malaffinity

Release 2.4.0

Aug 17, 2017

Getting Started

CHAPTER 1

Calculate affinity between MyAnimeList users

Contents:

Introduction

What is this?

malaffinity provides a simple way to calculate affinity (Pearson's correlation * 100) between a "base" user and another user on MyAnimeList.

Note: The term "base user" refers to the user whose scores other users' scores will be compared to (and affinities to said scores calculated for).

Just assume the "base user" is referring to you, or whoever will be running your script, unless you're getting into some advanced mumbo-jumbo, in which case you're on your own.

malaffinity is meant to be used in bulk, where one user (the "base")'s scores are compared against multiple people, but there's nothing stopping you from using this as a one-off.

But why should I bother using this? Doesn't MAL give me an affinity?

Let's consider what you'd have to do if you wanted MAL to give you an affinity value, and a good estimation as to whether it's "accurate" or not:

- Create a `requests.Session()`
- Make a `GET` request to MAL's login page
- Retrieve the `csrf_token` from one of the `meta` headers
- Make a `POST` request to the login page, providing a username, password, a bunch of stupid form data, and the `csrf_token` you've just obtained
- Confirm you are logged in, by seeing if the `is_logged_in` cookie is present in the `CookieJar`
- Visit a users' profile
- Look for the affinity value (hint: `.user-compatability-graph .anime .bar-inner` [sic])
- Read its `innerHTML`, retrieve the affinity value, add a case in to get rid of the double-negative that appears on any negative value for some reason
- Find how many rated anime you share. The value MAL gives you includes unrated anime, and PTW stuff. It's not an accurate indicator
  - Visit `/shared.php?u1=you&u2=them` and you find yourself trying to navigate through the dark and murky world of bad HTML table parsing

Congrats, you've just wasted a few hours of your life, and you're probably a bit stressed right now. HTML parsing does that to you.

Let's see how you could handle all this with `malaffinity`, assuming your username is `Xinil` and you want to calculate affinity with `Luna`:

```
from malaffinity import MALAffinity

ma = MALAffinity("Xinil")

affinity, shared = ma.calculate_affinity("Luna")

# Do whatever you like with ``affinity`` and ``shared``
print(affinity)
# 37.06659111674594
print(shared)  # Note: Is referring to shared, rated anime
# 171
```

Note: `ma` now holds your scores. You can easily call `ma.calculate_affinity` on anyone, and you'd get your affinity with them.

If you don't want your scores to be stored, an option exists for quick, one-off calculations:

```
import malaffinity

affinity, shared = malaffinity.calculate_affinity("Xinil", "Luna")

# ...
```

I'm no expert, but the code(s) above looks a lot neater than the alternative would've looked.

Getting Started

Install

```
$ pip install malaffinity
```

Alternatively, download this repo and run:

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

To use the development version (please don't), run:

```
$ pip install --upgrade https://github.com/erkghlerngm44/malaffinity/archive/master.
↪zip
```

Dependencies

- `BeautifulSoup4`
- `lxml`
- `Requests`

These should be installed when you install this package, so no need to worry about them.

`lxml` is a bit wonky sometimes. If install fails:

```
$ pip install --upgrade pip
$ pip install --upgrade lxml
```

If all else fails and you're on Windows, download the wheel yourself and:

```
$ pip install /path/to/wheel.whl
```

Development

This section demonstrates how documentation can be built, tests run, and how to check if you're adhering to PEP8 and PEP257. These should not be used unless you're contributing to the package.

Conventions

The `flake8` (PEP8) and `pydocstyle` (PEP257) packages can be run to check that PEP8 and PEP257 are being followed.

These can be installed as follows:

```
$ pip install flake8 pydocstyle
```

The following commands can then be run:

```
$ flake8
$ pydocstyle malaffinity
```

which will print any warnings/errors/other stuff. These should ideally be fixed, but in the event that they can't, place a `# noqa: ERROR_CODE` comment on the offending line(s).

