ProLIF

Cédric Bouysset

Feb 02, 2021

USER GUIDE

CHAPTER

ONE

PROLIF

Documentation	
Tutorials	
CI	
PyPI	
License	

1.1 Description

ProLIF (*Protein-Ligand Interaction Fingerprints*) is a tool designed to generate interaction fingerprints for protein-ligand and protein-protein interactions extracted from molecular dynamics trajectories and docking simulations. It also supports DNA-ligand, DNA-protein and DNA-DNA interactions.

You can try it out prior to any installation on Binder.

1.2 Documentation

The installation instructions, documentation and tutorials can be found online on ReadTheDocs.

1.3 Issues

If you have found a bug, please open an issue on the GitHub Issues page.

1.4 Discussion

If you have questions on how to use ProLIF, or if you want to give feedback or share ideas and new features, please head to the GitHub Discussions page.

1.5 Citing ProLIF

Please refer to the citation page on the documentation.

1.6 License

Unless otherwise noted, all files in this directory and all subdirectories are distributed under the Apache License, Version 2.0

```
Copyright 2017-2021 Cédric BOUYSSET

Licensed under the Apache License, Version 2.0 (the "License");
you may not use this file except in compliance with the License.
You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
```

1.6.1 Installation

Requirements:

- Python 3.6+
- RDKit (2020.03+)
- MDAnalysis (2.0+)
- Pandas (1.0+)
- NumPy
- SciPy
- tqdm

The simplest way to install ProLIF dependencies is to use conda:

```
# create a separate virtual environment
conda create -n prolif
# activate it
conda activate prolif
# install main dependencies
conda config --add channels conda-forge
conda install rdkit cython
```

We strongly encourage users to install ProLIF in a separate virtual environment, as it currently (and temporarily) depends on a custom fork of MDAnalysis.

The rest of the dependencies are automatically installed through pip when installing prolif:

```
pip install git+https://github.com/chemosim-lab/ProLIF.git
```

Alternatively, you can install a specific release version as follow:

```
pip install https://github.com/chemosim-lab/ProLIF/archive/v0.3.0.zip
```

Note: Until MDAnalysis version 2.0.0 is out, ProLIF can only be installed through our GitHub repository. Once MDAnalysis v2.0.0 is out, it will be made available as a standard PyPI package and installable with `pip install prolif`.

1.6.2 Citation

If you use ProLIF in your research, please cite the DOI corresponding to the release version you've used:

Table 1: ProLIF releases

Version	Release date	DOI
v0.3.0	2020-12-23	

1.6.3 Changelog

All notable changes to this project will be documented in this file.

The format is based on Keep a Changelog, and this project adheres to Semantic Versioning.

[Unreleased]

Added

- Integration with Zenodo to automatically generate a DOI for new releases
- Citation page
- Docking section in the Quickstart notebook (Issue #11)
- PDBQT, MOL2 and SDF molecule suppliers to make it easier for users to use docking results as input (Issue #11)
- `Molecule.from_rdkit` classmethod to easily prepare RDKit molecules for ProLIF

Changed

- The visualisation notebook now displays the protein with py3Dmol. Some examples for creating and displaying a graph from the interaction dataframe have been added
- Updated the installation instructions to show how to install a specific release
- The previous repr method of `ResidueId` was easy to confuse with a string, especially when trying to access the `Fingerprint.ifp` results by string. The new repr method is now more explicit.
- Added the `Fingerprint.run_from_iterable` method, which uses the new supplier functions to quickly generate a fingerprint.

- Sorted the output of `Fingerprint.list_available`

Deprecated

Removed

Fixed

- `Fingerprint.to_dataframe` is now much faster (Issue #7)
- `ResidueId.from_string` method now supports 1-letter and 2-letter codes for RNA/DNA (Issue #8)

[0.3.0] - 2020-12-23

Added

- Reading input directly from RDKit Mol as well as MDAnalysis AtomGroup objects
- Proper documentation and tests
- CI through GitHub Actions
- Publishing to PyPI triggered by GitHub releases

Changed

- All the API and the underlying code have been modified
- Repository has been moved from GitHub user @cbouy to organisation @chemosim-lab

Deprecated

Removed

- Custom MOL2 file reader
- Command-line interface

Fixed

- Interactions not detected properly

[0.2.1] - 2019-10-02

Base version for this changelog

1.6.4 Quickstart

This is a very short guide on how to use ProLIF to generate an interaction fingerprint for a ligand-protein complex from an MD simulation.

Let's start by importing MDAnalysis and ProLIF to read our input files:

[1]:
```
import MDAnalysis as mda
import prolif as plf
# load trajectory
u = mda.Universe(plf.datafiles.TOP, plf.datafiles.TRAJ)
# create selections for the ligand and protein
lig = u.atoms.select_atoms("resname LIG")
prot = u.atoms.select_atoms("protein")
lig, prot
```

[1]:
```
(<AtomGroup with 79 atoms>, <AtomGroup with 4988 atoms>)
```

MDAnalysis should automatically recognize the file type that you're using from its extension. Click here to learn more about loading files with MDAnalysis, and here to learn more about their atom selection language.

Next, lets make sure that our ligand was correctly read by MDAnalysis.

This next step is crucial if you're loading a structure from a file that doesn't explicitely contain bond orders and formal charges. MDAnalysis will infer those from the atoms connectivity, which requires all atoms including hydrogens to be present in the input file.

ProLIF molecules are built on top of RDKit and are compatible with its drawing code. Let's have a quick look at our ligand:

[2]:
```
from rdkit import Chem
from rdkit.Chem import Draw
# create a molecule from the MDAnalysis selection
lmol = plf.Molecule.from_mda(lig)
# cleanup before drawing
mol = Chem.RemoveHs(lmol)
mol.RemoveAllConformers()
Draw.MolToImage(mol, size=(400,200))
```

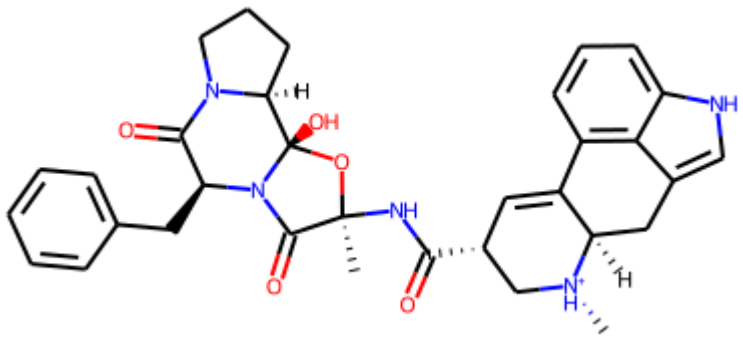
```
/home/docs/checkouts/readthedocs.org/user_builds/prolif/conda/v0.3.1/lib/python3.9/
↪site-packages/MDAnalysis/coordinates/RDKit.py:492: DeprecationWarning: `np.float`␣
↪is a deprecated alias for the builtin `float`. To silence this warning, use `float`␣
↪by itself. Doing this will not modify any behavior and is safe. If you specifically␣
↪wanted the numpy scalar type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/
↪release/1.20.0-notes.html#deprecations
  if isinstance(value, (float, np.float)):
/home/docs/checkouts/readthedocs.org/user_builds/prolif/conda/v0.3.1/lib/python3.9/
↪site-packages/MDAnalysis/coordinates/RDKit.py:494: DeprecationWarning: `np.int` is␣
↪a deprecated alias for the builtin `int`. To silence this warning, use `int` by␣
↪itself. Doing this will not modify any behavior and is safe. When replacing `np.
↪int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the precision.␣
↪If you wish to review your current use, check the release note link for additional␣
↪information.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/
↪release/1.20.0-notes.html#deprecations
```

(continues on next page)

(continued from previous page)

```
  elif isinstance(value, (int, np.int)):
```

[2]:

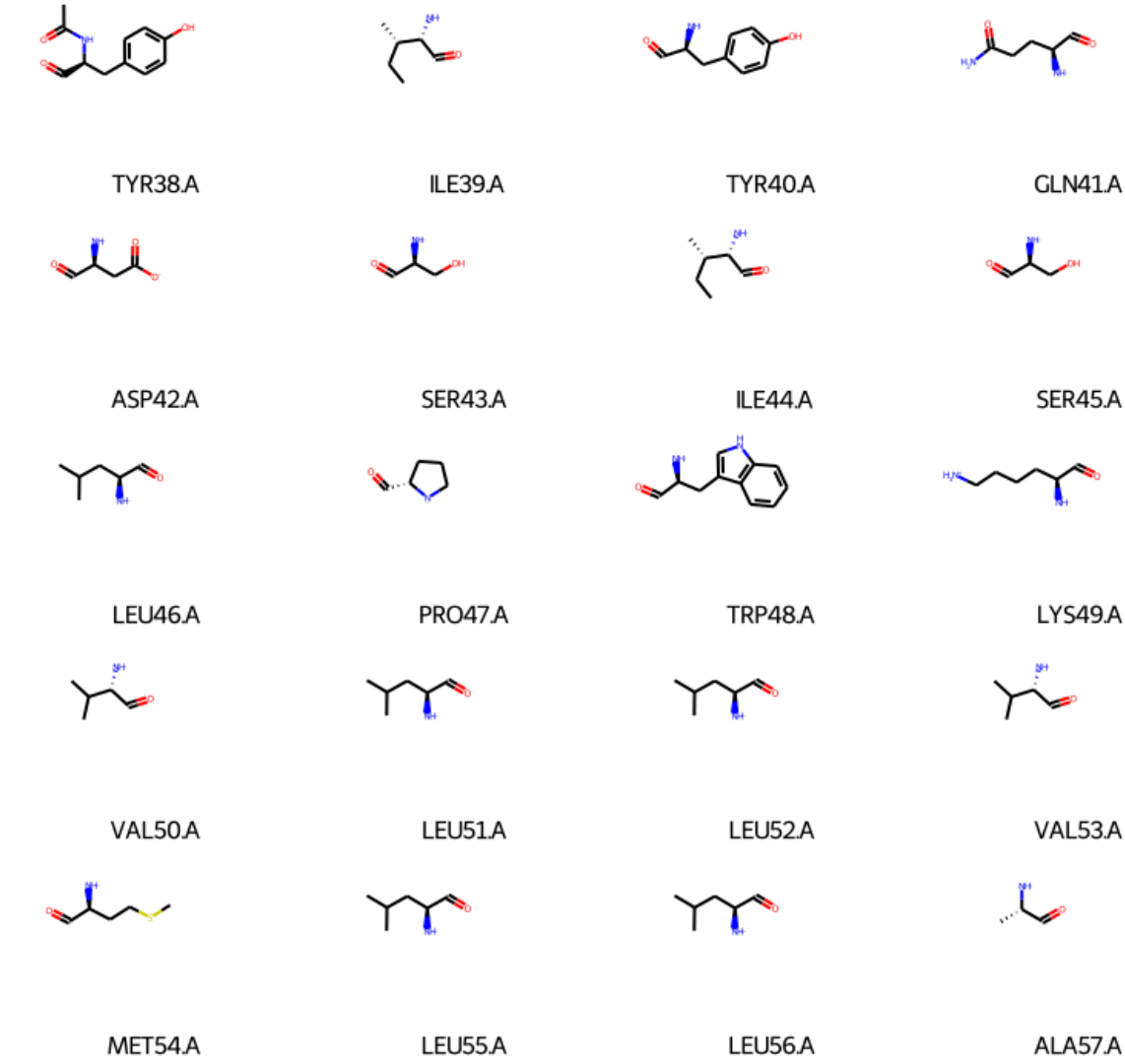


We can do the same for the residues in the protein (I'll only show the first 20 to keep the notebook short):

