid`), it makes sense to let users choose, as needed or desired.
  - First, `cld2-cffi` is now an optional dependency, i.e. is not installed by default. To install it, do `pip install textacy[lang]` or (for it and all other optional deps) do `pip install textacy[all]`. (PR #86)
  - Second, the `lang` param used to instantiate `Doc` objects may now be a callable that accepts a unicode string and returns a standard 2-letter language code. This could be a function that uses `langdetect` under the hood, or a function that always returns "de" – it's up to users. Note that the default value is now `textacy.text_utils.detect_language()`, which uses `cld2-cffi`, so the default behavior is unchanged.
- Customizable punctuation removal in the `preprocessing` module (Issue #91)
  - Users can now specify which punctuation marks they wish to remove, rather than always removing *all* marks.
  - In the case that all marks are removed, however, performance is now 5-10x faster by using Python's built-in `str.translate()` method instead of a regular expression.
- `textacy`, installable via `conda` (PR #100)
  - The package has been added to Conda-Forge (here), and installation instructions have been added to the docs. Hurray!
- `textacy`, now with helpful badges
  - Builds are now automatically tested via Travis CI, and there's a badge in the docs showing whether the build passed or not. The days of my ignoring broken tests in `master` are (probably) over...
  - There are also badges showing the latest releases on GitHub, pypi, and conda-forge (see above).

Fixed:

- Fixed the check for overlap between named entities and unigrams in the `Doc.to_terms_list()` method (PR #111)
- `Corpus.add_texts()` uses CPU_COUNT - 1 threads by default, rather than always assuming that 4 cores are available (Issue #89)
- Added a missing coding declaration to a test file, without which tests failed for Python 2 (PR #99)
- `readability_stats()` now catches an exception raised on empty documents and logs a message, rather than barfing with an unhelpful `ZeroDivisionError`. (Issue #88)
- Added a check for empty terms list in `terms_to_semantic_network` (Issue #105)
- Added and standardized module-specific loggers throughout the code base; not a bug per sé, but certainly some much-needed housecleaning
- Added a note to the docs about expectations for bytes vs. unicode text (PR #103)

Contributors:

Thanks to @henridwyer, @rolando, @pavlin99th, and @kyocum for their contributions! :raised_hands:

4.5.17 0.3.4 (2017-04-17)

New and Changed:

- Improved and expanded calculation of basic counts and readability statistics in `text_stats` module.
  - Added a `TextStats()` class for more convenient, granular access to individual values. See usage docs for more info. When calculating, say, just one readability statistic, performance with this class should be slightly better; if calculating *all* statistics, performance is worse owing to unavoidable, added overhead in Python for variable lookups. The legacy function `text_stats.readability_stats()` still exists and behaves as before, but a deprecation warning is displayed.
  - Added functions for calculating Wiener Sachtextformel (PR #77), LIX, and GULPease readability statistics.
  - Added number of long words and number of monosyllabic words to basic counts.
- Clarified the need for having spacy models installed for most use cases of textacy, in addition to just the spacy package.
  - README updated with comments on this, including links to more extensive spacy documentation. (Issues #66 and #68)
  - Added a function, `compat.get_config()` that includes information about which (if any) spacy models are installed.
  - Recent changes to spacy, including a warning message, will also make model problems more apparent.
- Added an `ngrams` parameter to `keyterms.sgrank()`, allowing for more flexibility in specifying valid keyterm candidates for the algorithm. (PR #75)
- Dropped dependency on `fuzzywuzzy` package, replacing usage of `fuzz.token_sort_ratio()` with a textacy equivalent in order to avoid license incompatibilities. As a bonus, the new code seems to perform faster! (Issue #62)
  - Note: Outputs are now floats in [0.0, 1.0], consistent with other similarity functions, whereas before outputs were ints in [0, 100]. This has implications for `match_threshold` values passed to `similarity.jaccard()`; a warning is displayed and the conversion is performed automatically, for now.
- A MANIFEST.in file was added to include docs, tests, and distribution files in the source distribution. This is just good practice. (PR #65)

Fixed:

- Known acronym-definition pairs are now properly handled in `extract.acronyms_and_definitions()` (Issue #61)
- WikiReader no longer crashes on null page element content while parsing (PR #64)
- Fixed a rare but perfectly legal edge case exception in `keyterms.sgrank()`, and added a window width sanity check. (Issue #72)
- Fixed assignment of 2-letter language codes to `Doc` and `Corpus` objects when the lang parameter is specified as a full spacy model name.

- Replaced several leftover print statements with proper logging functions.

Contributors:

Big thanks to @oroszgy, @rolando, @covuworie, and @RolandColored for the pull requests!

4.5.18 0.3.3 (2017-02-10)

New and Changed:

- Added a consistent `normalize` param to functions and methods that require token/span text normalization. Typically, it takes one of the following values: 'lemma' to lemmatize tokens, 'lower' to lowercase tokens, False-y to *not* normalize tokens, or a function that converts a spacy token or span into a string, in whatever way the user prefers (e.g. `spacy_utils.normalized_str()`).
  - Functions modified to use this param: `Doc.to_bag_of_terms()`, `Doc.to_bag_of_words()`, `Doc.to_terms_list()`, `Doc.to_semantic_network()`, `Corpus.word_freqs()`, `Corpus.word_doc_freqs()`, `keyterms.sgrank()`, `keyterms.textrank()`, `keyterms.singlerank()`, `keyterms.key_terms_from_semantic_network()`, `network.terms_to_semantic_network()`, `network.sents_to_semantic_network()`
- Tweaked `keyterms.sgrank()` for higher quality results and improved internal performance.
- When getting both n-grams and named entities with `Doc.to_terms_list()`, filtering out numeric spans for only one is automatically extended to the other. This prevents unexpected behavior, such as passing `filter_nums=True` but getting numeric named entities back in the terms list.

