
textblob Documentation

Release 0.16.0

Steven Loria

Apr 26, 2020

Contents

1	Features	3
2	Get it now	5
3	Guide	7
3.1	License	7
3.2	Installation	7
3.3	Tutorial: Quickstart	9
3.4	Tutorial: Building a Text Classification System	14
3.5	Advanced Usage: Overriding Models and the Blobber Class	17
3.6	Extensions	19
3.7	API Reference	20
4	Project info	51
4.1	Changelog	51
4.2	Authors	58
4.3	Contributing guidelines	59
	Python Module Index	63
	Index	65

Release v0.16.0. (*Changelog*)

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

```
from textblob import TextBlob

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

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

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

for sentence in blob.sentences:
    print(sentence.sentiment.polarity)
# 0.060
# -0.341
```

TextBlob stands on the giant shoulders of [NLTK](#) and [pattern](#), and plays nicely with both.

CHAPTER 1

Features

- Noun phrase extraction
- Part-of-speech tagging
- Sentiment analysis
- Classification (Naive Bayes, Decision Tree)
- Tokenization (splitting text into words and sentences)
- Word and phrase frequencies
- Parsing
- n-grams
- Word inflection (pluralization and singularization) and lemmatization
- Spelling correction
- Add new models or languages through extensions
- WordNet integration

CHAPTER 2

Get it now

```
$ pip install -U textblob  
$ python -m textblob.download_corpora
```

Ready to dive in? Go on to the [Quickstart guide](#).

3.1 License

Copyright 2013-2020 Steven Loria

Permission **is** hereby granted, free of charge, to **any** person obtaining a copy of this software **and** associated documentation files (the "**Software**"), to deal **in** the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, **and/or** sell copies of the Software, **and** to permit persons to whom the Software **is** furnished to do so, subject to the following conditions:

The above copyright notice **and** this permission notice shall be included **in** all copies **or** substantial portions of the Software.

THE SOFTWARE IS PROVIDED "**AS IS**", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

3.2 Installation

3.2.1 Installing/Upgrading From the PyPI

```
$ pip install -U textblob
$ python -m textblob.download_corpora
```

This will install TextBlob and download the necessary NLTK corpora. If you need to change the default download directory set the `NLTK_DATA` environment variable.

Downloading the minimum corpora

If you only intend to use TextBlob's default models (no model overrides), you can pass the `lite` argument. This downloads only those corpora needed for basic functionality.

```
$ python -m textblob.download_corpora lite
```

3.2.2 With conda

TextBlob is also available as a [conda](#) package. To install with `conda`, run

```
$ conda install -c conda-forge textblob
$ python -m textblob.download_corpora
```

3.2.3 From Source

TextBlob is actively developed on [Github](#).

You can clone the public repo:

```
$ git clone https://github.com/sloria/TextBlob.git
```

Or download one of the following:

- [tarball](#)
- [zipball](#)

Once you have the source, you can install it into your site-packages with

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

3.2.4 Get the bleeding edge version

To get the latest development version of TextBlob, run

```
$ pip install -U git+https://github.com/sloria/TextBlob.git@dev
```

3.2.5 Migrating from older versions (<=0.7.1)

As of TextBlob 0.8.0, TextBlob's core package was renamed to `textblob`, whereas earlier versions used a package called `text`. Therefore, migrating to newer versions should be as simple as rewriting your imports, like so:

New:

```
from textblob import TextBlob, Word, Blobber
from textblob.classifiers import NaiveBayesClassifier
from textblob.taggers import NLTKTagger
```

Old:

```
from text.blob import TextBlob, Word, Blobber
from text.classifiers import NaiveBayesClassifier
from text.taggers import NLTKTagger
```

Python

TextBlob supports Python ≥ 2.7 or ≥ 3.5 .

Dependencies

TextBlob depends on NLTK 3. NLTK will be installed automatically when you run `pip install textblob` or `python setup.py install`.

Some features, such as the maximum entropy classifier, require `numpy`, but it is not required for basic usage.

3.3 Tutorial: Quickstart

TextBlob aims to provide access to common text-processing operations through a familiar interface. You can treat *TextBlob* objects as if they were Python strings that learned how to do Natural Language Processing.

3.3.1 Create a TextBlob

First, the import.

```
>>> from textblob import TextBlob
```

Let's create our first *TextBlob*.

```
>>> wiki = TextBlob("Python is a high-level, general-purpose programming language.")
```

3.3.2 Part-of-speech Tagging

Part-of-speech tags can be accessed through the *tags* property.

```
>>> wiki.tags
[('Python', 'NNP'), ('is', 'VBZ'), ('a', 'DT'), ('high-level', 'JJ'), ('general-  
→purpose', 'JJ'), ('programming', 'NN'), ('language', 'NN')]
```

3.3.3 Noun Phrase Extraction

Similarly, noun phrases are accessed through the *noun_phrases* property.

```
>>> wiki.noun_phrases
WordList(['python'])
```

3.3.4 Sentiment Analysis

The `sentiment` property returns a namedtuple of the form `Sentiment(polarity, subjectivity)`. The polarity score is a float within the range [-1.0, 1.0]. The subjectivity is a float within the range [0.0, 1.0] where 0.0 is very objective and 1.0 is very subjective.

```
>>> testimonial = TextBlob("Textblob is amazingly simple to use. What great fun!")
>>> testimonial.sentiment
Sentiment(polarity=0.39166666666666666, subjectivity=0.4357142857142857)
>>> testimonial.sentiment.polarity
0.39166666666666666
```

3.3.5 Tokenization

You can break TextBlobs into words or sentences.

```
>>> zen = TextBlob("Beautiful is better than ugly. "
...                "Explicit is better than implicit. "
...                "Simple is better than complex.")
>>> zen.words
WordList(['Beautiful', 'is', 'better', 'than', 'ugly', 'Explicit', 'is', 'better',
↪ 'than', 'implicit', 'Simple', 'is', 'better', 'than', 'complex'])
>>> zen.sentences
[Sentence("Beautiful is better than ugly."), Sentence("Explicit is better than_
↪ implicit."), Sentence("Simple is better than complex.")]
```

`Sentence` objects have the same properties and methods as TextBlobs.

```
>>> for sentence in zen.sentences:
...     print(sentence.sentiment)
```

For more advanced tokenization, see the [Advanced Usage](#) guide.

3.3.6 Words Inflection and Lemmatization

Each word in `TextBlob.words` or `Sentence.words` is a `Word` object (a subclass of unicode) with useful methods, e.g. for word inflection.

```
>>> sentence = TextBlob("Use 4 spaces per indentation level.")
>>> sentence.words
WordList(['Use', '4', 'spaces', 'per', 'indentation', 'level'])
>>> sentence.words[2].singularize()
'space'
>>> sentence.words[-1].pluralize()
'levels'
```

Words can be lemmatized by calling the `lemmatize` method.

```
>>> from textblob import Word
>>> w = Word("octopi")
>>> w.lemmatize()
'octopus'
>>> w = Word("went")
>>> w.lemmatize("v") # Pass in WordNet part of speech (verb)
'go'
```

3.3.7 WordNet Integration

You can access the synsets for a *Word* via the *synsets* property or the *get_synsets* method, optionally passing in a part of speech.

```
>>> from textblob import Word
>>> from textblob.wordnet import VERB
>>> word = Word("octopus")
>>> word.synsets
[Synset('octopus.n.01'), Synset('octopus.n.02')]
>>> Word("hack").get_synsets(pos=VERB)
[Synset('hack.v.01'), Synset('hack.v.02'), Synset('hack.v.03'), Synset('hack.v.04'),
Synset('hack.v.05'), Synset('hack.v.06'), Synset('hack.v.07'), Synset('hack.v.08')]
```

You can access the definitions for each synset via the *definitions* property or the *define()* method, which can also take an optional part-of-speech argument.

```
>>> Word("octopus").definitions
['tentacles of octopus prepared as food', 'bottom-living cephalopod having a soft_
oval body with eight long tentacles']
```

You can also create synsets directly.

```
>>> from textblob.wordnet import Synset
>>> octopus = Synset('octopus.n.02')
>>> shrimp = Synset('shrimp.n.03')
>>> octopus.path_similarity(shrimp)
0.1111111111111111
```

For more information on the WordNet API, see the NLTK documentation on the [Wordnet Interface](#).

3.3.8 WordLists

A *WordList* is just a Python list with additional methods.

```
>>> animals = TextBlob("cat dog octopus")
>>> animals.words
WordList(['cat', 'dog', 'octopus'])
>>> animals.words.pluralize()
WordList(['cats', 'dogs', 'octopodes'])
```

3.3.9 Spelling Correction

Use the *correct()* method to attempt spelling correction.

```
>>> b = TextBlob("I havv goood speling!")
>>> print(b.correct())
I have good spelling!
```

Word objects have a *spellcheck()* *Word.spellcheck()* method that returns a list of (word, confidence) tuples with spelling suggestions.

```
>>> from textblob import Word
>>> w = Word('falibility')
```

(continues on next page)

(continued from previous page)

```
>>> w.spellcheck()  
[('fallibility', 1.0)]
```

Spelling correction is based on Peter Norvig’s “How to Write a Spelling Corrector”¹ as implemented in the pattern library. It is about 70% accurate².

3.3.10 Get Word and Noun Phrase Frequencies

There are two ways to get the frequency of a word or noun phrase in a *TextBlob*.

The first is through the `word_counts` dictionary.

```
>>> monty = TextBlob("We are no longer the Knights who say Ni. "  
...                 "We are now the Knights who say Ekki ekki ekki PTANG.")  
>>> monty.word_counts['ekki']  
3
```

If you access the frequencies this way, the search will *not* be case sensitive, and words that are not found will have a frequency of 0.

The second way is to use the `count()` method.

```
>>> monty.words.count('ekki')  
3
```

You can specify whether or not the search should be case-sensitive (default is `False`).

```
>>> monty.words.count('ekki', case_sensitive=True)  
2
```

Each of these methods can also be used with noun phrases.

```
>>> wiki.noun_phrases.count('python')  
1
```

3.3.11 Parsing

Use the `parse()` method to parse the text.

```
>>> b = TextBlob("And now for something completely different.")  
>>> print(b.parse())  
And/CC/O/O now/RB/B-ADVP/O for/IN/B-PP/B-PNP something/NN/B-NP/I-PNP completely/RB/B-  
→ADJP/O different/JJ/I-ADJP/O ././O/O
```

By default, *TextBlob* uses pattern’s parser³.

3.3.12 TextBlobs Are Like Python Strings!

You can use Python’s substring syntax.

¹ <http://norvig.com/spell-correct.html>

² <http://www.clips.ua.ac.be/pages/pattern-en#spelling>

³ <http://www.clips.ua.ac.be/pages/pattern-en#parser>


```
>>> zen[0:19]
TextBlob("Beautiful is better")
```

You can use common string methods.

```
>>> zen.upper()
TextBlob("BEAUTIFUL IS BETTER THAN UGLY. EXPLICIT IS BETTER THAN IMPLICIT. SIMPLE IS_
↳ BETTER THAN COMPLEX.")
>>> zen.find("Simple")
65
```

You can make comparisons between TextBlobs and strings.

```
>>> apple_blob = TextBlob('apples')
>>> banana_blob = TextBlob('bananas')
>>> apple_blob < banana_blob
True
>>> apple_blob == 'apples'
True
```

You can concatenate and interpolate TextBlobs and strings.

```
>>> apple_blob + ' and ' + banana_blob
TextBlob("apples and bananas")
>>> "{0} and {1}".format(apple_blob, banana_blob)
'apples and bananas'
```

3.3.13 n-grams

The `TextBlob.ngrams()` method returns a list of tuples of *n* successive words.

```
>>> blob = TextBlob("Now is better than never.")
>>> blob.ngrams(n=3)
[WordList(['Now', 'is', 'better']), WordList(['is', 'better', 'than']), WordList([
↳ 'better', 'than', 'never'])]
```

3.3.14 Get Start and End Indices of Sentences

Use `sentence.start` and `sentence.end` to get the indices where a sentence starts and ends within a `TextBlob`.

```
>>> for s in zen.sentences:
...     print(s)
...     print("---- Starts at index {}, Ends at index {}".format(s.start, s.end))
Beautiful is better than ugly.
---- Starts at index 0, Ends at index 30
Explicit is better than implicit.
---- Starts at index 31, Ends at index 64
Simple is better than complex.
---- Starts at index 65, Ends at index 95
```

Next Steps

Want to build your own text classification system? Check out the [Classifiers Tutorial](#).

Want to use a different POS tagger or noun phrase chunker implementation? Check out the [Advanced Usage](#) guide.

3.4 Tutorial: Building a Text Classification System

The `textblob.classifiers` module makes it simple to create custom classifiers.

As an example, let's create a custom sentiment analyzer.

3.4.1 Loading Data and Creating a Classifier

First we'll create some training and test data.

```
>>> train = [
...     ('I love this sandwich.', 'pos'),
...     ('this is an amazing place!', 'pos'),
...     ('I feel very good about these beers.', 'pos'),
...     ('this is my best work.', 'pos'),
...     ("what an awesome view", 'pos'),
...     ('I do not like this restaurant', 'neg'),
...     ('I am tired of this stuff.', 'neg'),
...     ("I can't deal with this", 'neg'),
...     ('he is my sworn enemy!', 'neg'),
...     ('my boss is horrible.', 'neg')
... ]
>>> test = [
...     ('the beer was good.', 'pos'),
...     ('I do not enjoy my job', 'neg'),
...     ("I ain't feeling dandy today.", 'neg'),
...     ("I feel amazing!", 'pos'),
...     ('Gary is a friend of mine.', 'pos'),
...     ("I can't believe I'm doing this.", 'neg')
... ]
```

Now we'll create a Naive Bayes classifier, passing the training data into the constructor.

```
>>> from textblob.classifiers import NaiveBayesClassifier
>>> cl = NaiveBayesClassifier(train)
```

Loading Data from Files

You can also load data from common file formats including CSV, JSON, and TSV.

CSV files should be formatted like so:

```
I love this sandwich.,pos
This is an amazing place!,pos
I do not like this restaurant,neg
```

JSON files should be formatted like so:

```
[
    {"text": "I love this sandwich.", "label": "pos"},
    {"text": "This is an amazing place!", "label": "pos"},
    {"text": "I do not like this restaurant", "label": "neg"}
]
```

You can then pass the opened file into the constructor.

```
>>> with open('train.json', 'r') as fp:
...     cl = NaiveBayesClassifier(fp, format="json")
```

3.4.2 Classifying Text

Call the `classify(text)` method to use the classifier.

```
>>> cl.classify("This is an amazing library!")
'pos'
```

You can get the label probability distribution with the `prob_classify(text)` method.

```
>>> prob_dist = cl.prob_classify("This one's a doozy.")
>>> prob_dist.max()
'pos'
>>> round(prob_dist.prob("pos"), 2)
0.63
>>> round(prob_dist.prob("neg"), 2)
0.37
```

3.4.3 Classifying TextBlobs

Another way to classify text is to pass a classifier into the constructor of `TextBlob` and call its `classify()` method.

```
>>> from textblob import TextBlob
>>> blob = TextBlob("The beer is good. But the hangover is horrible.", classifier=cl)
>>> blob.classify()
'pos'
```

The advantage of this approach is that you can classify sentences within a `TextBlob`.

```
>>> for s in blob.sentences:
...     print(s)
...     print(s.classify())
...
The beer is good.
pos
But the hangover is horrible.
neg
```

3.4.4 Evaluating Classifiers

To compute the accuracy on our test set, use the `accuracy(test_data)` method.

```
>>> cl.accuracy(test)
0.8333333333333334
```

Note: You can also pass in a file object into the `accuracy` method. The file can be in any of the formats listed in the [Loading Data](#) section.

