disamby Documentation

Release 0.2.3

Luca Verginer

Jul 01, 2017

Contents

- Free software: MIT license
- Documentation: https://disamby.readthedocs.io.

`disamby` is a python package designed to carry out entity disambiguation based on fuzzy string matching.

It works best for entities which if the same have very similar strings. Examples of situation where this disambiguation algorithm works fairly well is with company names and addresses which have typos, alternative spellings or composite names. Other use-cases include identifying people in a database where the name might be misspelled.

The algorithm works by exploiting how informative a given word/token is, based on the observed frequencies in the whole corpus of strings. For example the word 'inc' in the case of firm names is not very informative, however "Solomon" is, since the former appears repeatedly whereas the second rarely.

With these frequencies the algorithms computes for a given pair of instances how similar they are, and if they are above an arbitrary threshold they are connected in an "alias graph" (i.e. a directed network where an entity is connected to an other if it is similar enough). After all records have been connected in this way disamby returns sets of entities, which are strongly connected[2] . Strongly connected means in this case that there exists a path from all nodes to all nodes within the component.

[2] https://en.wikipedia.org/wiki/Strongly_connected_component

CHAPTER 1

Example

To use disamby in a project:

```
import pandas as pd
import disamby.preprocessors as pre
form disamby import Disamby

# create a dataframe with the fields you intend to match on as columns
df = pd.DataFrame({
    'name':     ['Luca Georger',        'Luca Geroger',         'Adrian Sulzer'],
    'address':  ['Mira, 34, Augsburg',  'Miri, 34, Augsburg',   'Milano, 34']},
    index=      ['L1',                  'L2',                   'O1']
)

# define the pipeline to process the strings, note that the last step must return
# a tuple of strings
pipeline = [
    pre.normalize_whitespace,
    pre.remove_punctuation,
    lambda x: pre.trigram(x) + pre.split_words(x)  # any python function is allowed
]

# instantiate the disamby object, it applies the given pre-processing pipeline and
# computes their frequency.
dis = Disamby(df, pipeline)

# let disamby compute disambiguated sets. Node that a threshold must be given or it
# defaults to 0.
dis.disambiguated_sets(threshold=0.5)
[{'L2', 'L1'}, {'O1'}]  # output

# To check if the sets are accurate you can get the rows from the
# pandas dataframe like so:
df.loc[['L2', 'L1']]
```

CHAPTER 2

Installation

To install disamby, run this command in your terminal:

```
$ pip install disamby
```

This is the preferred method to install disamby, as it will always install the most recent stable release. If you don't have pip installed, this Python installation guide can guide you through the process.

You can also install it from source as follows The sources for disamby can be downloaded from the Github repo. You can either clone the public repository:

```
$ git clone git://github.com/verginer/disamby
```

Or download the tarball:

```
$ curl  -OL https://github.com/verginer/disamby/tarball/master
```

Once you have a copy of the source, you can install it with:

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

CHAPTER 3

Credits

I got the inspiration for this package from the seminar "The SearchEngine - A Tool for Matching by Fuzzy Criteria" by Thorsten Doherr at the CISS[1] Summer School 2017

3.1 Indices and tables

3.1.1 disamby

disamby package

Submodules

disamby.cli module

Console script for disamby.

With the cli it is possible to carry out the disambiguation on the command line. The script returns a json file with a name taken from the first field as the key and a list of indices belonging to the same disambiguated cluster.

```
{'International Business Machines Inc': [1, 2, 5, 34, 90],
'Samsung': [4, 123],
'...': [...],
 ...}
```

To carry out the disambiguation you will need a csv file with proper headings and optionally an index column. If not index column is specified then the position in the csv is used in the json file to identify its members.

Only a basic processing pipeline is possible here, see the *–help* of the script to know which are available.