Fixed:

- `keyterms.sgrank()` no longer crashes if a term is missing from `idfs` mapping. (@jeremybmerrill, issue #53)
- Proper nouns are no longer excluded from consideration as keyterms in `keyterms.sgrank()` and `keyterms.textrank()`. (@jeremybmerrill, issue #53)
- Empty strings are now excluded from consideration as keyterms — a bug inherited from spaCy. (@mlehl88, issue #58)

4.5.19 0.3.2 (2016-11-15)

New and Changed:

- Preliminary inclusion of custom spaCy pipelines
  - updated `load_spacy()` to include explicit path and create_pipeline kwargs, and removed the already-deprecated `load_spacy_pipeline()` function to avoid confusion around spaCy languages and pipelines
  - added `spacy_pipelines` module to hold implementations of custom spaCy pipelines, including a basic one that merges entities into single tokens
  - note: necessarily bumped minimum spaCy version to 1.1.0+
  - see the announcement here: https://explosion.ai/blog/spacy-deep-learning-keras

- To reduce code bloat, made the `matplotlib` dependency optional and dropped the `gensim` dependency
  - to install `matplotlib` at the same time as textacy, do `$ pip install textacy[viz]`
  - bonus: `backports.csv` is now only installed for Py2 users
  - thanks to @mbatchkarov for the request
- Improved performance of `textacy.corpora.WikiReader().texts()`; results should stream faster and have cleaner plaintext content than when they were produced by `gensim`. This *should* also fix a bug reported in Issue #51 by @baisk
- Added a `Corpus.vectors` property that returns a matrix of shape (# documents, vector dim) containing the average word2vec-style vector representation of constituent tokens for all `Doc` s

4.5.20 0.3.1 (2016-10-19)

Changed:

- Updated spaCy dependency to the latest v1.0.1; set a floor on other dependencies' versions to make sure everyone's running reasonably up-to-date code

Fixed:

- Fixed incorrect kwarg in `sgrank` 's call to `extract.ngrams()` (@patcollis34, issue #44)
- Fixed import for `cachetool` 's `hashkey`, which changed in the v2.0 (@gramonov, issue #45)

4.5.21 0.3.0 (2016-08-23)

New and Changed:

- Refactored and streamlined `TextDoc`; changed name to `Doc`
  - simplified init params: `lang` can now be a language code string or an equivalent `spacy.Language` object, and `content` is either a string or `spacy.Doc`; param values and their interactions are better checked for errors and inconsistencies
  - renamed and improved methods transforming the Doc; for example, `.as_bag_of_terms()` is now `.to_bag_of_terms()`, and terms can be returned as integer ids (default) or as strings with absolute, relative, or binary frequencies as weights
  - added performant `.to_bag_of_words()` method, at the cost of less customizability of what gets included in the bag (no stopwords or punctuation); words can be returned as integer ids (default) or as strings with absolute, relative, or binary frequencies as weights
  - removed methods wrapping `extract` functions, in favor of simply calling that function on the Doc (see below for updates to `extract` functions to make this more convenient); for example, `TextDoc.words()` is now `extract.words(Doc)`
  - removed `.term_counts()` method, which was redundant with `Doc.to_bag_of_terms()`
  - renamed `.term_count()` => `.count()`, and checking + caching results is now smarter and faster
- Refactored and streamlined `TextCorpus`; changed name to `Corpus`
  - added init params: can now initialize a `Corpus` with a stream of texts, spacy or textacy Docs, and optional metadatas, analogous to `Doc`; accordingly, removed `.from_texts()` class method

  - refactored, streamlined, *bug-fixed*, and made consistent the process of adding, getting, and removing documents from `Corpus`
    * getting/removing by index is now equivalent to the built-in `list` API: `Corpus[:5]` gets the first 5 `Doc`s, and `del Corpus[:5]` removes the first 5, automatically keeping track of corpus statistics for total # docs, sents, and tokens
    * getting/removing by boolean function is now done via the `.get()` and `.remove()` methods, the latter of which now also correctly tracks corpus stats
    * adding documents is split across the `.add_text()`, `.add_texts()`, and `.add_doc()` methods for performance and clarity reasons
  - added `.word_freqs()` and `.word_doc_freqs()` methods for getting a mapping of word (int id or string) to global weight (absolute, relative, binary, or inverse frequency); akin to a vectorized representation (see: `textacy.vsm`) but in non-vectorized form, which can be useful
  - removed `.as_doc_term_matrix()` method, which was just wrapping another function; so, instead of `corpus.as_doc_term_matrix((doc.as_terms_list() for doc in corpus))`, do `textacy.vsm.doc_term_matrix((doc.to_terms_list(as_strings=True) for doc in corpus))`
- Updated several `extract` functions
  - almost all now accept either a `textacy.Doc` or `spacy.Doc` as input
  - renamed and improved parameters for filtering for or against certain POS or NE types; for example, `good_pos_tags` is now `include_pos`, and will accept either a single POS tag as a string or a set of POS tags to filter for; same goes for `exclude_pos`, and analogously `include_types`, and `exclude_types`
- Updated corpora classes for consistency and added flexibility
  - enforced a consistent API: `.texts()` for a stream of plain text documents and `.records()` for a stream of dicts containing both text and metadata
  - added filtering options for `RedditReader`, e.g. by date or subreddit, consistent with other corpora (similar tweaks to `WikiReader` may come later, but it's slightly more complicated...)
  - added a nicer `repr` for `RedditReader` and `WikiReader` corpora, consistent with other corpora
- Moved `vsm.py` and `network.py` into the top-level of `textacy` and thus removed the `representations` subpackage
  - renamed `vsm.build_doc_term_matrix()` => `vsm.doc_term_matrix()`, because the "build" part of it is obvious
- Renamed `distance.py` => `similarity.py`; all returned values are now similarity metrics in the interval [0, 1], where higher values indicate higher similarity
- Renamed `regexes_etc.py` => `constants.py`, without additional changes
- Renamed `fileio.utils.split_content_and_metadata()` => `fileio.utils.split_record_fields()`, without further changes (except for tweaks to the docstring)
- Added functions to read and write delimited file formats: `fileio.read_csv()` and `fileio.write_csv()`, where the delimiter can be any valid one-char string; gzip/bzip/lzma compression is handled automatically when available
- Added better and more consistent docstrings and usage examples throughout the code base

4.5.22 0.2.8 (2016-08-03)

New:

- Added two new corpora!
  - the CapitolWords corpus: a collection of 11k speeches (~7M tokens) 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
  - the SupremeCourt corpus: a collection of 8.4k court cases (~71M tokens) decided by the U.S. Supreme Court from 1946 through 2016, with metadata on subject matter categories, ideology, and voting patterns
  - **DEPRECATED:** the Bernie and Hillary corpus, which is a small subset of CapitolWords that can be easily recreated by filtering CapitolWords by `speaker_name={'Bernie Sanders', 'Hillary Clinton'}`

Changed:

