textblob-de Documentation

Release 0.4.2a1

Markus Killer

October 03, 2014

Release 0.4.2a1 (*Changelog*)

TextBlob is a Python (2 and 3) library for processing textual data. It is being developed by Steven Loria. 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.

'textblob-de' is the **German language extension** for TextBlob.

```
from textblob_de import TextBlobDE

text = '''
"Der Blob" macht in seiner unbekümmert-naiven Weise einfach nur Spass.
Er hat eben den gewissen Charme, bei dem auch die eher hölzerne Regie und
das konfuse Drehbuch nicht weiter stören.
'''

blob = TextBlobDE(text)
blob.tags           # [('Der', 'DT'), ('Blob', 'NN'), ('macht', 'VB'),
                    #  ('in', 'IN'), ('seiner', 'PRP$'), ...]

blob.noun_phrases   # WordList(['Der Blob', 'seiner unbekümmert-naiven Weise',
                    #           'den gewissen Charme', 'hölzerne Regie',
                    #           'konfuse Drehbuch'])


for sentence in blob.sentences:
    print(sentence.sentiment.polarity)
# 1.0
# 0.0

blob.translate(to="es")  # '" The Blob " hace a su manera ingenua...'
```

For a complete overview of *TextBlob* 's features, see documentation of the main TextBlob library.

The docs of the German language extension focus on additions/differences to TextBlob and provide a detailed API reference.

CHAPTER

ONE

GUIDE

1.1 textblob-de README

German language support for TextBlob by Steven Loria.

This python package is being developed as a `TextBlob` **Language Extension**. See Extension Guidelines for details.

1.1.1 Features

- All directly accessible `textblob_de` classes (e.g. `Sentence()` or `Word()`) are initialized with default models for German
- Properties or methods that do not yet work for German raise a `NotImplementedError`
- German sentence boundary detection and tokenization (`NLTKPunktTokenizer`)
- Consistent use of specified tokenizer for all tools (`NLTKPunktTokenizer` or `PatternTokenizer`)
- Part-of-speech tagging (`PatternTagger`) with keyword `include_punc=True` (defaults to `False`)
- Parsing (`PatternParser`) with all `pattern` keywords, plus `pprint=True` (defaults to `False`)
- Noun Phrase Extraction (`PatternParserNPExtractor`)
- Lemmatization (`PatternParserLemmatizer`)
- Polarity detection (`PatternAnalyzer`) - Still **EXPERIMENTAL**, does not yet have information on subjectivity
- Full `pattern.text.de` API support on Python3
- Supports Python 2 and 3
- See working features overview for details

1.1.2 Installing/Upgrading

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

Or the latest development release (apparently this does not always work on Windows see issues #1744/5 for details):

```
$ pip install -U git+https://github.com/markuskiller/textblob-de.git@dev
$ python -m textblob.download_corpora
```

Note: `TextBlob` will be installed/upgraded automatically when running `pip install`. The second line (`python -m textblob.download_corpora`) downloads/updates nltk corpora and language models used in `TextBlob`.

1.1.3 Usage

```
>>> from textblob_de import TextBlobDE as TextBlob
>>> text = '''Heute ist der 3. Mai 2014 und Dr. Meier feiert seinen 43. Geburtstag.
Ich muss unbedingt daran denken, Mehl, usw. für einen Kuchen einzukaufen. Aber leider
habe ich nur noch EUR 3.50 in meiner Brieftasche.'''
>>> blob = TextBlob(text)
>>> blob.sentences
[Sentence("Heute ist der 3. Mai 2014 und Dr. Meier feiert seinen 43. Geburtstag."),
 Sentence("Ich muss unbedingt daran denken, Mehl, usw. für einen Kuchen einzukaufen."),
 Sentence("Aber leider habe ich nur noch EUR 3.50 in meiner Brieftasche.")]
>>> blob.tokens
WordList(['Heute', 'ist', 'der', '3.', 'Mai', ...]
>>> blob.tags
[('Heute', 'RB'), ('ist', 'VB'), ('der', 'DT'), ('3.', 'LS'), ('Mai', 'NN'),
('2014', 'CD'), ...]
# Default: Only noun_phrases that consist of two or more meaningful parts are displayed.
# Not perfect, but a start (relies heavily on parser accuracy)
>>> blob.noun_phrases
WordList(['Mai 2014', 'Dr. Meier', 'seinen 43. Geburtstag', 'Kuchen einzukaufen',
'meiner Brieftasche'])

>>> blob = TextBlob("Das Auto ist sehr schön.")
>>> blob.parse()
'Das/DT/B-NP/O Auto/NN/I-NP/O ist/VB/B-VP/O sehr/RB/B-ADJP/O schön/JJ/I-ADJP/O'
>>> from textblob_de import PatternParser
>>> blob = TextBlobDE("Das ist ein schönes Auto.", parser=PatternParser(pprint=True, lemmata=True))
>>> blob.parse()
      WORD    TAG     CHUNK    ROLE    ID      PNP     LEMMA

       Das    DT      -        -       -       -       das
       ist    VB      VP       -       -       -       sein
       ein    DT      NP       -       -       -       ein
   schönes    JJ      NP ^     -       -       -       schön
      Auto    NN      NP ^     -       -       -       auto
         .    .       -        -       -       -       .
>>> from textblob_de import PatternTagger
>>> blob = TextBlob(text, pos_tagger=PatternTagger(include_punc=True))
[('Das', 'DT'), ('Auto', 'NN'), ('ist', 'VB'), ('sehr', 'RB'), ('schön', 'JJ'), ('.', '.')]

>>> blob = TextBlob("Das Auto ist sehr schön.")
>>> blob.sentiment
Sentiment(polarity=1.0, subjectivity=0.0)
>>> blob = TextBlob("Das ist ein hässliches Auto.")
>>> blob.sentiment
Sentiment(polarity=-1.0, subjectivity=0.0)
```

