aniffinity

Release 0.2.0

Apr 24, 2019

Getting Started

Calculate affinity between anime list users

CHAPTER 1

Contents:

1.1 Introduction

1.1.1 What is this?

Aniffinity provides a simple way to calculate affinity (Pearson's correlation * 100) between a "base" user and another user on anime list services.

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.

For a list of anime list services that can be used with Aniffinity, see *Available Services*.

Aniffinity 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.

1.2 Getting Started

1.2.1 Install

```
$ pip install aniffinity
```

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/aniffinity/archive/master.zip
```

1.2.2 Dependencies

- `json-api-doc`
- `requests`

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

1.2.3 Development

This section demonstrates how documentation can be built, tests run, and how to check if you're adhering to **PEP 8** and **PEP 257**. These should not be used unless you're contributing to the package.

Conventions

The `flake8` and `pydocstyle` packages can be used to check that PEP 8 and PEP 257 are being followed respectively.

These can be installed as follows:

```
$ pip install .[conventions]
```

The following commands can then be run:

```
$ flake8
$ pydocstyle aniffinity
```

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 .[docs]
```

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 .[tests]
```

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 aniffinity setup.py test
```

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

```
$ coverage report
```

1.3 Services

1.3.1 Available Services

The following anime list services can be used with Aniffinity:

- Anilist
  - **Aliases**: `AL`, `A`
- Kitsu[1][2]
  - **Aliases**: `K`
- MyAnimeList[3]
  - **Aliases**: `MAL`, `M`

Note: Do note, this package is designed for cross-service compatibility. You are able to, say, calculate affinity (or compare scores) with one user on AniList and another on Kitsu, for example.

There's nothing restricting you to stick with users from one service, unless that's your intention.

---

[1] Incredibly slow, due to constraints by the service API itself and not this package. Nothing I can do about it, sorry.

[2] When passing a username to Kitsu, use the "slug" (from the "profile url") instead of the "display name" as "display name"s aren't unique in Kitsu. The "slug" refers to this part of the user's profile URL: `https://kitsu.io/users/<SLUG>`.

[3] The MyAnimeList API being used is undocumented by them (probably because it's only meant to be used internally) and may change at any time without warning. Not much I can do about that, if MAL decides to cough up a real API in the (distant) future, I'll change it over to that. **Until then, if this API fails and I can't fix it, I'll just remove MyAnimeList support altogether.**

1.4 Walkthrough

This section will show the various ways the *Aniffinity* class can be initialised with the username `Foobar` on the service `AniList`, and used to calculate affinity or get a comparison with the user `baz` on the service `Kitsu`.

1.4.1 Passing Arguments to the Class and its Methods

As multiple services can be used in this package, there needs to be a way of telling it which service to use.

In general, the data that needs to be passed to the class and its methods are a user's "username" and the "service", so that the package can decide which service endpoint to call.

When initialising the class for the "base" user, the names of these params will be `base_user` and `base_service`, to denote that this information applies only to the "base" user. When using the methods inside the class, these params will be called `user` and `service`.

Using the class method *Aniffinity.calculate_affinity()* to demonstrate, which has the `user` and `service` params, the various ways of passing this info are as follows:

Method 1: Using the respective arguments

This is by far the fastest method, with the least amount of computing done to determine which service to use.

```
af.calculate_affinity("baz", service="Kitsu")
```

Method 2: Passing a tuple

If a tuple is passed, it must take the form `(username, service)`.

```
user = ("baz", "Kitsu")
af.calculate_affinity(user)
```

Note: A namedtuple exists in `aniffinity.models`, which is created for this very purpose. If you wish to use it, this can be created as follows:

```
user = models.User(username="baz", service="Kitsu")
```

This can then be passed where necessary.