- Refactored and improved `fileio` subpackage
  - moved shared (read/write) functions into separate `fileio.utils` module
  - almost all read/write functions now use `fileio.utils.open_sesame()`, enabling seamless fileio for uncompressed or gzip, bz2, and lzma compressed files; relative/user-home-based paths; and missing intermediate directories. NOTE: certain file mode / compression pairs simply don't work (this is Python's fault), so users may run into exceptions; in Python 3, you'll almost always want to use text mode ('wt' or 'rt'), but in Python 2, users can't read or write compressed files in text mode, only binary mode ('wb' or 'rb')
  - added options for writing json files (matching stdlib's `json.dump()`) that can help save space
  - `fileio.utils.get_filenames()` now matches for/against a regex pattern rather than just a contained substring; using the old params will now raise a deprecation warning
  - **BREAKING:** `fileio.utils.split_content_and_metadata()` now has `itemwise=False` by default, rather than `itemwise=True`, which means that splitting multi-document streams of content and metadata into parallel iterators is now the default action
  - added `compression` param to `TextCorpus.save()` and `.load()` to optionally write metadata json file in compressed form
  - moved `fileio.write_conll()` functionality to `export.doc_to_conll()`, which converts a spaCy doc into a ConLL-U formatted string; writing that string to disk would require a separate call to `fileio.write_file()`
- Cleaned up deprecated/bad Py2/3 `compat` imports, and added better functionality for Py2/3 strings
  - now `compat.unicode_type` used for text data, `compat.bytes_type` for binary data, and `compat.string_types` for when either will do
  - also added `compat.unicode_to_bytes()` and `compat.bytes_to_unicode()` functions, for converting between string types

Fixed:

- Fixed document(s) removal from `TextCorpus` objects, including correct decrementing of `.n_docs`, `.n_sents`, and `.n_tokens` attributes (@michelleful #29)
- Fixed OSError being incorrectly raised in `fileio.open_sesame()` on missing files
- `lang` parameter in `TextDoc` and `TextCorpus` can now be unicode *or* bytes, which was bug-like

4.5.23 0.2.5 (2016-07-14)

Fixed:

- Added (missing) `pyemd` and `python-levenshtein` dependencies to requirements and setup files
- Fixed bug in `data.load_depechemood()` arising from the Py2 `csv` module's inability to take unicode as input (thanks to @robclewley, issue #25)

4.5.24 0.2.4 (2016-07-14)

New and Changed:

- New features for `TextDoc` and `TextCorpus` classes
  - added `.save()` methods and `.load()` classmethods, which allows for fast serialization of parsed documents/corpora and associated metadata to/from disk — with an important caveat: if `spacy.Vocab` object used to serialize and deserialize is not the same, there will be problems, making this format useful as short-term but not long-term storage
  - `TextCorpus` may now be instantiated with an already-loaded spaCy pipeline, which may or may not have all models loaded; it can still be instantiated using a language code string ('en', 'de') to load a spaCy pipeline that includes all models by default
  - `TextDoc` methods wrapping `extract` and `keyterms` functions now have full documentation rather than forwarding users to the wrapped functions themselves; more irritating on the dev side, but much less irritating on the user side :)
- Added a `distance.py` module containing several document, set, and string distance metrics
  - word movers: document distance as distance between individual words represented by word2vec vectors, normalized
  - "word2vec": token, span, or document distance as cosine distance between (average) word2vec representations, normalized
  - jaccard: string or set(string) distance as intersection / overlap, normalized, with optional fuzzy-matching across set members
  - hamming: distance between two strings as number of substitutions, optionally normalized
  - levenshtein: distance between two strings as number of substitions, deletions, and insertions, optionally normalized (and removed a redundant function from the still-orphaned `math_utils.py` module)
  - jaro-winkler: distance between two strings with variable prefix weighting, normalized
- Added `most_discriminating_terms()` function to `keyterms` module to take a collection of documents split into two exclusive groups and compute the most discriminating terms for group1-and-not-group2 as well as group2-and-not-group1

Fixed:

- fixed variable name error in docs usage example (thanks to @licyeus, PR #23)

4.5.25 0.2.3 (2016-06-20)

New and Changed:

- Added `corpora.RedditReader()` class for streaming Reddit comments from disk, with `.texts()` method for a stream of plaintext comments and `.comments()` method for a stream of structured comments as dicts, with basic filtering by text length and limiting the number of comments returned
- Refactored functions for streaming Wikipedia articles from disk into a `corpora.WikiReader()` class, with `.texts()` method for a stream of plaintext articles and `.pages()` method for a stream of structured pages as dicts, with basic filtering by text length and limiting the number of pages returned
- Updated README and docs with a more comprehensive — and correct — usage example; also added tests to ensure it doesn't get stale
- Updated requirements to latest version of spaCy, as well as added matplotlib for `viz`

Fixed:

- `textacy.preprocess.preprocess_text()` is now, once again, imported at the top level, so easily reachable via `textacy.preprocess_text()` (@bretdabaker #14)
- `viz` subpackage now included in the docs' API reference
- missing dependencies added into `setup.py` so pip install handles everything for folks

4.5.26 0.2.2 (2016-05-05)

New and Changed:

- Added a `viz` subpackage, with two types of plots (so far):
  - `viz.draw_termite_plot()`, typically used to evaluate and interpret topic models; conveniently accessible from the `tm.TopicModel` class
  - `viz.draw_semantic_network()` for visualizing networks such as those output by `representations.network`
- Added a "Bernie & Hillary" corpus with 3000 congressional speeches made by Bernie Sanders and Hillary Clinton since 1996
  - `corpora.fetch_bernie_and_hillary()` function automatically downloads to and loads from disk this corpus
- Modified `data.load_depechemood` function, now downloads data from GitHub source if not found on disk
- Removed `resources/` directory from GitHub, hence all the downloadin'
- Updated to spaCy v0.100.7
  - German is now supported! although some functionality is English-only

  - added `textacy.load_spacy()` function for loading spaCy packages, taking advantage of the new `spacy.load()` API; added a DeprecationWarning for `textacy.data.load_spacy_pipeline()`
  - proper nouns' and pronouns' `.pos_` attributes are now correctly assigned 'PROPN' and 'PRON'; hence, modified `regexes_etc.POS_REGEX_PATTERNS['en']` to include 'PROPN'
  - modified `spacy_utils.preserve_case()` to check for language-agnostic 'PROPN' POS rather than English-specific 'NNP' and 'NNPS' tags
- Added `text_utils.clean_terms()` function for cleaning up a sequence of single- or multi-word strings by stripping leading/trailing junk chars, handling dangling parens and odd hyphenation, etc.

