Brigit Documentation

Release 0.1.0

Raúl Fernández Díaz

Nov 22, 2022

CONTENTS:

CHAPTER

ONE

BRIGITMETALPREDICTOR

BrigitMetalPredictor or Brigit, for short, is a computational tool designed for the prediction of protein-metal binding sites in proteins. It uses a novel scoring function powered by a deep learning model and previous domain knowledge regarding bioinorganic interactions.

The deep learning model used is a 3D Convolutional Neural Network (CNN) that interprets the physico-chemical environment. The previous domain knowledge is based on the works by Sánchez-Aparicio et al. (2017), and translates into the use of statistics to score the suitability of a point of space based on its relative position to certain atoms in the protein backbone.

1.1 Features

Diferent options for customizing the search:

- Search for binding sites of specific metals.
- Filter results according to how likely they are to bind metals.
- Takes into account binding with backbone nitrogens and oxygens.
- Scanning the whole protein, though, a region can also be provided as input (in PDB format).

Modular design:

- The modular design of this package allows for its use as a command-line tool or to be integrated into a larger Python program or pipeline.

Possible applications:

- Screening of a pool of *.pdb* structures.
- Identification of conformational changes that alter the formation of metal-binding sites.
- Identification of probable paths that the metals might have to traverse to transitorily binding regions before reaching the more stable, final, binding site.
- Metalloenzyme design.
- Metallodrug design.

1.2 License

Brigit is an open-source software licensed under the BSD-3 Clause License. Check the details in the LICENSE file.

1.3 TODO

- Modify clustering or check clustering functions to identify specific motifs.
- Modify clustering or check clustering to propose mutations.

1.4 History of versions

- **v.0.1:** First operative release version.

1.5 OS Compatibility

Brigit is currently only compatible with Linux, due to some of its dependencies.

If you find some dificulties when installing it in a concrete distribution, please use the issues page to report them.

1.6 Credits

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

CHAPTER

TWO

INSTALLATION

2.1 Stable release

To install Brigit, run this command in your terminal:

Still not implemented

2.2 Requirements

There are quite a few requirements that have to be installed in order for this program to work, so it is recommended to initialize a specific conda environment. Further, some of this commands will generate directories within your desktop so it is recommended to create a directory from where to install the program.

```
$ mkdir brigit_install

$ cd brigit_install

$ wget https://raw.githubusercontent.com/RaulFD-creator/brigit/master/environment.yml

$ conda env create -f environment.yml

$ conda activate brigit

$ git clone https://github.com/Acellera/moleculekit

$ pip install moleculekit
```

The installation of CUDA depends on the version of CUDA available in your machine, by default this is the recommended setup. However, if any problem arises, the version of pytorch-cuda should be changed.

```
$ conda install pytorch pytorch-cuda=11.6 pytorch-lightning torchmetrics -c pytorch -c␣
↪nvidia -c conda-forge
```

If you do not have a GPU with CUDA available, instead install the following:

```
$ conda install pytorch pytorch-lightning torchmetrics -c conda-forge -c pytorch
```

2.3 From sources

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

The first step will be to clone the repository. This process will create a directory in whichever location you are currently at. It is recommended to continue in the same directory from the previous step.

```
$ git clone https://github.com/RaulFD-creator/brigit
```

Once you have a copy of the source, without moving from the directory:

```
$ cd brigit
$ pip install .
```

Or alternatively,

```
$ cd brigit
$ python setup.py install
```

2.4 Solving problems with dependencies

CHAPTER

THREE

USAGE

3.1 Input parameters

The following list contains the possible parameters that can be introduced with the program, their default values, and a brief explanation.

First, the program has two mandatory arguments:

- **target:** PDB code to be analysed or Path to a local PDB file.
- **metal:** Nature of the metal for which possible binding sites are to be

predicted. The complete list of metals currently supported are: Li, Na, K, Rb, Cs, Be, Mg, Ca, Sr, Ba, V, Cr, Mn, Fe, Co, Ni, Cu, Zn, Mo, Ru, Rh, Pd, Ag, Cd, Ir, Pt, Au, and Hg. There is an additional mode, General, that uses statistics obtained from all metals in the MetalPDB database.

Optional parameters have to be introduced by '- -', examples are provided in the next section:

- **model:** Which of the trained CNN models do you want to use for the predictions. It accepts two possible values: 'BrigitCNN' which is the default, and 'tinyBrigit' which is a more lightweight version with sligthly worse predictive performance that is recommended for environments were computational efficiency is paramount.
- **device:** Whether the calculations are to be performed in CPU or GPU (default). The program subsequently verifies whether GPU devices are available and if not switches to CPU.
- **device_id:** In cases with multiple GPU devices specify in which the computations should be run. By default is set to device 0.
- **outputfile:** Base name of the output files generated by the program. It is recomended to not include extensions as it will add the extensions by itself. The idea is to generate 2 outputs with names: outputfile_brigit.pdb and outputfile_clusters.txt. The first file will contain information about the probes used to locate possible binding sites and/or metal migration pathways. Should be visualised with a protein/molecule visualizater like UCSF Chimera. To visualize the scores of each probe, select colour by property -> b-factor. The _clusters.txt file contains information regarding the residues closer than *cluster_radius* from the center of the cluster.
- **max_coordinators:** Is a normalization factor that considers the number of residues that are expected to be coordinating the metal center. By default: 2.
- **residues:** Number of residues to be considered when computing possible coordinations. It selects the most likely residues. This parameter affects mostly the performance. By default is set at 10, lower number will compute faster, though, there are some coordinations with specific residues that may not be considered. In the heading of the *_clusters.txt* file, the residues considered for each type of coordination are included.
- **stride:** Step of the CNN sliding window when evaluating the protein. By default: 1. Increase this number will significantly increase the computational speed though it may decrease equally both the accuracy and the sensitivity of the method. It has to be an integer. It is not recomended to change it, except for preliminary analysis where speed is important.

- **pH:** Acidic conditions of the environment. Can be changed when the effect of the protonation of the system is believed to be an important factor in the coordination. By default: 7.4.
- **cluster_radius:** Radius of the clusters. By default: 5 A.
- **cnn_threshold:** Value for the CNN score filter. By default is set to 0.5. It is not recommended to change, unless there is suspicion that there are binding sites that should be detected that are not. However, for this purpose it is recomended first to try different values of *cnn_weight* or *combined_threshold*.
- **combined_threshold:** Value for the final hybrid score to filter results. By default: 0.5. Can be modified depending on how stringent the final predictions are supposed to be. Higher values will only allow for higher scores and conversely.
- **cnn_weight:** Relative importance of the CNN score and the coordination score when computing the final hybrid score. By default: 0.5. Greater values rely more heavily on the predictions by the CNN, and thus, in the second coordination information; whereas, lower values rely more on the coordination analysis and, therefore, on the first coordination sphere.
- **voxelsize:** Resolution when creating the 3D representation in A. Modification might lead to visual artifacts. The accuracy of the CNN is unclear for resolutions different from the default (1 A).
- **residue_score:** Scoring function for the coordination analysis of side chains. By default: Gaussian. Other available options: 'discrete'.
- **verbose:** Information that will be displayed. 0: Only Moleculekit, 1: All.
- **threads:** Number of threads available for multithreading. By default it will create 2 threads per physical core.

3.2 Console use

The default use of Brigit is simple

```
$ brigit {target} {metal}
```

For example:

```
$ brigit 1dhy Fe
```

An output file name can be set:

```
$ brigit 1dhy Fe --outputfile output_name
```

Furthermore, the maximum number of coordinators to consider (*–max_coordinators*) or the threshold score (*–combined_threshold*), can also be modified:

```
$ brigit 1dhy Fe --max_coordinators 4 --combined_threshold 0.75
```

Finally, the importance of each of the elements of the hybrid scoring function can be tuned (*–cnn_weight*), as well as the scoring function used for the coordination analysis (*–residue_score*):

```
$ brigit 1dhy Fe --cnn_weight 0.3 --residue_score gaussian
```

3.3 Interpretation of results

The program generates 2 files with names: outputfile_brigit.pdb and outputfile_clusters.txt. The first file will contain information about the probes used to locate possible binding sites and/or metal migration pathways. It should be visualised with a protein/molecule visualizater like UCSF Chimera. To visualize the scores of each probe, select colour by property -> b-factor. The _clusters.txt file contains information regarding the residues closer than *cluster_radius* to the center of the cluster. It covers all possible coordinations that the cluster could experience.

3.4 Integration in other projects

To use Brigit in a project:

```
import brigit
predictor = brigit.Brigit()
Brigit.run(*args)
```

CHAPTER

FOUR

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:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at https://github.com/RaulFD-creator/brigit/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.

4.1.2 Fix Bugs

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

4.1.3 Implement Features

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

4.1.4 Write Documentation

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

4.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/RaulFD-creator/brigit/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 :)

4.2 Get Started!

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

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

```
$ git clone git@github.com:your_name_here/brigit.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 brigit
$ cd brigit/
$ 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 brigit tests
$ python setup.py test or pytest
$ 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.

4.3 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 3.5, 3.6, 3.7 and 3.8, and for PyPy. Check https://travis-ci.com/RaulFD-creator/brigit/pull_requests and make sure that the tests pass for all supported Python versions.

4.4 Tips

To run a subset of tests:

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

4.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bump2version patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

CHAPTER

FIVE

CREDITS

5.1 Development Lead

- Raúl Fernández Díaz <Raul.FernandezDi@autonoma.cat>

5.2 Contributors

None yet. Why not be the first?

CHAPTER

SIX

HISTORY

CHAPTER

SEVEN

INDICES AND TABLES

- genindex
- modindex
- search