Warning: **WORK IN PROGRESS:** The German polarity lexicon contains only uninflected forms and there are no subjectivity scores yet. As of version 0.2.3, lemmatized word forms are submitted to the `PatternAnalyzer`, increasing the accuracy of polarity values. New in version 0.2.7: return type of `.sentiment` is now adapted to the main TextBlob library (`:rtype: namedtuple`).

```
>>> blob.words.lemmatize()
WordList(['das', 'sein', 'ein', 'hässlich', 'Auto'])
>>> from textblob_de.lemmatizers import PatternParserLemmatizer
>>> _lemmatizer = PatternParserLemmatizer()
>>> _lemmatizer.lemmatize("Das ist ein hässliches Auto.")
[('das', 'DT'), ('sein', 'VB'), ('ein', 'DT'), ('hässlich', 'JJ'), ('Auto', 'NN')]
```

Note: Make sure that you use unicode strings on Python2 if your input contains non-ascii characters (e.g. `word = u"schön"`).

1.1.4 Access to `pattern` API in Python3

```
>>> from textblob_de.packages import pattern_de as pd
>>> print(pd.attributive("neugierig", gender=pd.FEMALE, role=pd.INDIRECT, article="die"))
neugierigen
```

Note: Alternatively, the path to `textblob_de/ext` can be added to the `PYTHONPATH`, which allows the use of `pattern.de` in almost the same way as described in its Documentation. The only difference is that you will have to prepend an underscore: `from _pattern.de import ...`. This is a precautionary measure in case the `pattern` library gets native Python3 support in the future.

1.1.5 Documentation and API Reference

- http://textblob-de.readthedocs.org/en/latest

1.1.6 Requirements

- Python >= 2.6 or >= 3.3

1.1.7 TODO

- Planned Extensions
- Additional PoS tagging options, e.g. NLTK tagging (`NLTKTagger`)
- Improve noun phrase extraction (e.g. based on `RFTagger` output)
- Improve sentiment analysis (find suitable subjectivity scores)
- Improve functionality of `Sentence()` and `Word()` objects
- Adapt more tests from the main TextBlob library (esp. for `TextBlobDE()` in `test_blob.py`)

1.1.8 License

MIT licensed. See the bundled `LICENSE` file for more details.

1.1.9 Thanks

Coded with Wing IDE 5.0 (free open source developer license)

1.2 Tutorial: Quickstart

Use the following line as your first import ...

```
from textblob_de import TextBlobDE as TextBlob
```

... and follow the quickstart guide in the documentation of the main package (using German examples and starting with "Let's create our first **TextBlob**").

1.3 Advanced Usage: Overriding Models and the Blobber Class

Follow the Advanced Usage guide in the documentation of the main package (using German examples). The following minimal replacements are necessary in order to enable the use of the German default models:

Instead of:	Use:
`textblob`	`textblob_de`
`TextBlob`	`TextBlobDE`
`Blobber`	`BlobberDE`

1.4 Extensions

Table 1.1: Planned extensions

Extension	Purpose	Status (in private repo)
`textblob-rftagger`	wrapper class for `RFTagger`	95% completed
`textblob-cmd`	command-line wrapper for `TextBlob`	50% completed
`textblob-stanfordparser`	wrapper class for `StanfordParser`	25% completed
`textblob-berkeleyparser`	wrapper class for `BerkeleyParser`	0% completed
`textblob-sent-align`	sentence alignment for parallel TextBlobs	40% completed
`textblob-converters`	various input and output conversions	20% completed

See also notes on Extensions in the documentation of the main package.