[3]:

```
pmol = plf.Molecule.from_mda(prot)
frags = []
# to show all residues, simply use `for res in pmol:`
for i in range(20):
    res = pmol[i]
    mol = Chem.RemoveHs(res)
    mol.RemoveAllConformers()
    frags.append(mol)
Draw.MolsToGridImage(frags,
                     legends=[str(res.resid) for res in pmol],
                     subImgSize=(200, 140),
                     molsPerRow=4,
                     maxMols=prot.n_residues)
```

[3]:



Everything looks good, we can now compute a fingerprint:

[4]:
```
# use default interactions
fp = plf.Fingerprint()
# run on a slice of frames from begining to end with a step of 10
fp.run(u.trajectory[::10], lig, prot)
```

```
  0%|          | 0/25 [00:00<?, ?it/s]
```

[4]:
```
<prolif.fingerprint.Fingerprint: 8 interactions: ['Hydrophobic', 'HBDonor',
↪'HBAcceptor', 'Cationic', 'Anionic', 'PiCation', 'CationPi', 'PiStacking'] at␣
↪0x7f20eb1be940>
```

The `run` method will automatically select residues that are close to the ligand (6.0 Å) when computing the fingerprint. Alternatively, you can pass a list of residues like so:

```
fp.run(..., residues=["TYR38.A", "ASP129.A"])
```

Or simply use `fp.run(..., residues="all")` to use all residues in the `prot` selection.

The best way to access our results is to export our interaction fingerprints to a Pandas DataFrame. By default, the

resulting DataFrame only keeps track of residues and interaction types that were seen in at least one of the frames in your trajectory. If needed, you can access the full results with `fp.to_dataframe(drop_empty=False)`.

[5]:
```
df = fp.to_dataframe()
# show only the 10 first frames
df.head(10)
```

[5]:
```
ligand           LIG1.G                                                  \
protein         TYR38.A    TYR109.A    THR110.A    TRP125.A    LEU126.A
interaction Hydrophobic Hydrophobic Hydrophobic Hydrophobic Hydrophobic
Frame
0                 False        True       False        True        True
10                False        True       False        True        True
20                False        True        True        True        True
30                 True        True        True        True        True
40                False        True        True        True        True
50                 True        True       False       False        True
60                False        True       False       False        True
70                False        True       False        True        True
80                False        True       False        True        True
90                 True       False       False        True       False

ligand                                                             ...  \
protein        ASP129.A                      ILE130.A    CYS133.A  ...
interaction Hydrophobic HBDonor Cationic Hydrophobic Hydrophobic  ...
Frame                                                              ...
0                  True    True     True        True        True  ...
10                 True    True     True        True        True  ...
20                 True    True     True        True        True  ...
30                 True    True     True        True        True  ...
40                 True    True     True        True        True  ...
50                 True    True     True        True        True  ...
60                 True    True     True        True        True  ...
70                 True    True     True        True        True  ...
80                 True    True     True        True       False  ...
90                 True    True     True        True        True  ...

ligand                                                                  \
protein        MET337.B    PRO338.B    ALA343.B    CYS344.B    LEU348.B
interaction Hydrophobic Hydrophobic Hydrophobic Hydrophobic Hydrophobic
Frame
0                  True        True       False       False       False
10                 True       False        True       False       False
20                 True       False        True       False       False
30                 True       False        True        True       False
40                False       False       False       False       False
50                 True       False       False       False       False
60                False       False       False       False       False
70                False       False       False       False        True
80                 True       False       False       False        True
90                False       False       False       False        True

ligand
protein        PHE351.B               ASP352.B    THR355.B    TYR359.B
interaction Hydrophobic PiStacking Hydrophobic Hydrophobic Hydrophobic
Frame
0                  True      False        True        True        True
10                 True       True       False        True        True
```

(continues on next page)

(continued from previous page)

```
20                 True      False       False       False        True
30                 True       True       False        True        True
40                 True       True        True       False        True
50                 True       True       False        True        True
60                 True      False        True        True        True
70                 True      False        True        True        True
80                 True      False        True        True        True
90                 True       True        True        True        True

[10 rows x 40 columns]
```

[6]:
```
# drop the ligand residue column since there's only a single ligand residue
df = df.droplevel("ligand", axis=1)
df.head(5)
```

[6]:
```
protein          TYR38.A    TYR109.A    THR110.A    TRP125.A    LEU126.A  \
interaction Hydrophobic Hydrophobic Hydrophobic Hydrophobic Hydrophobic
Frame
0                  False        True       False        True        True
10                 False        True       False        True        True
20                 False        True        True        True        True
30                  True        True        True        True        True
40                 False        True        True        True        True

protein         ASP129.A                     ILE130.A    CYS133.A  ...  \
interaction Hydrophobic HBDonor Cationic Hydrophobic Hydrophobic  ...
Frame                                                             ...
0                  True    True     True        True        True  ...
10                 True    True     True        True        True  ...
20                 True    True     True        True        True  ...
30                 True    True     True        True        True  ...
40                 True    True     True        True        True  ...

protein         MET337.B    PRO338.B    ALA343.B    CYS344.B    LEU348.B  \
interaction Hydrophobic Hydrophobic Hydrophobic Hydrophobic Hydrophobic
Frame
0                  True        True       False       False       False
10                 True       False        True       False       False
20                 True       False        True       False       False
30                 True       False        True        True       False
40                False       False       False       False       False

protein         PHE351.B               ASP352.B    THR355.B    TYR359.B
interaction Hydrophobic PiStacking Hydrophobic Hydrophobic Hydrophobic
Frame
0                  True      False        True        True        True
10                 True       True       False        True        True
20                 True      False       False       False        True
30                 True       True       False        True        True
40                 True       True        True       False        True

[5 rows x 40 columns]
```

[7]:
```
# show all pi-stacking interactions
df.xs("PiStacking", level="interaction", axis=1).head(5)
```

[7]:
```
protein  PHE330.B  PHE331.B  PHE351.B
Frame
0           False      True     False
10           True      True      True
20          False      True     False
30          False     False      True
40          False      True      True
```

[8]:
```
# show all interactions with a specific protein residue
df.xs("ASP129.A", level="protein", axis=1).head(5)
# or more simply
df["ASP129.A"].head(5)
```

[8]:
```
interaction  Hydrophobic  HBDonor  Cationic
Frame
0                   True     True      True
10                  True     True      True
20                  True     True      True
30                  True     True      True
40                  True     True      True
```

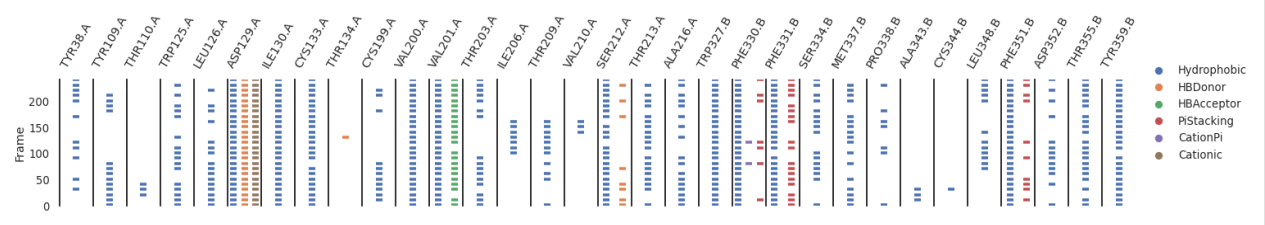
Here's a simple example to plot the interactions over time

[9]:
```
import seaborn as sns
import pandas as pd

# reorganize data
data = df.reset_index()
data = pd.melt(data, id_vars=["Frame"], var_name=["residue","interaction"])
data = data[data["value"] != False]
data.reset_index(inplace=True, drop=True)

# plot
sns.set_theme(font_scale=.8, style="white", context="talk")
g = sns.catplot(
    data=data, x="interaction", y="Frame", hue="interaction", col="residue",
    hue_order=["Hydrophobic", "HBDonor", "HBAcceptor", "PiStacking", "CationPi",
↪"Cationic"],
    height=3, aspect=0.2, jitter=0, sharex=False, marker="_", s=8, linewidth=3.5,
)
g.set_titles("{col_name}")
g.set(xticks=[], ylim=(-.5, data.Frame.max()+1))
g.set_xticklabels([])
g.set_xlabels("")
g.fig.subplots_adjust(wspace=0)
g.add_legend()
g.despine(bottom=True)
for ax in g.axes.flat:
    ax.invert_yaxis()
    ax.set_title(ax.get_title(), pad=15, rotation=60, ha="center", va="baseline")
```

```
/home/docs/checkouts/readthedocs.org/user_builds/prolif/conda/v0.3.1/lib/python3.9/
↪site-packages/seaborn/axisgrid.py:64: UserWarning: Tight layout not applied. tight_
↪layout cannot make axes width small enough to accommodate all axes decorations
  self.fig.tight_layout(*args, **kwargs)
```



[10]:
```
# calculate the occurence of each interaction on the trajectory
occ = df.mean()
# restrict to the frequent ones
occ.loc[occ > 0.3]
```

[10]:
```
protein   interaction
TYR38.A   Hydrophobic    0.44
TYR109.A  Hydrophobic    0.52
TRP125.A  Hydrophobic    0.64
LEU126.A  Hydrophobic    0.64
ASP129.A  Hydrophobic    1.00
          HBDonor        1.00
          Cationic       1.00
ILE130.A  Hydrophobic    1.00
CYS133.A  Hydrophobic    0.96
CYS199.A  Hydrophobic    0.44
VAL200.A  Hydrophobic    1.00
VAL201.A  Hydrophobic    1.00
          HBAcceptor     0.92
THR203.A  Hydrophobic    0.64
THR209.A  Hydrophobic    0.44
SER212.A  Hydrophobic    0.92
          HBDonor        0.32
THR213.A  Hydrophobic    0.76
ALA216.A  Hydrophobic    0.68
TRP327.B  Hydrophobic    1.00
PHE330.B  Hydrophobic    1.00
PHE331.B  Hydrophobic    0.96
          PiStacking     0.72
SER334.B  Hydrophobic    0.64
MET337.B  Hydrophobic    0.68
LEU348.B  Hydrophobic    0.48
PHE351.B  Hydrophobic    1.00
          PiStacking     0.40
ASP352.B  Hydrophobic    0.52
THR355.B  Hydrophobic    0.84
TYR359.B  Hydrophobic    0.96
dtype: float64
```