Example usage:

[1] http://www.euro-ciss.eu/ciss/home.html

```
$ disamby --col-headers name,address \
            --index inv_id \
            --threshold .7 \
            --prep APNX \
            input.csv output.json
```

disamby.core module

Main module.

class disamby.core.**Disamby**(*data: typing.Union[pandas.core.frame.DataFrame, pandas.core.series.Series] = None, preprocessors: list = None, field: str = None*)

Bases: object

Class for disambiguation fitting, scoring and ranking of potential matches

A *Disamby* instance stores the pre-processing pipeline applied to the strings for a given field as well as the the computed frequencies from the entire corpus of strings to match against. *Disamby* can be instantiated either with not arguments, with a list of strings, pandas.Series or pandas.DataFrame. This triggers the immediate call to the *fit* method, whose doc explains the parameters.

Examples

```
>>> import pandas as pd
>>> import disamby.preprocessors as pre
>>> df = pd.DataFrame(
... {'a': ['Luca Georger', 'Luca Geroger', 'Adrian Sulzer'],
... 'b': ['Mira, 34, Augsburg', 'Miri, 34, Augsburg', 'Milano, 34']
... }, index=['L1', 'L2', 'O1']
... )
>>> pipeline = [
...     pre.normalize_whitespace,
...     pre.remove_punctuation,
...     pre.trigram
... ]
>>> dis = Disamby(df, pipeline)
>>> dis.disambiguated_sets(threshold=0.5, verbose=False)
[{'L2', 'L1'}, {'O1'}]
```

alias_graph(*threshold=0.7, verbose=True, weights=None, **kwargs*) → networkx.classes.digraph.DiGraph

This function creates the directed network connecting an instance to an other through a directed edge if the the target instance has a similarity score above the threshold.

Parameters

- **weights** –
- **threshold** (*float*) – between 0 and 1
- **verbose** (*whether to show the progressbar*) –
- **kwargs** – arguments to pass to the score function (i.e. offset, smoother)

Returns

Return type DiGraph

disambiguated_sets (*threshold=0.7*, *verbose=True*, *weights=None*, ***kwargs*)

fields

find (*idx*, *threshold=0.0*, *weights: dict = None*, ***kwargs*) → list
returns the list of scored instances which have a score above the threshold. Note that strings which do not share any token are omitted since their score is 0 by default.

Parameters

- **idx** – index of the record to find
- **threshold** –
- **weights** (*dict*) –

fit (*data: typing.Union[pandas.core.frame.DataFrame, pandas.core.series.Series]*, *preprocessors: list*, *field: str = None*)
Computes the frequencies of the terms by field.

Parameters

- **data** (*pandas.DataFrame, pandas.Series or list of strings*) – list of strings or pandas.DataFrame if dataframe is given then the field defaults to the column name
- **preprocessors** (*list*) – list of functions to apply in that order note the first function must accept a string, the other functions must be such that a pipeline is possible the result is a tuple of strings.
- **field** (*str*) – string identifying which field this data belongs to

Examples

```
>>> import pandas as pd
>>> from disamby.preprocessors import split_words
>>> df = pd.DataFrame(
... {'a': ['Luca Georger', 'Luke Geroge', 'Adrian Sulzer'],
... 'b': ['Mira, 34, Augsburg', 'Miri, 32', 'Milano, 34']
... })
>>> dis = Disamby()
>>> prep = [split_words]
>>> dis.fit(df, prep)
```

id_potential (*term: typing.Union[tuple, str]*, *field: str*, *smoother: str = None*, *offset=0*) → dict
Computes the weights of the words based on the observed frequency and normalized.