Fixed:

- `textstats.readability_stats()` now correctly gets the number of words in a doc from its generator function (@gryBox #8)
- removed NLTK dependency, which wasn't actually required
- `text_utils.detect_language()` now warns via `logging` rather than a `print()` statement
- `fileio.write_conll()` documentation now correctly indicates that the filename param is not optional

4.5.27 0.2.0 (2016-04-11)

New and Changed:

- Added `representations` subpackage; includes modules for network and vector space model (VSM) document and corpus representations
  - Document-term matrix creation now takes documents represented as a list of terms (rather than as spaCy Docs); splits the tokenization step from vectorization for added flexibility
  - Some of this functionality was refactored from existing parts of the package
- Added `tm` (topic modeling) subpackage, with a main `TopicModel` class for training, applying, persisting, and interpreting NMF, LDA, and LSA topic models through a single interface
- Various improvements to `TextDoc` and `TextCorpus` classes
  - `TextDoc` can now be initialized from a spaCy Doc
  - Removed caching from `TextDoc`, because it was a pain and weird and probably not all that useful
  - `extract`-based methods are now generators, like the functions they wrap
  - Added `.as_semantic_network()` and `.as_terms_list()` methods to `TextDoc`
  - `TextCorpus.from_texts()` now takes advantage of multithreading via spaCy, if available, and document metadata can be passed in as a paired iterable of dicts
- Added read/write functions for sparse scipy matrices
- Added `fileio.read.split_content_and_metadata()` convenience function for splitting (text) content from associated metadata when reading data from disk into a `TextDoc` or `TextCorpus`
- Renamed `fileio.read.get_filenames_in_dir()` to `fileio.read.get_filenames()` and added functionality for matching/ignoring files by their names, file extensions, and ignoring invisible files
- Rewrote `export.docs_to_gensim()`, now significantly faster

- Imports in `__init__.py` files for main and subpackages now explicit

Fixed:

- `textstats.readability_stats()` no longer filters out stop words (@henningko #7)
- Wikipedia article processing now recursively removes nested markup
- `extract.ngrams()` now filters out ngrams with any space-only tokens
- functions with `include_nps` kwarg changed to `include_ncs`, to match the renaming of the associated function from `extract.noun_phrases()` to `extract.noun_chunks()`

4.5.28 0.1.4 (2016-02-26)

New:

- Added `corpora` subpackage with `wikipedia.py` module; functions for streaming pages from a Wikipedia db dump as plain text or structured data
- Added `fileio` subpackage with functions for reading/writing content from/to disk in common formats
  - JSON formats, both standard and streaming-friendly
  - text, optionally compressed
  - spacy documents to/from binary

4.5.29 0.1.3 (2016-02-22)

New:

- Added `export.py` module for exporting textacy/spacy objects into "third-party" formats; so far, just gensim and conll-u
- Added `compat.py` module for Py2/3 compatibility hacks
- Added `TextDoc.merge()` and `spacy_utils.merge_spans()` for merging spans into single tokens within a `spacy.Doc`, uses Spacy's recent implementation

Changed:

- Renamed `extract.noun_phrases()` to `extract.noun_chunks()` to match Spacy's API
- Changed extract functions to generators, rather than returning lists

Fixed:

- Whitespace tokens now always filtered out of `extract.words()` lists
- Some Py2/3 str/unicode issues fixed
- Broken tests in `test_extract.py` no longer broken

PYTHON MODULE INDEX

t

textacy.augmentation.augmenter, 130
textacy.augmentation.transforms, 132
textacy.augmentation.utils, 134
textacy.cache, 140
textacy.corpus, 27
textacy.datasets.capitol_words, 37
textacy.datasets.imdb, 51
textacy.datasets.oxford_text_archive, 49
textacy.datasets.reddit_comments, 46
textacy.datasets.supreme_court, 40
textacy.datasets.udhr, 53
textacy.datasets.wikimedia, 43
textacy.errors, 140
textacy.extensions, 34
textacy.extract.acros, 72
textacy.extract.basics, 65
textacy.extract.keyterms, 73
textacy.extract.keyterms.scake, 75
textacy.extract.keyterms.sgrank, 75
textacy.extract.keyterms.textrank, 73
textacy.extract.keyterms.yake, 74
textacy.extract.kwic, 72
textacy.extract.matches, 68
textacy.extract.triples, 70
textacy.extract.utils, 76
textacy.io.csv, 121
textacy.io.http, 124
textacy.io.json, 119
textacy.io.matrix, 122
textacy.io.spacy, 123
textacy.io.text, 118
textacy.io.utils, 125
textacy.lang_id.lang_identifier, 135
textacy.preprocessing.normalize, 62
textacy.preprocessing.pipeline, 61
textacy.preprocessing.remove, 63
textacy.preprocessing.replace, 64
textacy.representations.matrix_utils, 109
textacy.representations.network, 94
textacy.representations.sparse_vec, 97
textacy.representations.vectorizers, 100
textacy.resources.concept_net, 55
textacy.resources.depeche_mood, 57
textacy.similarity.edits, 89
textacy.similarity.hybrid, 92
textacy.similarity.sequences, 92
textacy.similarity.tokens, 90
textacy.spacier.core, 26
textacy.spacier.utils, 140
textacy.text_stats.api, 80
textacy.text_stats.basics, 83
textacy.text_stats.components, 88
textacy.text_stats.readability, 85
textacy.tm.topic_model, 112
textacy.types, 139
textacy.utils, 138
textacy.viz.network, 129
textacy.viz.termite, 127

INDEX

A

accents() (*in module textacy.preprocessing.remove*), 63
acronyms() (*in module textacy.extract.acros*), 72
acronyms_and_definitions() (*in module textacy.extract.acros*), 72
add() (*textacy.corpus.Corpus method*), 29
add_doc() (*textacy.corpus.Corpus method*), 31
add_docs() (*textacy.corpus.Corpus method*), 31
add_record() (*textacy.corpus.Corpus method*), 30
add_records() (*textacy.corpus.Corpus method*), 30
add_text() (*textacy.corpus.Corpus method*), 30
add_texts() (*textacy.corpus.Corpus method*), 30
agg_metadata() (*textacy.corpus.Corpus method*), 33
aggregate_term_variants() (*in module textacy.extract.utils*), 77
antonyms() (*textacy.resources.concept_net.ConceptNet property*), 56
apply_idf_weighting() (*in module textacy.representations.matrix_utils*), 111
apply_transforms() (*textacy.augmentation.augmenter.Augmenter method*), 131
Augmenter (*class in textacy.augmentation.augmenter*), 130
AugTok (*class in textacy.augmentation.utils*), 134
authors (*textacy.datasets.oxford_text_archive.OxfordTextArchive attribute*), 50
automated_readability_index() (*in module textacy.text_stats.readability*), 85
automated_readability_index() (*textacy.text_stats.api.TextStats property*), 82
automatic_arabic_readability_index() (*in module textacy.text_stats.readability*), 85
automatic_arabic_readability_index() (*textacy.text_stats.api.TextStats property*), 82

B

bag() (*in module textacy.similarity.tokens*), 92
brackets() (*in module textacy.preprocessing.remove*), 63
build_cooccurrence_network() (*in module textacy.representations.network*), 94
build_doc_term_matrix() (*in module textacy.representations.sparse_vec*), 97
build_grp_term_matrix() (*in module textacy.representations.sparse_vec*), 99
build_similarity_network() (*in module textacy.representations.network*), 95
bullet_points() (*in module textacy.preprocessing.normalize*), 62

