Wayward

Release 0.3.2

Jun 09, 2019

Contents

Wayward is a Python package that helps to identify characteristic terms from single documents or groups of documents. It can be used for keyword extraction and several related tasks, and can create efficient sparse representations for classifiers. It was originally created to provide term weights for word clouds.

Rather than use simple term frequency to estimate the importance of words and phrases, it weighs terms by statistical models known as *parsimonious language models*. These models are good at picking up the terms that distinguish a text document from other documents in a collection.

For this to work, a preferably large amount of documents is needed to serve as a background collection, to compare the documents of interest to. This could be a random sample of newspaper articles, for instance, but for many applications it works better to take a natural collection, such as a periodical publication, and to fit the model for separate parts (e.g. individual issues, or yearly groups of issues).

See the *References* section for more information about parsimonious language models and their applications.

Wayward does not do visualization of word clouds. For that, you can paste its output into a tool like http://wordle.net or the IBM Word-Cloud Generator.

CHAPTER 1

Installation

Either install the latest release from PyPI:

```
$ pip install wayward
```

or clone the git repository, and use Poetry to install the package in editable mode:

```
$ git clone https://github.com/aolieman/wayward.git
$ cd wayward/
$ poetry install
```

CHAPTER 2

Usage

```
>>> quotes = [
...     "Love all, trust a few, Do wrong to none",
...     ...
...     "A lover's eyes will gaze an eagle blind. "
...     "A lover's ear will hear the lowest sound.",
... ]
>>> doc_tokens = [
...     re.sub(r"[.,:;!?\"‘’]|'s\b", " ", quote).lower().split()
...     for quote in quotes
... ]
```

The ParsimoniousLM is initialized with all document tokens as a background corpus, and subsequently takes a single document's tokens as input. Its top() method returns the top terms and their probabilities:

```
>>> from wayward import ParsimoniousLM
>>> plm = ParsimoniousLM(doc_tokens, w=.1)
>>> plm.top(10, doc_tokens[-1])
[('lover', 0.1538461408077277),
 ('will', 0.1538461408077277),
 ('eyes', 0.0769230704038643),
 ('gaze', 0.0769230704038643),
 ('an', 0.0769230704038643),
 ('eagle', 0.0769230704038643),
 ('blind', 0.0769230704038643),
 ('ear', 0.0769230704038643),
 ('hear', 0.0769230704038643),
 ('lowest', 0.0769230704038643)]
```

The SignificantWordsLM is similarly initialized with a background corpus, but subsequently takes a group of document tokens as input. Its group_top method returns the top terms and their probabilities:

```
>>> from wayward import SignificantWordsLM
>>> swlm = SignificantWordsLM(doc_tokens, lambdas=(.7, .1, .2))
>>> swlm.group_top(10, doc_tokens[-2:], fix_lambdas=True)
```

(continues on next page)

(continued from previous page)

```
[('much', 0.09077675276900632),
 ('lover', 0.06298706244865138),
 ('will', 0.06298706244865138),
 ('you', 0.04538837638450315),
 ('your', 0.04538837638450315),
 ('rhymes', 0.04538837638450315),
 ('speak', 0.04538837638450315),
 ('neither', 0.04538837638450315),
 ('rhyme', 0.04538837638450315),
 ('nor', 0.04538837638450315)]
```

See `example/dickens.py` for a runnable example with more realistic data.

CHAPTER 3

Origin and Relaunch

This package started out as WeighWords, written by Lars Buitinck at the University of Amsterdam. It provides an efficient parsimonious LM implementation, and a very accessible API.

A recent innovation in language modeling, Significant Words Language Models, led to the addition of a two-way parsimonious language model to this package. This new version targets python 3.x, and after a long slumber deserved a fresh name. The name "Wayward" was chosen because it is a near-homophone of WeighWords, and as a nod to parsimonious language modeling: it uncovers which terms "depart" most from the background collection. The parsimonization algorithm discounts terms that are already well explained by the background model, until the most wayward terms come out on top.

See the Changelog for an overview of the most important changes.