1.5 API Reference

1.5.1 Blob Classes

Wrappers for various units of text.

This includes the main `TextBlobDE`, `Word`, and `WordList` classes.

Whenever possible, classes are inherited from the main TextBlob library, but in many cases, the models for German have to be initialised here in `textblob_de.blob`, resulting in a lot of duplicate code. The main reason are the `Word` objects. If they are generated from an inherited class, they will use the English models (e.g. for `pluralize`/`singularize`) used in the main library.

Example usage:

```
>>> from textblob_de import TextBlobDE
>>> b = TextBlobDE("Einfach ist besser als kompliziert.")
>>> b.tags
[('Einfach', 'RB'), ('ist', 'VB'), ('besser', 'RB'), ('als', 'IN'), ('kompliziert', 'JJ')]
>>> b.noun_phrases
WordList([])
>>> b.words
WordList(['Einfach', 'ist', 'besser', 'als', 'kompliziert'])
```

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

BaseBlob class initialised with German default models:

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** (*str*) – A string.
- **tokenizer** – (optional) A tokenizer instance. If None, defaults to NLTKPunktTokenizer().
- **np_extractor** – (optional) An NPExtractor instance. If None, defaults to PatternParserNPExtractor().
- **pos_tagger** – (optional) A Tagger instance. If None, defaults to PatternTagger.
- **analyzer** – (optional) A sentiment analyzer. If None, defaults to PatternAnalyzer.
- **classifier** – (optional) 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: (textblob)

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

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 occurence 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 he last (right-most) occurence 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)`

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=None*, *to='de'*)
Translate the blob to another language.

