Match Documentation

Release 0.1.0

Ken Van Haren

Mar 26, 2017

Contents

Contents:

CHAPTER 1

Match

Probabilistic Entity Matching

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

Match brings common-sense entity detection and matching to python. Match is:

- Dead-simple to use
- Fast
- Lightweight (no heavy dependencies)
- Magic!

Installation

- TODO

Usage

Automatic entity detection and matching

```
>>> import match

# Auto detect entity type
>>> match.detect_type('608-555-5555')
(1, PhoneNumberType)
>>> match.detect_type('joe.van.gogh@example.com')
(1, EmailType)
>>> match.detect_type('John R. Smith')
(.95, FullNameType)
```

```
>>> match.detect_type('Hi, how are you?')
(1, StringType)
>>> match.score_types('Score this! @squaredloss')
[(0, EmailType), (.05, FullNameType), (0, PhoneNumberType), (1, StringType), (0,␣
↪DateTimeType), ...

# Score similarities intelligently based on detected type
>>> match.score_similarity('Jonathan R. Smith', 'john r smith')
(.92, FullNameType)
>>> match.score_similarity('123 easy st, NY, NY', '123 Easy Street, New York City')
(.98, AddressType)
>>> match.score_similarity('223 easy st, NY, NY', '123 easy st, NY, NY')
(.6, AddressType)
>>> match.score_similarity('Hi, how are you Joe?', 'hi how are you doing joe?')
(.81, StringType)
>>> match.score_similarity_as_type('608-555-5555', '608-555-5554', 'phonenumber')
.0
>>> match.score_similarity_as_type('608-555-5555', '608-555-5554', 'string')
.9

# Parse entity (to normalized string or object) based on detected type
>>> match.parse('(608) 555-5555')
('+1 608 555 5555', PhoneNumberType)
>>> match.parse('6085555555')
('+1 608 555 5555', PhoneNumberType)
>>> match.parse(' march 3rd, 1997', to_object=True)
(datetime.datetime(1997, 3, 3), DateTimeType)
>>> match.parse_as_type(' march 3rd, 1997', 'email')
None
```

Probabilistic matching, based on frequencies in a given corpus.

```
>>> from match import similarities
>>> import random

# Build similarity model from weighted random corpus of a's, b's, c's, and d's
>>> corpus = random.sample('a'*10000 + ' '*10000 + 'b'*1000 + 'c'*100 + 'd'*10,␣
↪k=21110)
>>> psim = similarities.ProbabilisticNgramSimilarity(corpus, grams=2)
>>> psim.similarity('ab ba c', 'ab ba d')
.6  # Lower similarity since 'a' is common
>>> psim.similarity('db bd c', 'db bd a')
.8  # Higher similarity since 'd' is rare
```

Custom types

```
>>> from match.similarity import ProbabilisticDiceCoefficient

# Build similarity model from custom corpus
>>> corpus = ''.join(['cheddar', 'brie', 'guyere', 'mozzarella', 'parmesian', 'jack',
↪'colby'])
>>> cheese_sim = ProbabilisticDiceCoefficient(corpus)
>>> match.add_type('cheese', StringType(similarity_measure=cheese_sim))
>>> match.detect_type('colby jack')
(.8, 'cheese')
```

Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER 2

Installation

Stable release

To install Match, run this command in your terminal:

```
$ pip install match
```

This is the preferred method to install Match, 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.

From sources

The sources for Match can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/kvh/match
```

Or download the tarball:

```
$ curl  -OL https://github.com/kvh/match/tarball/master
```

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

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

CHAPTER 3

Usage

To use Match in a project:

```
import match
```

CHAPTER 4

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/kvh/match/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

Match could always use more documentation, whether as part of the official Match 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/kvh/match/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 *match* for local development.

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

```
$ git clone git@github.com:your_name_here/match.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 match
$ cd match/
$ 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 match 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/kvh/match/pull_requests and make sure that the tests pass for all supported Python versions.

Tips

To run a subset of tests:

```
$ python -m unittest tests.test_match
```

CHAPTER 5

Indices and tables

- genindex
- modindex
- search