C

Candidate (*class in textacy.extract.keyterms.sgrank*), 75
CapitolWords (*class in textacy.datasets.capitol_words*), 37
chambers (*textacy.datasets.capitol_words.CapitolWords attribute*), 38
character_ngrams() (*in module textacy.similarity.edits*), 90
classes (*textacy.lang_id.lang_identifier.LangIdentifier attribute*), 136
clean_term_strings() (*in module textacy.extract.utils*), 77
clear() (*in module textacy.cache*), 140
coerce_content_type() (*in module textacy.io.utils*), 126
coleman_liau_index() (*in module textacy.text_stats.readability*), 85
coleman_liau_index() (*textacy.text_stats.api.TextStats property*), 82
ConceptNet (*class in textacy.resources.concept_net*), 55
congresses (*textacy.datasets.capitol_words.CapitolWords attribute*), 38
content (*textacy.extract.triples.DQTriple attribute*), 70
Corpus (*class in textacy.corpus*), 27
cosine() (*in module textacy.similarity.tokens*), 91
count (*textacy.extract.keyterms.sgrank.Candidate attribute*), 75
cue (*textacy.extract.triples.DQTriple attribute*), 70
cue (*textacy.extract.triples.SSSTriple attribute*), 70

currency_symbols() (*in module textacy.preprocessing.replace*), 64

D

decision_directions (*textacy.datasets.supreme_court.SupremeCourt attribute*), 41
default() (*textacy.io.json.ExtendedJSONEncoder method*), 120
delete_chars() (*in module textacy.augmentation.transforms*), 134
delete_words() (*in module textacy.augmentation.transforms*), 133
DepecheMood (*class in textacy.resources.depeche_mood*), 58
deprecated() (*in module textacy.utils*), 138
direct_quotations() (*in module textacy.extract.triples*), 71
docs (*textacy.corpus.Corpus attribute*), 29
download() (*textacy.datasets.capitol_words.CapitolWords method*), 38
download() (*textacy.datasets.imdb.IMDB method*), 52
download() (*textacy.datasets.oxford_text_archive.OxfordTextArchive method*), 50
download() (*textacy.datasets.reddit_comments.RedditComments method*), 48
download() (*textacy.datasets.supreme_court.SupremeCourt method*), 42
download() (*textacy.datasets.udhr.UDHR method*), 54
download() (*textacy.datasets.wikimedia.Wikimedia method*), 43
download() (*textacy.lang_id.lang_identifier.LangIdentifier method*), 137
download() (*textacy.resources.concept_net.ConceptNet method*), 56
download() (*textacy.resources.depeche_mood.DepecheMood method*), 60
download_file() (*in module textacy.io.utils*), 127
DQTriple (*class in textacy.extract.triples*), 70
draw_semantic_network() (*in module textacy.viz.network*), 129
draw_termite_plot() (*in module textacy.viz.termite*), 127

E

emails() (*in module textacy.preprocessing.replace*), 64
emojis() (*in module textacy.preprocessing.replace*), 64
entities() (*in module textacy.extract.basics*), 67
entity (*textacy.extract.triples.SSSTriple attribute*), 70
entropy() (*in module textacy.text_stats.basics*), 85
entropy() (*textacy.text_stats.api.TextStats property*), 82
expand_noun() (*in module textacy.extract.triples*), 72
expand_verb() (*in module textacy.extract.triples*), 72
ExtendedJSONEncoder (*class in textacy.io.json*), 120

F

filepath() (*textacy.datasets.capitol_words.CapitolWords property*), 38
filepath() (*textacy.datasets.supreme_court.SupremeCourt property*), 42
filepath() (*textacy.datasets.wikimedia.Wikimedia property*), 43
filepath() (*textacy.resources.concept_net.ConceptNet property*), 56
filepath() (*textacy.resources.depeche_mood.DepecheMood property*), 60
filepaths() (*textacy.datasets.reddit_comments.RedditComments property*), 48
filter_terms_by_df() (*in module textacy.representations.matrix_utils*), 111
filter_terms_by_ic() (*in module textacy.representations.matrix_utils*), 112
fit() (*textacy.representations.vectorizers.GroupVectorizer method*), 108
fit() (*textacy.representations.vectorizers.Vectorizer method*), 104
fit_transform() (*textacy.representations.vectorizers.GroupVectorizer method*), 108
fit_transform() (*textacy.representations.vectorizers.Vectorizer method*), 104
flesch_kincaid_grade_level() (*in module textacy.text_stats.readability*), 86
flesch_kincaid_grade_level() (*textacy.text_stats.api.TextStats property*), 82
flesch_reading_ease() (*in module textacy.text_stats.readability*), 86
flesch_reading_ease() (*textacy.text_stats.api.TextStats property*), 82
fragment (*textacy.extract.triples.SSSTriple attribute*), 70
full_date_range (*textacy.datasets.capitol_words.CapitolWords attribute*), 38
full_date_range (*textacy.datasets.oxford_text_archive.OxfordTextArchive attribute*), 50
full_date_range (*textacy.datasets.reddit_comments.RedditComments attribute*), 48
full_date_range (*textacy.datasets.supreme_court.SupremeCourt attribute*), 41

full_rating_range (*textacy.datasets.imdb.IMDB attribute*), 52

G

get() (*textacy.corpus.Corpus method*), 31
get_antonyms() (*textacy.resources.concept_net.ConceptNet method*), 56
get_char_weights() (*in module textacy.augmentation.utils*), 135
get_config() (*in module textacy.utils*), 138
get_doc_extensions() (*in module textacy.extensions*), 36
get_doc_freqs() (*in module textacy.representations.matrix_utils*), 109
get_doc_lengths() (*in module textacy.representations.matrix_utils*), 110
get_doc_topic_matrix() (*textacy.tm.topic_model.TopicModel method*), 114
get_emotional_valence() (*textacy.resources.depeche_mood.DepecheMood method*), 60
get_filename_from_url() (*in module textacy.io.utils*), 127
get_filepaths() (*in module textacy.io.utils*), 126
get_filtered_topn_terms() (*in module textacy.extract.utils*), 78
get_hyponyms() (*textacy.resources.concept_net.ConceptNet method*), 57
get_information_content() (*in module textacy.representations.matrix_utils*), 110
get_inverse_doc_freqs() (*in module textacy.representations.matrix_utils*), 110
get_kwargs_for_func() (*in module textacy.utils*), 139
get_longest_subsequence_candidates() (*in module textacy.extract.utils*), 77
get_main_verbs_of_sent() (*in module textacy.spacier.utils*), 141
get_meronyms() (*textacy.resources.concept_net.ConceptNet method*), 57
get_meta() (*in module textacy.extensions*), 34
get_ngram_candidates() (*in module textacy.extract.utils*), 78
get_normalized_text() (*in module textacy.spacier.utils*), 141
get_objects_of_verb() (*in module textacy.spacier.utils*), 141
get_pattern_matching_candidates() (*in module textacy.extract.utils*), 78
get_preview() (*in module textacy.extensions*), 34
get_span_for_compound_noun() (*in module textacy.spacier.utils*), 141
get_span_for_verb_auxiliaries() (*in module textacy.spacier.utils*), 141
get_subjects_of_verb() (*in module textacy.spacier.utils*), 141
get_synonyms() (*textacy.resources.concept_net.ConceptNet method*), 57
get_term_freqs() (*in module textacy.representations.matrix_utils*), 109
GroupVectorizer (*class in textacy.representations.vectorizers*), 105
grps_list() (*textacy.representations.vectorizers.GroupVectorizer property*), 108
gulpease_index() (*in module textacy.text_stats.readability*), 86
gulpease_index() (*textacy.text_stats.api.TextStats property*), 82
gunning_fog_index() (*in module textacy.text_stats.readability*), 86
gunning_fog_index() (*textacy.text_stats.api.TextStats property*), 82