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_de.blob.**BlobberDE**(*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_de import BlobberDE
>>> from textblob_de.taggers import PatternTagger
>>> from textblob.tokenizers import PatternTokenizer
>>> tb = Blobber(pos_tagger=PatternTagger(), tokenizer=PatternTokenizer())
>>> blob1 = tb("Das ist ein Blob.")
>>> blob2 = tb("Dieser Blob benutzt die selben Tagger und Tokenizer.")
>>> blob1.pos_tagger is blob2.pos_tagger
True
```

Parameters

- **text** (*str*) – A string.
- **tokenizer** – (optional) A tokenizer instance. If `None`, defaults to `NLTKPunktTokenizer()`.
- **np_extractor** – (optional) An NPExtractor instance. If `None`, defaults to `PatternParserNPExtractor()`.
- **pos_tagger** – (optional) A Tagger instance. If `None`, defaults to `PatternTagger`.
- **analyzer** – (optional) A sentiment analyzer. If `None`, defaults to `PatternAnalyzer`.
- **classifier** – (optional) A classifier.

New in version 0.4.0: (`textblob`)

class textblob_de.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: (`textblob`)

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

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 occurence 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 he last (right-most) occurence 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)`

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 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=None*, *to='de'*)
Translate the blob to another language.

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_de.blob.**TextBlobDE**(*text*, *tokenizer=None*, *pos_tagger=None*, *np_extractor=None*, *analyzer=None*, *parser=None*, *classifier=None*, *clean_html=False*)
`TextBlob` class initialised with German default models:

Parameters

- **text** (*str*) – A string.
- **tokenizer** – (optional) A tokenizer instance. If `None`, defaults to `NLTKPunktTokenizer()`.
- **np_extractor** – (optional) An NPExtractor instance. If `None`, defaults to `PatternParserNPExtractor()`.
- **pos_tagger** – (optional) A Tagger instance. If `None`, defaults to `PatternTagger`.
- **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: (`textblob`)

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

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 occurence 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 he last (right-most) occurence 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 named tuple of the form `Sentiment(polarity=0.0, subjectivity=0.0)`

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: (textblob)

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=None*, *to='de'*)
Translate the blob to another language.

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_de.blob.**Word**(*string*, *pos_tag=None*)
A simple word representation.

Includes methods for inflection, translation, and WordNet integration.

capitalize() → str
Return a capitalized version of S, i.e. make the first character have upper case and the rest lower case.

casefold() → str
Return a version of S suitable for caseless comparisons.

center(*width*[, *fillchar*]) → str
Return S centered in a 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: (textblob)

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

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: (`textblob`)

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

New in version 0.7.0: (`textblob`)

detect_language()
Detect the word's language using Google's Translate API.

New in version 0.5.0: (`textblob`)

encode(*encoding='utf-8'*, *errors='strict'*) → bytes
Encode S using the codec registered for encoding. Default encoding is 'utf-8'. 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=8*) → str
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*) → str
Return a formatted version of S, using substitutions from args and kwargs. The substitutions are identified by braces ('{' and '}').

format_map(*mapping*) → str
Return a formatted version of S, using substitutions from mapping. 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: (`textblob`)

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.

isidentifier() → bool
Return True if S is a valid identifier according to the language definition.

Use keyword.iskeyword() to test for reserved identifiers such as "def" and "class".

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.

isprintable() → bool
Return True if all characters in S are considered printable in repr() or S is empty, 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*) → str
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(*pos=None*)
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: (`textblob`)

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

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

lstrip([*chars*]) → str
Return a copy of the string S with leading whitespace removed. If chars is given and not None, remove characters in chars instead.

maketrans()
Return a translation table usable for str.translate().

If there is only one argument, it must be a dictionary mapping Unicode ordinals (integers) or characters to Unicode ordinals, strings or None. Character keys will be then converted to ordinals. If there are two

arguments, they must be strings of equal length, and in the resulting dictionary, each character in x will be mapped to the character at the same position in y. If there is a third argument, it must be a string, whose characters will be mapped to None in the result.

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*]) → str
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*]) → str
Return S right-justified in a 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=None*, *maxsplit=-1*) → 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*]) → str
Return a copy of the string S with trailing whitespace removed. If chars is given and not None, remove characters in chars instead.

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: (`textblob`)

split(*sep=None*, *maxsplit=-1*) → 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*]) → 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.

strip([*chars*]) → str
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.

swapcase() → str
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: (`textblob`)

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

translate(*from_lang=None*, *to='de'*)
Translate the word to another language using Google's Translate API.

New in version 0.5.0: (`textblob`)

upper() → str
Return a copy of S converted to uppercase.

zfill(*width*) → str
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_de.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.

clear() → None – remove all items from L

copy() → list – a shallow copy of L

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.

Currently using NLTKPunktTokenizer() for all lemmatization tasks. This might cause slightly different tokenization results compared to the TextBlob.words property.

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(*value*) → None – 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(*key=None*, *reverse=False*) → None – stable sort *IN PLACE*

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

1.5.2 Base Classes

Extensions to Abstract base classes in `textblob.base`

class `textblob_de.base.`**BaseLemmatizer**
Abstract base class from which all Lemmatizer classes inherit. Descendant classes must implement a `lemmatize(text)` method that returns a WordList of Word object with updated lemma properties.

New in version 0.2.3: (`textblob_de`)

lemmatize(*text*)
Return a list of (lemma, tag) tuples.

1.5.3 Tokenizers

Various tokenizer implementations.

class `textblob_de.tokenizers.`**NLTKPunktTokenizer**
Tokenizer included in `nltk.tokenize.punkt` package.

This is the default tokenizer in `textblob-de`

PROs:

•trained model available for German

•deals with many abbreviations and common German tokenization problems oob

CONs:

•not very flexible (model has to be re-trained on your own corpus)

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

New in version 0.6.0.

Return type generator

sent_tokenize(*text*, ***kwargs*)
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.

tokenize(*text*, *include_punc=True*, *nested=False*)
Return a list of word tokens.

Parameters

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

word_tokenize(*text*, *include_punc=True*)
The Treebank tokenizer uses regular expressions to tokenize text as in Penn Treebank.

It assumes that the text has already been segmented into sentences, e.g. using `self.sent_tokenize()`.

This tokenizer performs the following steps:

•split standard contractions, e.g. `don't` -> `do n't` and `they'll` -> `they 'll`

•treat most punctuation characters as separate tokens

•split off commas and single quotes, when followed by whitespace

•separate periods that appear at the end of line

Source: NLTK's docstring of `TreebankWordTokenizer` (accessed: 02/10/2014)

class `textblob_de.tokenizers.`**PatternTokenizer**
Tokenizer included in `pattern.de` package.

PROs:

•handling of emoticons

•flexible implementations of abbreviations

•can be adapted very easily

CONs:

•ordinal numbers cause sentence breaks

•indices of Sentence() objects cannot be computed

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

New in version 0.6.0.

Return type generator

sent_tokenize(*text*, ***kwargs*)
Returns a list of sentences.

Each sentence is a space-separated string of tokens (words). Handles common cases of abbreviations (e.g., etc., ...). Punctuation marks are split from other words. Periods (or ?!) mark the end of a sentence. Headings without an ending period are inferred by line breaks.

tokenize(*text*, *include_punc=True*, *nested=False*)
Return a list of word tokens.

Parameters

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

class textblob_de.tokenizers.**SentenceTokenizer**(*tokenizer=None*, **args*, ***kwargs*)
Generic sentence tokenization class, using tokenizer specified in TextBlobDE() instance.

Enables SentenceTokenizer().itokenize generator that would be lost otherwise.

Aim: Not to break core API of the main TextBlob library.

Parameters **tokenizer** – (optional) A tokenizer instance. If `None`, defaults to `NLTKPunktTokenizer()`.

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

New in version 0.6.0.

Return type generator

sent_tokenize(*text*, ***kwargs*)
Compatibility method to tokenizers included in `textblob-de`

tokenize(*text*, ***kwargs*)
Return a list of word tokens.

Parameters

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

class textblob_de.tokenizers.**WordTokenizer**(*tokenizer=None*, **args*, ***kwargs*)
Generic word tokenization class, using tokenizer specified in TextBlobDE() instance.

You can also submit the tokenizer as keyword argument: `WordTokenizer(tokenizer=NLTKPunktTokenizer())`

Enables WordTokenizer().itokenize generator that would be lost otherwise.

Default: NLTKPunktTokenizer().word_tokenize(text, include_punc=True)

Aim: Not to break core API of the main TextBlob library.

Parameters **tokenizer** – (optional) A tokenizer instance. If `None`, defaults to `NLTKPunktTokenizer()`.

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*, ***kwargs*)
Return a list of word tokens.

Parameters

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

word_tokenize(*text*, *include_punc=True*)
Compatibility method to tokenizers included in `textblob-de`

`textblob_de.tokenizers.`**sent_tokenize**(*text*, *tokenizer=None*)
Convenience function for tokenizing sentences (not iterable).

If tokenizer is not specified, the default tokenizer NLTKPunktTokenizer() is used (same behaviour as in the main TextBlob library).

This function returns the sentences as a generator object.

`textblob_de.tokenizers.`**word_tokenize**(*text*, *tokenizer=None*, *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.

This function returns an itertools chain object (generator).

1.5.4 POS Taggers

Default taggers for German.