[11]:
```
# regroup all interactions together and do the same
g = (df.groupby(level=["protein"], axis=1)
       .sum()
       .astype(bool)
       .mean())
g.loc[g > 0.3]
```

[11]:
```
protein
ALA216.A    0.68
ASP129.A    1.00
```

(continues on next page)

(continued from previous page)

```
ASP352.B    0.52
CYS133.A    0.96
CYS199.A    0.44
ILE130.A    1.00
LEU126.A    0.64
LEU348.B    0.48
MET337.B    0.68
PHE330.B    1.00
PHE331.B    0.96
PHE351.B    1.00
SER212.A    0.92
SER334.B    0.64
THR203.A    0.64
THR209.A    0.44
THR213.A    0.76
THR355.B    0.84
TRP125.A    0.64
TRP327.B    1.00
TYR109.A    0.52
TYR359.B    0.96
TYR38.A     0.44
VAL200.A    1.00
VAL201.A    1.00
dtype: float64
```

You can also compute a Tanimoto similarity between each frame:

[12]:
```
from rdkit import DataStructs
bvs = fp.to_bitvectors()
tanimoto_sims = DataStructs.BulkTanimotoSimilarity(bvs[0], bvs)
tanimoto_sims
```

[12]:
```
[1.0,
 0.71875,
 0.6451612903225806,
 0.6470588235294118,
 0.7419354838709677,
 0.7741935483870968,
 0.8275862068965517,
 0.8,
 0.78125,
 0.6774193548387096,
 0.6666666666666666,
 0.65625,
 0.5882352941176471,
 0.6333333333333333,
 0.6129032258064516,
 0.7,
 0.7,
 0.8275862068965517,
 0.7241379310344828,
 0.75,
 0.71875,
 0.696969696969697,
 0.6451612903225806,
 0.7741935483870968,
 0.625]
```

1.6.5 How-to

This notebook serves as a practical guide to common questions users might have.

Table of content

[1]:
```
import MDAnalysis as mda
import prolif as plf
import pandas as pd
import numpy as np
```

[2]:
```
u = mda.Universe(plf.datafiles.TOP, plf.datafiles.TRAJ)
lig = u.select_atoms("resname LIG")
prot = u.select_atoms("protein")
lmol = plf.Molecule.from_mda(lig)
pmol = plf.Molecule.from_mda(prot)
```

```
/home/docs/checkouts/readthedocs.org/user_builds/prolif/conda/v0.3.1/lib/python3.9/
↪site-packages/MDAnalysis/coordinates/RDKit.py:492: DeprecationWarning: `np.float`␣
↪is a deprecated alias for the builtin `float`. To silence this warning, use `float`␣
↪by itself. Doing this will not modify any behavior and is safe. If you specifically␣
↪wanted the numpy scalar type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/
↪release/1.20.0-notes.html#deprecations
  if isinstance(value, (float, np.float)):
/home/docs/checkouts/readthedocs.org/user_builds/prolif/conda/v0.3.1/lib/python3.9/
↪site-packages/MDAnalysis/coordinates/RDKit.py:494: DeprecationWarning: `np.int` is␣
↪a deprecated alias for the builtin `int`. To silence this warning, use `int` by␣
↪itself. Doing this will not modify any behavior and is safe. When replacing `np.
↪int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the precision.␣
↪If you wish to review your current use, check the release note link for additional␣
↪information.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/
↪release/1.20.0-notes.html#deprecations
  elif isinstance(value, (int, np.int)):
```

Changing the parameters for an interaction

You can list all the available interactions as follow:

[3]:
```
plf.Fingerprint.list_available(show_hidden=True)
```

[3]:
```
['Anionic',
 'CationPi',
 'Cationic',
 'EdgeToFace',
 'FaceToFace',
 'HBAcceptor',
 'HBDonor',
 'Hydrophobic',
 'Interaction',
```

(continues on next page)

(continued from previous page)

```
 'MetalAcceptor',
 'MetalDonor',
 'PiCation',
 'PiStacking',
 'XBAcceptor',
 'XBDonor',
 '_BaseCationPi',
 '_BaseHBond',
 '_BaseIonic',
 '_BaseMetallic',
 '_BaseXBond',
 '_Distance']
```

In this example, we'll redefine the hydrophobic interaction with a shorter distance.

You have the choice between overwriting the original hydrophobic interaction with the new one, or giving it an original name.

Let's start with a test case: with the default parameters, Y109 is interacting with our ligand.

[4]:
```
fp = plf.Fingerprint()
fp.hydrophobic(lmol, pmol["TYR109.A"])
```

[4]:
```
True
```

Overwriting the original interaction

You have to define a class that inherits one of the classes listed in the `prolif.interactions` module.

[5]:
```
class Hydrophobic(plf.interactions.Hydrophobic):
    def __init__(self):
        super().__init__(distance=4.0)
```

```
/home/docs/checkouts/readthedocs.org/user_builds/prolif/conda/v0.3.1/lib/python3.9/
↪site-packages/prolif/interactions.py:55: UserWarning: The 'Hydrophobic' interaction␣
↪has been superseded by a new class with id 0x55a1ec27d3d0
  warnings.warn(f"The {name!r} interaction has been superseded by a "
```

[6]:
```
fp = plf.Fingerprint()
fp.hydrophobic(lmol, pmol["TYR109.A"])
```

[6]:
```
False
```

The interaction is not detected anymore. You can reset to the default interaction like so:

[7]:
```
class Hydrophobic(plf.interactions.Hydrophobic):
    pass

fp = plf.Fingerprint()
fp.hydrophobic(lmol, pmol["TYR109.A"])
```

```
/home/docs/checkouts/readthedocs.org/user_builds/prolif/conda/v0.3.1/lib/python3.9/
↪site-packages/prolif/interactions.py:55: UserWarning: The 'Hydrophobic' interaction␣
↪has been superseded by a new class with id 0x55a1eba8e6c0
  warnings.warn(f"The {name!r} interaction has been superseded by a "
```

[7]:
```
True
```

Reparameterizing an interaction with another name

The steps are identical to above, just give the class a different name:

```
[8]: class CustomHydrophobic(plf.interactions.Hydrophobic):
         def __init__(self):
             super().__init__(distance=4.0)

     fp = plf.Fingerprint()
     fp.hydrophobic(lmol, pmol["TYR109.A"])
```

```
[8]: True
```

```
[9]: fp.customhydrophobic(lmol, pmol["TYR109.A"])
```

```
[9]: False
```

```
[10]: fp = plf.Fingerprint(["Hydrophobic", "CustomHydrophobic"])
      fp.bitvector(lmol, pmol["TYR109.A"])
```

```
[10]: array([ True, False])
```

Writing your own interaction

Before you dive into this section, make sure that there isn't already an interaction that could just be **reparameterized** to do what you want!

For this, the best is to check the section of the documentation corresponding to the `prolif.interactions` module. There are some generic interactions, like the `_Distance` class, if you just need to define two chemical moieties within a certain distance. Both the Hydrophobic, Ionic, and Metallic interactions inherit from this class!

With that being said, there are a few rules that you must respect when writing your own interaction:

- **Inherit the ProLIF Interaction class**

This class is located in `prolif.interactions.Interaction`. If for any reason you must inherit from another class, you can also define the `prolif.interactions._InteractionMeta` as a metaclass.

- **Naming convention**

Your class name must not start with _ or be named `Interaction`. For non-symmetrical interactions, like hydrogen bonds or salt-bridges, the convention used here is to named the class after the function of the ligand. For example, the class `HBDonor` detects if a ligand acts as a hydrogen bond donor, and the class `Cationic` detects if a ligand acts as a cation.

- **Define a ``detect`` method**

This method takes exactly two positional arguments (and as many named arguments as you need): a ligand Residue or Molecule and a protein Residue or Molecule (in this order).

- **Return value(s) for the ``detect`` method**

There are two possibilities here, depending on whether or not you want to access the indices of atoms responsible for the interaction. If you don't need this information, just return `True` if the interaction is detected, `False` otherwise. If you need to access atomic indices, you must return the following items in this order:

- `True` or `False` for the detection of the interaction
- The index of the ligand atom, or None if not detected
- The index of the protein atom, or None if not detected

[11]:
```
from scipy.spatial import distance_matrix

# without atom indices
class CloseContact(plf.interactions.Interaction):
    def detect(self, res1, res2, threshold=2.0):
        dist_matrix = distance_matrix(res1.xyz, res2.xyz)
        if (dist_matrix <= threshold).any():
            return True
        return False

fp = plf.Fingerprint()
fp.closecontact(lmol, pmol["ASP129.A"])
```

[11]:
```
True
```

[12]:
```
# with atom indices
class CloseContact(plf.interactions.Interaction):
    def detect(self, res1, res2, threshold=2.0):
        dist_matrix = distance_matrix(res1.xyz, res2.xyz)
        mask = dist_matrix <= threshold
        if mask.any():
            res1_i, res2_i = np.where(mask)
            # return the first solution
            return True, res1_i[0], res2_i[0]
        return False, None, None

fp = plf.Fingerprint()
fp.closecontact(lmol, pmol["ASP129.A"])
```

```
/home/docs/checkouts/readthedocs.org/user_builds/prolif/conda/v0.3.1/lib/python3.9/
↪site-packages/prolif/interactions.py:55: UserWarning: The 'CloseContact'␣
↪interaction has been superseded by a new class with id 0x55a1eb4e33d0
  warnings.warn(f"The {name!r} interaction has been superseded by a "
```

[12]:
```
True
```

By default, the fingerprint will modify all interaction classes to only return the boolean value. To get the atom indices when using your custom class, you must choose one of the following options:

- Use the `__wrapped__` argument when calling the class as a fingerprint method:

[13]:
```
fp.closecontact.__wrapped__(lmol, pmol["ASP129.A"])
```

[13]:
```
(True, 52, 10)
```

- Use the `bitvector_atoms` method instead of `bitvector`:

[14]:
```
fp = plf.Fingerprint(["CloseContact"])
bv, indices = fp.bitvector_atoms(lmol, pmol["ASP129.A"])
bv, indices
```

[14]:
```
(array([ True]), [[52, 10]])
```

- Use the `return_atoms=True` argument when calling the `run` method:

[15]:
```
fp.run(u.trajectory[0:1], lig, prot, return_atoms=True)
fp.ifp
```

```
  0%|          | 0/1 [00:00<?, ?it/s]
```