Use the `show_informative_features()` method to display a listing of the most informative features.

```
>>> cl.show_informative_features(5)
Most Informative Features
contains(my) = True          neg : pos = 1.7 : 1.0
contains(an) = False        neg : pos = 1.6 : 1.0
contains(I) = True          neg : pos = 1.4 : 1.0
contains(I) = False        pos : neg = 1.4 : 1.0
contains(my) = False        pos : neg = 1.3 : 1.0
```

3.4.5 Updating Classifiers with New Data

Use the `update(new_data)` method to update a classifier with new training data.

```
>>> new_data = [('She is my best friend.', 'pos'),
...             ('I'm happy to have a new friend.', 'pos'),
...             ('Stay thirsty, my friend.', 'pos'),
...             ('He ain't from around here.', 'neg')]
>>> cl.update(new_data)
True
>>> cl.accuracy(test)
1.0
```

3.4.6 Feature Extractors

By default, the `NaiveBayesClassifier` uses a simple feature extractor that indicates which words in the training set are contained in a document.

For example, the sentence “*I feel happy*” might have the features `contains(happy): True` or `contains(angry): False`.

You can override this feature extractor by writing your own. A feature extractor is simply a function with `document` (the text to extract features from) as the first argument. The function may include a second argument, `train_set` (the training dataset), if necessary.

The function should return a dictionary of features for document.

For example, let’s create a feature extractor that just uses the first and last words of a document as its features.

```
>>> def end_word_extractor(document):
...     tokens = document.split()
...     first_word, last_word = tokens[0], tokens[-1]
...     feats = {}
...     feats["first({})".format(first_word)] = True
...     feats["last({})".format(last_word)] = False
...     return feats
>>> features = end_word_extractor("I feel happy")
>>> assert features == {'last(happy)': False, 'first(I)': True}
```

We can then use the feature extractor in a classifier by passing it as the second argument of the constructor.

```
>>> cl2 = NaiveBayesClassifier(test, feature_extractor=end_word_extractor)
>>> blob = TextBlob("I'm excited to try my new classifier.", classifier=cl2)
>>> blob.classify()
'pos'
```

3.4.7 Next Steps

Be sure to check out the [API Reference](#) for the *classifiers module*.

Want to try different POS taggers or noun phrase chunkers with TextBlobs? Check out the [Advanced Usage](#) guide.

3.5 Advanced Usage: Overriding Models and the Blobber Class

TextBlob allows you to specify which algorithms you want to use under the hood of its simple API.

3.5.1 Sentiment Analyzers

New in version 0.5.0.

The `textblob.sentiments` module contains two sentiment analysis implementations, `PatternAnalyzer` (based on the [pattern](#) library) and `NaiveBayesAnalyzer` (an [NLTK](#) classifier trained on a movie reviews corpus).

The default implementation is `PatternAnalyzer`, but you can override the analyzer by passing another implementation into a `TextBlob`'s constructor.

For instance, the `NaiveBayesAnalyzer` returns its result as a `namedtuple` of the form: `Sentiment(classification, p_pos, p_neg)`.

```
>>> from textblob import TextBlob
>>> from textblob.sentiments import NaiveBayesAnalyzer
>>> blob = TextBlob("I love this library", analyzer=NaiveBayesAnalyzer())
>>> blob.sentiment
Sentiment(classification='pos', p_pos=0.7996209910191279, p_neg=0.2003790089808724)
```

3.5.2 Tokenizers

New in version 0.4.0.

The `words` and `sentences` properties are helpers that use the `textblob.tokenizers.WordTokenizer` and `textblob.tokenizers.SentenceTokenizer` classes, respectively.

You can use other tokenizers, such as those provided by [NLTK](#), by passing them into the `TextBlob` constructor then accessing the `tokens` property.

```
>>> from textblob import TextBlob
>>> from nltk.tokenize import TabTokenizer
>>> tokenizer = TabTokenizer()
>>> blob = TextBlob("This is\ta rather tabby\tblob.", tokenizer=tokenizer)
>>> blob.tokens
WordList(['This is', 'a rather tabby', 'blob.'])
```

You can also use the `tokenize([tokenizer])` method.

```
>>> from textblob import TextBlob
>>> from nltk.tokenize import BlanklineTokenizer
>>> tokenizer = BlanklineTokenizer()
>>> blob = TextBlob("A token\n\nof appreciation")
>>> blob.tokenize(tokenizer)
WordList(['A token', 'of appreciation'])
```

3.5.3 Noun Phrase Chunkers

TextBlob currently has two noun phrases chunker implementations, `textblob.np_extractors.FastNPExtractor` (default, based on Shlomi Babluki's implementation from [this blog post](#)) and `textblob.np_extractors.ConllExtractor`, which uses the CoNLL 2000 corpus to train a tagger.

You can change the chunker implementation (or even use your own) by explicitly passing an instance of a noun phrase extractor to a TextBlob's constructor.

```
>>> from textblob import TextBlob
>>> from textblob.np_extractors import ConllExtractor
>>> extractor = ConllExtractor()
>>> blob = TextBlob("Python is a high-level programming language.", np_
↳ extractor=extractor)
>>> blob.noun_phrases
WordList(['python', 'high-level programming language'])
```

3.5.4 POS Taggers

TextBlob currently has two POS tagger implementations, located in `textblob.taggers`. The default is the `PatternTagger` which uses the same implementation as the `pattern` library.

The second implementation is `NLTKTagger` which uses NLTK's TreeBank tagger. *Numpy is required to use the NLTKTagger.*

Similar to the tokenizers and noun phrase chunkers, you can explicitly specify which POS tagger to use by passing a tagger instance to the constructor.

```
>>> from textblob import TextBlob
>>> from textblob.taggers import NLTKTagger
>>> nltk_tagger = NLTKTagger()
>>> blob = TextBlob("Tag! You're It!", pos_tagger=nltk_tagger)
>>> blob.pos_tags
[(Word('Tag'), u'NN'), (Word('You'), u'PRP'), (Word(''), u'VBZ'), (Word('re'), u'NN
↳ ), (Word('It'))
, u'PRP')]
```

3.5.5 Parsers

New in version 0.6.0.

Parser implementations can also be passed to the TextBlob constructor.

```
>>> from textblob import TextBlob
>>> from textblob.parsers import PatternParser
>>> blob = TextBlob("Parsing is fun.", parser=PatternParser())
```

(continues on next page)

(continued from previous page)

```
>>> blob.parse()
'Parsing/VBG/B-VP/O is/VBZ/I-VP/O fun/VBG/I-VP/O ././O/O'
```

3.5.6 Blobber: A TextBlob Factory

New in 0.4.0.

It can be tedious to repeatedly pass taggers, NP extractors, sentiment analyzers, classifiers, and tokenizers to multiple TextBlobs. To keep your code **DRY**, you can use the `Blobber` class to create TextBlobs that share the same models.

First, instantiate a `Blobber` with the tagger, NP extractor, sentiment analyzer, classifier, and/or tokenizer of your choice.

```
>>> from textblob import Blobber
>>> from textblob.taggers import NLTKTagger
>>> tb = Blobber(pos_tagger=NLTKTagger())
```

You can now create new TextBlobs like so:

```
>>> blob1 = tb("This is a blob.")
>>> blob2 = tb("This is another blob.")
>>> blob1.pos_tagger is blob2.pos_tagger
True
```

3.6 Extensions

TextBlob supports adding custom models and new languages through “extensions”.

Extensions can be installed from the PyPI.

```
$ pip install textblob-name
```

where “name” is the name of the package.

3.6.1 Available extensions

Languages

- `textblob-fr`: French
- `textblob-de`: German

Part-of-speech Taggers

- `textblob-aptagger`: A fast and accurate tagger based on the Averaged Perceptron.

Interested in creating an extension?

See the *Contributing guide*.

3.7 API Reference

3.7.1 Blob Classes

Wrappers for various units of text, including the main *TextBlob*, *Word*, and *WordList* classes. Example usage:

```
>>> from textblob import TextBlob
>>> b = TextBlob("Simple is better than complex.")
>>> b.tags
[(u'Simple', u'NN'), (u'is', u'VBZ'), (u'better', u'JJR'), (u'than', u'IN'), (u
→ 'complex', u'NN')]
>>> b.noun_phrases
WordList([u'simple'])
>>> b.words
WordList([u'Simple', u'is', u'better', u'than', u'complex'])
>>> b.sentiment
(0.06666666666666667, 0.41904761904761906)
>>> b.words[0].synsets()[0]
Synset('simple.n.01')
```

Changed in version 0.8.0: These classes are now imported from `textblob` rather than `text.blob`.

class `textblob.blob.BaseBlob` (*text*, *tokenizer=None*, *pos_tagger=None*, *np_extractor=None*, *analyzer=None*, *parser=None*, *classifier=None*, *clean_html=False*)

An abstract base class that all textblob classes will inherit from. Includes words, POS tag, NP, and word count properties. Also includes basic dunder and string methods for making objects like Python strings.

Parameters

- **text** – A string.
- **tokenizer** – (optional) A tokenizer instance. If *None*, defaults to *WordTokenizer()*.
- **np_extractor** – (optional) An *NPEExtractor* instance. If *None*, defaults to *FastNPEExtractor()*.
- **pos_tagger** – (optional) A *Tagger* instance. If *None*, defaults to *NLTKTagger*.
- **analyzer** – (optional) A sentiment analyzer. If *None*, defaults to *PatternAnalyzer*.
- **parser** – A parser. If *None*, defaults to *PatternParser*.
- **classifier** – A classifier.

Changed in version 0.6.0: `clean_html` parameter deprecated, as it was in NLTK.

classify()

Classify the blob using the blob's classifier.

correct()

Attempt to correct the spelling of a blob.

New in version 0.6.0.

Return type *BaseBlob*

detect_language()

Detect the blob's language using the Google Translate API.

Requires an internet connection.

Usage:


```
>>> b = TextBlob("bonjour")
>>> b.detect_language()
u'fr'
```

Language code reference: https://developers.google.com/translate/v2/using_rest#language-params

Deprecated since version 0.16.0: Use the official Google Translate API instead.

New in version 0.5.0.

Return type str

ends_with (*suffix*, *start*=0, *end*=9223372036854775807)

Returns True if the blob ends with the given suffix.

endswith (*suffix*, *start*=0, *end*=9223372036854775807)

Returns True if the blob ends with the given suffix.

find (*sub*, *start*=0, *end*=9223372036854775807)

Behaves like the built-in `str.find()` method. Returns an integer, the index of the first occurrence of the substring argument *sub* in the sub-string given by [*start*:*end*].

format (**args*, ***kwargs*)

Perform a string formatting operation, like the built-in `str.format(*args, **kwargs)`. Returns a blob object.

index (*sub*, *start*=0, *end*=9223372036854775807)

Like `blob.find()` but raise `ValueError` when the substring is not found.

join (*iterable*)

Behaves like the built-in `str.join(iterable)` method, except returns a blob object.

Returns a blob which is the concatenation of the strings or blobs in the iterable.

lower ()

Like `str.lower()`, returns new object with all lower-cased characters.

ngrams (*n*=3)

Return a list of n-grams (tuples of *n* successive words) for this blob.

Return type List of *WordLists*

noun_phrases

Returns a list of noun phrases for this blob.

np_counts

Dictionary of noun phrase frequencies in this text.

parse (*parser*=None)

Parse the text.

Parameters **parser** – (optional) A parser instance. If None, defaults to this blob's default parser.

New in version 0.6.0.

polarity

Return the polarity score as a float within the range [-1.0, 1.0]

Return type float

pos_tags

Returns an list of tuples of the form (word, POS tag).

Example:

```
[('At', 'IN'), ('eight', 'CD'), ('o'clock', 'JJ'), ('on', 'IN'),  
 ('Thursday', 'NNP'), ('morning', 'NN')]
```

Return type list of tuples

replace (*old*, *new*, *count*=9223372036854775807)

Return a new blob object with all the occurrence of *old* replaced by *new*.

rfind (*sub*, *start*=0, *end*=9223372036854775807)

Behaves like the built-in `str.rfind()` method. Returns an integer, the index of the last (right-most) occurrence of the substring argument *sub* in the sub-sequence given by `[start:end]`.

rindex (*sub*, *start*=0, *end*=9223372036854775807)

Like `blob.rfind()` but raise `ValueError` when substring is not found.

sentiment

Return a tuple of form (polarity, subjectivity) where polarity is a float within the range [-1.0, 1.0] and subjectivity is a float within the range [0.0, 1.0] where 0.0 is very objective and 1.0 is very subjective.

Return type namedtuple of the form `Sentiment(polarity, subjectivity)`

sentiment_assessments

Return a tuple of form (polarity, subjectivity, assessments) where polarity is a float within the range [-1.0, 1.0], subjectivity is a float within the range [0.0, 1.0] where 0.0 is very objective and 1.0 is very subjective, and assessments is a list of polarity and subjectivity scores for the assessed tokens.

Return type namedtuple of the form `“Sentiment(polarity, subjectivity, assessments)”`

split (*sep*=None, *maxsplit*=9223372036854775807)

Behaves like the built-in `str.split()` except returns a `WordList`.

Return type `WordList`

starts_with (*prefix*, *start*=0, *end*=9223372036854775807)

Returns True if the blob starts with the given prefix.

startswith (*prefix*, *start*=0, *end*=9223372036854775807)

Returns True if the blob starts with the given prefix.

strip (*chars*=None)

Behaves like the built-in `str.strip([chars])` method. Returns an object with leading and trailing whitespace removed.

subjectivity

Return the subjectivity score as a float within the range [0.0, 1.0] where 0.0 is very objective and 1.0 is very subjective.

Return type float

tags

Returns an list of tuples of the form (word, POS tag).

Example:

```
[('At', 'IN'), ('eight', 'CD'), ('o'clock', 'JJ'), ('on', 'IN'),
 ('Thursday', 'NNP'), ('morning', 'NN')]
```

Return type list of tuples

title()

Returns a blob object with the text in title-case.

tokenize (*tokenizer=None*)

Return a list of tokens, using *tokenizer*.

Parameters **tokenizer** – (optional) A tokenizer object. If *None*, defaults to this blob's default tokenizer.

tokens

Return a list of tokens, using this blob's tokenizer object (defaults to *WordTokenizer*).

translate (*from_lang=u'auto', to=u'en'*)

Translate the blob to another language. Uses the Google Translate API. Returns a new TextBlob.