Documentation

To install the dependencies needed to build the docs, run:

```
$ pip install .[doc]
```

The docs can then be built by navigating to the `docs` directory, and running:

```
$ make html
```

The built docs will now be in `./_build/html`. You can either run them by clicking and viewing them, or by running a server in that directory, which you can view in your browser.

Note: Any warnings that show up when building will be interpreted as errors when the tests get run on Travis, which will cause the build to fail. You'll want to make sure these are taken care of.

Test Suite

To install the dependencies needed for the test suite, run:

```
$ pip install .[test]
```

It is advised to run the test suite through `coverage`, so a coverage report can be generated as well. To do this, run:

```
$ coverage run --source malaffinity setup.py test
```

The tests should then run. You can view the coverage report by running:

```
$ coverage report
```

Examples

This section will show the various ways the `MALAffinity` class can be initialised with the user `Xinil` (MAL creator), and used to calculate affinity or get a comparison with the user `Luna` (MAL database admin).

Initialising the Class

The class can be initialised in either one of two ways:

Method 1: Normal initialisation

The class is initialised, with a "base user" passed as an argument to `MALAffinity`.

```
ma = MALAffinity("Xinil")
```

Method 2: Specifying a "base user" after initialisation

The class is initialised, with a "base user" passed sometime later after initialisation, which may be useful in scripts where creating globals inside functions or classes or different files is a pain.

```
ma = MALAffinity()

# This can be done anywhere, as long as it has access to ``ma``,
# but MUST be done before ``calculate_affinity`` or ``comparison``
# are called
ma.init("Xinil")
```

Rounding of the final affinity value

Note: This doesn't affect *comparison()*, so don't worry about it if you're just using that.

Do note that the class also has a `round` parameter, which is used to round the final affinity value. This must be specified at class initialisation if wanted, as it isn't available in *init()*. A value for this can be passed as follows:

```
# To round to two decimal places
ma = MALAffinity("Xinil", round=2)

# Alternatively, the following can also work, if you decide to follow
# method 2 for initialising the class
ma = MALAffinity(round=2)
ma.init("Xinil")
```

Doing Things with the Initialised Class

The initialised class, now stored in `ma`, can now perform the following actions:

Calculate affinity with a user

Note: Values may or may not be rounded, depending on the value you passed for the `round` parameter at class initialisation.

```
print(ma.calculate_affinity("Luna"))
# (37.06659111674594, 171)
```

Note that what is being returned is a tuple, containing the affinity and shared rated anime. This can be separated into different variables as follows:

```
affinity, shared = ma.calculate_affinity("Luna")

print(affinity)
# 37.06659111674594
print(shared)
# 171
```

Comparing scores with a user

```
comparison = ma.comparison("Luna")

print(comparison)
# Note: this won't be prettified for you. Run it
# through a prettifier if you want it to look nice.
# {
#     1: [10, 6],
#     5: [8, 6],
#     6: [10, 7],
#     15: [7, 9],
#     16: [8, 5],
#     ...
# }
```

This can now be manipulated in whatever way you like, to suit your needs. I like to just get the arrays on their own, zip them and plot a graph with it.

Extras

One-off affinity calculations

This is mainly used if you don't want the "base user"'s scores saved to a variable, and you're only interested in the affinity with one person.

Warning: This sends two GET requests over to MAL in a short amount of time, with no wait inbetween them. If you're getting in trouble with them for breaking their rate limit, you might have a few problems getting this to work without `MALRateLimitExceededError` getting raised.

```
# Note that ``round`` can also be specified here if needed.
affinity, shared = calculate_affinity("Xinil", "Luna")

print(affinity)
# 37.06659111674594
print(shared)
# 171
```

Note: Don't use this if you're planning on calculating affinity again with one of the users you've specified when using this.

It's better to create an instance of the `MALAffinity` class with said user, and calculating affinity with the other user(s) that way.