[15]:
```
[{'Frame': 0,
  (ResidueId(LIG, 1, G), ResidueId(TYR, 359, B)): [[None, None]],
  (ResidueId(LIG, 1, G), ResidueId(TRP, 327, B)): [[None, None]],
  (ResidueId(LIG, 1, G), ResidueId(ILE, 130, A)): [[None, None]],
  (ResidueId(LIG, 1, G), ResidueId(GLY, 358, B)): [[None, None]],
  (ResidueId(LIG, 1, G), ResidueId(PHE, 330, B)): [[None, None]],
  (ResidueId(LIG, 1, G), ResidueId(SER, 106, A)): [[None, None]],
  (ResidueId(LIG, 1, G), ResidueId(ILE, 350, B)): [[None, None]],
  (ResidueId(LIG, 1, G), ResidueId(THR, 110, A)): [[None, None]],
  (ResidueId(LIG, 1, G), ResidueId(THR, 209, A)): [[None, None]],
  (ResidueId(LIG, 1, G), ResidueId(THR, 134, A)): [[None, None]],
  (ResidueId(LIG, 1, G), ResidueId(MET, 337, B)): [[None, None]],
  (ResidueId(LIG, 1, G), ResidueId(ALA, 349, B)): [[None, None]],
  (ResidueId(LIG, 1, G), ResidueId(GLU, 198, A)): [[None, None]],
  (ResidueId(LIG, 1, G), ResidueId(ASP, 352, B)): [[None, None]],
  (ResidueId(LIG, 1, G), ResidueId(PHE, 354, B)): [[None, None]],
  (ResidueId(LIG, 1, G), ResidueId(ILE, 180, A)): [[None, None]],
  (ResidueId(LIG, 1, G), ResidueId(TRP, 125, A)): [[None, None]],
  (ResidueId(LIG, 1, G), ResidueId(THR, 213, A)): [[None, None]],
  (ResidueId(LIG, 1, G), ResidueId(TYR, 211, A)): [[None, None]],
  (ResidueId(LIG, 1, G), ResidueId(PRO, 338, B)): [[None, None]],
  (ResidueId(LIG, 1, G), ResidueId(VAL, 210, A)): [[None, None]],
  (ResidueId(LIG, 1, G), ResidueId(VAL, 102, A)): [[None, None]],
  (ResidueId(LIG, 1, G), ResidueId(VAL, 201, A)): [[28, 6]],
  (ResidueId(LIG, 1, G), ResidueId(SER, 334, B)): [[None, None]],
  (ResidueId(LIG, 1, G), ResidueId(TRP, 115, A)): [[None, None]],
  (ResidueId(LIG, 1, G), ResidueId(ASP, 129, A)): [[52, 10]],
  (ResidueId(LIG, 1, G), ResidueId(PHE, 217, A)): [[None, None]],
  (ResidueId(LIG, 1, G), ResidueId(TYR, 208, A)): [[None, None]],
  (ResidueId(LIG, 1, G), ResidueId(TYR, 38, A)): [[None, None]],
  (ResidueId(LIG, 1, G), ResidueId(PHE, 351, B)): [[None, None]],
  (ResidueId(LIG, 1, G), ResidueId(VAL, 214, A)): [[None, None]],
  (ResidueId(LIG, 1, G), ResidueId(THR, 131, A)): [[None, None]],
  (ResidueId(LIG, 1, G), ResidueId(LEU, 126, A)): [[None, None]],
  (ResidueId(LIG, 1, G), ResidueId(ALA, 216, A)): [[None, None]],
  (ResidueId(LIG, 1, G), ResidueId(TYR, 109, A)): [[None, None]],
  (ResidueId(LIG, 1, G), ResidueId(ILE, 333, B)): [[None, None]],
  (ResidueId(LIG, 1, G), ResidueId(GLN, 41, A)): [[None, None]],
  (ResidueId(LIG, 1, G), ResidueId(LEU, 335, B)): [[None, None]],
  (ResidueId(LIG, 1, G), ResidueId(PHE, 331, B)): [[None, None]],
  (ResidueId(LIG, 1, G), ResidueId(PHE, 353, B)): [[None, None]],
  (ResidueId(LIG, 1, G), ResidueId(ILE, 137, A)): [[None, None]],
  (ResidueId(LIG, 1, G), ResidueId(LEU, 348, B)): [[None, None]],
  (ResidueId(LIG, 1, G), ResidueId(TRP, 356, B)): [[None, None]],
  (ResidueId(LIG, 1, G), ResidueId(GLY, 215, A)): [[None, None]],
  (ResidueId(LIG, 1, G), ResidueId(THR, 203, A)): [[None, None]],
  (ResidueId(LIG, 1, G), ResidueId(VAL, 200, A)): [[None, None]],
  (ResidueId(LIG, 1, G), ResidueId(SER, 212, A)): [[None, None]],
  (ResidueId(LIG, 1, G), ResidueId(CYS, 133, A)): [[None, None]],
  (ResidueId(LIG, 1, G), ResidueId(ASN, 202, A)): [[None, None]],
  (ResidueId(LIG, 1, G), ResidueId(CYS, 199, A)): [[None, None]],
  (ResidueId(LIG, 1, G), ResidueId(THR, 355, B)): [[None, None]],
  (ResidueId(LIG, 1, G), ResidueId(TYR, 40, A)): [[None, None]]}]
```

- Directly use your class:

[16]:
```
cc = CloseContact()
cc.detect(lmol, pmol["ASP129.A"])
```

[16]:
```
(True, 52, 10)
```

Working with docking poses instead of MD simulations

ProLIF currently provides file readers for MOL2, SDF and PDBQT files. The API is slightly different compared to the quickstart example but the end result is the same.

Please note that this part of the tutorial is only suitable for interactions between one protein and several ligands, or in more general terms, between one molecule with multiple residues and one molecule with a single residue. This is not suitable for protein-protein or DNA-protein interactions.

Let's start by loading the protein. Here I'm using a PDB file but you can use any format supported by MDAnalysis as long as it contains explicit hydrogens.

[17]:
```
# load protein
prot = mda.Universe(plf.datafiles.datapath / "vina" / "rec.pdb")
prot = plf.Molecule.from_mda(prot)
prot.n_residues
```

[17]:
```
302
```

Using an SDF file

[18]:
```
# load ligands
path = str(plf.datafiles.datapath / "vina" / "vina_output.sdf")
lig_suppl = list(plf.sdf_supplier(path))
# generate fingerprint
fp = plf.Fingerprint()
fp.run_from_iterable(lig_suppl, prot)
df = fp.to_dataframe()
df
```

```
  0%|          | 0/9 [00:00<?, ?it/s]
```

[18]:
```
ligand             UNL1                                                \
protein          TYR38.A                TYR40.A    SER106.A    TYR109.A
interaction Hydrophobic HBAcceptor Hydrophobic Hydrophobic Hydrophobic
Frame
0                  False      False       False       False        True
1                  False      False       False       False        True
2                  False      False       False       False        True
3                   True      False       False       False       False
4                   True       True       False       False        True
5                  False      False       False       False        True
6                   True      False        True        True        True
7                   True      False       False       False        True
8                  False      False       False       False       False

ligand                                                              ...  \
protein                    CYS122.A    ASP123.A    TRP125.A             ...
interaction PiStacking Hydrophobic Hydrophobic Hydrophobic PiStacking  ...
Frame                                                               ...
0                 False       False       False       False      False  ...
1                 False       False       False        True      False  ...
2                 False       False       False        True      False  ...
```

(continues on next page)

(continued from previous page)

```
3                False       False       False       False      False  ...
4                False       False       False       False      False  ...
5                False       False       False        True      False  ...
6                 True       False       False        True       True  ...
7                False       False       False        True      False  ...
8                False        True        True       False      False  ...

ligand                                                                    \
protein        PRO338.B    PHE346.B    HSE347.B    LEU348.B    PHE351.B
interaction Hydrophobic Hydrophobic Hydrophobic Hydrophobic Hydrophobic
Frame
0                 False       False       False       False        True
1                 False       False       False       False        True
2                 False       False       False       False        True
3                 False       False       False       False        True
4                  True        True        True        True        True
5                 False       False       False       False        True
6                 False       False       False        True        True
7                  True       False        True        True        True
8                 False       False       False       False       False

ligand
protein                   ASP352.B    THR355.B    TYR359.B
interaction PiStacking Hydrophobic Hydrophobic Hydrophobic PiStacking
Frame
0                 True        True        True       False      False
1                 True        True        True       False      False
2                False        True        True       False      False
3                False        True        True        True      False
4                 True        True       False       False      False
5                False        True        True       False      False
6                 True       False        True        True       True
7                 True       False        True       False      False
8                False       False       False       False      False

[9 rows x 47 columns]
```

Please note that converting the `lig_suppl` to a list is optionnal (and maybe not suitable for large files) as it will load all the ligands in memory, but it's nicer to track the progression with the progress bar.

If you want to calculate the Tanimoto similarity between your docked poses and a reference ligand, here's how to do it.

We first need to generate the interaction fingerprint for the reference, and concatenate it to the previous one

[19]:
```
# load the reference
ref = mda.Universe(plf.datafiles.datapath / "vina" / "lig.pdb")
ref = plf.Molecule.from_mda(ref)
# generate IFP
fp.run_from_iterable([ref], prot)
df0 = fp.to_dataframe()
df0.rename({0: "ref"}, inplace=True)
# drop the ligand level on both dataframes
df0.columns = df0.columns.droplevel(0)
df.columns = df.columns.droplevel(0)
# concatenate them
df = (pd.concat([df0, df])
```

(continues on next page)

(continued from previous page)

```
        .fillna(False)
        .sort_index(axis=1, level=0,
                    key=lambda index: [plf.ResidueId.from_string(x) for x in index]))
df
```

```
/home/docs/checkouts/readthedocs.org/user_builds/prolif/conda/v0.3.1/lib/python3.9/
↪site-packages/MDAnalysis/coordinates/RDKit.py:492: DeprecationWarning: `np.float`␣
↪is a deprecated alias for the builtin `float`. To silence this warning, use `float`␣
↪by itself. Doing this will not modify any behavior and is safe. If you specifically␣
↪wanted the numpy scalar type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/
↪release/1.20.0-notes.html#deprecations
  if isinstance(value, (float, np.float)):
/home/docs/checkouts/readthedocs.org/user_builds/prolif/conda/v0.3.1/lib/python3.9/
↪site-packages/MDAnalysis/coordinates/RDKit.py:494: DeprecationWarning: `np.int` is␣
↪a deprecated alias for the builtin `int`. To silence this warning, use `int` by␣
↪itself. Doing this will not modify any behavior and is safe. When replacing `np.
↪int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the precision.␣
↪If you wish to review your current use, check the release note link for additional␣
↪information.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/
↪release/1.20.0-notes.html#deprecations
  elif isinstance(value, (int, np.int)):
```

```
  0%|          | 0/1 [00:00<?, ?it/s]
```