CHAPTER 4

References

D. Hiemstra, S. Robertson, and H. Zaragoza (2004). Parsimonious Language Models for Information Retrieval. Proc. SIGIR‘04.

R. Kaptein, D. Hiemstra, and J. Kamps (2010). How different are Language Models and word clouds?. Proc. ECIR‘10.

M. Dehghani, H. Azarbonyad, J. Kamps, D. Hiemstra, and M. Marx (2016). Luhn Revisited: Significant Words Language Models. Proc. CKIM‘16.

CHAPTER 5

Contents

Wayward is a Python package that helps to identify characteristic terms from single documents or groups of documents. It can be used for keyword extraction and several related tasks, and can create efficient sparse representations for classifiers. It was originally created to provide term weights for word clouds.

Rather than use simple term frequency to estimate the importance of words and phrases, it weighs terms by statistical models known as *parsimonious language models*. These models are good at picking up the terms that distinguish a text document from other documents in a collection.

For this to work, a preferably large amount of documents is needed to serve as a background collection, to compare the documents of interest to. This could be a random sample of newspaper articles, for instance, but for many applications it works better to take a natural collection, such as a periodical publication, and to fit the model for separate parts (e.g. individual issues, or yearly groups of issues).

See the *References* section for more information about parsimonious language models and their applications.

Wayward does not do visualization of word clouds. For that, you can paste its output into a tool like http://wordle.net or the IBM Word-Cloud Generator.

5.1 Installation

Either install the latest release from PyPI:

```
$ pip install wayward
```

or clone the git repository, and use Poetry to install the package in editable mode:

```
$ git clone https://github.com/aolieman/wayward.git
$ cd wayward/
$ poetry install
```

5.2 Usage

```
>>> quotes = [
...     "Love all, trust a few, Do wrong to none",
...     ...
...     "A lover's eyes will gaze an eagle blind. "
...     "A lover's ear will hear the lowest sound.",
... ]
>>> doc_tokens = [
...     re.sub(r"[.,:;!?\"‘’]|'s\b", " ", quote).lower().split()
...     for quote in quotes
... ]
```

The `ParsimoniousLM` is initialized with all document tokens as a background corpus, and subsequently takes a single document’s tokens as input. Its `top()` method returns the top terms and their probabilities:

```
>>> from wayward import ParsimoniousLM
>>> plm = ParsimoniousLM(doc_tokens, w=.1)
>>> plm.top(10, doc_tokens[-1])
[('lover', 0.1538461408077277),
 ('will', 0.1538461408077277),
 ('eyes', 0.0769230704038643),
 ('gaze', 0.0769230704038643),
 ('an', 0.0769230704038643),
 ('eagle', 0.0769230704038643),
 ('blind', 0.0769230704038643),
 ('ear', 0.0769230704038643),
 ('hear', 0.0769230704038643),
 ('lowest', 0.0769230704038643)]
```

The `SignificantWordsLM` is similarly initialized with a background corpus, but subsequently takes a group of document tokens as input. Its `group_top` method returns the top terms and their probabilities:

```
>>> from wayward import SignificantWordsLM
>>> swlm = SignificantWordsLM(doc_tokens, lambdas=(.7, .1, .2))
>>> swlm.group_top(10, doc_tokens[-2:], fix_lambdas=True)
[('much', 0.09077675276900632),
 ('lover', 0.06298706244865138),
 ('will', 0.06298706244865138),
 ('you', 0.04538837638450315),
 ('your', 0.04538837638450315),
 ('rhymes', 0.04538837638450315),
 ('speak', 0.04538837638450315),
 ('neither', 0.04538837638450315),
 ('rhyme', 0.04538837638450315),
 ('nor', 0.04538837638450315)]
```

See `example/dickens.py` for a runnable example with more realistic data.

5.3 Origin and Relaunch

This package started out as WeighWords, written by Lars Buitinck at the University of Amsterdam. It provides an efficient parsimonious LM implementation, and a very accessible API.