H

hamming() (*in module textacy.similarity.edits*), 90
hashtags() (*in module textacy.preprocessing.replace*), 64
html_tags() (*in module textacy.preprocessing.remove*), 63
hyphenated_words() (*in module textacy.preprocessing.normalize*), 62
hyponyms() (*textacy.resources.concept_net.ConceptNet property*), 57

I

id_to_grp() (*textacy.representations.vectorizers.GroupVectorizer property*), 108
id_to_term() (*textacy.representations.vectorizers.Vectorizer property*), 104
identify_lang() (*in module textacy.lang_id.lang_identifier*), 137
identify_lang() (*textacy.lang_id.lang_identifier.LangIdentifier method*), 137
identify_topn_langs() (*in module textacy.lang_id.lang_identifier*), 137
identify_topn_langs() (*textacy.lang_id.lang_identifier.LangIdentifier method*), 137
idx (*textacy.extract.keyterms.sgrank.Candidate attribute*), 75
IMDB (*class in textacy.datasets.imdb*), 51

insert_chars() (*in module textacy.augmentation.transforms*), 133
insert_word_synonyms() (*in module textacy.augmentation.transforms*), 132
is_acronym() (*in module textacy.extract.acros*), 72
is_record() (*in module textacy.utils*), 138
is_word (*textacy.augmentation.utils.AugTok attribute*), 134
issue_area_codes (*textacy.datasets.supreme_court.SupremeCourt attribute*), 41
issue_codes (*textacy.datasets.supreme_court.SupremeCourt attribute*), 42

J

jaccard() (*in module textacy.similarity.tokens*), 91
jaro() (*in module textacy.similarity.edits*), 90

K

keyword_in_context() (*in module textacy.extract.kwic*), 73

L

lang (*textacy.corpus.Corpus attribute*), 29
LangIdentifier (*class in textacy.lang_id.lang_identifier*), 136
langs (*textacy.datasets.udhr.UDHR attribute*), 54
length (*textacy.extract.keyterms.sgrank.Candidate attribute*), 75
levenshtein() (*in module textacy.similarity.edits*), 90
lix() (*in module textacy.text_stats.readability*), 87
lix() (*textacy.text_stats.api.TextStats property*), 82
load() (*textacy.corpus.Corpus class method*), 33
load_hyphenator() (*in module textacy.text_stats.api*), 83
load_model() (*textacy.lang_id.lang_identifier.LangIdentifier method*), 136
load_spacy_lang() (*in module textacy.spacier.core*), 26
LRU_CACHE (*in module textacy.cache*), 140

M

make_doc_from_text_chunks() (*in module textacy.spacier.utils*), 140
make_pipeline() (*in module textacy.preprocessing.pipeline*), 61
make_spacy_doc() (*in module textacy.spacier.core*), 26
matching_subsequences_ratio() (*in module textacy.similarity.sequences*), 92
merge_spans() (*in module textacy.spacier.utils*), 140
meronyms() (*textacy.resources.concept_net.ConceptNet property*), 57
meta (*textacy.types.Record attribute*), 140
metadata() (*textacy.datasets.oxford_text_archive.OxfordTextArchive property*), 50
model (*textacy.lang_id.lang_identifier.LangIdentifier attribute*), 136
module
    textacy.augmentation.augmenter, 130
    textacy.augmentation.transforms, 132
    textacy.augmentation.utils, 134
    textacy.cache, 140
    textacy.corpus, 27
    textacy.datasets.capitol_words, 37
    textacy.datasets.imdb, 51
    textacy.datasets.oxford_text_archive, 49
    textacy.datasets.reddit_comments, 46
    textacy.datasets.supreme_court, 40
    textacy.datasets.udhr, 53
    textacy.datasets.wikimedia, 43
    textacy.errors, 140
    textacy.extensions, 34
    textacy.extract.acros, 72
    textacy.extract.basics, 65
    textacy.extract.keyterms, 73
    textacy.extract.keyterms.scake, 75
    textacy.extract.keyterms.sgrank, 75
    textacy.extract.keyterms.textrank, 73
    textacy.extract.keyterms.yake, 74
    textacy.extract.kwic, 72
    textacy.extract.matches, 68
    textacy.extract.triples, 70
    textacy.extract.utils, 76
    textacy.io.csv, 121
    textacy.io.http, 124
    textacy.io.json, 119
    textacy.io.matrix, 122
    textacy.io.spacy, 123
    textacy.io.text, 118
    textacy.io.utils, 125
    textacy.lang_id.lang_identifier, 135
    textacy.preprocessing.normalize, 62
    textacy.preprocessing.pipeline, 61
    textacy.preprocessing.remove, 63
    textacy.preprocessing.replace, 64
    textacy.representations.matrix_utils, 109
    textacy.representations.network, 94
    textacy.representations.sparse_vec, 97
    textacy.representations.vectorizers, 100
    textacy.resources.concept_net, 55
    textacy.resources.depeche_mood, 57

textacy.similarity.edits, 89
textacy.similarity.hybrid, 92
textacy.similarity.sequences, 92
textacy.similarity.tokens, 90
textacy.spacier.core, 26
textacy.spacier.utils, 140
textacy.text_stats.api, 80
textacy.text_stats.basics, 83
textacy.text_stats.components, 88
textacy.text_stats.readability, 85
textacy.tm.topic_model, 112
textacy.types, 139
textacy.utils, 138
textacy.viz.network, 129
textacy.viz.termite, 127
monge_elkan() (*in module textacy.similarity.hybrid*), 93
mu_legibility_index() (*in module textacy.text_stats.readability*), 87
mu_legibility_index() (*textacy.text_stats.api.TextStats property*), 83