[19]:

```
protein        TYR38.A                TYR40.A    SER106.A    TYR109.A  \
interaction HBAcceptor Hydrophobic Hydrophobic Hydrophobic Hydrophobic
Frame
ref              False       False       False       False        True
0                False       False       False       False        True
1                False       False       False       False        True
2                False       False       False       False        True
3                False        True       False       False       False
4                 True        True       False       False        True
5                False       False       False       False        True
6                False        True        True        True        True
7                False        True       False       False        True
8                False       False       False       False       False

protein                   CYS122.A    ASP123.A    TRP125.A             ...  \
interaction PiStacking Hydrophobic Hydrophobic Hydrophobic PiStacking  ...
Frame                                                                  ...
ref              False       False       False        True      False  ...
0                False       False       False       False      False  ...
1                False       False       False        True      False  ...
2                False       False       False        True      False  ...
3                False       False       False       False      False  ...
4                False       False       False       False      False  ...
5                False       False       False        True      False  ...
6                 True       False       False        True       True  ...
7                False       False       False        True      False  ...
8                False        True        True       False      False  ...

protein        PRO338.B    PHE346.B    HSE347.B    LEU348.B    PHE351.B  \
interaction Hydrophobic Hydrophobic Hydrophobic Hydrophobic Hydrophobic
Frame
ref               True       False       False       False        True
```

(continues on next page)

(continued from previous page)

```
0                False       False       False       False        True
1                False       False       False       False        True
2                False       False       False       False        True
3                False       False       False       False        True
4                 True        True        True        True        True
5                False       False       False       False        True
6                False       False       False        True        True
7                 True       False        True        True        True
8                False       False       False       False       False

protein                    ASP352.B    THR355.B    TYR359.B
interaction PiStacking Hydrophobic Hydrophobic Hydrophobic PiStacking
Frame
ref               False        True        True        True      False
0                  True        True        True       False      False
1                  True        True        True       False      False
2                 False        True        True       False      False
3                 False        True        True        True      False
4                  True        True       False       False      False
5                 False        True        True       False      False
6                  True       False        True        True       True
7                  True       False        True       False      False
8                 False       False       False       False      False

[10 rows x 50 columns]
```

Lastly, we can convert the dataframe to a list of RDKit bitvectors to finally compute the Tanimoto similarity between our reference pose and the docking poses generated by Vina:

[20]:
```
from rdkit import DataStructs

bvs = plf.to_bitvectors(df)
for i, bv in enumerate(bvs[1:]):
    tc = DataStructs.TanimotoSimilarity(bvs[0], bv)
    print(f"{i}: {tc:.3f}")
```

```
0: 0.633
1: 0.455
2: 0.484
3: 0.433
4: 0.286
5: 0.690
6: 0.278
7: 0.469
8: 0.297
```

Interestingly, the best scored docking pose (#0) isn't the most similar to the reference (#5)

Using a MOL2 file

The input mol2 file can contain multiple ligands in different conformations.

[21]:
```
# load ligands
path = plf.datafiles.datapath / "vina" / "vina_output.mol2"
lig_suppl = list(plf.mol2_supplier(path))
# generate fingerprint
fp = plf.Fingerprint()
fp.run_from_iterable(lig_suppl, prot)
df = fp.to_dataframe()
df
```

```
  0%|          | 0/9 [00:00<?, ?it/s]
```

[21]:
```
ligand             UNL1                                                  \
protein         TYR38.A               TYR40.A    SER106.A    TYR109.A
interaction Hydrophobic HBAcceptor Hydrophobic Hydrophobic Hydrophobic
Frame
0                 False      False       False       False        True
1                 False      False       False       False        True
2                 False      False       False       False        True
3                  True      False       False       False       False
4                  True       True       False       False        True
5                 False      False       False       False        True
6                  True      False        True        True        True
7                  True      False       False       False        True
8                 False      False       False       False       False

ligand                                                                ...  \
protein                   CYS122.A    ASP123.A    TRP125.A             ...
interaction PiStacking Hydrophobic Hydrophobic Hydrophobic PiStacking  ...
Frame                                                                 ...
0                 False       False       False       False      False  ...
1                 False       False       False        True      False  ...
2                 False       False       False        True      False  ...
3                 False       False       False       False      False  ...
4                 False       False       False       False      False  ...
5                 False       False       False        True      False  ...
6                  True       False       False        True       True  ...
7                 False       False       False        True      False  ...
8                 False        True        True       False      False  ...

ligand                                                                 \
protein        PRO338.B    PHE346.B    HSE347.B    LEU348.B    PHE351.B
interaction Hydrophobic Hydrophobic Hydrophobic Hydrophobic Hydrophobic
Frame
0                 False       False       False       False        True
1                 False       False       False       False        True
2                 False       False       False       False        True
3                 False       False       False       False        True
4                  True        True        True        True        True
5                 False       False       False       False        True
6                 False       False       False        True        True
7                  True       False        True        True        True
8                 False       False       False       False       False

ligand
```

(continues on next page)

(continued from previous page)

```
protein                    ASP352.B    THR355.B    TYR359.B
interaction PiStacking Hydrophobic Hydrophobic Hydrophobic PiStacking
Frame
0                  True        True        True       False      False
1                  True        True        True       False      False
2                 False        True        True       False      False
3                 False        True        True        True      False
4                  True        True       False       False      False
5                 False        True        True       False      False
6                  True       False        True        True       True
7                  True       False        True       False      False
8                 False       False       False       False      False

[9 rows x 47 columns]
```

Using PDBQT files

The typical use case here is getting the IFP from AutoDock Vina's output. It requires a few additional steps and informations compared to other formats like MOL2, since the PDBQT format gets rid of most hydrogen atoms and doesn't contain bond order information.

The prerequisites for a successfull usage of ProLIF in this case is having external files that contain bond orders and formal charges for your ligand (like SMILES, SDF or MOL2), or at least a file with explicit hydrogen atoms.

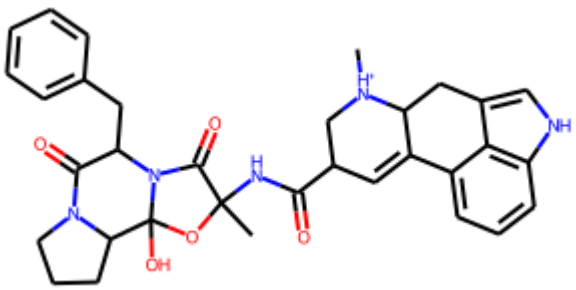
Please note that your PDBQT input must have a single model per file (this is required by MDAnalysis). Splitting a multi-model file can be done using the `vina_split` command-line tool that comes with AutoDock Vina: `vina_split --input vina_output.pdbqt`

Let's start by loading our "template" file with bond orders. It can be a SMILES string, MOL2, SDF file or anything supported by RDKit.

[22]:
```
from rdkit import Chem
from rdkit.Chem import AllChem

template = Chem.MolFromSmiles("C[NH+]1CC(C(=O)NC2(C)OC3(O)C4CCCN4C(=O)C"
                              "(Cc4ccccc4)N3C2=O)C=C2c3cccc4[nH]cc(c34)CC21")
template
```

[22]:



Next, we'll use the PDBQT supplier which loads each file from a list of paths, and assigns bond orders and charges using the template. The template and PDBQT file must have the exact same atoms, **even hydrogens**, otherwise no match will be found. Since PDBQT files partially keep the hydrogen atoms, we have the choice between:

- Manually selecting where to add the hydrogens on the template, do the matching, then add the remaining hydrogens (not covered here)
- Or just remove the hydrogens from the PDBQT file, do the matching, then add all hydrogens.

This last option will delete the coordinates of your hydrogens atoms and replace them by the ones generated by RDKit, but unless you're working with an exotic system this should be fine.

For the protein, there's usually no need to load the PDBQT that was used by Vina. The original file that was used to generate the PDBQT can be used directly, but **it must contain explicit hydrogen atoms**:

[23]:
```
# load ligands
pdbqt_files = sorted(plf.datafiles.datapath.glob("vina/*.pdbqt"))
lig_suppl = list(plf.pdbqt_supplier(pdbqt_files, template))
# generate fingerprint
fp = plf.Fingerprint()
fp.run_from_iterable(lig_suppl, prot)
df = fp.to_dataframe()
df
```

```
/home/docs/checkouts/readthedocs.org/user_builds/prolif/conda/v0.3.1/lib/python3.9/
↪site-packages/MDAnalysis/topology/guessers.py:146: UserWarning: Failed to guess the␣
↪mass for the following atom types: A
  warnings.warn("Failed to guess the mass for the following atom types: {}".
↪format(atom_type))
/home/docs/checkouts/readthedocs.org/user_builds/prolif/conda/v0.3.1/lib/python3.9/
↪site-packages/MDAnalysis/topology/guessers.py:146: UserWarning: Failed to guess the␣
↪mass for the following atom types: HD
  warnings.warn("Failed to guess the mass for the following atom types: {}".
↪format(atom_type))
/home/docs/checkouts/readthedocs.org/user_builds/prolif/conda/v0.3.1/lib/python3.9/
↪site-packages/MDAnalysis/topology/guessers.py:146: UserWarning: Failed to guess the␣
↪mass for the following atom types: OA
  warnings.warn("Failed to guess the mass for the following atom types: {}".
↪format(atom_type))
/home/docs/checkouts/readthedocs.org/user_builds/prolif/conda/v0.3.1/lib/python3.9/
↪site-packages/MDAnalysis/coordinates/RDKit.py:492: DeprecationWarning: `np.float`␣
↪is a deprecated alias for the builtin `float`. To silence this warning, use `float`␣
↪by itself. Doing this will not modify any behavior and is safe. If you specifically␣
↪wanted the numpy scalar type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/
↪release/1.20.0-notes.html#deprecations
  if isinstance(value, (float, np.float)):
/home/docs/checkouts/readthedocs.org/user_builds/prolif/conda/v0.3.1/lib/python3.9/
↪site-packages/MDAnalysis/coordinates/RDKit.py:494: DeprecationWarning: `np.int` is␣
↪a deprecated alias for the builtin `int`. To silence this warning, use `int` by␣
↪itself. Doing this will not modify any behavior and is safe. When replacing `np.
↪int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the precision.␣
↪If you wish to review your current use, check the release note link for additional␣
↪information.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/
↪release/1.20.0-notes.html#deprecations
  elif isinstance(value, (int, np.int)):
/home/docs/checkouts/readthedocs.org/user_builds/prolif/conda/v0.3.1/lib/python3.9/
↪site-packages/MDAnalysis/coordinates/RDKit.py:416: UserWarning: No `bonds`␣
↪attribute in this AtomGroup. Guessing bonds based on atoms coordinates
  warnings.warn(
```

```
  0%|          | 0/9 [00:00<?, ?it/s]
```

[23]:
```
ligand           LIG1.G                                                \
protein         TYR38.A               TYR40.A    SER106.A    TYR109.A
interaction Hydrophobic HBAcceptor Hydrophobic Hydrophobic Hydrophobic
Frame
0                 False      False       False       False        True
```