A recent innovation in language modeling, Significant Words Language Models, led to the addition of a two-way parsimonious language model to this package. This new version targets python 3.x, and after a long slumber deserved a fresh name. The name “Wayward” was chosen because it is a near-homophone of WeighWords, and as a nod to parsimonious language modeling: it uncovers which terms “depart” most from the background collection. The parsimonization algorithm discounts terms that are already well explained by the background model, until the most wayward terms come out on top.

See the Changelog for an overview of the most important changes.

5.4 References

D. Hiemstra, S. Robertson, and H. Zaragoza (2004). Parsimonious Language Models for Information Retrieval. Proc. SIGIR‘04.

R. Kaptein, D. Hiemstra, and J. Kamps (2010). How different are Language Models and word clouds?. Proc. ECIR‘10.

M. Dehghani, H. Azarbonyad, J. Kamps, D. Hiemstra, and M. Marx (2016). Luhn Revisited: Significant Words Language Models. Proc. CKIM‘16.

5.5 Changelog

All notable changes to this project should be documented in this file.

The format is based on Keep a Changelog, and this project adheres to Semantic Versioning.

5.5.1 [Unreleased]

-

5.5.2 [0.3.2] - 2019-06-09

Added

- Package documentation:
  - Transclude basic instructions from README.
  - Generate API documentation.
  - Configuration for Read the Docs.
  - Incorporate changelog via symlink.
  - Add a Dickens example page.
- Docs build status and PyPI version badges in README.

5.5.3 [0.3.1] - 2019-06-05

Added

- This changelog.

Changed

- Explicitly specified the readme in `pyproject.toml`.
- Updated install instructions for Poetry.

5.5.4 [0.3.0] - 2019-06-04

Added

- Significant Words Language Model.
- Pluggable specific terms estimator.
- Tests for PLM document model.
- Tests for SWLM model fit.
- Tests for model (non-)equivalence between PLM and SWLM.
- SWLM example in `exmaple/dickens.py`.
- Usage examples in README.
- Type hints in function annotations.

Changed

- Renamed package to Wayward.
- Replaced `setup.py` with `pyproject.toml`.
- `ParsimoniousLM.top()` now returns linear probabilities instead of log-probabilities.

Removed

- Dropped python 2.7 compatibility in favor of ^3.7.

Fixed

- `KeyError` when out-of-vocabulary terms occurred in a document.

5.5.5 [0.2.x] - 2011-11-13 to 2013-04-18

The WeighWords version from which Wayward was forked.

Some commits have been put on the master branch after bumping the version to 0.2. Since there is no git tag to pin down what's part of 0.2, I've mentioned both the version bump date, and the date of the latest commit that we use here.

5.6 Indices and tables

- genindex
- modindex
- search

5.6.1 Dickens Example

In this example, three books by Charles Dickens are used as a background corpus. Each of the books is subsequently used as a foreground model, and is parsimonized against the background corpus. This results in top terms that are characteristic for specific books, when compared to common Dickensian language.

This is a minimalistic example, which only analyzes unigrams, and uses a background corpus of limited size. As an exercise, one could expand this example with phrase modeling (e.g. as provided by gensim.phrases) to analyze higher-order ngrams.

The full text of the input books was obtained from Project Gutenberg.

Running

Download (or clone) the source files from GitHub.

```
$ cd wayward/example
$ python dickens.
```

Output

```
INFO:__main__:Fetching terms from Oliver Twist
INFO:__main__:Fetching terms from David Copperfield
INFO:__main__:Fetching terms from Great Expectations
INFO:wayward.parsimonious:Building corpus model
INFO:wayward.parsimonious:Building corpus model
INFO:wayward.parsimonious:Gathering term probabilities
INFO:wayward.parsimonious:EM with max_iter=50, eps=1e-05

... *omitted numpy warnings*

INFO:wayward.significant_words:Lambdas initialized to: Corpus=0.9, Group=0.01,␣
 ↪Specific=0.09

Top 20 words in Oliver Twist:

PLM term         PLM p        SWLM term        SWLM p
oliver           0.0824       oliver           0.1361
bumble           0.0372       sikes            0.0526
sikes            0.0332       bumble           0.0520
jew              0.0297       fagin            0.0477
fagin            0.0289       jew              0.0475
brownlow         0.0163       replied          0.0372
monks            0.0126       brownlow         0.0244
noah             0.0124       rose             0.0235
rose             0.0116       gentleman        0.0223
```