That instance will hold said users' scores, so they won't have to be retrieved again. See the other examples.

One-off comparison of scores

This is mainly used if you don't want the "base user"'s scores saved to a variable, and you're only interested in getting a comparison of scores with another user.

Warning: This sends two GET requests over to MAL in a short amount of time, with no wait inbetween them. If you're getting in trouble with them for breaking their rate limit, you might have a few problems getting this to work without `MALRateLimitExceededError` getting raised.

```
print(comparison("Xinil", "Luna"))

# Note: this won't be prettified for you. Run it
# through a prettifier if you want it to look nice.
# {
#     1: [10, 6],
#     5: [8, 6],
#     6: [10, 7],
#     15: [7, 9],
#     16: [8, 5],
#     ...
# }
```

The MALAffinity class

class `malaffinity.`**`MALAffinity`**(*base_user=None*, *round=False*)

The MALAffinity class.

The purpose of this class is to store a "base user"'s scores, so affinity with other users can be calculated easily.

For the user `Xinil`, the class can be initialised as follows:

```
from malaffinity import MALAffinity

ma = MALAffinity("Xinil")
```

The instance, stored in `ma`, will now hold `Xinil`'s scores.

`comparison()` and *`calculate_affinity()`* can now be called, to perform operations on this data.

`__init__`(*base_user=None*, *round=False*)

Initialise an instance of *MALAffinity*.

Note: To avoid dealing with dodgy globals, this class MAY be initialised without the `base_user` argument, in the global scope (if you wish), but *`init()`* MUST be called sometime afterwards, with a `base_user` passed, before affinity calculations take place.

Example (for the user `Xinil`):

```
from malaffinity import MALAffinity

ma = MALAffinity()

ma.init("Xinil")
```

The class should then be good to go.

Parameters

- **base_user** (*str or None*) – Base MAL username
- **round** (*int or False*) – Decimal places to round affinity values to. Specify `False` for no rounding

calculate_affinity(*username*)

Get the affinity between the "base user" and `username`.

Note: The data returned will be a tuple, with the affinity and shared rated anime. This can easily be separated as follows (using the user `Luna` as `username`):

```
affinity, shared = ma.calculate_affinity("Luna")
```

Note: The final affinity value may or may not be rounded, depending on the value of `_round`, set at class initialisation.

Parameters **username** (*str*) – The username to calculate affinity with

Returns (float affinity, int shared)

Return type tuple

comparison(*username*)

Get a comparison of scores between the "base user" and `username`.

A Key-Value returned will consist of the following:

```
{
    ANIME_ID: [BASE_USER_SCORE, OTHER_USER_SCORE],
    ...
}
```

Example:

```
{
    30831: [3, 8],
    31240: [4, 7],
    32901: [1, 5],
    ...
}
```

> **Warning:** The JSON returned isn't valid JSON. The keys are stored as integers instead of the JSON standard of strings. You'll want to force the keys to strings if you'll be using the ids elsewhere.

Parameters **username** (*str*) – The username to compare the base users' scores to

Returns Key-value pairs as described above

Return type dict

init(*base_user*)

Retrieve a "base user"'s list, and store it in `_base_scores`.

Parameters **base_user** (*str*) – Base users' username

Handling Exceptions

The types of exceptions that can be raised when calculating affinities are:

class malaffinity.exceptions.**NoAffinityError**

Raised when either the shared rated anime between the base user and another user is less than 10, the user does not have any rated anime, or the standard deviation of either users' scores is zero.

class malaffinity.exceptions.**InvalidUsernameError**

Raised when username specified does not exist.

class malaffinity.exceptions.**MALRateLimitExceededError**

Raised when MAL's blocking your request, because you're going over their rate limit of one request every two seconds. Slow down and try again.