```
>>> from textblob_de.taggers import PatternTagger
```

or

```
>>> from textblob_de import PatternTagger
```

class `textblob_de.taggers.`**PatternTagger**(*tokenizer=None*, *include_punc=False*)
Tagger that uses the implementation in Tom de Smedt's pattern library (http://www.clips.ua.ac.be/pattern).

Parameters

- **tokenizer** – (optional) A tokenizer instance. If `None`, defaults to `PatternTokenizer()`.
- **include_punc** – (optional) whether to include punctuation as separate tokens. Default to `False`.

tag(*sentence*, *tokenize=True*)
Tag a string *sentence*.

Parameters

- **or list sentence** (*str*) – A string or a list of sentence strings.
- **tokenize** – (optional) If `False` string has to be tokenized before (space separated string).

1.5.5 Noun Phrase Extractors

Various noun phrase extractor implementations.

`PatternParserNPExtractor()`.

class `textblob_de.np_extractors.`**`PatternParserNPExtractor`**(*tokenizer=None*)
Extract noun phrases (NP) from PatternParser() output.

Very naïve and resource hungry approach:

•get parser output

•try to correct as many obvious parser errors as you can (e.g. eliminate wrongly tagged verbs)

•filter insignificant words

Parameters **tokenizer** – (optional) A tokenizer instance. If `None`, defaults to `PatternTokenizer()`.

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

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

1.5.6 Sentiment Analyzers

German sentiment analysis implementations.

Main resource for `de-sentiment.xml`:

- German Polarity Lexicon
- See xml comment section in `de-sentiment.xml` for details

class `textblob_de.sentiments.`**`PatternAnalyzer`**(*tokenizer=None*, *lemmatizer=None*, *lemmatize=True*)
Sentiment analyzer that uses the same implementation as the pattern library. Returns results as a tuple of the form:

```
(polarity, subjectivity)
```

`RETURN_TYPE`
Return type declaration

alias of `Sentiment`

`analyze`(*text*)
Return the sentiment as a tuple of the form: `(polarity, subjectivity)`

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

`kind = 'co'`
adapted from 'textblob.en.sentiments.py'

1.5.7 Parsers

Default parsers for German.