(continues on next page)

(continued from previous page)

```
giles           0.0112      girl            0.0178
nancy           0.0109      nancy           0.0164
dodger          0.0107      dodger          0.0161
maylie          0.0093      monks           0.0159
bates           0.0088      noah            0.0156
beadle          0.0081      bates           0.0133
sowerberry      0.0079      giles           0.0118
yer             0.0077      maylie          0.0117
grimwig         0.0062      bill            0.0115
charley         0.0062      rejoined        0.0113
corney          0.0061      lady            0.0110

INFO:wayward.parsimonious:Gathering term probabilities
INFO:wayward.parsimonious:EM with max_iter=50, eps=1e-05

... *omitted wayward logging output*

INFO:wayward.significant_words:Lambdas initialized to: Corpus=0.9, Group=0.01,␣
↪Specific=0.09

Top 20 words in David Copperfield:

PLM term        PLM p       SWLM term       SWLM p
micawber        0.0367      micawber        0.0584
peggotty        0.0335      peggotty        0.0533
aunt            0.0330      aunt            0.0517
copperfield     0.0226      copperfield     0.0359
traddles        0.0218      traddles        0.0346
dora            0.0216      my              0.0295
agnes           0.0182      dora            0.0290
steerforth      0.0169      agnes           0.0285
murdstone       0.0138      steerforth      0.0259
uriah           0.0100      murdstone       0.0200
ly              0.0088      her             0.0171
dick            0.0085      mother          0.0157
wickfield       0.0084      uriah           0.0145
davy            0.0073      dick            0.0142
barkis          0.0067      ly              0.0140
trotwood        0.0065      wickfield       0.0128
spenlow         0.0064      davy            0.0105
ham             0.0057      trotwood        0.0099
heep            0.0055      barkis          0.0097
creakle         0.0054      ham             0.0094

INFO:wayward.parsimonious:Gathering term probabilities
INFO:wayward.parsimonious:EM with max_iter=50, eps=1e-05

... *omitted wayward logging output*

INFO:wayward.significant_words:Lambdas initialized to: Corpus=0.9, Group=0.01,␣
↪Specific=0.09

Top 20 words in Great Expectations:

PLM term        PLM p       SWLM term       SWLM p
joe             0.0732      joe             0.1346
pip             0.0335      pip             0.0614
```

(continues on next page)

(continued from previous page)

```
havisham         0.0314       havisham         0.0559
herbert          0.0309       herbert          0.0502
wemmick          0.0280       estella          0.0471
estella          0.0265       wemmick          0.0456
jaggers          0.0239       jaggers          0.0409
biddy            0.0227       biddy            0.0404
pumblechook      0.0161       pumblechook      0.0275
wopsle           0.0118       wopsle           0.0192
drummle          0.0087       pocket           0.0186
provis           0.0067       sister           0.0152
orlick           0.0058       drummle          0.0132
compeyson        0.0057       aged             0.0097
aged             0.0056       marshes          0.0092
marshes          0.0052       orlick           0.0088
handel           0.0051       forge            0.0088
forge            0.0050       handel           0.0082
guardian         0.0047       provis           0.0074
trabb            0.0045       convict          0.0068
```

5.6.2 parsimonious module

class wayward.parsimonious.**ParsimoniousLM**(*documents: Iterable[Iterable[str]], w: numpy.floating, thresh: int = 0*)

Bases: `object`

Language model for a set of documents.

Constructing an object of this class fits a background model. The top method can then be used to fit document-specific models, also for unseen documents (with the same vocabulary as the background corpus).

References

D. Hiemstra, S. Robertson, and H. Zaragoza (2004). Parsimonious Language Models for Information Retrieval. Proc. SIGIR‘04.