Parameters

- **term** (*str, tuple*) – term to look for or a tuple of proper tokens
- **field** (*str*) – field the word falls under
- **smoother** (*str (optional)*) – one of {None, 'offset', 'log'}
- **offset** (*int*) – offset to add to count only needed for smoothers 'log' and 'offset'

Returns

Return type float

static pre_process (*base_name*, *functions: list*)
apply every function consecutively to base_name

score (*term: str*, *other_term: str*, *field: str*, *smoother=None*, *offset=0*) → float
Computes the score between the two strings using the frequency data

Parameters

- **term** (*str*) – term to search for
- **other_term** (*str*) – the other term to compare too
- **field** (*str*) – the name of the column to which this term belongs
- **smoother** (*str (optional)*) – one of {None, 'offset', 'log'}
- **offset** (*int*) – offset to add to count only needed for smoothers 'log' and 'offset'

Returns

Return type float

Notes

The score is not commutative (i.e. score(A,B) != score(B,A))

class disamby.core.**ScoredElement** (*index*, *score*)
Bases: tuple

index
Alias for field number 0

score
Alias for field number 1

disamby.preprocessors module

This module contains the various string preprocessors

disamby.preprocessors.**compact_abbreviations** (*string: str*) → str
Removes dots between single letters and concatenates them

Parameters **string** –

Returns

Return type str

Examples

```
>>> compact_abbreviations('an other A.B.M this')
'AN OTHER ABM THIS'
```

disamby.preprocessors.**normalize_whitespace** (*string: str*) → str
removes duplicates whitespaces as well as replace tabs and newlines with a space

Parameters **string** –

Returns

Return type str

Examples

```
>>> normalize_whitespace('this is a          long  string')
'THIS IS A LONG STRING'
```

disamby.preprocessors.**ngram**(*string: str*, *n: int*) → tuple
constructs all possible ngrams from the given string. If the string is shorter then the n then the string is returned

Parameters

- **string** –
- **n** (*int*) – value must be larger at least 2

Returns

Return type tuple of strings

Examples

```
>>> ngram('this', 2)
('th', 'hi', 'is')
```

disamby.preprocessors.**trigram**(*string: str*) → tuple

disamby.preprocessors.**split_words**(*string: str*) → tuple
splits words on whitespace. This function is more reliable then *.split(' ')* since it works with any whitespace character (i.e. those recognized by regex)

Parameters **string** –

Returns

Return type tuple of strings

Examples

```
>>> len(split_words('a new day'))
3
```

disamby.preprocessors.**remove_punctuation**(*word: str*) → str
removes all punctuation symbols from the string

Parameters **word** (*str*) –

Returns

Return type str

Examples

```
>>> remove_punctuation('.has -a .few!')
'has a few'
```

disamby.preprocessors.**nword**(*word: str*, *k: int*) → tuple
concatenates k consecutive words into a tuple

Parameters

- **word** –
- **k** –

Returns

Return type tuple of strings

Examples

```
>>> nword('this that the other', 2)
('thisthat', 'thatthe', 'theother')
```

Module contents

Top-level package for disamby.

3.1.2 Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

Types of Contributions

Report Bugs

Report bugs at https://github.com/verginer/disamby/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

Write Documentation

disamby could always use more documentation, whether as part of the official disamby docs, in docstrings, or even on the web in blog posts, articles, and such.

Submit Feedback

The best way to send feedback is to file an issue at https://github.com/verginer/disamby/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

Get Started!

Ready to contribute? Here's how to set up *disamby* for local development.

1. Fork the *disamby* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:verginer/disamby.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv disamby
$ cd disamby/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 disamby tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, 3.3, 3.4 and 3.5, and for PyPy. Check https://travis-ci.org/verginer/disamby/pull_requests and make sure that the tests pass for all supported Python versions.

Tips

To run a subset of tests:

```
$ py.test tests.test_disamby
```

3.1.3 Credits

Development Lead

- Luca Verginer <luca@verginer.eu>

Contributors

None yet. Why not be the first?

3.1.4 History

0.2.3 (2017-07-01)

- Fixes formatting breaking pypi display

0.2.2 (2017-06-30)

- working release with minimal documentation
- works with multiple field matching
- carries out all steps autonomously from string pre-processing to identifying the strongly connected components

0.1.0 (2017-06-24)

- First release on PyPI.
- genindex
- modindex
- search

Python Module Index

d

Index