N

n_chars() (*in module textacy.text_stats.basics*), 84
n_chars() (*textacy.text_stats.api.TextStats property*), 81
n_chars_per_word() (*in module textacy.text_stats.basics*), 83
n_chars_per_word() (*textacy.text_stats.api.TextStats property*), 81
n_docs (*textacy.corpus.Corpus attribute*), 29
n_long_words() (*in module textacy.text_stats.basics*), 84
n_long_words() (*textacy.text_stats.api.TextStats property*), 81
n_monosyllable_words() (*in module textacy.text_stats.basics*), 84
n_monosyllable_words() (*textacy.text_stats.api.TextStats property*), 81
n_polysyllable_words() (*in module textacy.text_stats.basics*), 84
n_polysyllable_words() (*textacy.text_stats.api.TextStats property*), 81
n_sents (*textacy.corpus.Corpus attribute*), 29
n_sents() (*in module textacy.text_stats.basics*), 84
n_sents() (*textacy.text_stats.api.TextStats property*), 81
n_syllables() (*in module textacy.text_stats.basics*), 84
n_syllables() (*textacy.text_stats.api.TextStats property*), 81
n_syllables_per_word() (*in module textacy.text_stats.basics*), 84
n_syllables_per_word() (*textacy.text_stats.api.TextStats property*), 81
n_tokens (*textacy.corpus.Corpus attribute*), 29
n_unique_words() (*in module textacy.text_stats.basics*), 83
n_unique_words() (*textacy.text_stats.api.TextStats property*), 81
n_words() (*in module textacy.text_stats.basics*), 83
n_words() (*textacy.text_stats.api.TextStats property*), 81
ngrams() (*in module textacy.extract.basics*), 66
noun_chunks() (*in module textacy.extract.basics*), 67
numbers() (*in module textacy.preprocessing.replace*), 64

O

object (*textacy.extract.triples.SVOTriple attribute*), 70
open_sesame() (*in module textacy.io.utils*), 125
opinion_author_codes (*textacy.datasets.supreme_court.SupremeCourt attribute*), 41
OxfordTextArchive (*class in textacy.datasets.oxford_text_archive*), 49

P

perspicuity_index() (*in module textacy.text_stats.readability*), 87
perspicuity_index() (*textacy.text_stats.api.TextStats property*), 83
phone_numbers() (*in module textacy.preprocessing.replace*), 64
pos (*textacy.augmentation.utils.AugTok attribute*), 134
preserve_case() (*in module textacy.spacier.utils*), 141
print_markdown() (*in module textacy.utils*), 138
punctuation() (*in module textacy.preprocessing.remove*), 64

Q

quotation_marks() (*in module textacy.preprocessing.normalize*), 62

R

rank_nodes_by_bestcoverage() (*in module textacy.representations.network*), 96
rank_nodes_by_divrank() (*in module textacy.representations.network*), 97
rank_nodes_by_pagerank() (*in module textacy.representations.network*), 96
read_csv() (*in module textacy.io.csv*), 121
read_http_stream() (*in module textacy.io.http*), 124
read_json() (*in module textacy.io.json*), 119
read_json_mash() (*in module textacy.io.json*), 119

read_spacy_docs() (*in module textacy.io.spacy*), 123
read_sparse_matrix() (*in module textacy.io.matrix*), 122
read_text() (*in module textacy.io.text*), 118
Record (*class in textacy.types*), 139
records() (*textacy.datasets.capitol_words.CapitolWords method*), 39
records() (*textacy.datasets.imdb.IMDB method*), 53
records() (*textacy.datasets.oxford_text_archive.OxfordTextArchive method*), 51
records() (*textacy.datasets.reddit_comments.RedditComments method*), 48
records() (*textacy.datasets.supreme_court.SupremeCourt method*), 42
records() (*textacy.datasets.udhr.UDHR method*), 54
records() (*textacy.datasets.wikimedia.Wikimedia method*), 44
RedditComments (*class in textacy.datasets.reddit_comments*), 47
regex_matches() (*in module textacy.extract.matches*), 70
remove() (*textacy.corpus.Corpus method*), 31
remove_doc_extensions() (*in module textacy.extensions*), 36
repeating_chars() (*in module textacy.preprocessing.normalize*), 62

S

save() (*textacy.corpus.Corpus method*), 33
save_model() (*textacy.lang_id.lang_identifier.LangIdentifier method*), 136
scake() (*in module textacy.extract.keyterms.scake*), 75
semistructured_statements() (*in module textacy.extract.triples*), 71
set_doc_extensions() (*in module textacy.extensions*), 36
set_meta() (*in module textacy.extensions*), 34
sgrank() (*in module textacy.extract.keyterms.sgrank*), 75
smog_index() (*in module textacy.text_stats.readability*), 87
smog_index() (*textacy.text_stats.api.TextStats property*), 83
sorensen_dice() (*in module textacy.similarity.tokens*), 91
spacy_lang (*textacy.corpus.Corpus attribute*), 29
speaker (*textacy.extract.triples.DQTriple attribute*), 70
speaker_names (*textacy.datasets.capitol_words.CapitolWords attribute*), 38
speaker_parties (*textacy.datasets.capitol_words.CapitolWords attribute*), 38
split_records() (*in module textacy.io.utils*), 126
SSSTriple (*class in textacy.extract.triples*), 70
subject (*textacy.extract.triples.SVOTriple attribute*), 70
subject_verb_object_triples() (*in module textacy.extract.triples*), 70
substitute_chars() (*in module textacy.augmentation.transforms*), 133
substitute_word_synonyms() (*in module textacy.augmentation.transforms*), 132
SupremeCourt (*class in textacy.datasets.supreme_court*), 40
SVOTriple (*class in textacy.extract.triples*), 70
swap_chars() (*in module textacy.augmentation.transforms*), 134
swap_words() (*in module textacy.augmentation.transforms*), 132
synonyms() (*textacy.resources.concept_net.ConceptNet property*), 57
syns (*textacy.augmentation.utils.AugTok attribute*), 134