Parameters

- **documents** (*iterable over iterable of str terms*) – All documents that should be included in the corpus model.
- **w** (*float*) – Weight of document model (1 - weight of corpus model).
- **thresh** (*int*) – Don’t include words that occur fewer than *thresh* times.

vocab

Mapping of terms to numeric indices

Type dict of term -> int

p_corpus

Log probability of terms in background model (indexed by *vocab*)

Type array of float

p_document

Log probability of terms in the last processed document model (indexed by *vocab*)

Type array of float

get_term_probabilities(*log_prob_distribution: numpy.ndarray*) → Dict[str, float]

Align a term distribution with the vocabulary, and transform the term log probabilities to linear probabilities.

Parameters **log_prob_distribution** (*array of float*) – Log probability of terms which is indexed by the vocabulary.

Returns **t_p_map** – Dictionary of terms and their probabilities in the (sub-)model.

Return type dict of term -> float

top(*k: int, d: Iterable[str], max_iter: int = 50, eps: float = 1e-05, w: Optional[numpy.floating] = None*) → List[Tuple[str, float]]

Get the top *k* terms of a document *d* and their log probabilities.

Uses the Expectation Maximization (EM) algorithm to estimate term probabilities.

Parameters

- **k** (*int*) – Number of top terms to return.
- **d** (*iterable of str terms*) – Terms that make up the document.
- **max_iter** (*int, optional*) – Maximum number of iterations of EM algorithm to run.
- **eps** (*float, optional*) – Epsilon: convergence threshold for EM algorithm.
- **w** (*float, optional*) – Weight of document model; overrides value given to *ParsimoniousLM*

Returns **t_p** – Terms and their probabilities in the parsimonious model.

Return type list of (str, float)

5.6.3 significant_words module

class wayward.significant_words.**SignificantWordsLM**(*documents: Iterable[Iterable[str]], lambdas: Tuple[numpy.floating, numpy.floating, numpy.floating], thresh: int = 0*)

Bases: *wayward.parsimonious.ParsimoniousLM*

Language model that consists of three sub-models:

- Corpus model: represents term probabilities in a (large) background collection;
- Group model: parsimonious term probabilities in a group of documents;
- Specific model: represents the same group, but is biased towards terms that occur with a high frequency in single docs, and a low frequency in others.

References

M. Dehghani, H. Azarbonyad, J. Kamps, D. Hiemstra, and M. Marx (2016). Luhn Revisited: Significant Words Language Models. Proc. CKIM'16.

Parameters

- **documents** (*iterable over iterable of str terms*) – All documents that should be included in the corpus model.

- **lambdas** (*3-tuple of float*) – Weight of corpus, group, and specific models. Will be normalized if the weights in the tuple don't sum to one.
- **thresh** (*int*) – Don't include words that occur fewer than *thresh* times.

vocab
Mapping of terms to numeric indices

Type dict of term -> int

p_corpus
Log probability of terms in background model (indexed by *vocab*)

Type array of float

p_group
Log probability of terms in the last processed group model (indexed by *vocab*)

Type array of float

p_specific
Log probability of terms in the last processed specific model (indexed by *vocab*)

Type array of float

lambda_corpus
Log probability (weight) of corpus model for documents

Type array of float

lambda_group
Log probability (weight) of group model for documents

Type array of float

lambda_specific
Log probability (weight) of specific model for documents

Type array of float

See also:

wayward.parsimonious.ParsimoniousLM one-sided parsimonious model

fit_parsimonious_group(*document_group: Iterable[Iterable[str]], max_iter: int = 50, eps: float = 1e-05, lambdas: Optional[Tuple[numpy.floating, numpy.floating, numpy.floating]] = None, fix_lambdas: bool = False, parsimonize_specific: bool = False, post_parsimonize: bool = False, specific_estimator: Callable[[Sequence[numpy.ndarray]], numpy.ndarray] = <function mutual_exclusion>*) → Dict[str, float]
Estimate a document group model, and parsimonize it against fixed corpus and specific models. The documents may be unseen, but any terms that are not in the vocabulary will be ignored.

Parameters

- **document_group** (*iterable over iterable of str terms*) – All documents that should be included in the group model.
- **max_iter** (*int, optional*) – Maximum number of iterations of EM algorithm to run.
- **eps** (*float, optional*) – Epsilon: convergence threshold for EM algorithm.
- **lambdas** (*3-tuple of float, optional*) – Weight of corpus, group, and specific models. Will be normalized if the weights in the tuple don't sum to one.