Requires an internet connection.

Usage:

```
>>> b = TextBlob("Simple is better than complex")
>>> b.translate(to="es")
TextBlob('Lo simple es mejor que complejo')
```

Language code reference: https://developers.google.com/translate/v2/using_rest#language-params

Deprecated since version 0.16.0: Use the official Google Translate API instead.

New in version 0.5.0..

Parameters

- **from_lang** (*str*) – Language to translate from. If *None*, will attempt to detect the language.
- **to** (*str*) – Language to translate to.

Return type *BaseBlob*

upper()

Like *str.upper()*, returns new object with all upper-cased characters.

word_counts

Dictionary of word frequencies in this text.

words

Return a list of word tokens. This excludes punctuation characters. If you want to include punctuation characters, access the *tokens* property.

Returns A *WordList* of word tokens.

class `textblob.blob.Blobber` (*tokenizer=None, pos_tagger=None, np_extractor=None, analyzer=None, parser=None, classifier=None*)

A factory for TextBlobs that all share the same tagger, tokenizer, parser, classifier, and *np_extractor*.

Usage:

```
>>> from textblob import Blobber
>>> from textblob.taggers import NLTKTagger
>>> from textblob.tokenizers import SentenceTokenizer
>>> tb = Blobber(pos_tagger=NLTKTagger(), tokenizer=SentenceTokenizer())
>>> blob1 = tb("This is one blob.")
>>> blob2 = tb("This blob has the same tagger and tokenizer.")
>>> blob1.pos_tagger is blob2.pos_tagger
True
```

Parameters

- **tokenizer** – (optional) A tokenizer instance. If None, defaults to *WordTokenizer()*.
- **np_extractor** – (optional) An NPE extractor instance. If None, defaults to *FastNPExtractor()*.
- **pos_tagger** – (optional) A Tagger instance. If None, defaults to *NLTKTagger*.
- **analyzer** – (optional) A sentiment analyzer. If None, defaults to *PatternAnalyzer*.
- **parser** – A parser. If None, defaults to *PatternParser*.
- **classifier** – A classifier.

New in version 0.4.0.

class textblob.blob.**Sentence** (*sentence*, *start_index*=0, *end_index*=None, *args, **kwargs)

A sentence within a TextBlob. Inherits from *BaseBlob*.

Parameters

- **sentence** – A string, the raw sentence.
- **start_index** – An int, the index where this sentence begins in a TextBlob. If not given, defaults to 0.
- **end_index** – An int, the index where this sentence ends in a TextBlob. If not given, defaults to the length of the sentence - 1.

classify()

Classify the blob using the blob's classifier.

correct()

Attempt to correct the spelling of a blob.

New in version 0.6.0.

Return type *BaseBlob*

detect_language()

Detect the blob's language using the Google Translate API.

Requires an internet connection.

Usage:

```
>>> b = TextBlob("bonjour")
>>> b.detect_language()
u'fr'
```

Language code reference: https://developers.google.com/translate/v2/using_rest#language-params

Deprecated since version 0.16.0: Use the official Google Translate API instead.

New in version 0.5.0.

Return type str

dict

The dict representation of this sentence.

end = None

The end index within a textBlob

end_index = None

The end index within a textBlob

ends_with (*suffix*, *start*=0, *end*=9223372036854775807)

Returns True if the blob ends with the given suffix.

endswith (*suffix*, *start*=0, *end*=9223372036854775807)

Returns True if the blob ends with the given suffix.

find (*sub*, *start*=0, *end*=9223372036854775807)

Behaves like the built-in str.find() method. Returns an integer, the index of the first occurrence of the substring argument sub in the sub-string given by [start:end].

format (**args*, ***kwargs*)

Perform a string formatting operation, like the built-in str.format(*args, **kwargs). Returns a blob object.

index (*sub*, *start*=0, *end*=9223372036854775807)

Like blob.find() but raise ValueError when the substring is not found.

join (*iterable*)

Behaves like the built-in str.join(iterable) method, except returns a blob object.

Returns a blob which is the concatenation of the strings or blobs in the iterable.

lower ()

Like str.lower(), returns new object with all lower-cased characters.

ngrams (*n*=3)

Return a list of n-grams (tuples of n successive words) for this blob.

Return type List of *WordLists*

noun_phrases

Returns a list of noun phrases for this blob.

np_counts

Dictionary of noun phrase frequencies in this text.

parse (*parser*=None)

Parse the text.

Parameters parser – (optional) A parser instance. If None, defaults to this blob's default parser.

New in version 0.6.0.

polarity

Return the polarity score as a float within the range [-1.0, 1.0]

Return type float

pos_tags

Returns an list of tuples of the form (word, POS tag).

Example:

```
[('At', 'IN'), ('eight', 'CD'), ('o'clock', 'JJ'), ('on', 'IN'),  
 ('Thursday', 'NNP'), ('morning', 'NN')]
```

Return type list of tuples

replace (*old*, *new*, *count*=9223372036854775807)

Return a new blob object with all the occurrence of *old* replaced by *new*.

rfind (*sub*, *start*=0, *end*=9223372036854775807)

Behaves like the built-in `str.rfind()` method. Returns an integer, the index of the last (right-most) occurrence of the substring argument *sub* in the sub-sequence given by [*start*:*end*].

rindex (*sub*, *start*=0, *end*=9223372036854775807)

Like `blob.rfind()` but raise `ValueError` when substring is not found.

sentiment

Return a tuple of form (polarity, subjectivity) where polarity is a float within the range [-1.0, 1.0] and subjectivity is a float within the range [0.0, 1.0] where 0.0 is very objective and 1.0 is very subjective.

Return type namedtuple of the form `Sentiment(polarity, subjectivity)`

sentiment_assessments

Return a tuple of form (polarity, subjectivity, assessments) where polarity is a float within the range [-1.0, 1.0], subjectivity is a float within the range [0.0, 1.0] where 0.0 is very objective and 1.0 is very subjective, and assessments is a list of polarity and subjectivity scores for the assessed tokens.

Return type namedtuple of the form `“Sentiment(polarity, subjectivity, assessments)”`

split (*sep*=None, *maxsplit*=9223372036854775807)

Behaves like the built-in `str.split()` except returns a `WordList`.

Return type `WordList`

start = None

The start index within a `TextBlob`

start_index = None

The start index within a `TextBlob`

starts_with (*prefix*, *start*=0, *end*=9223372036854775807)

Returns True if the blob starts with the given prefix.

startswith (*prefix*, *start*=0, *end*=9223372036854775807)

Returns True if the blob starts with the given prefix.

strip (*chars*=None)

Behaves like the built-in `str.strip([chars])` method. Returns an object with leading and trailing whitespace removed.

subjectivity

Return the subjectivity score as a float within the range [0.0, 1.0] where 0.0 is very objective and 1.0 is very subjective.

Return type float

tags

Returns a list of tuples of the form (word, POS tag).

Example:

```
[('At', 'IN'), ('eight', 'CD'), ('o'clock', 'JJ'), ('on', 'IN'),
 ('Thursday', 'NNP'), ('morning', 'NN')]
```

Return type list of tuples

title()

Returns a blob object with the text in title-case.

tokenize (*tokenizer=None*)

Return a list of tokens, using *tokenizer*.

Parameters *tokenizer* – (optional) A tokenizer object. If *None*, defaults to this blob's default tokenizer.

tokens

Return a list of tokens, using this blob's tokenizer object (defaults to *WordTokenizer*).

translate (*from_lang=u'auto', to=u'en'*)

Translate the blob to another language. Uses the Google Translate API. Returns a new *TextBlob*.

Requires an internet connection.

Usage:

```
>>> b = TextBlob("Simple is better than complex")
>>> b.translate(to="es")
TextBlob('Lo simple es mejor que complejo')
```

Language code reference: https://developers.google.com/translate/v2/using_rest#language-params

Deprecated since version 0.16.0: Use the official Google Translate API instead.

New in version 0.5.0..

Parameters

- **from_lang** (*str*) – Language to translate from. If *None*, will attempt to detect the language.
- **to** (*str*) – Language to translate to.

Return type *BaseBlob*

upper()

Like *str.upper()*, returns new object with all upper-cased characters.

word_counts

Dictionary of word frequencies in this text.

words

Return a list of word tokens. This excludes punctuation characters. If you want to include punctuation characters, access the *tokens* property.

Returns A *WordList* of word tokens.

class textblob.blob.**TextBlob** (*text*, *tokenizer=None*, *pos_tagger=None*, *np_extractor=None*, *analyzer=None*, *parser=None*, *classifier=None*, *clean_html=False*)

A general text block, meant for larger bodies of text (esp. those containing sentences). Inherits from *BaseBlob*.

Parameters

- **text** (*str*) – A string.
- **tokenizer** – (optional) A tokenizer instance. If *None*, defaults to *WordTokenizer()*.
- **np_extractor** – (optional) An *NPEExtractor* instance. If *None*, defaults to *FastNPEExtractor()*.
- **pos_tagger** – (optional) A *Tagger* instance. If *None*, defaults to *NLTKTagger*.
- **analyzer** – (optional) A sentiment analyzer. If *None*, defaults to *PatternAnalyzer*.
- **classifier** – (optional) A classifier.

classify()

Classify the blob using the blob's classifier.

correct()

Attempt to correct the spelling of a blob.

New in version 0.6.0.

Return type *BaseBlob*

detect_language()

Detect the blob's language using the Google Translate API.

Requires an internet connection.

Usage:

```
>>> b = TextBlob("bonjour")
>>> b.detect_language()
u'fr'
```

Language code reference: https://developers.google.com/translate/v2/using_rest#language-params

Deprecated since version 0.16.0: Use the official Google Translate API instead.

New in version 0.5.0.

Return type *str*

ends_with (*suffix*, *start=0*, *end=9223372036854775807*)

Returns True if the blob ends with the given suffix.

endswith (*suffix*, *start=0*, *end=9223372036854775807*)

Returns True if the blob ends with the given suffix.

find (*sub*, *start=0*, *end=9223372036854775807*)

Behaves like the built-in *str.find()* method. Returns an integer, the index of the first occurrence of the substring argument *sub* in the sub-string given by [*start:end*].

format (**args*, ***kwargs*)

Perform a string formatting operation, like the built-in *str.format(*args, **kwargs)*. Returns a blob object.

index (*sub*, *start=0*, *end=9223372036854775807*)

Like *blob.find()* but raise *ValueError* when the substring is not found.

join (*iterable*)

Behaves like the built-in `str.join(iterable)` method, except returns a blob object.

Returns a blob which is the concatenation of the strings or blobs in the iterable.

json

The json representation of this blob.

Changed in version 0.5.1: Made `json` a property instead of a method to restore backwards compatibility that was broken after version 0.4.0.

lower ()

Like `str.lower()`, returns new object with all lower-cased characters.

ngrams (*n=3*)

Return a list of n-grams (tuples of n successive words) for this blob.

Return type List of *WordLists*

noun_phrases

Returns a list of noun phrases for this blob.

np_counts

Dictionary of noun phrase frequencies in this text.

parse (*parser=None*)

Parse the text.

Parameters **parser** – (optional) A parser instance. If `None`, defaults to this blob's default parser.

New in version 0.6.0.

polarity

Return the polarity score as a float within the range [-1.0, 1.0]

Return type float

pos_tags

Returns an list of tuples of the form (word, POS tag).

Example:

```
[('At', 'IN'), ('eight', 'CD'), ('o'clock', 'JJ'), ('on', 'IN'),
 ('Thursday', 'NNP'), ('morning', 'NN')]
```

Return type list of tuples

raw_sentences

List of strings, the raw sentences in the blob.

replace (*old, new, count=9223372036854775807*)

Return a new blob object with all the occurrence of `old` replaced by `new`.

rfind (*sub, start=0, end=9223372036854775807*)

Behaves like the built-in `str.rfind()` method. Returns an integer, the index of the last (right-most) occurrence of the substring argument `sub` in the sub-sequence given by `[start:end]`.

rindex (*sub, start=0, end=9223372036854775807*)

Like `blob.rfind()` but raise `ValueError` when substring is not found.

sentences

Return list of *Sentence* objects.

sentiment

Return a tuple of form (polarity, subjectivity) where polarity is a float within the range [-1.0, 1.0] and subjectivity is a float within the range [0.0, 1.0] where 0.0 is very objective and 1.0 is very subjective.

Return type namedtuple of the form `Sentiment (polarity, subjectivity)`

sentiment_assessments

Return a tuple of form (polarity, subjectivity, assessments) where polarity is a float within the range [-1.0, 1.0], subjectivity is a float within the range [0.0, 1.0] where 0.0 is very objective and 1.0 is very subjective, and assessments is a list of polarity and subjectivity scores for the assessed tokens.

Return type namedtuple of the form “`Sentiment(polarity, subjectivity, assessments)`”

serialized

Returns a list of each sentence’s dict representation.

split (*sep=None, maxsplit=9223372036854775807*)

Behaves like the built-in `str.split()` except returns a `WordList`.

Return type `WordList`

starts_with (*prefix, start=0, end=9223372036854775807*)

Returns True if the blob starts with the given prefix.

startswith (*prefix, start=0, end=9223372036854775807*)

Returns True if the blob starts with the given prefix.

strip (*chars=None*)

Behaves like the built-in `str.strip([chars])` method. Returns an object with leading and trailing whitespace removed.

subjectivity

Return the subjectivity score as a float within the range [0.0, 1.0] where 0.0 is very objective and 1.0 is very subjective.

Return type float

tags

Returns an list of tuples of the form (word, POS tag).

Example:

```
[('At', 'IN'), ('eight', 'CD'), ('o'clock', 'JJ'), ('on', 'IN'),
 ('Thursday', 'NNP'), ('morning', 'NN')]
```

Return type list of tuples

title()

Returns a blob object with the text in title-case.

to_json (**args, **kwargs*)

Return a json representation (str) of this blob. Takes the same arguments as `json.dumps`.

New in version 0.5.1.

tokenize (*tokenizer=None*)

Return a list of tokens, using `tokenizer`.

Parameters `tokenizer` – (optional) A tokenizer object. If None, defaults to this blob’s default tokenizer.

tokens

Return a list of tokens, using this blob's tokenizer object (defaults to *WordTokenizer*).

translate (*from_lang=u'auto', to=u'en'*)

Translate the blob to another language. Uses the Google Translate API. Returns a new TextBlob.

Requires an internet connection.

Usage:

```
>>> b = TextBlob("Simple is better than complex")
>>> b.translate(to="es")
TextBlob('Lo simple es mejor que complejo')
```

Language code reference: https://developers.google.com/translate/v2/using_rest#language-params

Deprecated since version 0.16.0: Use the official Google Translate API instead.

New in version 0.5.0..

Parameters

- **from_lang** (*str*) – Language to translate from. If *None*, will attempt to detect the language.
- **to** (*str*) – Language to translate to.