(continues on next page)

(continued from previous page)

```
1                False      False       False       False        True
2                False      False       False       False        True
3                 True      False       False       False       False
4                 True       True       False       False        True
5                False      False       False       False        True
6                 True      False        True        True        True
7                 True      False       False       False        True
8                False      False       False       False       False

ligand                                                                ...  \
protein                   CYS122.A    ASP123.A    TRP125.A             ...
interaction PiStacking Hydrophobic Hydrophobic Hydrophobic PiStacking  ...
Frame                                                                 ...
0                False       False       False       False      False  ...
1                False       False       False        True      False  ...
2                False       False       False        True      False  ...
3                False       False       False       False      False  ...
4                False       False       False       False      False  ...
5                False       False       False        True      False  ...
6                 True       False       False        True       True  ...
7                False       False       False        True      False  ...
8                False        True        True       False      False  ...

ligand                                                                 \
protein        PRO338.B    PHE346.B    HSE347.B    LEU348.B    PHE351.B
interaction Hydrophobic Hydrophobic Hydrophobic Hydrophobic Hydrophobic
Frame
0                 False       False       False       False        True
1                 False       False       False       False        True
2                 False       False       False       False        True
3                 False       False       False       False        True
4                  True        True        True        True        True
5                 False       False       False       False        True
6                 False       False       False        True        True
7                  True       False        True        True        True
8                 False       False       False       False       False

ligand
protein                   ASP352.B    THR355.B    TYR359.B
interaction PiStacking Hydrophobic Hydrophobic Hydrophobic PiStacking
Frame
0                 True        True        True       False      False
1                 True        True        True       False      False
2                False        True        True       False      False
3                False        True        True        True      False
4                 True        True       False       False      False
5                False        True        True       False      False
6                 True       False        True        True       True
7                 True       False        True       False      False
8                False       False       False       False      False

[9 rows x 47 columns]
```

1.6.6 Protein-protein interactions

This notebooks shows how to compute a fingerprint for protein-protein interactions.

Here we will investigate the interactions in a G-protein coupled receptor (GPCR) between a particular helix (called TM3) and the rest of the protein.

This can obviously be applied to proteins that don't belong to the same chain/segment, as long as you can figure out an appropriate MDAnalysis selection

[1]:
```
import MDAnalysis as mda
import prolif as plf
```

[2]:
```
# load traj
u = mda.Universe(plf.datafiles.TOP, plf.datafiles.TRAJ)
tm3 = u.select_atoms("resid 119:152")
prot = u.select_atoms("protein and not group tm3", tm3=tm3)
tm3, prot
```

[2]:
```
(<AtomGroup with 531 atoms>, <AtomGroup with 4457 atoms>)
```

[3]:
```
# prot-prot interactions
fp = plf.Fingerprint(["HBDonor", "HBAcceptor", "PiStacking", "PiCation", "CationPi",
↪"Anionic", "Cationic"])
fp.run(u.trajectory[::10], tm3, prot)
```

```
  0%|          | 0/25 [00:00<?, ?it/s]
```

```
/home/docs/checkouts/readthedocs.org/user_builds/prolif/conda/v0.3.1/lib/python3.9/
↪site-packages/MDAnalysis/coordinates/RDKit.py:492: DeprecationWarning: `np.float`␣
↪is a deprecated alias for the builtin `float`. To silence this warning, use `float`␣
↪by itself. Doing this will not modify any behavior and is safe. If you specifically␣
↪wanted the numpy scalar type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/
↪release/1.20.0-notes.html#deprecations
  if isinstance(value, (float, np.float)):
/home/docs/checkouts/readthedocs.org/user_builds/prolif/conda/v0.3.1/lib/python3.9/
↪site-packages/MDAnalysis/coordinates/RDKit.py:494: DeprecationWarning: `np.int` is␣
↪a deprecated alias for the builtin `int`. To silence this warning, use `int` by␣
↪itself. Doing this will not modify any behavior and is safe. When replacing `np.
↪int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the precision.␣
↪If you wish to review your current use, check the release note link for additional␣
↪information.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/
↪release/1.20.0-notes.html#deprecations
  elif isinstance(value, (int, np.int)):
```

[3]:
```
<prolif.fingerprint.Fingerprint: 7 interactions: ['HBDonor', 'HBAcceptor', 'Cationic',
↪ 'Anionic', 'PiCation', 'CationPi', 'PiStacking'] at 0x7f36e6a61d90>
```

[4]:
```
df = fp.to_dataframe()
df.head()
```

[4]:
```
ligand        GLN119.A                                                  \
protein       GLN189.A ALA190.A ALA192.A            GLU194.A
interaction HBAcceptor  HBDonor  HBDonor HBAcceptor  HBDonor HBAcceptor
Frame
0                False    False     True      False    False       True
10               False    False    False      False    False      False
```

(continues on next page)

(continued from previous page)

```
20              False    False    False      False    False      False
30              False    False    False      False    False      False
40              False    False    False      False     True      False

ligand                                                ...   ALA150.A  \
protein     VAL196.A            SER197.A              ...   ALA154.A
interaction  HBDonor HBAcceptor  HBDonor HBAcceptor  ... HBAcceptor
Frame                                                 ...
0             False      False    False      False  ...      False
10            False      False    False      False  ...      False
20            False      False    False       True  ...      False
30            False      False    False      False  ...      False
40            False       True    False      False  ...      False

ligand                   ILE151.A                          THR152.A             \
protein       ARG238.A   ALA154.A   ARG238.A   ASN245.A ASP153.A
interaction HBAcceptor HBAcceptor HBAcceptor HBAcceptor  HBDonor HBAcceptor
Frame
0               False      False      False      False    False      False
10              False      False       True      False    False      False
20              False      False       True      False    False       True
30              False      False      False      False    False      False
40              False       True       True      False    False      False

ligand
protein     GLU234.A   ARG238.A   LYS241.A
interaction  HBDonor HBAcceptor HBAcceptor
Frame
0             False      False      False
10            False      False      False
20            False       True      False
30            False      False      False
40            False      False      False

[5 rows x 55 columns]
```

[5]:
```
# show interactions for a specific ligand residue
df.xs("ARG147.A", level="ligand", axis=1).head(5)
```

[5]:
```
protein     ALA84.A GLU309.B          THR313.B
interaction HBDonor  HBDonor Cationic  HBDonor
Frame
0             False    False    False    False
10            False    False    False    False
20             True    False    False    False
30            False    False    False    False
40            False     True     True    False
```

[6]:
```
# same for a protein residue
df.xs("GLU309.B", level="protein", axis=1).head(5)
```

[6]:
```
ligand      ARG147.A
interaction  HBDonor Cationic
Frame
0             False    False
10            False    False
20            False    False
```

(continues on next page)

(continued from previous page)

```
30             False    False
40              True     True
```

[7]:
```
# display a specific type of interaction
df.xs("Cationic", level="interaction", axis=1).head(5)
```

[7]:
```
ligand  ARG147.A
protein GLU309.B
Frame
0           False
10          False
20          False
30          False
40           True
```

[8]:
```
# calculate the occurence of each interaction on the trajectory
occ = df.mean()
# restrict to the frequent ones
occ.loc[occ > 0.3]
```

[8]:
```
ligand    protein   interaction
CYS122.A  GLY118.A  HBDonor        0.84
ASP123.A  LYS191.A  Anionic        0.32
TRP125.A  VAL102.A  HBDonor        0.60
          TYR109.A  PiStacking     0.48
          TRP115.A  PiStacking     0.80
SER127.A  SER181.A  HBDonor        0.32
ASP129.A  TYR359.B  HBAcceptor     0.96
HSD139.A  TRP174.A  PiStacking     0.36
ASP146.A  TYR157.A  HBAcceptor     0.88
          ARG161.A  HBAcceptor     0.48
                    Anionic        0.36
ARG147.A  GLU309.B  HBDonor        0.44
                    Cationic       0.44
TRP149.A  ASP153.A  HBAcceptor     0.36
dtype: float64
```

[9]:
```
# regroup all interactions together and do the same
g = (df.groupby(level=["ligand", "protein"], axis=1)
       .sum()
       .astype(bool)
       .mean())
g.loc[g > 0.3]
```

[9]:
```
ligand    protein
ARG147.A  GLU309.B    0.44
ASP123.A  LYS191.A    0.32
ASP129.A  TYR359.B    0.96
ASP146.A  ARG161.A    0.48
          TYR157.A    0.88
CYS122.A  GLY118.A    0.84
HSD139.A  TRP174.A    0.36
SER127.A  SER181.A    0.32
TRP125.A  TRP115.A    0.80
          TYR109.A    0.48
          VAL102.A    0.60
TRP149.A  ASP153.A    0.36
```

(continues on next page)

(continued from previous page)

```
dtype: float64
```

1.6.7 Visualisation

This notebook showcases different ways of visualizing lig-prot and prot-prot interactions, either with atomistic details or simply at the residue level.

[1]:
```
import MDAnalysis as mda
import prolif as plf
import numpy as np
# load topology
u = mda.Universe(plf.datafiles.TOP, plf.datafiles.TRAJ)
lig = u.select_atoms("resname LIG")
prot = u.select_atoms("protein")
```

[2]:
```
# create RDKit-like molecules for visualisation
lmol = plf.Molecule.from_mda(lig)
pmol = plf.Molecule.from_mda(prot)
```

```
/home/docs/checkouts/readthedocs.org/user_builds/prolif/conda/v0.3.1/lib/python3.9/
↪site-packages/MDAnalysis/coordinates/RDKit.py:492: DeprecationWarning: `np.float`␣
↪is a deprecated alias for the builtin `float`. To silence this warning, use `float`␣
↪by itself. Doing this will not modify any behavior and is safe. If you specifically␣
↪wanted the numpy scalar type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/
↪release/1.20.0-notes.html#deprecations
  if isinstance(value, (float, np.float)):
/home/docs/checkouts/readthedocs.org/user_builds/prolif/conda/v0.3.1/lib/python3.9/
↪site-packages/MDAnalysis/coordinates/RDKit.py:494: DeprecationWarning: `np.int` is␣
↪a deprecated alias for the builtin `int`. To silence this warning, use `int` by␣
↪itself. Doing this will not modify any behavior and is safe. When replacing `np.
↪int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the precision.␣
↪If you wish to review your current use, check the release note link for additional␣
↪information.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/
↪release/1.20.0-notes.html#deprecations
  elif isinstance(value, (int, np.int)):
```

[3]:
```
# get lig-prot interactions with atom info
fp = plf.Fingerprint(["HBDonor", "HBAcceptor", "Cationic", "PiStacking"])
fp.run(u.trajectory[0:1], lig, prot, return_atoms=True)
df = fp.to_dataframe()
df.T
```

```
  0%|          | 0/1 [00:00<?, ?it/s]
```

[3]:
```
Frame                                0
ligand protein  interaction
LIG1.G ASP129.A Cationic      (13, 10)
                HBDonor       (52, 10)
       PHE331.B PiStacking      (3, 9)
       SER212.A HBDonor        (44, 2)
       VAL201.A HBAcceptor     (28, 6)
```

py3Dmol (3Dmol.js)

With py3Dmol we can easily display the interactions.