```
>>> from textblob_de.parsers import PatternParser
```

or

```
>>> from textblob_de import PatternParser
```

class textblob_de.parsers.**PatternParser**(*tokenizer=None*, *tokenize=True*, *pprint=False*, *tags=True*, *chunks=True*, *relations=False*, *lemmata=False*, *encoding='utf-8'*, *tagset=None*)

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

Parameters

- **tokenizer** – (optional) A tokenizer instance. If `None`, defaults to `PatternTokenizer()`.
- **tokenize** – (optional) Split punctuation marks from words? (Default `True`)
- **pprint** – (optional) Use `pattern`'s `pprint` function to display parse trees (Default `False`)
- **tags** – (optional) Parse part-of-speech tags? (NN, JJ, ...) (Default `True`)
- **chunks** – (optional) Parse chunks? (NP, VP, PNP, ...) (Default `True`)
- **relations** – (optional) Parse chunk relations? (-SBJ, -OBJ, ...) (Default `False`)
- **lemmata** – (optional) Parse lemmata? (schönes => schön) (Default `False`)
- **encoding** – (optional) Input string encoding. (Default `utf-8`)
- **tagset** – (optional) Penn Treebank II (default) or ('penn'|'universal'|'stts').

parse(*text*)

Parses the text.

`pattern.de.parse(**kwargs)` can be passed to the parser instance and are documented in the main docstring of `PatternParser()`.

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

parsetree(*text*)

Returns a parsed `pattern` Text object from the given string.

1.5.8 Classifiers (from TextBlob main package)

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 at 0x0000000007317598>*, *format=None*, ***kwargs*)

Abstract classifier class from which all classifers 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 at 0x0000000007317598>*, *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.

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

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 at 0x0000000007317598>*, *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:

```
                        dotprod(weights, encode(fs,label))
prob(fs|label) = ---------------------------------------------------
                 sum(dotprod(weights, encode(fs,l)) for l in labels)
```

Where `dotprod` is the dot product:

```
dotprod(a,b) = sum(x*y for (x,y) 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 `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 at 0x0000000007317598>*, *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 at 0x0000000007317598>*, *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 `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*, *unlabeled_set*, *feature_extractor=<function contains_extractor at 0x0000000007317620>*, *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 catched 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)`.

textblob.classifiers.**contains_extractor**(*document*)
A basic document feature extractor that returns a dict of words that the document contains.

1.5.9 Blobber

class textblob_de.blob.**BlobberDE**(*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_de import BlobberDE
>>> from textblob_de.taggers import PatternTagger
>>> from textblob.tokenizers import PatternTokenizer
>>> tb = Blobber(pos_tagger=PatternTagger(), tokenizer=PatternTokenizer())
>>> blob1 = tb("Das ist ein Blob.")
>>> blob2 = tb("Dieser Blob benutzt die selben Tagger und Tokenizer.")
>>> blob1.pos_tagger is blob2.pos_tagger
True
```

Parameters

- **text** (*str*) – A string.
- **tokenizer** – (optional) A tokenizer instance. If `None`, defaults to `NLTKPunktTokenizer()`.
- **np_extractor** – (optional) An NPExtractor instance. If `None`, defaults to `PatternParserNPExtractor()`.
- **pos_tagger** – (optional) A Tagger instance. If `None`, defaults to `PatternTagger`.
- **analyzer** – (optional) A sentiment analyzer. If `None`, defaults to `PatternAnalyzer`.
- **classifier** – (optional) A classifier.

New in version 0.4.0: (`textblob`)

__call__(*text*)

Return a new TextBlob object with this Blobber's `np_extractor`, `pos_tagger`, `tokenizer`, `analyzer`, and `classifier`.

Returns A new `TextBlob`.

1.5.10 File Formats (from TextBlob main package)

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
```

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`.

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.

1.5.11 Exceptions (from TextBlob main package)

`textblob.exceptions.`**`MissingCorpusException`**
alias of `MissingCorpusError`

CHAPTER

TWO

PROJECT INFO

2.1 Changelog

2.1.1 0.4.2 (unreleased)

-

2.1.2 0.4.1 (03/10/2014)

- Removed dependency on nltk's depricated `PunktWordTokenizer` and replaced it with `TreebankWordTokenizer` see nltk/nltk#746 (comment) for details

2.1.3 0.4.0 (17/09/2014)

- Fixed Issue #7 (restore `textblob>=0.9.0` compatibility)
- Depend on `nltk3`. Vendorized `nltk` was removed in `textblob>=0.9.0`
- Fixed `ImportError` on Python2 (`unicodecsv`)

2.1.4 0.3.1 (29/08/2014)