Return type *BaseBlob*

upper ()

Like *str.upper()*, returns new object with all upper-cased characters.

word_counts

Dictionary of word frequencies in this text.

words

Return a list of word tokens. This excludes punctuation characters. If you want to include punctuation characters, access the *tokens* property.

Returns A *WordList* of word tokens.

class *textblob.blob.Word* (*string, pos_tag=None*)

A simple word representation. Includes methods for inflection, translation, and WordNet integration.

capitalize () → unicode

Return a capitalized version of *S*, i.e. make the first character have upper case and the rest lower case.

center (*width[, fillchar]*) → unicode

Return *S* centered in a Unicode string of length *width*. Padding is done using the specified fill character (default is a space)

correct ()

Correct the spelling of the word. Returns the word with the highest confidence using the spelling corrector.

New in version 0.6.0.

count (*sub[, start[, end]]*) → int

Return the number of non-overlapping occurrences of substring *sub* in Unicode string *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

decode (*[encoding[, errors]]*) → string or unicode

Decodes *S* using the codec registered for encoding. *encoding* defaults to the default encoding. *errors* may be given to set a different error handling scheme. Default is 'strict' meaning that encoding errors raise a

UnicodeDecodeError. Other possible values are 'ignore' and 'replace' as well as any other name registered with codecs.register_error that is able to handle UnicodeDecodeErrors.

define (*pos=None*)

Return a list of definitions for this word. Each definition corresponds to a synset for this word.

Parameters *pos* – A part-of-speech tag to filter upon. If *None*, definitions for all parts of speech will be loaded.

Return type List of strings

New in version 0.7.0.

definitions

The list of definitions for this word. Each definition corresponds to a synset.

New in version 0.7.0.

detect_language ()

Detect the word's language using Google's Translate API.

Deprecated since version 0.16.0: Use the official Google Translate API instead.

New in version 0.5.0.

encode ([*encoding* [, *errors*]]) → string or unicode

Encodes *S* using the codec registered for encoding. encoding defaults to the default encoding. errors may be given to set a different error handling scheme. Default is 'strict' meaning that encoding errors raise a UnicodeEncodeError. Other possible values are 'ignore', 'replace' and 'xmlcharrefreplace' as well as any other name registered with codecs.register_error that can handle UnicodeEncodeErrors.

endswith (*suffix* [, *start* [, *end*]]) → bool

Return True if *S* ends with the specified suffix, False otherwise. With optional start, test *S* beginning at that position. With optional end, stop comparing *S* at that position. *suffix* can also be a tuple of strings to try.

expandtabs ([*tabsize*]) → unicode

Return a copy of *S* where all tab characters are expanded using spaces. If *tabsize* is not given, a tab size of 8 characters is assumed.

find (*sub* [, *start* [, *end*]]) → int

Return the lowest index in *S* where substring *sub* is found, such that *sub* is contained within *S*[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Return -1 on failure.

format (**args*, ***kwargs*) → unicode

Return a formatted version of *S*, using substitutions from *args* and *kwargs*. The substitutions are identified by braces ('{' and '}').

get_synsets (*pos=None*)

Return a list of Synset objects for this word.

Parameters *pos* – A part-of-speech tag to filter upon. If *None*, all synsets for all parts of speech will be loaded.

Return type list of Synsets

New in version 0.7.0.

index (*sub* [, *start* [, *end*]]) → int

Like *S*.find() but raise ValueError when the substring is not found.

isalnum() → bool
Return True if all characters in S are alphanumeric and there is at least one character in S, False otherwise.

isalpha() → bool
Return True if all characters in S are alphabetic and there is at least one character in S, False otherwise.

isdecimal() → bool
Return True if there are only decimal characters in S, False otherwise.

isdigit() → bool
Return True if all characters in S are digits and there is at least one character in S, False otherwise.

islower() → bool
Return True if all cased characters in S are lowercase and there is at least one cased character in S, False otherwise.

isnumeric() → bool
Return True if there are only numeric characters in S, False otherwise.

isspace() → bool
Return True if all characters in S are whitespace and there is at least one character in S, False otherwise.

istitle() → bool
Return True if S is a titlecased string and there is at least one character in S, i.e. upper- and titlecase characters may only follow uncased characters and lowercase characters only cased ones. Return False otherwise.

isupper() → bool
Return True if all cased characters in S are uppercase and there is at least one cased character in S, False otherwise.

join(iterable) → unicode
Return a string which is the concatenation of the strings in the iterable. The separator between elements is S.

lemma
Return the lemma of this word using Wordnet's morphy function.

lemmatize(kwargs)**
Return the lemma for a word using WordNet's morphy function.

Parameters pos – Part of speech to filter upon. If None, defaults to `_wordnet.NOUN`.

New in version 0.8.1.

ljust(width[, fillchar]) → int
Return S left-justified in a Unicode string of length width. Padding is done using the specified fill character (default is a space).

lower() → unicode
Return a copy of the string S converted to lowercase.

lstrip([chars]) → unicode
Return a copy of the string S with leading whitespace removed. If chars is given and not None, remove characters in chars instead. If chars is a str, it will be converted to unicode before stripping

partition(sep) → (head, sep, tail)
Search for the separator sep in S, and return the part before it, the separator itself, and the part after it. If the separator is not found, return S and two empty strings.

pluralize()
Return the plural version of the word as a string.

replace (*old*, *new*[, *count*]) → unicode

Return a copy of S with all occurrences of substring *old* replaced by *new*. If the optional argument *count* is given, only the first *count* occurrences are replaced.

rfind (*sub*[, *start*[, *end*]]) → int

Return the highest index in S where substring *sub* is found, such that *sub* is contained within S[*start*:*end*]. Optional arguments *start* and *end* are interpreted as in slice notation.

Return -1 on failure.

rindex (*sub*[, *start*[, *end*]]) → int

Like S.rfind() but raise ValueError when the substring is not found.

rjust (*width*[, *fillchar*]) → unicode

Return S right-justified in a Unicode string of length *width*. Padding is done using the specified fill character (default is a space).

rpartition (*sep*) -> (*head*, *sep*, *tail*)

Search for the separator *sep* in S, starting at the end of S, and return the part before it, the separator itself, and the part after it. If the separator is not found, return two empty strings and S.

rsplit ([*sep*[, *maxsplit*]]) → list of strings

Return a list of the words in S, using *sep* as the delimiter string, starting at the end of the string and working to the front. If *maxsplit* is given, at most *maxsplit* splits are done. If *sep* is not specified, any whitespace string is a separator.

rstrip ([*chars*]) → unicode

Return a copy of the string S with trailing whitespace removed. If *chars* is given and not None, remove characters in *chars* instead. If *chars* is a str, it will be converted to unicode before stripping

singularize ()

Return the singular version of the word as a string.

spellcheck ()

Return a list of (word, confidence) tuples of spelling corrections.

Based on: Peter Norvig, “How to Write a Spelling Corrector” (<http://norvig.com/spell-correct.html>) as implemented in the pattern library.

New in version 0.6.0.

split ([*sep*[, *maxsplit*]]) → list of strings

Return a list of the words in S, using *sep* as the delimiter string. If *maxsplit* is given, at most *maxsplit* splits are done. If *sep* is not specified or is None, any whitespace string is a separator and empty strings are removed from the result.

splitlines (*keepends*=False) → list of strings

Return a list of the lines in S, breaking at line boundaries. Line breaks are not included in the resulting list unless *keepends* is given and true.

startswith (*prefix*[, *start*[, *end*]]) → bool

Return True if S starts with the specified prefix, False otherwise. With optional *start*, test S beginning at that position. With optional *end*, stop comparing S at that position. *prefix* can also be a tuple of strings to try.

stem (*stemmer*=<PorterStemmer>)

Stem a word using various NLTK stemmers. (Default: Porter Stemmer)

New in version 0.12.0.

strip (*[chars]*) → unicode

Return a copy of the string *S* with leading and trailing whitespace removed. If *chars* is given and not *None*, remove characters in *chars* instead. If *chars* is a str, it will be converted to unicode before stripping

swapcase () → unicode

Return a copy of *S* with uppercase characters converted to lowercase and vice versa.

synsets

The list of Synset objects for this Word.

Return type list of Synsets

New in version 0.7.0.

title () → unicode

Return a titlecased version of *S*, i.e. words start with title case characters, all remaining cased characters have lower case.

translate (*from_lang=u'auto', to=u'en'*)

Translate the word to another language using Google's Translate API.

Deprecated since version 0.16.0: Use the official Google Translate API instead.

New in version 0.5.0.

upper () → unicode

Return a copy of *S* converted to uppercase.

zfill (*width*) → unicode

Pad a numeric string *S* with zeros on the left, to fill a field of the specified width. The string *S* is never truncated.

class textblob.blob.**WordList** (*collection*)

A list-like collection of words.

append (*obj*)

Append an object to end. If the object is a string, appends a *Word* object.

count (*strg, case_sensitive=False, *args, **kwargs*)

Get the count of a word or phrase *s* within this WordList.

Parameters

- **strg** – The string to count.
- **case_sensitive** – A boolean, whether or not the search is case-sensitive.

extend (*iterable*)

Extend WordList by appending elements from *iterable*. If an element is a string, appends a *Word* object.

index (*value[, start[, stop]]*) → integer – return first index of value.

Raises ValueError if the value is not present.

insert ()

L.insert(index, object) – insert object before index

lemmatize ()

Return the lemma of each word in this WordList.

lower ()

Return a new WordList with each word lower-cased.

pluralize ()

Return the plural version of each word in this WordList.

pop (*[index]*) → item – remove and return item at index (default last).
Raises `IndexError` if list is empty or index is out of range.

remove ()
L.remove(value) – remove first occurrence of value. Raises `ValueError` if the value is not present.

reverse ()
L.reverse() – reverse *IN PLACE*

singularize ()
Return the single version of each word in this `WordList`.

sort ()
L.sort(cmp=None, key=None, reverse=False) – stable sort *IN PLACE*; cmp(x, y) -> -1, 0, 1

stem (*args, **kwargs)
Return the stem for each word in this `WordList`.

upper ()
Return a new `WordList` with each word upper-cased.

3.7.2 Base Classes

Abstract base classes for models (taggers, noun phrase extractors, etc.) which define the interface for descendant classes.

Changed in version 0.7.0: All base classes are defined in the same module, `textblob.base`.

class `textblob.base.BaseNPExtractor`

Abstract base class from which all `NPExtractor` classes inherit. Descendant classes must implement an `extract(text)` method that returns a list of noun phrases as strings.

extract (text)
Return a list of noun phrases (strings) for a body of text.

class `textblob.base.BaseParser`

Abstract parser class from which all parsers inherit from. All descendants must implement a `parse()` method.

parse (text)
Parses the text.

class `textblob.base.BaseSentimentAnalyzer`

Abstract base class from which all sentiment analyzers inherit. Should implement an `analyze(text)` method which returns either the results of analysis.

analyze (text)
Return the result of analysis. Typically returns either a tuple, float, or dictionary.

class `textblob.base.BaseTagger`

Abstract tagger class from which all taggers inherit from. All descendants must implement a `tag()` method.

tag (text, tokenize=True)
Return a list of tuples of the form (word, tag) for a given set of text or `BaseBlob` instance.

class `textblob.base.BaseTokenizer`

Abstract base class from which all `Tokenizer` classes inherit. Descendant classes must implement a `tokenize(text)` method that returns a list of noun phrases as strings.

itokenize (text, *args, **kwargs)
Return a generator that generates tokens “on-demand”.

New in version 0.6.0.

Return type generator

tokenize (*text*)

Return a list of tokens (strings) for a body of text.

Return type list

3.7.3 Tokenizers

Various tokenizer implementations.

New in version 0.4.0.

class textblob.tokenizers.SentenceTokenizer

NLTK's sentence tokenizer (currently PunktSentenceTokenizer). Uses an unsupervised algorithm to build a model for abbreviation words, collocations, and words that start sentences, then uses that to find sentence boundaries.

itokenize (*text*, **args*, ***kwargs*)

Return a generator that generates tokens “on-demand”.

New in version 0.6.0.

Return type generator

tokenize (***kwargs*)

Return a list of sentences.

class textblob.tokenizers.WordTokenizer

NLTK's recommended word tokenizer (currently the TreeBankTokenizer). Uses regular expressions to tokenize text. Assumes text has already been segmented into sentences.

Performs the following steps:

- split standard contractions, e.g. don't -> do n't
- split commas and single quotes
- separate periods that appear at the end of line

itokenize (*text*, **args*, ***kwargs*)

Return a generator that generates tokens “on-demand”.

New in version 0.6.0.

Return type generator

tokenize (*text*, *include_punc=True*)

Return a list of word tokens.

Parameters

- **text** – string of text.
- **include_punc** – (optional) whether to include punctuation as separate tokens. Default to True.

textblob.tokenizers.sent_tokenize = <bound method SentenceTokenizer.itokenize of <textblob>
Convenience function for tokenizing sentences

`textblob.tokenizers.word_tokenize` (*text*, *include_punc=True*, *args, **kwargs)

Convenience function for tokenizing text into words.

NOTE: NLTK's word tokenizer expects sentences as input, so the text will be tokenized to sentences before being tokenized to words.

3.7.4 POS Taggers

Parts-of-speech tagger implementations.

class `textblob.en.taggers.NLTKTagger`

Tagger that uses NLTK's standard TreeBank tagger. NOTE: Requires numpy. Not yet supported with PyPy.

tag (*kwargs)

Tag a string or BaseBlob.

class `textblob.en.taggers.PatternTagger`

Tagger that uses the implementation in Tom de Smedt's pattern library (<http://www.clips.ua.ac.be/pattern>).

tag (*text*, *tokenize=True*)

Tag a string or BaseBlob.

3.7.5 Noun Phrase Extractors

Various noun phrase extractors.

class `textblob.en.np_extractors.ChunkParser`

evaluate (*gold*)

Score the accuracy of the chunker against the gold standard. Remove the chunking the gold standard text, rechunk it using the chunker, and return a `ChunkScore` object reflecting the performance of this chunk parser.

Parameters *gold* (*list* (*Tree*)) – The list of chunked sentences to score the chunker on.

Return type `ChunkScore`

grammar ()

Returns The grammar used by this parser.

parse (*sentence*)

Return the parse tree for the sentence.

parse_all (*sent*, *args, **kwargs)

Return type `list(Tree)`

parse_one (*sent*, *args, **kwargs)

Return type `Tree` or `None`

parse_sents (*sents*, *args, **kwargs)

Apply `self.parse()` to each element of *sents*. :rtype: `iter(iter(Tree))`

train (*kwargs)

Train the Chunker on the ConLL-2000 corpus.

class `textblob.en.np_extractors.ConllExtractor` (*parser=None*)

A noun phrase extractor that uses chunk parsing trained with the ConLL-2000 training corpus.

extract (*text*)

Return a list of noun phrases (strings) for body of text.

class textblob.en.np_extractors.**FastNPExtractor**

A fast and simple noun phrase extractor.