For interactions involving a ring (pi-cation, pi-stacking…etc.) ProLIF returns the index of one of the ring atoms, but for visualisation having the centroid of the ring looks nicer. We'll start by writing a function to find the centroid, given the index of one of the ring atoms.

[4]:
```
from rdkit import Chem
from rdkit import Geometry

def get_ring_centroid(mol, index):
    # find ring using the atom index
    Chem.SanitizeMol(mol, Chem.SanitizeFlags.SANITIZE_SETAROMATICITY)
    ri = mol.GetRingInfo()
    for r in ri.AtomRings():
        if index in r:
            break
    else:
        raise ValueError("No ring containing this atom index was found in the given molecule")
    # get centroid
    coords = mol.xyz[list(r)]
    ctd = plf.utils.get_centroid(coords)
    return Geometry.Point3D(*ctd)
```

Finally, the actual visualisation code. The API of py3Dmol is exactly the same as the GLViewer class of 3Dmol.js, for which the documentation can be found here.

[5]:
```
import py3Dmol

colors = {
    "HBAcceptor": "blue",
    "HBDonor": "red",
    "Cationic": "green",
    "PiStacking": "purple",
}

# JavaScript functions
resid_hover = """function(atom,viewer) {{
    if(!atom.label) {{
        atom.label = viewer.addLabel('{0}:'+atom.atom+atom.serial,
            {{position: atom, backgroundColor: 'mintcream', fontColor:'black'}});
    }}
}}"""
hover_func = """
function(atom,viewer) {
    if(!atom.label) {
        atom.label = viewer.addLabel(atom.interaction,
            {position: atom, backgroundColor: 'black', fontColor:'white'});
    }
}"""
unhover_func = """
function(atom,viewer) {
    if(atom.label) {
        viewer.removeLabel(atom.label);
        delete atom.label;
    }
```

(continues on next page)

(continued from previous page)

```
}"""

v = py3Dmol.view(650, 600)
v.removeAllModels()

models = {}
mid = -1
for i, row in df.T.iterrows():
    lresid, presid, interaction = i
    lindex, pindex = row[0]
    lres = lmol[lresid]
    pres = pmol[presid]
    # set model ids for reusing later
    for resid, res, style in [(lresid, lres, {"colorscheme": "cyanCarbon"}),
                              (presid, pres, {})]:
        if resid not in models.keys():
            mid += 1
            v.addModel(Chem.MolToMolBlock(res), "sdf")
            model = v.getModel()
            model.setStyle({}, {"stick": style})
            # add residue label
            model.setHoverable({}, True, resid_hover.format(resid), unhover_func)
            models[resid] = mid
    # get coordinates for both points of the interaction
    if interaction in ["PiStacking", "EdgeToFace", "FaceToFace", "PiCation"]:
        p1 = get_ring_centroid(lres, lindex)
    else:
        p1 = lres.GetConformer().GetAtomPosition(lindex)
    if interaction in ["PiStacking", "EdgeToFace", "FaceToFace", "CationPi"]:
        p2 = get_ring_centroid(pres, pindex)
    else:
        p2 = pres.GetConformer().GetAtomPosition(pindex)
    # add interaction line
    v.addCylinder({"start": dict(x=p1.x, y=p1.y, z=p1.z),
                   "end":   dict(x=p2.x, y=p2.y, z=p2.z),
                   "color": colors[interaction],
                   "radius": .15,
                   "dashed": True,
                   "fromCap": 1,
                   "toCap": 1,
                  })
    # add label when hovering the middle of the dashed line by adding a dummy atom
    c = Geometry.Point3D(*plf.utils.get_centroid([p1, p2]))
    modelID = models[lresid]
    model = v.getModel(modelID)
    model.addAtoms([{"elem": 'Z',
                     "x": c.x, "y": c.y, "z": c.z,
                     "interaction": interaction}])
    model.setStyle({"interaction": interaction}, {"clicksphere": {"radius": .5}})
    model.setHoverable(
        {"interaction": interaction}, True,
        hover_func, unhover_func)

# show protein:
# first we need to reorder atoms as in the original MDAnalysis file.
# needed because the RDKitConverter reorders them when infering bond order
# and 3Dmol.js doesn't like when atoms from the same residue are spread accross the␣
↪whole file
```

(continues on next page)

(continued from previous page)

```
order = np.argsort([atom.GetIntProp("_MDAnalysis_index") for atom in pmol.GetAtoms()])
mol = Chem.RenumberAtoms(pmol, order.astype(int).tolist())
mol = Chem.RemoveAllHs(mol)
pdb = Chem.MolToPDBBlock(mol, flavor=0x20 | 0x10)
v.addModel(pdb, "pdb")
model = v.getModel()
model.setStyle({}, {"cartoon": {"style":"edged"}})

v.zoomTo({"model": list(models.values())})
```

Data type cannot be displayed: application/3dmoljs_load.v0, text/html

[5]:
```
<py3Dmol.view at 0x7f75285e1ac0>
```

RDKit

This script is given as a starting point for visualising the interactions with RDKit. It's definitely not ideal for 3D visualisation as the drawing ends up being crowded and quite unreadable very quickly. I'm sure there's a better way to do this but it will do as a code snippet.

[6]:
```
from rdkit.Chem import Draw
from IPython import display

colors = {
    "HBAcceptor": (0, 1, 1),
    "HBDonor": (1, .7,.3),
    "Cationic": (0, 1, 0),
    "PiStacking": (.5, 0, .5),
}

d = Draw.MolDraw2DSVG(650, 600)
opts = Draw.MolDrawOptions()
opts.fixedBondLength = 25
d.SetDrawOptions(opts)

displayed = []
for i, row in df.T.iterrows():
    lresid, presid, interaction = i
    lindex, pindex = row[0]
    lres = lmol[lresid]
    pres = pmol[presid]
    for resid, res in [(lresid, lres), (presid, pres)]:
        if resid not in displayed:
            displayed.append(resid)
            d.DrawMolecule(res)
    if interaction in ["PiStacking", "EdgeToFace", "FaceToFace", "PiCation"]:
        p1 = get_ring_centroid(lres, lindex)
    else:
        p1 = lres.GetConformer().GetAtomPosition(lindex)
    if interaction in ["PiStacking", "EdgeToFace", "FaceToFace", "CationPi"]:
        p2 = get_ring_centroid(pres, pindex)
    else:
        p2 = pres.GetConformer().GetAtomPosition(pindex)
    p1 = Geometry.Point2D(p1)
```

(continues on next page)

(continued from previous page)

```
    p2 = Geometry.Point2D(p2)
    d.DrawWavyLine(p1, p2, colors[interaction], colors[interaction])

# display
d.FinishDrawing()
display.SVG(d.GetDrawingText())
```

[6]:

NetworkX and pyvis

NetworkX is a great library for working with graphs, but the drawing options are quickly limited so we will use networkx to create a graph, and pyvis to create interactive plots. The following code snippet will calculate the IFP, each residue (ligand or protein) is converted to a node, each interaction to an edge, and the occurence of each interaction between residues will be used to control the weight and thickness of each edge.

[7]:
```
import networkx as nx
from pyvis.network import Network
from tqdm.auto import tqdm
from matplotlib import cm, colors
from IPython.display import IFrame
```

[8]:
```
# get lig-prot interactions and distance between residues

fp = plf.Fingerprint()
fp.run(u.trajectory[::10], lig, prot)
df = fp.to_dataframe()
df.head()
```

```
  0%|          | 0/25 [00:00<?, ?it/s]
```

[8]:
```
ligand           LIG1.G                                                  \
protein         TYR38.A    TYR109.A    THR110.A    TRP125.A    LEU126.A
interaction Hydrophobic Hydrophobic Hydrophobic Hydrophobic Hydrophobic
Frame
0                 False        True       False        True        True
10                False        True       False        True        True
20                False        True        True        True        True
30                 True        True        True        True        True
40                False        True        True        True        True

ligand                                                             ...  \
protein        ASP129.A                     ILE130.A    CYS133.A  ...
interaction Hydrophobic HBDonor Cationic Hydrophobic Hydrophobic  ...
Frame                                                              ...
0                 True    True     True        True        True  ...
10                True    True     True        True        True  ...
20                True    True     True        True        True  ...
30                True    True     True        True        True  ...
40                True    True     True        True        True  ...

ligand                                                                  \
protein        MET337.B    PRO338.B    ALA343.B    CYS344.B    LEU348.B
interaction Hydrophobic Hydrophobic Hydrophobic Hydrophobic Hydrophobic
Frame
0                 True        True       False       False       False
10                True       False        True       False       False
```

(continues on next page)

(continued from previous page)

```
20                 True       False        True       False       False
30                 True       False        True        True       False
40                False       False       False       False       False

ligand
protein        PHE351.B               ASP352.B    THR355.B    TYR359.B
interaction Hydrophobic PiStacking Hydrophobic Hydrophobic Hydrophobic
Frame
0                  True      False        True        True        True
10                 True       True       False        True        True
20                 True      False       False       False        True
30                 True       True       False        True        True
40                 True       True        True       False        True

[5 rows x 40 columns]
```

[9]:
```
def make_graph(values, df=None,
               node_color=["#FFB2AC", "#ACD0FF"], node_shape="dot",
               edge_color="#a9a9a9", width_multiplier=1):
    """Convert a pandas DataFrame to a NetworkX object

    Parameters
    ----------
    values : pandas.Series
        Series with 'ligand' and 'protein' levels, and a unique value for
        each lig-prot residue pair that will be used to set the width and weigth
        of each edge. For example:

            ligand  protein
            LIG1.G  ALA216.A    0.66
                    ALA343.B    0.10

    df : pandas.DataFrame
        DataFrame obtained from the fp.to_dataframe() method
        Used to label each edge with the type of interaction

    node_color : list
        Colors for the ligand and protein residues, respectively

    node_shape : str
        One of ellipse, circle, database, box, text or image, circularImage,
        diamond, dot, star, triangle, triangleDown, square, icon.

    edge_color : str
        Color of the edge between nodes

    width_multiplier : int or float
        Each edge's width is defined as `width_multiplier * value`
    """
    lig_res = values.index.get_level_values("ligand").unique().tolist()
    prot_res = values.index.get_level_values("protein").unique().tolist()

    G = nx.Graph()
    # add nodes
    # https://pyvis.readthedocs.io/en/latest/documentation.html#pyvis.network.Network.
↪add_node
```

(continues on next page)

(continued from previous page)

```
    for res in lig_res:
        G.add_node(res, title=res, shape=node_shape,
                   color=node_color[0], dtype="ligand")
    for res in prot_res:
        G.add_node(res, title=res, shape=node_shape,
                   color=node_color[1], dtype="protein")

    for resids, value in values.items():
        label = "{} - {}<br>{}".format(*resids, "<br>".join([f"{k}: {v}"
                                       for k, v in (df.xs(resids,
                                                          level=["ligand", "protein"],
                                                          axis=1)
                                                     .sum()
                                                     .to_dict()
                                                     .items())]))
        # https://pyvis.readthedocs.io/en/latest/documentation.html#pyvis.network.
↪Network.add_edge
        G.add_edge(*resids, title=label, color=edge_color,
                   weight=value, width=value*width_multiplier)

    return G
```

Regrouping all interactions

We will regroup all interactions as if they were equivalent.