Method 3: Passing a URL

```
# Do note that this method is somewhat lenient, and also somewhat strict,
# in the types of URL that it will take. A link to either the user's
# profile, or their anime list will work.
af.calculate_affinity("https://kitsu.io/users/baz")
```

Method 4: Passing a string

If a string is passed that is not a URL, it should take one of the following forms:

- `SERVICE:USERNAME`
- `SERVICE/USERNAME`

Note: Note the lack of a space between the `:` and `/`. This will not work if there are any spaces between these characters.

```
af.calculate_affinity("Kitsu:baz")
# or
af.calculate_affinity("Kitsu/baz")
```

Method 5: Passing a username only

Warning: This is highly unrecommended - the behaviour of this cannot be guaranteed, but if you are in a rush then this option does exist.

If a username and no service is passed, the package will use the default service which, at the time of writing, is AniList.

```
af.calculate_affinity("baz")
```

Aliases

For methods 1, 2 and 4, there exist aliases for the service names, which can be used in place of the full service name. For a list of aliases and services, see *Available Services*.

1.4.2 Initialising the Class

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

Method 1: Normal initialisation

The class is initialised, with the necessary arguments passed to the *`Aniffinity`* class.

```
af = Aniffinity("Foobar", service="AniList")
```

Method 2: Specifying the arguments after initialisation

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

```
af = Aniffinity()

# This can be done anywhere, as long as it has access to ``af``,
# but MUST be done before ``calculate_affinity`` or ``comparison``
# are called
af.init("Foobar", service="AniList")
```

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
af = Aniffinity(..., round=2)

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

1.4.3 Doing Things with the Initialised Class

The initialised class, now stored in `af`, 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(af.calculate_affinity("baz", service="Kitsu"))
# Affinity(value=37.06659111674594, shared=171)
```

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

```
affinity, shared = af.calculate_affinity("baz", service="Kitsu")

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

Alternatively, the following also works (as this is a namedtuple):

```
affinity = af.calculate_affinity("baz", service="Kitsu")

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

Comparing scores with a user

```
comparison = af.comparison("baz", service="Kitsu")

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],
#     ...
# }
```

Note that a key-value pair returned here consist of: `"MYANIMELIST_ID": [BASE_USER_SCORE, OTHER_USER_SCORE]`.

Note: MyAnimeList IDs are used here as a cross-service-compatible identifier is needed to match up each anime across services, as the anime ids used in different services may differ from each other.

If you wish to use the anime ids for the service you specify, set the param <TO_BE_IMPLEMENTED> to <TO_BE_IMPLEMENTED>

This data 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.

1.4.4 Extras

Warning: These send a request over to each service 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 these to work without *exceptions.RateLimitExceededError* getting raised.

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

It's better to create an instance of the *Aniffinity* class with said user, and using that 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.

For each of these functions below, assume the following variables were set in advance:

```
user1 = models.User("Foobar", service="AniList")
user2 = models.User("Baz", service="Kitsu")
```

Note: As there are no params to specify which service to use for each user, specify this information for both `user1` and `user2` by passing a tuple for each of these, containing (username, service).

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.

```
# Note that ``round`` can also be specified here if needed.
affinity, shared = calculate_affinity(user1, user2)

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

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.

```
print(comparison(user1, user2))

# 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],
#     ...
# }
```

1.5 API

class aniffinity.**Aniffinity**(*base_user=None*, *base_service=None*, *round=10*, ***kws*)

The Aniffinity 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 username `Josh` on the service `AniList`, the class can be initialised as follows:

```
from aniffinity import Aniffinity

af = Aniffinity("Josh", base_service="AniList")
```

There are multiple ways of specifying this information, and multiple ways to initialise this class. For more info, read the documentation.

The instance, stored in `af`, will now hold `Josh`'s scores.

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

`__init__`(*base_user=None*, *base_service=None*, *round=10*, ***kws*)
Initialise an instance of `Aniffinity`.

The information required to retrieve a users' score from a service are their "username" and "service". For a list of "service"s, read the documentation.

Note: As this applies to the "base" user, the params used are `base_user` and `base_service` respectively.