Credit to Shlomi Babluk. Link to original blog post:

<http://thetokenizer.com/2013/05/09/efficient-way-to-extract-the-main-topics-of-a-sentence/>

extract (*sentence*)

Return a list of noun phrases (strings) for body of text.

3.7.6 Sentiment Analyzers

Sentiment analysis implementations.

New in version 0.5.0.

class textblob.en.sentiments.**NaiveBayesAnalyzer** (*feature_extractor=<function _default_feature_extractor>*)

Naive Bayes analyzer that is trained on a dataset of movie reviews. Returns results as a named tuple of the form: `Sentiment(classification, p_pos, p_neg)`

Parameters **feature_extractor** (*callable*) – Function that returns a dictionary of features, given a list of words.

RETURN_TYPE

Return type declaration

alias of `Sentiment`

analyze (*text*)

Return the sentiment as a named tuple of the form: `Sentiment(classification, p_pos, p_neg)`

train (***kwargs*)

Train the Naive Bayes classifier on the movie review corpus.

class textblob.en.sentiments.**PatternAnalyzer**

Sentiment analyzer that uses the same implementation as the pattern library. Returns results as a named tuple of the form:

`Sentiment(polarity, subjectivity, [assessments])`

where `[assessments]` is a list of the assessed tokens and their polarity and subjectivity scores

RETURN_TYPE

alias of `Sentiment`

analyze (*text, keep_assessments=False*)

Return the sentiment as a named tuple of the form: `Sentiment(polarity, subjectivity, [assessments])`.

3.7.7 Parsers

Various parser implementations.

New in version 0.6.0.

class textblob.en.parsers.**PatternParser**

Parser that uses the implementation in Tom de Smedt's pattern library. <http://www.clips.ua.ac.be/pages/pattern-en#parser>

parse (*text*)

Parses the text.

3.7.8 Classifiers

Various classifier implementations. Also includes basic feature extractor methods.

Example Usage:

```
>>> from textblob import TextBlob
>>> from textblob.classifiers import NaiveBayesClassifier
>>> train = [
...     ('I love this sandwich.', 'pos'),
...     ('This is an amazing place!', 'pos'),
...     ('I feel very good about these beers.', 'pos'),
...     ('I do not like this restaurant', 'neg'),
...     ('I am tired of this stuff.', 'neg'),
...     ('I can't deal with this', 'neg'),
...     ('My boss is horrible.', 'neg')
... ]
>>> cl = NaiveBayesClassifier(train)
>>> cl.classify("I feel amazing!")
'pos'
>>> blob = TextBlob("The beer is good. But the hangover is horrible.", classifier=cl)
>>> for s in blob.sentences:
...     print(s)
...     print(s.classify())
...
The beer is good.
pos
But the hangover is horrible.
neg
```

New in version 0.6.0.

class textblob.classifiers.**BaseClassifier**(*train_set*, *feature_extractor*=<function *basic_extractor*>, *format*=None, ****kwargs**)

Abstract classifier class from which all classifiers inherit. At a minimum, descendant classes must implement a `classify` method and have a `classifier` property.

Parameters

- **train_set** – The training set, either a list of tuples of the form (*text*, *classification*) or a file-like object. *text* may be either a string or an iterable.
- **feature_extractor** (*callable*) – A feature extractor function that takes one or two arguments: document and *train_set*.
- **format** (*str*) – If *train_set* is a filename, the file format, e.g. "csv" or "json". If None, will attempt to detect the file format.
- **kwargs** – Additional keyword arguments are passed to the constructor of the *Format* class used to read the data. Only applies when a file-like object is passed as *train_set*.

New in version 0.6.0.

classifier

The classifier object.

classify (*text*)

Classifies a string of text.

extract_features (*text*)

Extracts features from a body of text.

Return type dictionary of features

labels ()

Returns an iterable containing the possible labels.

train (*labeled_featureset*)

Trains the classifier.

```
class textblob.classifiers.DecisionTreeClassifier (train_set, feature_extractor=<function basic_extractor>, format=None, **kwargs)
```

A classifier based on the decision tree algorithm, as implemented in NLTK.

Parameters

- **train_set** – The training set, either a list of tuples of the form (*text*, *classification*) or a filename. *text* may be either a string or an iterable.
- **feature_extractor** – A feature extractor function that takes one or two arguments: *document* and *train_set*.
- **format** – If *train_set* is a filename, the file format, e.g. "csv" or "json". If None, will attempt to detect the file format.

New in version 0.6.2.

accuracy (*test_set*, *format*=None)

Compute the accuracy on a test set.

Parameters

- **test_set** – A list of tuples of the form (*text*, *label*), or a file pointer.
- **format** – If *test_set* is a filename, the file format, e.g. "csv" or "json". If None, will attempt to detect the file format.

classifier

The classifier.

classify (*text*)

Classifies the text.

Parameters **text** (*str*) – A string of text.

extract_features (*text*)

Extracts features from a body of text.

Return type dictionary of features

labels ()

Return an iterable of possible labels.

nlk_class

alias of `nlk.classify.decisiontree.DecisionTreeClassifier`

pprint (*args, **kwargs)

Return a string containing a pretty-printed version of this decision tree. Each line in the string corresponds to a single decision tree node or leaf, and indentation is used to display the structure of the tree.

Return type str

pretty_format (*args, **kwargs)

Return a string containing a pretty-printed version of this decision tree. Each line in the string corresponds to a single decision tree node or leaf, and indentation is used to display the structure of the tree.

Return type str

pseudocode (*args, **kwargs)

Return a string representation of this decision tree that expresses the decisions it makes as a nested set of pseudocode if statements.

Return type str

train (*args, **kwargs)

Train the classifier with a labeled feature set and return the classifier. Takes the same arguments as the wrapped NLTK class. This method is implicitly called when calling `classify` or `accuracy` methods and is included only to allow passing in arguments to the `train` method of the wrapped NLTK class.

New in version 0.6.2.

Return type A classifier

update (new_data, *args, **kwargs)

Update the classifier with new training data and re-trains the classifier.

Parameters **new_data** – New data as a list of tuples of the form (text, label).

class textblob.classifiers.**MaxEntClassifier**(train_set, feature_extractor=<function basic_extractor>, format=None, **kwargs)

A maximum entropy classifier (also known as a “conditional exponential classifier”). This classifier is parameterized by a set of “weights”, which are used to combine the joint-features that are generated from a featureset by an “encoding”. In particular, the encoding maps each (featureset, label) pair to a vector. The probability of each label is then computed using the following equation:

$$\text{prob}(\text{fs}|\text{label}) = \frac{\text{dotprod}(\text{weights}, \text{encode}(\text{fs}, \text{label}))}{\sum(\text{dotprod}(\text{weights}, \text{encode}(\text{fs}, l)) \text{ for } l \text{ in labels})}$$

Where `dotprod` is the dot product:

$$\text{dotprod}(a, b) = \sum(x * y \text{ for } (x, y) \text{ in zip}(a, b))$$

accuracy (test_set, format=None)

Compute the accuracy on a test set.

Parameters

- **test_set** – A list of tuples of the form (text, label), or a file pointer.
- **format** – If `test_set` is a filename, the file format, e.g. "csv" or "json". If None, will attempt to detect the file format.

classifier

The classifier.

classify (text)

Classifies the text.

Parameters **text** (*str*) – A string of text.

extract_features (*text*)

Extracts features from a body of text.

Return type dictionary of features

labels ()

Return an iterable of possible labels.

nltk_class

alias of `nltk.classify.maxent.MaxentClassifier`

prob_classify (*text*)

Return the label probability distribution for classifying a string of text.

Example:

```
>>> classifier = MaxEntClassifier(train_data)
>>> prob_dist = classifier.prob_classify("I feel happy this morning.")
>>> prob_dist.max()
'positive'
>>> prob_dist.prob("positive")
0.7
```

Return type `nltk.probability.DictionaryProbDist`

train (**args*, ***kwargs*)

Train the classifier with a labeled feature set and return the classifier. Takes the same arguments as the wrapped NLTK class. This method is implicitly called when calling `classify` or `accuracy` methods and is included only to allow passing in arguments to the `train` method of the wrapped NLTK class.

New in version 0.6.2.

Return type A classifier

update (*new_data*, **args*, ***kwargs*)

Update the classifier with new training data and re-trains the classifier.

Parameters **new_data** – New data as a list of tuples of the form (*text*, *label*).

class `textblob.classifiers.NLTKClassifier` (*train_set*, *feature_extractor*=<function *basic_extractor*>, *format*=None, ***kwargs*)

An abstract class that wraps around the `nltk.classify` module.

Expects that descendant classes include a class variable `nltk_class` which is the class in the `nltk.classify` module to be wrapped.

Example:

```
class MyClassifier(NLTKClassifier):
    nltk_class = nltk.classify.svm.SvmClassifier
```

accuracy (*test_set*, *format*=None)

Compute the accuracy on a test set.

Parameters

- **test_set** – A list of tuples of the form (*text*, *label*), or a file pointer.
- **format** – If *test_set* is a filename, the file format, e.g. "csv" or "json". If None, will attempt to detect the file format.

classifier

The classifier.

classify (*text*)

Classifies the text.

Parameters *text* (*str*) – A string of text.

extract_features (*text*)

Extracts features from a body of text.

Return type dictionary of features

labels ()

Return an iterable of possible labels.

nltk_class = **None**

The NLTK class to be wrapped. Must be a class within nltk.classify

train (*args, **kwargs)

Train the classifier with a labeled feature set and return the classifier. Takes the same arguments as the wrapped NLTK class. This method is implicitly called when calling `classify` or `accuracy` methods and is included only to allow passing in arguments to the `train` method of the wrapped NLTK class.

New in version 0.6.2.

Return type A classifier

update (*new_data*, *args, **kwargs)

Update the classifier with new training data and re-trains the classifier.

Parameters *new_data* – New data as a list of tuples of the form (*text*, *label*).

class textblob.classifiers.NaiveBayesClassifier (*train_set*, *feature_extractor*=<function *basic_extractor*>, *format*=None, **kwargs)

A classifier based on the Naive Bayes algorithm, as implemented in NLTK.

Parameters

- **train_set** – The training set, either a list of tuples of the form (*text*, *classification*) or a filename. *text* may be either a string or an iterable.
- **feature_extractor** – A feature extractor function that takes one or two arguments: *document* and *train_set*.
- **format** – If *train_set* is a filename, the file format, e.g. "csv" or "json". If None, will attempt to detect the file format.

New in version 0.6.0.

accuracy (*test_set*, *format*=None)

Compute the accuracy on a test set.

Parameters

- **test_set** – A list of tuples of the form (*text*, *label*), or a file pointer.
- **format** – If *test_set* is a filename, the file format, e.g. "csv" or "json". If None, will attempt to detect the file format.

classifier

The classifier.

classify (*text*)

Classifies the text.

Parameters `text` (*str*) – A string of text.

extract_features (*text*)

Extracts features from a body of text.

Return type dictionary of features

informative_features (**args, **kwargs*)

Return the most informative features as a list of tuples of the form (*feature_name*, *feature_value*).

Return type list

labels ()

Return an iterable of possible labels.

nltk_class

alias of `nltk.classify.naivebayes.NaiveBayesClassifier`

prob_classify (*text*)

Return the label probability distribution for classifying a string of text.

Example:

```
>>> classifier = NaiveBayesClassifier(train_data)
>>> prob_dist = classifier.prob_classify("I feel happy this morning.")
>>> prob_dist.max()
'positive'
>>> prob_dist.prob("positive")
0.7
```

Return type `nltk.probability.DictionaryProbDist`

show_informative_features (**args, **kwargs*)

Displays a listing of the most informative features for this classifier.

Return type None

train (**args, **kwargs*)

Train the classifier with a labeled feature set and return the classifier. Takes the same arguments as the wrapped NLTK class. This method is implicitly called when calling `classify` or `accuracy` methods and is included only to allow passing in arguments to the `train` method of the wrapped NLTK class.

New in version 0.6.2.

Return type A classifier

update (*new_data, *args, **kwargs*)

Update the classifier with new training data and re-trains the classifier.

Parameters `new_data` – New data as a list of tuples of the form (*text*, *label*).

```
class textblob.classifiers.PositiveNaiveBayesClassifier(positive_set, un-
                                                         labeled_set, fea-
                                                         ture_extractor=<function
                                                         contains_extractor>,
                                                         positive_prob_prior=0.5,
                                                         **kwargs)
```

A variant of the Naive Bayes Classifier that performs binary classification with partially-labeled training sets, i.e. when only one class is labeled and the other is not. Assuming a prior distribution on the two labels, uses the unlabeled set to estimate the frequencies of the features.

Example usage:

```
>>> from text.classifiers import PositiveNaiveBayesClassifier
>>> sports_sentences = ['The team dominated the game',
...                     'They lost the ball',
...                     'The game was intense',
...                     'The goalkeeper caught the ball',
...                     'The other team controlled the ball']
>>> various_sentences = ['The President did not comment',
...                      'I lost the keys',
...                      'The team won the game',
...                      'Sara has two kids',
...                      'The ball went off the court',
...                      'They had the ball for the whole game',
...                      'The show is over']
>>> classifier = PositiveNaiveBayesClassifier(positive_set=sports_sentences,
...                                           unlabeled_set=various_sentences)
>>> classifier.classify("My team lost the game")
True
>>> classifier.classify("And now for something completely different.")
False
```

Parameters

- **positive_set** – A collection of strings that have the positive label.
- **unlabeled_set** – A collection of unlabeled strings.
- **feature_extractor** – A feature extractor function.
- **positive_prob_prior** – A prior estimate of the probability of the label `True`.

New in version 0.7.0.

accuracy (*test_set*, *format=None*)

Compute the accuracy on a test set.

Parameters

- **test_set** – A list of tuples of the form (*text*, *label*), or a file pointer.
- **format** – If *test_set* is a filename, the file format, e.g. "csv" or "json". If `None`, will attempt to detect the file format.

classifier

The classifier.

classify (*text*)

Classifies the text.

Parameters *text* (*str*) – A string of text.

extract_features (*text*)

Extracts features from a body of text.

Return type dictionary of features

labels ()

Return an iterable of possible labels.

train (**args*, ***kwargs*)

Train the classifier with a labeled and unlabeled feature sets and return the classifier. Takes the same arguments as the wrapped NLTK class. This method is implicitly called when calling `classify` or

accuracy methods and is included only to allow passing in arguments to the `train` method of the wrapped NLTK class.

Return type A classifier

update (*new_positive_data=None, new_unlabeled_data=None, positive_prob_prior=0.5, *args, **kwargs*)

Update the classifier with new data and re-trains the classifier.

Parameters

- **new_positive_data** – List of new, labeled strings.
- **new_unlabeled_data** – List of new, unlabeled strings.

`textblob.classifiers.basic_extractor` (*document, train_set*)

A basic document feature extractor that returns a dict indicating what words in `train_set` are contained in `document`.

Parameters

- **document** – The text to extract features from. Can be a string or an iterable.
- **train_set** (*list*) – Training data set, a list of tuples of the form (`words, label`) OR an iterable of strings.