T

termite_df_plot() (*in module textacy.viz.termite*), 128
termite_plot() (*textacy.tm.topic_model.TopicModel method*), 116
terms() (*in module textacy.extract.basics*), 68
terms_list() (*textacy.representations.vectorizers.Vectorizer property*), 104
terms_to_strings() (*in module textacy.extract.utils*), 76
text (*textacy.augmentation.utils.AugTok attribute*), 134
text (*textacy.extract.keyterms.sgrank.Candidate attribute*), 75
text (*textacy.types.Record attribute*), 140
text_to_char_ngrams() (*in module textacy.utils*), 139
textacy.augmentation.augmenter module, 130
textacy.augmentation.transforms module, 132
textacy.augmentation.utils module, 134
textacy.cache module, 140
textacy.corpus module, 27
textacy.datasets.capitol_words module, 37
textacy.datasets.imdb module, 51
textacy.datasets.oxford_text_archive module, 49

textacy.datasets.reddit_comments
    module, 46
textacy.datasets.supreme_court
    module, 40
textacy.datasets.udhr
    module, 53
textacy.datasets.wikimedia
    module, 43
textacy.errors
    module, 140
textacy.extensions
    module, 34
textacy.extract.acros
    module, 72
textacy.extract.basics
    module, 65
textacy.extract.keyterms
    module, 73
textacy.extract.keyterms.scake
    module, 75
textacy.extract.keyterms.sgrank
    module, 75
textacy.extract.keyterms.textrank
    module, 73
textacy.extract.keyterms.yake
    module, 74
textacy.extract.kwic
    module, 72
textacy.extract.matches
    module, 68
textacy.extract.triples
    module, 70
textacy.extract.utils
    module, 76
textacy.io.csv
    module, 121
textacy.io.http
    module, 124
textacy.io.json
    module, 119
textacy.io.matrix
    module, 122
textacy.io.spacy
    module, 123
textacy.io.text
    module, 118
textacy.io.utils
    module, 125
textacy.lang_id.lang_identifier
    module, 135
textacy.preprocessing.normalize
    module, 62
textacy.preprocessing.pipeline
    module, 61
textacy.preprocessing.remove
    module, 63
textacy.preprocessing.replace
    module, 64
textacy.representations.matrix_utils
    module, 109
textacy.representations.network
    module, 94
textacy.representations.sparse_vec
    module, 97
textacy.representations.vectorizers
    module, 100
textacy.resources.concept_net
    module, 55
textacy.resources.depeche_mood
    module, 57
textacy.similarity.edits
    module, 89
textacy.similarity.hybrid
    module, 92
textacy.similarity.sequences
    module, 92
textacy.similarity.tokens
    module, 90
textacy.spacier.core
    module, 26
textacy.spacier.utils
    module, 140
textacy.text_stats.api
    module, 80
textacy.text_stats.basics
    module, 83
textacy.text_stats.components
    module, 88
textacy.text_stats.readability
    module, 85
textacy.tm.topic_model
    module, 112
textacy.types
    module, 139
textacy.utils
    module, 138
textacy.viz.network
    module, 129
textacy.viz.termite
    module, 127
textrank() (*in module textacy.extract.keyterms.textrank*), 73
texts() (*textacy.datasets.capitol_words.CapitolWords method*), 39
texts() (*textacy.datasets.imdb.IMDB method*), 52
texts() (*textacy.datasets.oxford_text_archive.OxfordTextArchive method*), 50

texts() (*textacy.datasets.reddit_comments.RedditComments method*), 48
texts() (*textacy.datasets.supreme_court.SupremeCourt method*), 42
texts() (*textacy.datasets.udhr.UDHR method*), 54
texts() (*textacy.datasets.wikimedia.Wikimedia method*), 44
TextStats (*class in textacy.text_stats.api*), 80
TextStatsComponent (*class in textacy.text_stats.components*), 88
to_aug_toks() (*in module textacy.augmentation.utils*), 134
to_bag_of_terms() (*in module textacy.extensions*), 35
to_bag_of_words() (*in module textacy.extensions*), 34
to_bytes() (*in module textacy.utils*), 138
to_collection() (*in module textacy.utils*), 138
to_path() (*in module textacy.utils*), 138
to_tokenized_text() (*in module textacy.extensions*), 34
to_unicode() (*in module textacy.utils*), 138
token_matches() (*in module textacy.extract.matches*), 69
token_sort_ratio() (*in module textacy.similarity.hybrid*), 92
top_doc_topics() (*textacy.tm.topic_model.TopicModel method*), 115
top_topic_docs() (*textacy.tm.topic_model.TopicModel method*), 115
top_topic_terms() (*textacy.tm.topic_model.TopicModel method*), 114
topic_weights() (*textacy.tm.topic_model.TopicModel method*), 116
TopicModel (*class in textacy.tm.topic_model*), 112
transform() (*textacy.representations.vectorizers.GroupVectorizer method*), 109
transform() (*textacy.representations.vectorizers.Vectorizer method*), 105
tversky() (*in module textacy.similarity.tokens*), 91

U

UDHR (*class in textacy.datasets.udhr*), 53
unicode() (*in module textacy.preprocessing.normalize*), 62
unpack_archive() (*in module textacy.io.utils*), 127
unzip() (*in module textacy.io.utils*), 126
urls() (*in module textacy.preprocessing.replace*), 64
user_handles() (*in module textacy.preprocessing.replace*), 64

V

validate_and_clip_range() (*in module textacy.utils*), 139
validate_set_members() (*in module textacy.utils*), 138
vector_norms() (*textacy.corpus.Corpus property*), 32
Vectorizer (*class in textacy.representations.vectorizers*), 100
vectors() (*textacy.corpus.Corpus property*), 32
verb (*textacy.extract.triples.SVOTriple attribute*), 70
vocabulary_grps (*textacy.representations.vectorizers.GroupVectorizer attribute*), 108
vocabulary_terms (*textacy.representations.vectorizers.GroupVectorizer attribute*), 107
vocabulary_terms (*textacy.representations.vectorizers.Vectorizer attribute*), 104

W

weighting() (*textacy.representations.vectorizers.Vectorizer property*), 105
weights() (*textacy.resources.depeche_mood.DepecheMood property*), 60
whitespace() (*in module textacy.preprocessing.normalize*), 63
wiener_sachtextformel() (*in module textacy.text_stats.readability*), 87
wiener_sachtextformel() (*textacy.text_stats.api.TextStats property*), 83
Wikimedia (*class in textacy.datasets.wikimedia*), 43
Wikinews (*class in textacy.datasets.wikimedia*), 45
Wikipedia (*class in textacy.datasets.wikimedia*), 44
word_counts() (*textacy.corpus.Corpus method*), 32
word_doc_counts() (*textacy.corpus.Corpus method*), 32
words() (*in module textacy.extract.basics*), 66
write_csv() (*in module textacy.io.csv*), 121
write_http_stream() (*in module textacy.io.http*), 124
write_json() (*in module textacy.io.json*), 119
write_spacy_docs() (*in module textacy.io.spacy*), 123
write_sparse_matrix() (*in module textacy.io.matrix*), 122
write_text() (*in module textacy.io.text*), 118
ws (*textacy.augmentation.utils.AugTok attribute*), 134

Y

yake() (*in module textacy.extract.keyterms.yake*), 74