- Improved `PatternParserNPExtractor` (less false positives in verb filter)
- Made sure that all keyword arguments with default `None` are checked with `is not None`
- Fixed shortcut to `_pattern.de` in vendorized library
- Added `Makefile` to facilitate development process
- Added docs and API reference

2.1.5 0.3.0 (14/08/2014)

- Fixed Issue #5 (text + space + period)

2.1.6 0.2.9 (14/08/2014)

- Fixed tokenization in `PatternParser` (if initialized manually, punctuation was not always separated from words)
- Improved handling of empty strings (Issue #3) and of strings containing single punctuation marks (Issue #4) in `PatternTagger` and `PatternParser`
- Added tests for empty strings and for strings containing single punctuation marks

2.1.7 0.2.8 (14/08/2014)

- Fixed Issue #3 (empty string)
- Fixed Issue #4 (space + punctuation)

2.1.8 0.2.7 (13/08/2014)

- Fixed Issue #1 lemmatization of strings containing a forward slash (`/`)
- Enhancement Issue #2 use the same rtype as `textblob` for sentiment detection.
- Fixed tokenization in `PatternParserLemmatizer`

2.1.9 0.2.6 (04/08/2014)

- Fixed `MANIFEST.in` for package data in `sdist`

2.1.10 0.2.5 (04/08/2014)

- `sdist` is non-functional as important files are missing due to a misconfiguration in `MANIFEST.in` - does not affect `wheels`
- Major internal refactoring (but no backwards-incompatible API changes) with the aim of restoring complete compatibility to original `pattern>=2.6` library on Python2
- Separation of `textblob` and `pattern` code
- On Python2 the vendorized version of `pattern.text.de` is only used, if original is not installed (same as `nltk`)
- Made `pattern.de.pprint` function and all parser keywords accessible to customise parser output
- Access to complete `pattern.text.de` API on Python2 and Python3 `from textblob_de.packages import pattern_de as pd`
- `tox` passed on all major platforms (Win/Linux/OSX)

2.1.11 0.2.3 (26/07/2014)

- Lemmatizer: `PatternParserLemmatizer()` extracts lemmata from Parser output
- Improved polarity analysis through look-up of lemmatised word forms

2.1.12 0.2.2 (22/07/2014)

- Option: Include punctuation in `tags`/`pos_tags` properties (`b = TextBlobDE(text, tagger=PatternTagger(include_punc=True))`)
- Added `BlobberDE()` class initialized with German models
- `TextBlobDE()`, `Sentence()`, `WordList()` and `Word()` classes are now all initialized with German models
- Restored complete API compatibility with `textblob.tokenizers` module of the main TextBlob library

2.1.13 0.2.1 (20/07/2014)

- Noun Phrase Extraction: `PatternParserNPExtractor()` extracts NPs from Parser output
- Refactored the way `TextBlobDE()` passes on arguments and keyword arguments to individual tools
- *Backwards-incompatible*: Deprecate `parser_show_lemmata=True` keyword in `TextBlob()`. Use `parser=PatternParser(lemmata=True)` instead.

2.1.14 0.2.0 (18/07/2014)

- vastly improved tokenization (`NLTKPunktTokenizer` and `PatternTokenizer` with tests)
- consistent use of specified tokenizer for all tools
- `TextBlobDE` with initialized default models for German
- Parsing (`PatternParser`) plus `test_parsers.py`
- **EXPERIMENTAL** implementation of Polarity detection (`PatternAnalyzer`)
- first attempt at extracting German Polarity clues into `de-sentiment.xml`
- tox tests passing for py26, py27, py33 and py34

2.1.15 0.1.3 (09/07/2014)

- First release on PyPI

2.1.16 0.1.0 - 0.1.2 (09/07/2014)

- First release on github
- A number of experimental releases for testing purposes
- Adapted version badges, tests & travis-ci config
- Code adapted from sample extension textblob-fr
- Language specific linguistic resources copied from pattern-de

2.2 Credits

2.2.1 TextBlob Development Lead

- Steven Loria <sloria1@gmail.com>

2.2.2 textblob-de Author/Maintainer

- Markus Killer <m.killer@langui.ch>

2.2.3 Contributors

- Hocdoc (Issues #1 - #5)
- ups1974 (Issue #7)

2.3 LICENSE

Human readable generic MIT License

2.4 Contributing guidelines

2.4.1 In General

- PEP 8, when sensible.
- Test ruthlessly. Write docs for new features.
- Even more important than Test-Driven Development–*Human-Driven Development*.

2.4.2 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-de on Github.

   ```
   $ git clone https://github.com/markuskiller/textblob-de.git
   $ cd textblob-de
   ```

2. (recommended) Create and activate virtual python environment.