`textblob.classifiers.contains_extractor` (*document*)

A basic document feature extractor that returns a dict of words that the document contains.

3.7.9 Blobber

class `textblob.blob.Blobber` (*tokenizer=None, pos_tagger=None, np_extractor=None, analyzer=None, parser=None, classifier=None*)

A factory for TextBlobs that all share the same tagger, tokenizer, parser, classifier, and `np_extractor`.

Usage:

```
>>> from textblob import Blobber
>>> from textblob.taggers import NLTKTagger
>>> from textblob.tokenizers import SentenceTokenizer
>>> tb = Blobber(pos_tagger=NLTKTagger(), tokenizer=SentenceTokenizer())
>>> blob1 = tb("This is one blob.")
>>> blob2 = tb("This blob has the same tagger and tokenizer.")
>>> blob1.pos_tagger is blob2.pos_tagger
True
```

Parameters

- **tokenizer** – (optional) A tokenizer instance. If `None`, defaults to `WordTokenizer()`.
- **np_extractor** – (optional) An `NPEExtractor` instance. If `None`, defaults to `FastNPEExtractor()`.
- **pos_tagger** – (optional) A `Tagger` instance. If `None`, defaults to `NLTKTagger`.
- **analyzer** – (optional) A sentiment analyzer. If `None`, defaults to `PatternAnalyzer`.
- **parser** – A parser. If `None`, defaults to `PatternParser`.
- **classifier** – A classifier.

New in version 0.4.0.

__call__ (*text*)
Return a new TextBlob object with this Blobber's `np_extractor`, `pos_tagger`, `tokenizer`, `analyzer`, and `classifier`.
Returns A new *TextBlob*.

__init__ (*tokenizer=None, pos_tagger=None, np_extractor=None, analyzer=None, parser=None, classifier=None*)
`x.__init__(...)` initializes `x`; see `help(type(x))` for signature

__repr__ () $\leq \Rightarrow$ *repr*(*x*)

__str__ ()
`x.__repr__()` $\leq \Rightarrow$ *repr*(*x*)

3.7.10 File Formats

File formats for training and testing data.

Includes a registry of valid file formats. New file formats can be added to the registry like so:

```
from textblob import formats

class PipeDelimitedFormat(formats.DelimitedFormat):
    delimiter = '|'

formats.register('psv', PipeDelimitedFormat)
```

Once a format has been registered, classifiers will be able to read data files with that format.

```
from textblob.classifiers import NaiveBayesAnalyzer

with open('training_data.psv', 'r') as fp:
    cl = NaiveBayesAnalyzer(fp, format='psv')
```

class `textblob.formats.BaseFormat` (*fp, **kwargs*)
Interface for format classes. Individual formats can decide on the composition and meaning of ***kwargs*.

Parameters *fp* (*File*) – A file-like object.

Changed in version 0.9.0: Constructor receives a file pointer rather than a file path.

classmethod `detect` (*stream*)
Detect the file format given a filename. Return True if a stream is this file format.

Changed in version 0.9.0: Changed from a static method to a class method.

to_iterable ()
Return an iterable object from the data.

class `textblob.formats.CSV` (*fp, **kwargs*)
CSV format. Assumes each row is of the form `text, label`.

```
Today is a good day,pos
I hate this car.,pos
```

classmethod `detect` (*stream*)
Return True if stream is valid.

to_iterable ()
Return an iterable object from the data.

class textblob.formats.DelimitedFormat (*fp*, ***kwargs*)

A general character-delimited format.

classmethod detect (*stream*)

Return True if stream is valid.

to_iterable ()

Return an iterable object from the data.

class textblob.formats.JSON (*fp*, ***kwargs*)

JSON format.

Assumes that JSON is formatted as an array of objects with `text` and `label` properties.

```
[
    {"text": "Today is a good day.", "label": "pos"},
    {"text": "I hate this car.", "label": "neg"}
]
```

classmethod detect (*stream*)

Return True if stream is valid JSON.

to_iterable ()

Return an iterable object from the JSON data.

class textblob.formats.TSV (*fp*, ***kwargs*)

TSV format. Assumes each row is of the form `text label`.

classmethod detect (*stream*)

Return True if stream is valid.

to_iterable ()

Return an iterable object from the data.

textblob.formats.detect (*fp*, *max_read=1024*)

Attempt to detect a file's format, trying each of the supported formats. Return the format class that was detected.

If no format is detected, return `None`.

textblob.formats.get_registry ()

Return a dictionary of registered formats.

textblob.formats.register (*name*, *format_class*)

Register a new format.

Parameters

- **name** (*str*) – The name that will be used to refer to the format, e.g. 'csv'
- **format_class** (*type*) – The format class to register.

3.7.11 Wordnet

3.7.12 Exceptions

exception textblob.exceptions.TextBlobError

A TextBlob-related error.

exception `textblob.exceptions.MissingCorpusError` (*message="nLooks like you are missing some required data for this feature.nnTo download the necessary data, simply runnn python -m textblob.download_corporannor use the NLTK downloader to download the missing data: http://nltk.org/data.htmlnlf this doesn't fix the problem, file an issue at https://github.com/sloria/TextBlob/issues.n", *args, **kwargs)*)

Exception thrown when a user tries to use a feature that requires a dataset or model that the user does not have on their system.

exception `textblob.exceptions.DeprecationError`

Raised when user uses a deprecated feature.

exception `textblob.exceptions.TranslatorError`

Raised when an error occurs during language translation or detection.

exception `textblob.exceptions.NotTranslated`

Raised when text is unchanged after translation. This may be due to the language being unsupported by the translator.

exception `textblob.exceptions.FormatError`

Raised if a data file with an unsupported format is passed to a classifier.

4.1 Changelog

4.1.1 0.16.0 (2020-04-26)

Deprecations:

- `TextBlob.translate()` and `TextBlob.detect_language` are deprecated. Use the official Google Translate API instead (#215).

Other changes:

- *Backwards-incompatible*: Drop support for Python 3.4.
- Test against Python 3.7 and Python 3.8.
- Pin NLTK to `nltk<3.5` on Python 2 (#315).

4.1.2 0.15.3 (2019-02-24)

Bug fixes:

- Fix bug when `Word` string type after `pos_tags` is not a `str` (#255). Thanks @roman-y-korolev for the patch.

4.1.3 0.15.2 (2018-11-21)

Bug fixes:

- Fix bug that raised a `RuntimeError` when executing methods that delegate to `pattern.en` (#230). Thanks @vvaezian for the report and thanks @danong for the fix.
- Fix methods of `WordList` that modified the list in-place by removing the internal `_collection` variable (#235). Thanks @jammmo for the PR.

4.1.4 0.15.1 (2018-01-20)

Bug fixes:

- Convert POS tags from treebank to wordnet when calling `lemmatize` to prevent `MissingCorpusError` (#160). Thanks @jschnurr.

4.1.5 0.15.0 (2017-12-02)

Features:

- Add `TextBlob.sentiment_assessments` property which exposes pattern's sentiment assessments (#170). Thanks @jeffakolb.

4.1.6 0.14.0 (2017-11-20)

Features:

- Use specified tokenizer when tagging (#167). Thanks @jschnurr for the PR.

4.1.7 0.13.1 (2017-11-11)

Bug fixes:

- Avoid `AttributeError` when using pattern's sentiment analyzer (#178). Thanks @tylerjharden for the catch and patch.
- Correctly pass `format` argument to `NLTKClassifier.accuracy` (#177). Thanks @pavelmalai for the catch and patch.

4.1.8 0.13.0 (2017-08-15)

Features:

- Performance improvements to `NaiveBayesClassifier` (#63, #77, #123). Thanks @jcalbert for the PR.

4.1.9 0.12.0 (2017-02-27)

Features:

- Add `Word.stem` and `WordList.stem` methods (#145). Thanks @nitkul.

Bug fixes:

- Fix translation and language detection (#137). Thanks @EpicJhon for the fix.

Changes:

- *Backwards-incompatible*: Remove Python 2.6 and 3.3 support.

4.1.10 0.11.1 (2016-02-17)

Bug fixes:

- Fix translation and language detection (#115, #117, #119). Thanks @AdrianLC and @jschnurr for the fix. Thanks @AdrianLC, @edgaralts, and @pouya-cognitiv for reporting.

4.1.11 0.11.0 (2015-11-01)

Changes:

- Compatible with nltk>=3.1. NLTK versions < 3.1 are no longer supported.
- Change default tagger to NLTKTagger (uses NLTK's averaged perceptron tagger).
- Tested on Python 3.5.

Bug fixes:

- Fix singularization of a number of words. Thanks @jonmcoe.
- Fix spelling correction when nltk>=3.1 is installed (#99). Thanks @shubham12101 for reporting.

4.1.12 0.10.0 (2015-10-04)

Changes:

- Unchanged text is now considered a translation error. Raises NotTranslated (#76). Thanks @jschnurr.

Bug fixes:

- Translator.translate will detect language of input text by default (#85). Thanks again @jschnurr.
- Fix matching of tagged phrases with CFG in ConllExtractor. Thanks @lagnarsson.
- Fix inflection of a few irregular English nouns. Thanks @jonmcoe.

4.1.13 0.9.1 (2015-06-10)

Bug fixes:

- Fix DecisionTreeClassifier.pprint for compatibility with nltk>=3.0.2.
- Translation no longer adds erroneous whitespace around punctuation characters (#83). Thanks @AdrianLC for reporting and thanks @jschnurr for the patch.

4.1.14 0.9.0 (2014-09-15)

- TextBlob now depends on NLTK 3. The vendored version of NLTK has been removed.
- Fix bug that raised a SyntaxError when translating text with non-ascii characters on Python 3.
- Fix bug that showed “double-escaped” unicode characters in translator output (issue #56). Thanks Evan Dempsey.
- *Backwards-incompatible*: Completely remove import text.blob. You should import textblob instead.

- *Backwards-incompatible:* Completely remove `PerceptronTagger`. Install `textblob-aptagger` instead.
- *Backwards-incompatible:* Rename `TextBlobException` to `TextBlobError` and `MissingCorpusException` to `MissingCorpusError`.
- *Backwards-incompatible:* Format classes are passed a file object rather than a file path.
- *Backwards-incompatible:* If training a classifier with data from a file, you must pass a file object (rather than a file path).
- Updated English sentiment corpus.
- Add `feature_extractor` parameter to `NaiveBayesAnalyzer`.
- Add `textblob.formats.get_registry()` and `textblob.formats.register()` which allows users to register custom data source formats.
- Change `BaseClassifier.detect` from a staticmethod to a classmethod.
- Improved docs.
- Tested on Python 3.4.

4.1.15 0.8.4 (2014-02-02)

- Fix display (`__repr__`) of `WordList` slices on Python 3.
- Add `download_corpora` module. Corpora must now be downloaded using `python -m textblob.download_corpora`.

4.1.16 0.8.3 (2013-12-29)

- Sentiment analyzers return namedtuples, e.g. `Sentiment(polarity=0.12, subjectivity=0.34)`.
- Memory usage improvements to `NaiveBayesAnalyzer` and `basic_extractor` (default feature extractor for classifiers module).
- Add `textblob.tokenizers.sent_tokenize` and `textblob.tokenizers.word_tokenize` convenience functions.
- Add `textblob.classifiers.MaxEntClassifier`.
- Improved `NLTKTagger`.

4.1.17 0.8.2 (2013-12-21)

- Fix bug in spelling correction that stripped some punctuation (Issue #48).
- Various improvements to spelling correction: preserves whitespace characters (Issue #12); handle contractions and punctuation between words. Thanks @davidnk.
- Make `TextBlob.words` more memory-efficient.
- Translator now sends POST instead of GET requests. This allows for larger bodies of text to be translated (Issue #49).
- Update pattern tagger for better accuracy.

4.1.18 0.8.1 (2013-11-16)

- Fix bug that caused `ValueError` upon sentence tokenization. This removes modifications made to the NLTK sentence tokenizer.
- Add `Word.lemmatize()` method that allows passing in a part-of-speech argument.
- `Word.lemma` returns correct part of speech for `Word` objects that have their `pos` attribute set. Thanks @RomanYankovsky.

4.1.19 0.8.0 (2013-10-23)

- *Backwards-incompatible*: Renamed package to `textblob`. This avoids clashes with other namespaces called `text`. `TextBlob` should now be imported with `from textblob import TextBlob`.
- Update pattern resources for improved parser accuracy.
- Update NLTK.
- Allow `Translator` to connect to proxy server.
- `PerceptronTagger` completely deprecated. Install the `textblob-aptagger` extension instead.

4.1.20 0.7.1 (2013-09-30)

- Bugfix updates.
- Fix bug in feature extraction for `NaiveBayesClassifier`.
- `basic_extractor` is now case-sensitive, e.g. `contains(I) != contains(i)`
- Fix `repr` output when a `TextBlob` contains non-ascii characters.
- Fix part-of-speech tagging with `PatternTagger` on Windows.
- Suppress warning about not having `scikit-learn` installed.

4.1.21 0.7.0 (2013-09-25)

- Wordnet integration. `Word` objects have `synsets` and `definitions` properties. The `text.wordnet` module allows you to create `Synset` and `Lemma` objects directly.
- Move all English-specific code to its own module, `text.en`.
- Basic extensions framework in place. `TextBlob` has been refactored to make it easier to develop extensions.
- Add `text.classifiers.PositiveNaiveBayesClassifier`.
- Update NLTK.
- `NLTKTagger` now working on Python 3.
- Fix `__str__` behavior. `print(blob)` should now print non-ascii text correctly in both Python 2 and 3.
- *Backwards-incompatible*: All abstract base classes have been moved to the `text.base` module.
- *Backwards-incompatible*: `PerceptronTagger` will now be maintained as an extension, `textblob-aptagger`. Instantiating a `text.taggers.PerceptronTagger()` will raise a `DeprecationWarning`.

4.1.22 0.6.3 (2013-09-15)

- Word tokenization fix: Words that stem from a contraction will still have an apostrophe, e.g. "Let's" => ["Let", "'s"].
- Fix bug with comparing blobs to strings.
- Add `text.taggers.PerceptronTagger`, a fast and accurate POS tagger. Thanks @syllogism.
- Note for Python 3 users: You may need to update your corpora, since NLTK master has reorganized its corpus system. Just run `curl https://raw.githubusercontent.com/sloria/TextBlob/master/download_corpora.py | python` again.
- Add `download_corpora_lite.py` script for getting the minimum corpora requirements for TextBlob's basic features.

4.1.23 0.6.2 (2013-09-05)

- Fix bug that resulted in a `UnicodeEncodeError` when tagging text with non-ascii characters.
- Add `DecisionTreeClassifier`.
- Add `labels()` and `train()` methods to classifiers.

4.1.24 0.6.1 (2013-09-01)

- Classifiers can be trained and tested on CSV, JSON, or TSV data.
- Add basic WordNet lemmatization via the `Word.lemma` property.
- `WordList.pluralize()` and `WordList.singularize()` methods return `WordList` objects.