There are multiple ways of specifying the above information, and multiple aliases for services that can be used as shorthand. As docstrings are annoying to write, please refer to the documentation for a list of these. For an example of the simplest method to use, refer to the docstring for the *`Aniffinity`* class.

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` and `base_service` passed, before affinity calculations take place.

Example (for the username `Josh` on the service `AniList`):

```
from aniffinity import Aniffinity

af = Aniffinity()

ma.init("Josh", base_service="AniList")
```

The class should then be good to go.

Parameters

- **`base_user`** (*`str or tuple`*) – Base user
- **`base_service`** (*`str or None`*) – The service to use. If no value is specified for this param, specify the service in the `base_user` param, either as part of a url, or in a tuple
- **`round`** (*`int or False`*) – Decimal places to round affinity values to. Specify `False` for no rounding
- **`wait_time`** (*`int`*) – Wait time in seconds between paginated requests (default: 2)

`calculate_affinity`(*user*, *service=None*)
Get the affinity between the "base user" and `user`.

Note: The data returned will be a namedtuple, with the affinity and shared rated anime. This can easily be separated as follows:

```
affinity, shared = af.calculate_affinity(...)
```

Alternatively, the following also works:

```
affinity = af.calculate_affinity(...)
```

with the affinity and shared available as `affinity.value` and `affinity.shared` respectively.

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

Parameters

- **user** (*str or tuple*) – The user to calculate affinity with.
- **service** (*str or None*) – The service to use. If no value is specified for this param, specify the service in the `user` param, either as part of a url, or in a tuple

Returns (float affinity, int shared)

Return type tuple

comparison(*user*, *service=None*)

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

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],
    ...
}
```

Note: The `ANIME_ID` s will be the MyAnimeList anime ids. As annoying as it is, cross-compatibility is needed between services to get this module to work, and MAL ids are the best ones to use as other APIs are able to specify it. If you wish to use the anime ids for the service you specified, set the param `<TO BE IMPLEMENTED>` to `<TO BE IMPLEMENTED>`.

Parameters

- **user** (*str or tuple*) – The user to compare the base users' scores to.
- **service** (*str or None*) – The service to use. If no value is specified for this param, specify the service in the `user` param, either as part of a url, or in a tuple

Returns Mapping of `id` to `score` as described above

Return type dict

init(*base_user*, *base_service=None*)

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

Parameters

- **base_user** (*str or tuple*) – Base user
- **base_service** (*str or None*) – The service to use. If no value is specified for this param, specify the service in the `base_user` param, either as part of a url, or in a tuple

1.6 Handling Exceptions

1.6.1 Which exceptions can be raised?

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

exception aniffinity.exceptions.**NoAffinityError**
Raised when either the shared rated anime between the base user and another user is less than 11, the user does not have any rated anime, or the standard deviation of either users' scores is zero.

exception aniffinity.exceptions.**InvalidUserError**
Raised when username specified does not exist in the service, or the service does not exist.

exception aniffinity.exceptions.**RateLimitExceededError**
Raised when the service is blocking your request, because you're going over their rate limit. Slow down and try again.

If you're planning on using this package in an automated or unsupervised script, you'll want to make sure you account for these getting raised, as not doing so will mean you'll be bumping into a lot of exceptions, unless you can guarantee none of the above will get raised. For an example snippet of code that can demonstrate this, see *Exception Handling Snippet*.

1.6.2 AniffinityException

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

exception aniffinity.exceptions.**AniffinityException**
Base class for Aniffinity 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.

1.6.3 What to do if RateLimitExceededError gets raised

exceptions.RateLimitExceededError rarely gets raised if you abide by the rate limits of the services you are using. This may be something like one request a second, or 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.

Note: The name of the service ratelimiting you will be in the exception message, should this be of any use to you.

1.6.4 Exception Handling Snippet

The above can be demonstrated via something along these lines. Do note that this probably isn't the best method, but it works.

This should be placed in the section where you are attempting to calculate affinity, or get a comparison, with another user.