   ```
   $ pip install -U virtualenv
   $ virtualenv tb-de
   $ <activate virtual environment>
   ```

3. Install development requirements and run `setupy.py develop`. (see Makefile help for overview of available `make` targets):

   ```
   $ make develop
   ```

2.5 `make` command

This project adopts the `Makefile` approach, proposed by Jeff Knupp in his blog post Open Sourcing a Python Project the Right Way .

On Linux/OSX the `make` command should work out-of-the-box:

```
$ make help
```

Shows all available tasks.

2.5.1 Using `make` on Windows

The two `Makefile` s in this project should work on all three major platforms. On Windows, `make.exe` included in the MinGW/msys distribution has been successfully tested. Once `msys` is installed on a Windows system, the `path/to/msys/1.0/bin` needs to be added to the `PATH` environment variable.

A good place to update the `PATH` variable are the `Activate.ps1` or `activate.bat` scripts of a virtual python build environment, created using `virtualenv` (`pip install virtualenv`) or `pyvenv` (added to Python3.3's standard library).

Windows PowerShell

Add the following line at the end of `path\to\virtual\python\env\Scripts\Activate.ps1`:

```
# Add msys binaries to PATH
$env:PATH = "path\to\MinGW\msys\1.0\bin;$env:PATH"
```

Windows `cmd.exe`

Add the following line at the end of `path\to\virtual\python\env\Scripts\activate.bat`:

```
# Add msys binaries to PATH
set "PATH=path\to\MinGW\msys\1.0\bin;%PATH%"
```

Now the `make` command should work as documented in `$ make help`.

2.6 Project `Makefile`

```
generated: 03 October 2014 - 19:28


   Please use 'make <target>' where where <target> is one of

   SETUP & CLEAN
   -------------

   install            run 'python setup.py install'
   uninstall          run 'pip uninstall <package>'
   develop            install links to source files in current Python environment
   reset-dev          uninstall all links and console scripts and make clean
   clean              remove all artifacts
   clean-build        remove build artifacts
   clean-docs         remove documentation build artifacts
   clean-pyc          remove Python file artifacts (except in 'ext')
   clean-test         remove test artifacts (e.g. 'htmlcov')
   clean-logs         remove log artifacts and place empty file in 'log_dir'

   TESTING
   -------

   autopep8           automatically correct 'pep8' violations
   lint               check style with 'flake8'
   test               run tests quickly with the default Python
   test-all           run tests on every Python version with tox
   coverage           check code coverage quickly with the default Python

   PUBLISHING
   ----------

   docs               generate Sphinx HTML documentation, including API docs
   docs-pdf           generate Sphinx HTML and PDF documentation, including API docs
   sdist              package
   publish            package and upload sdist and universal wheel to PyPI
   register           update README.rst on PyPI
   push-github        push all changes to git repository on github.com
   push-bitbucket     push all changes to git repository on bitbucket.org
                           --> include commit message as M='your message'

   VARIABLES ACCESSIBLE FROM COMMAND-LINE
   --------------------------------------

   M='your message'   mandatory git commit message
   N='package name'   specify python package name (optional)
   O='open|xdg-open|start'
```

```
                          --> specify platform specific 'open' cmd (optional)
   P='path/to/python' specify python executable (optional)
```

2.7 Documentation `Makefile`

```
generated: 03 October 2014 - 19:28

   Please use `make <target>' where <target> is one of

   html       to make standalone HTML files
   dirhtml    to make HTML files named index.html in directories
   singlehtml to make a single large HTML file
   pickle     to make pickle files
   json       to make JSON files
   htmlhelp   to make HTML files and a HTML help project
   qthelp     to make HTML files and a qthelp project
   devhelp    to make HTML files and a Devhelp project
   epub       to make an epub
   latex      to make LaTeX files, you can set PAPER=a4 or PAPER=letter
   latexpdf   to make LaTeX files and run them through pdflatex
   latexpdfja to make LaTeX files and run them through platex/dvipdfmx
   text       to make text files
   man        to make manual pages
   texinfo    to make Texinfo files
   info       to make Texinfo files and run them through makeinfo
   gettext    to make PO message catalogs
   changes    to make an overview of all changed/added/deprecated items
   xml        to make Docutils-native XML files
   pseudoxml  to make pseudoxml-XML files for display purposes
   linkcheck  to check all external links for integrity
   doctest    to run all doctests embedded in the documentation (if enabled)
```

t

Symbols

A

B

C

D

M

N

P

R

S

T

U

W

Z