4.1.25 0.6.0 (2013-08-25)

- Add Naive Bayes classification. New `text.classifiers` module, `TextBlob.classify()`, and `Sentence.classify()` methods.
- Add parsing functionality via the `TextBlob.parse()` method. The `text.parsers` module currently has one implementation (`PatternParser`).
- Add spelling correction. This includes the `TextBlob.correct()` and `Word.spellcheck()` methods.
- Update NLTK.
- Backwards incompatible: `clean_html` has been deprecated, just as it has in NLTK. Use Beautiful Soup's `soup.get_text()` method for HTML-cleaning instead.
- Slight API change to language translation: if `from_lang` isn't specified, attempts to detect the language.
- Add `itokenize()` method to tokenizers that returns a generator instead of a list of tokens.

4.1.26 0.5.3 (2013-08-21)

- Unicode fixes: This fixes a bug that sometimes raised a `UnicodeEncodeError` upon creating accessing sentences for TextBlobs with non-ascii characters.
- Update NLTK

4.1.27 0.5.2 (2013-08-14)

- Important patch update for NLTK users: Fix bug with importing TextBlob if local NLTK is installed.
- Fix bug with computing start and end indices of sentences.

4.1.28 0.5.1 (2013-08-13)

- Fix bug that disallowed display of non-ascii characters in the Python REPL.
- Backwards incompatible: Restore `blob.json` property for backwards compatibility with `textblob<=0.3.10`. Add a `to_json()` method that takes the same arguments as `json.dumps`.
- Add `WordList.append` and `WordList.extend` methods that append `Word` objects.

4.1.29 0.5.0 (2013-08-10)

- Language translation and detection API!
- Add `text.sentiments` module. Contains the `PatternAnalyzer` (default implementation) as well as a `NaiveBayesAnalyzer`.
- Part-of-speech tags can be accessed via `TextBlob.tags` or `TextBlob.pos_tags`.
- Add `polarity` and `subjectivity` helper properties.

4.1.30 0.4.0 (2013-08-05)

- New `text.tokenizers` module with `WordTokenizer` and `SentenceTokenizer`. Tokenizer instances (from either `textblob` itself or `NLTK`) can be passed to `TextBlob`'s constructor. Tokens are accessed through the new `tokens` property.
- New `Blobber` class for creating `TextBlobs` that share the same tagger, tokenizer, and `np_extractor`.
- Add `ngrams` method.
- Backwards-incompatible: `TextBlob.json()` is now a method, not a property. This allows you to pass arguments (the same that you would pass to `json.dumps()`).
- New home for documentation: <https://textblob.readthedocs.io/>
- Add parameter for cleaning HTML markup from text.
- Minor improvement to word tokenization.
- Updated `NLTK`.
- Fix bug with adding blobs to bytestrings.

4.1.31 0.3.10 (2013-08-02)

- Bundled `NLTK` no longer overrides local installation.
- Fix sentiment analysis of text with non-ascii characters.

4.1.32 0.3.9 (2013-07-31)

- Updated nltk.
- ConllExtractor is now Python 3-compatible.
- Improved sentiment analysis.
- Blobs are equal (with `==`) to their string counterparts.
- Added instructions to install textblob without nltk bundled.
- Dropping official 3.1 and 3.2 support.

4.1.33 0.3.8 (2013-07-30)

- Importing TextBlob is now **much faster**. This is because the noun phrase parsers are trained only on the first call to `noun_phrases` (instead of training them every time you import TextBlob).
- Add `text.taggers` module which allows user to change which POS tagger implementation to use. Currently supports PatternTagger and NLTKTagger (NLTKTagger only works with Python 2).
- NPExtractor and Tagger objects can be passed to TextBlob's constructor.
- Fix bug with POS-tagger not tagging one-letter words.
- Rename `text/np_extractor.py` -> `text/np_extractors.py`
- Add `run_tests.py` script.

4.1.34 0.3.7 (2013-07-28)

- Every word in a Blob or Sentence is a Word instance which has methods for inflection, e.g `word.pluralize()` and `word.singularize()`.
- Updated the `np_extractor` module. Now has a new implementation, `ConllExtractor` that uses the Conll2000 chunking corpus. Only works on Py2.

4.2 Authors

4.2.1 Development Lead

- Steven Loria <sloria1@gmail.com> @sloria

4.2.2 Contributors (chronological)

- Pete Keen @peterkeen
- Matthew Honnibal @syllogism
- Roman Yankovsky @RomanYankovsky
- David Karesh @davidnk
- Evan Dempsey @evandempsey
- Wesley Childs @mrchilds

- Jeff Schnurr @jschnurr
- Adel Qalieh @adelq
- Lage Ragnarsson @lragnarsson
- Jonathon Coe @jonmcoe
- Adrián López Calvo @AdrianLC
- Nitish Kulshrestha @nitkul
- Jhon Eslava @EpicJhon
- @jcalbert
- Tyler James Harden @tylerjharden
- @pavelmalai
- Jeff Kolb @jeffakolb
- Daniel Ong @danong
- Jamie Moschella @jammmo
- Roman Korolev @roman-y-korolev

4.3 Contributing guidelines

4.3.1 In General

- [PEP 8](#), when sensible.
- Conventions *and* configuration.
- TextBlob wraps functionality in NLTK and pattern.en. Anything outside of that should be written as an extension.
- Test ruthlessly. Write docs for new features.
- Even more important than Test-Driven Development—*Human-Driven Development*.
- These guidelines may—and probably will—change.

4.3.2 In Particular

Questions, Feature Requests, Bug Reports, and Feedback...

... should all be reported on the [Github Issue Tracker](#).

Setting Up for Local Development

1. Fork [TextBlob](#) on Github.

```
$ git clone https://github.com/sloria/TextBlob.git
$ cd TextBlob
```

2. Install development requirements. It is highly recommended that you use a virtualenv.

```
# After activating your virtualenv
$ pip install -r dev-requirements.txt
```

3. Install TextBlob in develop mode.

```
$ python setup.py develop
```

Developing Extensions

Extensions are packages with the name `textblob-something`, where “something” is the name of your extension. Extensions should be imported with `import textblob_something`.

Model Extensions

To create a new extension for a part-of-speech tagger, sentiment analyzer, noun phrase extractor, classifier, tokenizer, or parser, simply create a module that has a class that implements the correct interface from `textblob.base`. For example, a tagger might look like this:

```
from textblob.base import BaseTagger

class MyTagger(BaseTagger):
    def tag(self, text):
        # Your implementation goes here
```

Language Extensions

The process for developing language extensions is the same as developing model extensions. Create your part-of-speech taggers, tokenizers, parsers, etc. in the language of your choice. Packages should be named `textblob-xx` where “xx” is the two- or three-letter language code ([Language code reference](#)).

To see examples of existing extensions, visit the [Extensions](#) page.

Check out the [API reference](#) for more info on the model interfaces.

Git Branch Structure

TextBlob loosely follows Vincent Driessen’s [Successful Git Branching Model](#). In practice, the following branch conventions are used:

dev The next release branch.

master Current production release on PyPI.

Pull Requests

1. Create a new local branch.

```
$ git checkout -b name-of-feature
```

2. Commit your changes. Write [good commit messages](#).


```
$ git commit -m "Detailed commit message"
$ git push origin name-of-feature
```

3. Before submitting a pull request, check the following:

- If the pull request adds functionality, it is tested and the docs are updated.
- If you’ve developed an extension, it is on the [Extensions List](#).
- The pull request works on Python 2.7, 3.4, 3.5, 3.6, and PyPy. Use `tox` to verify that it does.
- You’ve added yourself to `AUTHORS.rst`.

4. Submit a pull request to the `sloria:dev` branch.

Running tests

To run all the tests:

```
$ python run_tests.py
```

To skip slow tests:

```
$ python run_tests.py fast
```

To skip tests that require internet:

```
$ python run_tests.py no-internet
```

To get test coverage reports (must have coverage installed):

```
$ python run_tests.py cover
```

To run tests on Python 2.7, 3.4, 3.5, and 3.6 virtual environments (must have each interpreter installed):

```
$ tox
```

Documentation

Contributions to the documentation are welcome. Documentation is written in [reStructured Text](#) (rST). A quick rST reference can be found [here](#). Builds are powered by [Sphinx](#).

To build docs:

```
$ invoke docs -b
```

The `-b` (for “browse”) automatically opens up the docs in your browser after building.

t

- `textblob.base`, 36
- `textblob.blob`, 9
- `textblob.classifiers`, 40
- `textblob.en.np_extractors`, 38
- `textblob.en.parsers`, 39
- `textblob.en.sentiments`, 39
- `textblob.en.taggers`, 38
- `textblob.exceptions`, 49
- `textblob.formats`, 48
- `textblob.tokenizers`, 37

Symbols

`__call__()` (*textblob.blob.Blobber* method), 47
`__init__()` (*textblob.blob.Blobber* method), 48
`__repr__()` (*textblob.blob.Blobber* method), 48
`__str__()` (*textblob.blob.Blobber* method), 48

A

`accuracy()` (*textblob.classifiers.DecisionTreeClassifier* method), 41
`accuracy()` (*textblob.classifiers.MaxEntClassifier* method), 42
`accuracy()` (*textblob.classifiers.NaiveBayesClassifier* method), 44
`accuracy()` (*textblob.classifiers.NLTKClassifier* method), 43
`accuracy()` (*textblob.classifiers.PositiveNaiveBayesClassifier* method), 46
`analyze()` (*textblob.base.BaseSentimentAnalyzer* method), 36
`analyze()` (*textblob.en.sentiments.NaiveBayesAnalyzer* method), 39
`analyze()` (*textblob.en.sentiments.PatternAnalyzer* method), 39
`append()` (*textblob.blob.WordList* method), 35

B

`BaseBlob` (class in *textblob.blob*), 20
`BaseClassifier` (class in *textblob.classifiers*), 40
`BaseFormat` (class in *textblob.formats*), 48
`BaseNPExtractor` (class in *textblob.base*), 36
`BaseParser` (class in *textblob.base*), 36
`BaseSentimentAnalyzer` (class in *textblob.base*), 36
`BaseTagger` (class in *textblob.base*), 36
`BaseTokenizer` (class in *textblob.base*), 36
`basic_extractor()` (in module *textblob.classifiers*), 47
`Blobber` (class in *textblob.blob*), 23, 47

C

`capitalize()` (*textblob.blob.Word* method), 31
`center()` (*textblob.blob.Word* method), 31
`ChunkParser` (class in *textblob.en.np_extractors*), 38
`classifier` (*textblob.classifiers.BaseClassifier* attribute), 40
`classifier` (*textblob.classifiers.DecisionTreeClassifier* attribute), 41
`classifier` (*textblob.classifiers.MaxEntClassifier* attribute), 42
`classifier` (*textblob.classifiers.NaiveBayesClassifier* attribute), 44
`classifier` (*textblob.classifiers.NLTKClassifier* attribute), 43
`classifier` (*textblob.classifiers.PositiveNaiveBayesClassifier* attribute), 46
`classify()` (*textblob.blob.BaseBlob* method), 20
`classify()` (*textblob.blob.Sentence* method), 24
`classify()` (*textblob.blob.TextBlob* method), 28
`classify()` (*textblob.classifiers.BaseClassifier* method), 41
`classify()` (*textblob.classifiers.DecisionTreeClassifier* method), 41
`classify()` (*textblob.classifiers.MaxEntClassifier* method), 42
`classify()` (*textblob.classifiers.NaiveBayesClassifier* method), 44
`classify()` (*textblob.classifiers.NLTKClassifier* method), 44
`classify()` (*textblob.classifiers.PositiveNaiveBayesClassifier* method), 46
`ConllExtractor` (class in *textblob.en.np_extractors*), 38
`contains_extractor()` (in module *textblob.classifiers*), 47
`correct()` (*textblob.blob.BaseBlob* method), 20
`correct()` (*textblob.blob.Sentence* method), 24
`correct()` (*textblob.blob.TextBlob* method), 28
`correct()` (*textblob.blob.Word* method), 31

count() (*textblob.blob.Word* method), 31
count() (*textblob.blob.WordList* method), 35
CSV (*class in textblob.formats*), 48

D

DecisionTreeClassifier (*class in textblob.classifiers*), 41
decode() (*textblob.blob.Word* method), 31
define() (*textblob.blob.Word* method), 32
definitions (*textblob.blob.Word* attribute), 32
DelimitedFormat (*class in textblob.formats*), 48
DeprecationError, 50
detect() (*in module textblob.formats*), 49
detect() (*textblob.formats.BaseFormat* class method), 48
detect() (*textblob.formats.CSV* class method), 48
detect() (*textblob.formats.DelimitedFormat* class method), 49
detect() (*textblob.formats.JSON* class method), 49
detect() (*textblob.formats.TSV* class method), 49
detect_language() (*textblob.blob.BaseBlob* method), 20
detect_language() (*textblob.blob.Sentence* method), 24
detect_language() (*textblob.blob.TextBlob* method), 28
detect_language() (*textblob.blob.Word* method), 32
dict (*textblob.blob.Sentence* attribute), 25

E

encode() (*textblob.blob.Word* method), 32
end (*textblob.blob.Sentence* attribute), 25
end_index (*textblob.blob.Sentence* attribute), 25
ends_with() (*textblob.blob.BaseBlob* method), 21
ends_with() (*textblob.blob.Sentence* method), 25
ends_with() (*textblob.blob.TextBlob* method), 28
endswith() (*textblob.blob.BaseBlob* method), 21
endswith() (*textblob.blob.Sentence* method), 25
endswith() (*textblob.blob.TextBlob* method), 28
endswith() (*textblob.blob.Word* method), 32
evaluate() (*textblob.en.np_extractors.ChunkParser* method), 38
expandtabs() (*textblob.blob.Word* method), 32
extend() (*textblob.blob.WordList* method), 35
extract() (*textblob.base.BaseNPEExtractor* method), 36
extract() (*textblob.en.np_extractors.ConllExtractor* method), 38
extract() (*textblob.en.np_extractors.FastNPEExtractor* method), 39
extract_features() (*textblob.classifiers.BaseClassifier* method), 41

extract_features() (*textblob.classifiers.DecisionTreeClassifier* method), 41
extract_features() (*textblob.classifiers.MaxEntClassifier* method), 43
extract_features() (*textblob.classifiers.NaiveBayesClassifier* method), 45
extract_features() (*textblob.classifiers.NLTKClassifier* method), 44
extract_features() (*textblob.classifiers.PositiveNaiveBayesClassifier* method), 46

F

FastNPEExtractor (*class in textblob.en.np_extractors*), 39
find() (*textblob.blob.BaseBlob* method), 21
find() (*textblob.blob.Sentence* method), 25
find() (*textblob.blob.TextBlob* method), 28
find() (*textblob.blob.Word* method), 32
format() (*textblob.blob.BaseBlob* method), 21
format() (*textblob.blob.Sentence* method), 25
format() (*textblob.blob.TextBlob* method), 28
format() (*textblob.blob.Word* method), 32
FormatError, 50