```
time.sleep(2)

success = False

for _ in range(2):
    try:
        affinity, shared = af.calculate_affinity("Baz", service="Kitsu")

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

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

    # Exceptions not covered by aniffinity. Not sure what
    # you could do here. Feel free to handle however you like
    except Exception as e:
        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.

To see the above snippet in action, visit erkghlerngm44/r-anime-soulmate-finder.

1.7 Changelog

1.7.1 v0.2.0 (2019-04-24)

- Make `.calcs.pearson` raise a `ZeroDivisionError` when the standard deviation of one/both sets of data is zero. This will be caught by `Aniffinity.calculate_affinity` and will then raise the usual `NoAffinityError`, so scripts using this package will not need to be modified.
- Fix the faulty URL resolving in the resolving function. Valid usernames starting with "http" will now be handled correctly, instead of having an exception raised when no accompanying service is specified.
- Move the user/service resolving functions to `resolver.py`, and rename these functions to more meaningful names. Additionally, make these functions non-protected, adding in official support for them.
- Change the repr of the `Aniffinity` class to make it more accurate.
- Update various docstrings to make more sense and be more accurate.
- Bump the version for the dependency `json-api-doc` to `v0.7.x`.
- Handle the `Decimal` handling in `.calcs.pearson` better, by converting all of the values in each list to strings before passing them to `decimal.Decimal`.
- Round affinity values by default to 10dp, so floating-point issues no longer need to be accounted for. This can be bypassed by the user, should they wish to do so.
- Don't convert the scores lists to `list`-s to make them non-lazy, as this is already done.
- Allow the username & service to be specified as a string, in the form `service:username`.
- Resolve the `user` before raising exceptions, allowing the exception messages to include the service as well as the username.
- Include the relevant usernames in the "standard deviation is zero" exception message in `NoAffinityError`.

1.7.2 v0.1.2 (2019-04-15)

- No code changes have been made. This release is to confirm that Travis has been fixed. (third time lucky, hopefully)

1.7.3 v0.1.1 (2019-04-15)

- No code changes have been made. This release is to confirm that Travis has been fixed.

1.7.4 v0.1.0 (2019-04-15)

- Discontinue Python 2 support.
- Rename this package to `Aniffinity` and replace all occurances with this.
- Add AniList and Kitsu endpoints and support.
- Rename exception `MALRateLimitExceededError` to `RateLimitExceededError`.
- Add in a way to specify which service to use, either through the `service` param, using a tuple with the username and service, or passing the URL to a users' profile.
- Create the `User` namedtuple.

- Make AniList the default service to use (as it's the most stable).
- Rename the `affinity` field in the `Affinity` namedtuple to `value`.
- Rename exception `InvalidUsernameError` to `InvalidUserError`.
- Force the `_base_scores` `id` keys to strings.
- Change the MyAnimeList API to an official-yet-unofficial semi-working one.
- Speed up the creation of the `comparison` dict.
- Add the service name to all service-specific exceptions.
- Add the `wait_time` arg to the `Aniffinity` class to slow down paginated requests.

For older changes, read the changelog at erkghlerngm44/malaffinity.

1.8 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.

Note: Please *Contact* and notify me if you use the above, as this isn't my main GitHub account, so I won't be checking it that much. I'll probably see it weeks/months later if you don't.

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

1.8.1 How to Contribute

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

1.8.2 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 **PEP 8** and **PEP 257** (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.

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

1.9 Contact

On the off chance that someone wants to contact me, I can be reached via the following (ordered from fastest to slowest in terms of time it'll take to get a response from me):

- Reddit (/u/erkghlerngm44)
- Discord (erkghlerngm44#9210)
- Email (erkghlerngm44@protonmail.com)

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

1.10 License

Licensed under MIT.

```
MIT License

Copyright (c) 2017 erkghlerngm44

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE
SOFTWARE.
```

Index

Symbols

A

C

I

N

P

R