---

exceptions.NoAffinityError and *exceptions.InvalidUsernameError* are descendants of:

class malaffinity.exceptions.**MALAffinityException**

Base class for MALAffinity exceptions.

which means if that base exception gets raised, you know you won't be able to calculate affinity with that person for some reason, so your script should just move on.

---

exceptions.MALRateLimitExceededError rarely gets raised if you abide by the rate limit of one request every two seconds. If it does get raised, the following should happen:

- Halt the script for a few seconds. I recommend five.
- Try again.
- If you get roadblocked again, just give up. MAL obviously hates you.

These can be achieved via something like this:

```
success = False

for _ in range(2):
    try:
        affinity, shared = ma.calculate_affinity("OTHER_USERNAME")
```

```
    # Rate limit exceeded. Halt your script and try again
    except malaffinity.exceptions.MALRateLimitExceededError:
        time.sleep(5)
        continue

    # Any other malaffinity exception.
    # Affinity can't be calculated for some reason.
    # ``MALAffinityException`` is the base exception class for
    # all malaffinity exceptions
    except malaffinity.exceptions.MALAffinityException:
        break

    # Exceptions not covered by malaffinity. Not sure what
    # you could do here. Feel free to handle however you like
    except Exception as e:
        print("Something went wrong. Please contact Xinil for further assistance:")
        print("* https://myanimelist.net/profile/Xinil")
        print("* https://www.reddit.com/user/Xinil")
        print("Please also nag him to create a half-decent MAL API for gods sake.")
        print("")
        print("Exception: `{}`".format(e))
        break

    # Success!
    else:
        success = True
        break

# ``success`` will still be ``False`` if affinity can't been calculated.
# If this is the case, you'll want to stop doing anything with this person
# and move onto the next, so use the statement that will best accomplish this,
# given the layout of your script
if not success:
    return

# Assume from here on that ``affinity`` and ``shared`` hold their corresponding
# values, and feel free to do whatever you want with them
```

Feel free to use a `while` loop instead of the above. I'm just a bit wary of them, in case something happens and the script gets stuck in an infinite loop. Your choice.

Contributing

In the unlikely event that someone finds this package, and in the even unlikelier event that someone wants to contribute, send me a pull request or create an issue.

Feel free to use those for anything regarding the package, they're there to be used, I guess.

How to Contribute

- Fork the repo.
- `git clone https://github.com/YOUR_USERNAME/malaffinity.git`
- `cd malaffinity`

- `git checkout -b new_feature`
- Make changes.
- `git commit -am "Commit message"`
- `git push origin new_feature`
- Navigate to https://github.com/YOUR_USERNAME/malaffinity
- Create a pull request.

Notes and Stuff

I had a whole section on conventions to follow and other stuff, but that seemed a bit weird, so I just scratched it. If someone out there wants to contribute to this package in any way, shape or form, have at it. I'd prefer the changes to be non-breaking (i.e. existing functionality is not affected), but breaking changes are still welcome.

I only ask that you try to adhere to PEP8 and PEP257 (if you can), and try to achieve 100% coverage in tests (again, if you can). For information on how to check if you're adhering to those conventions, see *Conventions*.

For information on how to build docs and run tests, see *Documentation* and *Test Suite* respectively.

This package is based off a class I wrote for `erkghlerngm44/r-anime-soulmate-finder`, and while I have tried to modify it for general uses (and tried to clean the bad code up a bit), there are still a few iffy bits around. I'd appreciate any PRs to fix this up.

I think the package is mostly complete, so my main focus right now is making it as fast as can-be, as every fraction of a second counts when you're using this to calculate affinity with tens of thousands of people. PRs regarding this are especially welcome.

That's it, I guess. *Contact* me if you need help or anything.

Contact

Contact me on Reddit or by Email.

Note: Emailing me is pretty much pointless, since I rarely check that address. Contact me on Reddit if you need anything.

License

Licensed under MIT.

See `LICENSE` in the GitHub repo for the boring text.

Index

Symbols

C

I

M

N