G

get_registry() (*in module textblob.formats*), 49
get_synsets() (*textblob.blob.Word* method), 32
grammar() (*textblob.en.np_extractors.ChunkParser* method), 38

I

index() (*textblob.blob.BaseBlob* method), 21
index() (*textblob.blob.Sentence* method), 25
index() (*textblob.blob.TextBlob* method), 28
index() (*textblob.blob.Word* method), 32
index() (*textblob.blob.WordList* method), 35
informative_features() (*textblob.classifiers.NaiveBayesClassifier* method), 45
insert() (*textblob.blob.WordList* method), 35
isalnum() (*textblob.blob.Word* method), 32
isalpha() (*textblob.blob.Word* method), 33
isdecimal() (*textblob.blob.Word* method), 33
isdigit() (*textblob.blob.Word* method), 33
islower() (*textblob.blob.Word* method), 33
isnumeric() (*textblob.blob.Word* method), 33
isspace() (*textblob.blob.Word* method), 33
istitle() (*textblob.blob.Word* method), 33
isupper() (*textblob.blob.Word* method), 33

`itokenize()` (*textblob.base.BaseTokenizer method*), 36
`itokenize()` (*textblob.tokenizers.SentenceTokenizer method*), 37
`itokenize()` (*textblob.tokenizers.WordTokenizer method*), 37

J

`join()` (*textblob.blob.BaseBlob method*), 21
`join()` (*textblob.blob.Sentence method*), 25
`join()` (*textblob.blob.TextBlob method*), 28
`join()` (*textblob.blob.Word method*), 33
JSON (*class in textblob.formats*), 49
`json` (*textblob.blob.TextBlob attribute*), 29

L

`labels()` (*textblob.classifiers.BaseClassifier method*), 41
`labels()` (*textblob.classifiers.DecisionTreeClassifier method*), 41
`labels()` (*textblob.classifiers.MaxEntClassifier method*), 43
`labels()` (*textblob.classifiers.NaiveBayesClassifier method*), 45
`labels()` (*textblob.classifiers.NLTKClassifier method*), 44
`labels()` (*textblob.classifiers.PositiveNaiveBayesClassifier method*), 46
`lemma` (*textblob.blob.Word attribute*), 33
`lemmatize()` (*textblob.blob.Word method*), 33
`lemmatize()` (*textblob.blob.WordList method*), 35
`ljust()` (*textblob.blob.Word method*), 33
`lower()` (*textblob.blob.BaseBlob method*), 21
`lower()` (*textblob.blob.Sentence method*), 25
`lower()` (*textblob.blob.TextBlob method*), 29
`lower()` (*textblob.blob.Word method*), 33
`lower()` (*textblob.blob.WordList method*), 35
`lstrip()` (*textblob.blob.Word method*), 33

M

MaxEntClassifier (*class in textblob.classifiers*), 42
MissingCorpusError, 49

N

NaiveBayesAnalyzer (*class in textblob.en.sentiments*), 39
NaiveBayesClassifier (*class in textblob.classifiers*), 44
`ngrams()` (*textblob.blob.BaseBlob method*), 21
`ngrams()` (*textblob.blob.Sentence method*), 25
`ngrams()` (*textblob.blob.TextBlob method*), 29
`nltk_class` (*textblob.classifiers.DecisionTreeClassifier attribute*), 41

`nltk_class` (*textblob.classifiers.MaxEntClassifier attribute*), 43
`nltk_class` (*textblob.classifiers.NaiveBayesClassifier attribute*), 45
`nltk_class` (*textblob.classifiers.NLTKClassifier attribute*), 44
NLTKClassifier (*class in textblob.classifiers*), 43
NLTKTagger (*class in textblob.en.taggers*), 38
NotTranslated, 50
`noun_phrases` (*textblob.blob.BaseBlob attribute*), 21
`noun_phrases` (*textblob.blob.Sentence attribute*), 25
`noun_phrases` (*textblob.blob.TextBlob attribute*), 29
`np_counts` (*textblob.blob.BaseBlob attribute*), 21
`np_counts` (*textblob.blob.Sentence attribute*), 25
`np_counts` (*textblob.blob.TextBlob attribute*), 29

P

`parse()` (*textblob.base.BaseParser method*), 36
`parse()` (*textblob.blob.BaseBlob method*), 21
`parse()` (*textblob.blob.Sentence method*), 25
`parse()` (*textblob.blob.TextBlob method*), 29
`parse()` (*textblob.en.np_extractors.ChunkParser method*), 38
`parse()` (*textblob.en.parsers.PatternParser method*), 40
`parse_all()` (*textblob.en.np_extractors.ChunkParser method*), 38
`parse_one()` (*textblob.en.np_extractors.ChunkParser method*), 38
`parse_sents()` (*textblob.en.np_extractors.ChunkParser method*), 38
`partition()` (*textblob.blob.Word method*), 33
PatternAnalyzer (*class in textblob.en.sentiments*), 39
PatternParser (*class in textblob.en.parsers*), 39
PatternTagger (*class in textblob.en.taggers*), 38
`pluralize()` (*textblob.blob.Word method*), 33
`pluralize()` (*textblob.blob.WordList method*), 35
`polarity` (*textblob.blob.BaseBlob attribute*), 21
`polarity` (*textblob.blob.Sentence attribute*), 25
`polarity` (*textblob.blob.TextBlob attribute*), 29
`pop()` (*textblob.blob.WordList method*), 35
`pos_tags` (*textblob.blob.BaseBlob attribute*), 21
`pos_tags` (*textblob.blob.Sentence attribute*), 25
`pos_tags` (*textblob.blob.TextBlob attribute*), 29
PositiveNaiveBayesClassifier (*class in textblob.classifiers*), 45
`pprint()` (*textblob.classifiers.DecisionTreeClassifier method*), 41
`pretty_format()` (*textblob.classifiers.DecisionTreeClassifier method*), 42
`prob_classify()` (*textblob.classifiers.MaxEntClassifier method*), 43

`prob_classify()` (*textblob.classifiers.NaiveBayesClassifier* method), 45
`pseudocode()` (*textblob.classifiers.DecisionTreeClassifier* method), 42

R

`raw_sentences` (*textblob.blob.TextBlob* attribute), 29
`register()` (in module *textblob.formats*), 49
`remove()` (*textblob.blob.WordList* method), 36
`replace()` (*textblob.blob.BaseBlob* method), 22
`replace()` (*textblob.blob.Sentence* method), 26
`replace()` (*textblob.blob.TextBlob* method), 29
`replace()` (*textblob.blob.Word* method), 33
`RETURN_TYPE` (*textblob.en.sentiments.NaiveBayesAnalyzer* attribute), 39
`RETURN_TYPE` (*textblob.en.sentiments.PatternAnalyzer* attribute), 39
`reverse()` (*textblob.blob.WordList* method), 36
`rfind()` (*textblob.blob.BaseBlob* method), 22
`rfind()` (*textblob.blob.Sentence* method), 26
`rfind()` (*textblob.blob.TextBlob* method), 29
`rfind()` (*textblob.blob.Word* method), 34
`rindex()` (*textblob.blob.BaseBlob* method), 22
`rindex()` (*textblob.blob.Sentence* method), 26
`rindex()` (*textblob.blob.TextBlob* method), 29
`rindex()` (*textblob.blob.Word* method), 34
`rjust()` (*textblob.blob.Word* method), 34
`rpartition()` (*textblob.blob.Word* method), 34
`rsplit()` (*textblob.blob.Word* method), 34
`rstrip()` (*textblob.blob.Word* method), 34

S

`sent_tokenize` (in module *textblob.tokenizers*), 37
`Sentence` (class in *textblob.blob*), 24
`sentences` (*textblob.blob.TextBlob* attribute), 29
`SentenceTokenizer` (class in *textblob.tokenizers*), 37
`sentiment` (*textblob.blob.BaseBlob* attribute), 22
`sentiment` (*textblob.blob.Sentence* attribute), 26
`sentiment` (*textblob.blob.TextBlob* attribute), 29
`sentiment_assessments` (*textblob.blob.BaseBlob* attribute), 22
`sentiment_assessments` (*textblob.blob.Sentence* attribute), 26
`sentiment_assessments` (*textblob.blob.TextBlob* attribute), 30
`serialized` (*textblob.blob.TextBlob* attribute), 30
`show_informative_features()` (*textblob.classifiers.NaiveBayesClassifier* method), 45
`singularize()` (*textblob.blob.Word* method), 34
`singularize()` (*textblob.blob.WordList* method), 36
`sort()` (*textblob.blob.WordList* method), 36
`spellcheck()` (*textblob.blob.Word* method), 34

`split()` (*textblob.blob.BaseBlob* method), 22
`split()` (*textblob.blob.Sentence* method), 26
`split()` (*textblob.blob.TextBlob* method), 30
`split()` (*textblob.blob.Word* method), 34
`splitlines()` (*textblob.blob.Word* method), 34
`start` (*textblob.blob.Sentence* attribute), 26
`start_index` (*textblob.blob.Sentence* attribute), 26
`starts_with()` (*textblob.blob.BaseBlob* method), 22
`starts_with()` (*textblob.blob.Sentence* method), 26
`starts_with()` (*textblob.blob.TextBlob* method), 30
`startswith()` (*textblob.blob.BaseBlob* method), 22
`startswith()` (*textblob.blob.Sentence* method), 26
`startswith()` (*textblob.blob.TextBlob* method), 30
`startswith()` (*textblob.blob.Word* method), 34
`stem()` (*textblob.blob.Word* method), 34
`stem()` (*textblob.blob.WordList* method), 36
`strip()` (*textblob.blob.BaseBlob* method), 22
`strip()` (*textblob.blob.Sentence* method), 26
`strip()` (*textblob.blob.TextBlob* method), 30
`strip()` (*textblob.blob.Word* method), 34
`subjectivity` (*textblob.blob.BaseBlob* attribute), 22
`subjectivity` (*textblob.blob.Sentence* attribute), 26
`subjectivity` (*textblob.blob.TextBlob* attribute), 30
`swapcase()` (*textblob.blob.Word* method), 35
`synsets` (*textblob.blob.Word* attribute), 35

T

`tag()` (*textblob.base.BaseTagger* method), 36
`tag()` (*textblob.en.taggers.NLTKTagger* method), 38
`tag()` (*textblob.en.taggers.PatternTagger* method), 38
`tags` (*textblob.blob.BaseBlob* attribute), 22
`tags` (*textblob.blob.Sentence* attribute), 26
`tags` (*textblob.blob.TextBlob* attribute), 30
`TextBlob` (class in *textblob.blob*), 27
`textblob.base` (module), 36
`textblob.blob` (module), 9, 20
`textblob.classifiers` (module), 40
`textblob.en.np_extractors` (module), 38
`textblob.en.parsers` (module), 39
`textblob.en.sentiments` (module), 39
`textblob.en.taggers` (module), 38
`textblob.exceptions` (module), 49
`textblob.formats` (module), 48
`textblob.tokenizers` (module), 37
`TextBlobError`, 49
`title()` (*textblob.blob.BaseBlob* method), 23
`title()` (*textblob.blob.Sentence* method), 27
`title()` (*textblob.blob.TextBlob* method), 30
`title()` (*textblob.blob.Word* method), 35
`to_iterable()` (*textblob.formats.BaseFormat* method), 48
`to_iterable()` (*textblob.formats.CSV* method), 48
`to_iterable()` (*textblob.formats.DelimitedFormat* method), 49

[to_iterable\(\)](#) (*textblob.formats.JSON method*), 49
[to_iterable\(\)](#) (*textblob.formats.TSV method*), 49
[to_json\(\)](#) (*textblob.blob.TextBlob method*), 30
[tokenize\(\)](#) (*textblob.base.BaseTokenizer method*), 37
[tokenize\(\)](#) (*textblob.blob.BaseBlob method*), 23
[tokenize\(\)](#) (*textblob.blob.Sentence method*), 27
[tokenize\(\)](#) (*textblob.blob.TextBlob method*), 30
[tokenize\(\)](#) (*textblob.tokenizers.SentenceTokenizer method*), 37
[tokenize\(\)](#) (*textblob.tokenizers.WordTokenizer method*), 37
[tokens](#) (*textblob.blob.BaseBlob attribute*), 23
[tokens](#) (*textblob.blob.Sentence attribute*), 27
[tokens](#) (*textblob.blob.TextBlob attribute*), 30
[train\(\)](#) (*textblob.classifiers.BaseClassifier method*), 41
[train\(\)](#) (*textblob.classifiers.DecisionTreeClassifier method*), 42
[train\(\)](#) (*textblob.classifiers.MaxEntClassifier method*), 43
[train\(\)](#) (*textblob.classifiers.NaiveBayesClassifier method*), 45
[train\(\)](#) (*textblob.classifiers.NLTKClassifier method*), 44
[train\(\)](#) (*textblob.classifiers.PositiveNaiveBayesClassifier method*), 46
[train\(\)](#) (*textblob.en.np_extractors.ChunkParser method*), 38
[train\(\)](#) (*textblob.en.sentiments.NaiveBayesAnalyzer method*), 39
[translate\(\)](#) (*textblob.blob.BaseBlob method*), 23
[translate\(\)](#) (*textblob.blob.Sentence method*), 27
[translate\(\)](#) (*textblob.blob.TextBlob method*), 31
[translate\(\)](#) (*textblob.blob.Word method*), 35
[TranslatorError](#), 50
[TSV](#) (*class in textblob.formats*), 49

U

[update\(\)](#) (*textblob.classifiers.DecisionTreeClassifier method*), 42
[update\(\)](#) (*textblob.classifiers.MaxEntClassifier method*), 43
[update\(\)](#) (*textblob.classifiers.NaiveBayesClassifier method*), 45
[update\(\)](#) (*textblob.classifiers.NLTKClassifier method*), 44
[update\(\)](#) (*textblob.classifiers.PositiveNaiveBayesClassifier method*), 47
[upper\(\)](#) (*textblob.blob.BaseBlob method*), 23
[upper\(\)](#) (*textblob.blob.Sentence method*), 27
[upper\(\)](#) (*textblob.blob.TextBlob method*), 31
[upper\(\)](#) (*textblob.blob.Word method*), 35
[upper\(\)](#) (*textblob.blob.WordList method*), 36

W

[Word](#) (*class in textblob.blob*), 31
[word_counts](#) (*textblob.blob.BaseBlob attribute*), 23
[word_counts](#) (*textblob.blob.Sentence attribute*), 27
[word_counts](#) (*textblob.blob.TextBlob attribute*), 31
[word_tokenize\(\)](#) (*in module textblob.tokenizers*), 37
[WordList](#) (*class in textblob.blob*), 35
[words](#) (*textblob.blob.BaseBlob attribute*), 23
[words](#) (*textblob.blob.Sentence attribute*), 27
[words](#) (*textblob.blob.TextBlob attribute*), 31
[WordTokenizer](#) (*class in textblob.tokenizers*), 37

Z

[zfill\(\)](#) (*textblob.blob.Word method*), 35