- **fix_lambdas** (*bool, optional*) – Fix the weights of the three sub-models (i.e. don't estimate lambdas as part of the M-step).
- **parsimonize_specific** (*bool, optional*) – Bias the specific model towards uncommon terms before applying the EM algorithm to the group model. This generally results in a group model that stands out less from the corpus model.
- **post_parsimonize** (*bool, optional*) – Bias the group model towards uncommon terms after applying the EM algorithm. This may be used to compensate when the frequency of common terms varies much between the documents in the group.
- **specific_estimator** (*callable, optional*) – Function that estimates the specific terms model based on the document term frequencies of the doc group.

Returns **t_p_map** – Dictionary of terms and their probabilities in the group model.

Return type dict of term -> float

group_top (*k: int, document_group: Iterable[Iterable[str]], **kwargs*) → List[Tuple[str, float]]

Get the top *k* terms of a *document_group* and their probabilities. This is a shortcut to retrieve the top terms found by *fit_parsimonious_group()*.

Parameters

- **k** (*int*) – Number of top terms to return.
- **document_group** (*iterable over iterable of str terms*) – All documents that should be included in the group model.
- **kwargs** – Optional keyword arguments for *fit_parsimonious_group()*.

Returns **t_p** – Terms and their probabilities in the group model.

Return type list of (str, float)

See also:

SignificantWordsLM.fit_parsimonious_group()

static normalize_lambdas (*lambdas: Tuple[numpy.floating, numpy.floating, numpy.floating]*) → Tuple[numpy.floating, numpy.floating, numpy.floating]

Check and normalize the initial lambdas of the three sub-models.

Parameters **lambdas** (*3-tuple of float*) – Weight of corpus, group, and specific models.

Returns **lambdas** – Normalized probability of corpus, group, and specific models.

Return type 3-tuple of float

5.6.4 specific_term_estimators module

exception wayward.specific_term_estimators.**RequiresMultipleDocuments**

Bases: Exception

wayward.specific_term_estimators.**idf_fallback_for_many_docs** (*document_term_frequencies: Sequence[numpy.ndarray], primary_estimator: Callable[[Sequence[numpy.ndarray]], numpy.ndarray], fallback_thresh: int*) `wayward.specific_term_estimators.`**`inverse_doc_frequency`**(*document_term_frequencies: Sequence[numpy.ndarray]*) → numpy.ndarray

Estimate the fixed specific model with the inverse doc frequency method.

`wayward.specific_term_estimators.`**`me_up_to_40_docs`**(*document_term_frequencies: Sequence[np.ndarray], *, primary_estimator: SpecificTermEstimator = <function mutual_exclusion>, fallback_thresh: int = 40*)

`wayward.specific_term_estimators.`**`mutual_exclusion`**(*document_term_frequencies: Sequence[numpy.ndarray]*) → numpy.ndarray

Estimate the fixed specific model with the mutual exclusion method.

`wayward.specific_term_estimators.`**`requires_multiple_docs`**(*estimator_func: Callable[[Sequence[numpy.ndarray]], numpy.ndarray]*)

Do not let the decorated function be called with fewer than two docs.

Parameters **`estimator_func`** (*`SpecificTermEstimator`*) –

Raises *`RequiresMultipleDocuments`*

Returns **decorated_func**

Return type SpecificTermEstimator

5.6.5 logsum module

Safe addition in log-space, taken from scikit-learn.

Authors: G. Varoquaux, A. Gramfort, A. Passos, O. Grisel

License: BSD

`wayward.logsum.`**`logsum`**(*x: numpy.ndarray*) → numpy.ndarray

Computes the sum of x assuming x is in the log domain.

Returns `log(sum(exp(x)))` while minimizing the possibility of over/underflow.

Examples

```
>>> import numpy as np
>>> a = np.arange(10)
>>> np.log(np.sum(np.exp(a)))
9.4586297444267107
>>> logsum(a)
9.4586297444267107
```

Python Module Index

Index