[10]:
```
data = (df.groupby(level=["ligand", "protein"], axis=1)
          .sum()
          .astype(bool)
          .mean())

G = make_graph(data, df, width_multiplier=3)

# display graph
net = Network(width=600, height=500, notebook=True, heading="")
net.from_nx(G)
net.write_html("lig-prot_graph.html")
IFrame("lig-prot_graph.html", width=610, height=510)
```

[10]:
```
<IPython.lib.display.IFrame at 0x7f75285d4160>
```

Only plotting a specific interaction

We can also plot a specific type of interaction.

[11]:
```
data = (df.xs("Hydrophobic", level="interaction", axis=1)
          .mean())

G = make_graph(data, df, width_multiplier=3)

# display graph
net = Network(width=600, height=500, notebook=True, heading="")
net.from_nx(G)
```

(continues on next page)

(continued from previous page)

```
net.write_html("lig-prot_hydrophobic_graph.html")
IFrame("lig-prot_hydrophobic_graph.html", width=610, height=510)
```

[11]:
```
<IPython.lib.display.IFrame at 0x7f75285d40d0>
```

Protein-protein interaction

This kind of "residue-level" visualisation is especially suitable for protein-protein interactions. Here we'll show the interactions between one helix of our G-Protein coupled receptor (transmembrane helix 3, or TM3) in red and the rest of the protein in blue.

[12]:
```
tm3 = u.select_atoms("resid 119:152")
prot = u.select_atoms("protein and not group tm3", tm3=tm3)
fp = plf.Fingerprint()
fp.run(u.trajectory[::10], tm3, prot)
df = fp.to_dataframe()
df.head()
```

```
  0%|          | 0/25 [00:00<?, ?it/s]
```

```
/home/docs/checkouts/readthedocs.org/user_builds/prolif/conda/v0.3.1/lib/python3.9/
↪site-packages/MDAnalysis/coordinates/RDKit.py:492: DeprecationWarning: `np.float`␣
↪is a deprecated alias for the builtin `float`. To silence this warning, use `float`␣
↪by itself. Doing this will not modify any behavior and is safe. If you specifically␣
↪wanted the numpy scalar type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/
↪release/1.20.0-notes.html#deprecations
  if isinstance(value, (float, np.float)):
/home/docs/checkouts/readthedocs.org/user_builds/prolif/conda/v0.3.1/lib/python3.9/
↪site-packages/MDAnalysis/coordinates/RDKit.py:494: DeprecationWarning: `np.int` is␣
↪a deprecated alias for the builtin `int`. To silence this warning, use `int` by␣
↪itself. Doing this will not modify any behavior and is safe. When replacing `np.
↪int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the precision.␣
↪If you wish to review your current use, check the release note link for additional␣
↪information.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/
↪release/1.20.0-notes.html#deprecations
  elif isinstance(value, (int, np.int)):
```

[12]:
```
ligand         GLN119.A                                              \
protein        GLY118.A    GLN189.A           ALA190.A    LYS191.A
interaction Hydrophobic Hydrophobic HBAcceptor  HBDonor Hydrophobic
Frame
0                 True       False      False    False       False
10                True       False      False    False       False
20                True       False      False    False       False
30                True       False      False    False       False
40                True       False      False    False        True

ligand                                                          ...  \
protein        ALA192.A                      GLU193.A    GLU194.A  ...
interaction Hydrophobic HBDonor HBAcceptor Hydrophobic Hydrophobic  ...
Frame                                                           ...
0                False    True      False        True        True  ...
10               False   False      False       False       False  ...
20               False   False      False        True       False  ...
```

(continues on next page)

(continued from previous page)

```
30                 True   False      False       False       False  ...
40                False   False      False       False       False  ...

ligand         THR152.A                                                \
protein        ASP153.A                       ALA154.A    GLU234.A
interaction Hydrophobic HBDonor HBAcceptor Hydrophobic Hydrophobic HBDonor
Frame
0                  True   False      False       False        True   False
10                 True   False      False        True        True   False
20                 True   False       True        True        True   False
30                 True   False      False       False       False   False
40                 True   False      False       False       False   False

ligand
protein        ARG238.A               LYS241.A
interaction Hydrophobic HBAcceptor Hydrophobic HBAcceptor
Frame
0                 False      False       False      False
10                False      False       False      False
20                False       True       False      False
30                 True      False       False      False
40                False      False       False      False

[5 rows x 232 columns]
```

[13]:
```
data = (df.groupby(level=["ligand", "protein"], axis=1, sort=False)
          .sum()
          .astype(bool)
          .mean())

G = make_graph(data, df, width_multiplier=8)

# color each node based on its degree
max_nbr = len(max(G.adj.values(), key=lambda x: len(x)))
blues = cm.get_cmap('Blues', max_nbr)
reds = cm.get_cmap('Reds', max_nbr)
for n, d in G.nodes(data=True):
    n_neighbors = len(G.adj[n])
    # show TM3 in red and the rest of the protein in blue
    palette = reds if d["dtype"] == "ligand" else blues
    d["color"] = colors.to_hex( palette(n_neighbors / max_nbr) )

# convert to pyvis network
net = Network(width=640, height=500, notebook=True, heading="")
net.from_nx(G)
net.write_html("prot-prot_graph.html")
IFrame("prot-prot_graph.html", width=650, height=510)
```

[13]:
```
<IPython.lib.display.IFrame at 0x7f7521fcef40>
```

Residue interaction network

Another possible application is the visualisation of the residue interaction network of the whole protein. Since this protein is a GPCR, the graph will mostly display the HBond interactions reponsible for the secondary structure of the protein (7 alpha-helices). It would also show hydrophobic interactions between neighbor residues, so I'm simply going to disable it in the Fingerprint.

[14]:
```
prot = u.select_atoms("protein")
fp = plf.Fingerprint(['HBDonor', 'HBAcceptor', 'PiStacking', 'Anionic', 'Cationic',
↪'CationPi', 'PiCation'])
fp.run(u.trajectory[::10], prot, prot)
df = fp.to_dataframe()
df.head()
```

```
  0%|          | 0/25 [00:00<?, ?it/s]
```

```
/home/docs/checkouts/readthedocs.org/user_builds/prolif/conda/v0.3.1/lib/python3.9/
↪site-packages/MDAnalysis/coordinates/RDKit.py:492: DeprecationWarning: `np.float`␣
↪is a deprecated alias for the builtin `float`. To silence this warning, use `float`␣
↪by itself. Doing this will not modify any behavior and is safe. If you specifically␣
↪wanted the numpy scalar type, use `np.float64` here.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/
↪release/1.20.0-notes.html#deprecations
  if isinstance(value, (float, np.float)):
/home/docs/checkouts/readthedocs.org/user_builds/prolif/conda/v0.3.1/lib/python3.9/
↪site-packages/MDAnalysis/coordinates/RDKit.py:494: DeprecationWarning: `np.int` is␣
↪a deprecated alias for the builtin `int`. To silence this warning, use `int` by␣
↪itself. Doing this will not modify any behavior and is safe. When replacing `np.
↪int`, you may wish to use e.g. `np.int64` or `np.int32` to specify the precision.␣
↪If you wish to review your current use, check the release note link for additional␣
↪information.
Deprecated in NumPy 1.20; for more details and guidance: https://numpy.org/devdocs/
↪release/1.20.0-notes.html#deprecations
  elif isinstance(value, (int, np.int)):
```

[14]:
```
ligand         TYR38.A                                                    \
protein        TYR38.A    TYR40.A    GLN41.A    ASP42.A ARG114.A   TRP345.B
interaction PiStacking PiStacking HBAcceptor HBAcceptor PiCation PiStacking
Frame
0                True      False      False      False    False      False
10               True      False      False      False    False      False
20               True      False      False      False    False      False
30               True      False      False      False    False      False
40               True      False      False      False    False      False

ligand                                     ILE39.A  ... ARG385.B            \
protein     PHE346.B   HSE347.B ASP352.B    SER43.A  ... LEU383.B   LYS387.B
interaction  HBDonor PiStacking  HBDonor HBAcceptor  ...  HBDonor HBAcceptor
Frame                                                ...
0              False      False    False      False  ...     True      False
10              True      False    False      False  ...     True      False
20             False      False    False      False  ...     True      False
30              True       True    False      False  ...    False      False
40              True       True    False      False  ...    False      False

ligand        PHE386.B                                                  \
protein       PHE380.B HSD381.B                         ILE384.B   PHE386.B
interaction PiStacking  HBDonor HBAcceptor PiStacking  HBDonor PiStacking
Frame
```

(continues on next page)

(continued from previous page)

```
0                False     True      False       True    False       True
10               False     True      False       True    False       True
20               False     True      False      False    False       True
30               False     True       True      False    False       True
40               False     True      False      False    False       True

ligand        LYS387.B
protein       HSD381.B ARG385.B
interaction HBAcceptor  HBDonor
Frame
0                False    False
10                True    False
20               False    False
30               False    False
40               False    False

[5 rows x 1337 columns]
```

To hide most of the HBond interactions responsible for the alpha-helix structuration, I will show how to do it on the pandas DataFrame for simplicity, but ideally you should copy-paste the source code inside the `fp.run` method and add the condition shown below before calculating the bitvector for a residue pair, then use the custom function instead of `fp.run`. This would make the analysis faster and more memory efficient.

[15]:
```
# remove interactions between residues i and i±4 or less
mask = []
for l, p, interaction in df.columns:
    lr = plf.ResidueId.from_string(l)
    pr = plf.ResidueId.from_string(p)
    if (pr == lr) or (abs(pr.number - lr.number) <= 4
                      and interaction in ["HBDonor", "HBAcceptor", "Hydrophobic"]):
        mask.append(False)
    else:
        mask.append(True)
df = df[df.columns[mask]]
df.head()
```

[15]:
```
ligand         TYR38.A                                                  \
protein        TYR40.A ARG114.A   TRP345.B PHE346.B   HSE347.B ASP352.B
interaction PiStacking PiCation PiStacking  HBDonor PiStacking  HBDonor
Frame
0                False    False      False    False      False    False
10               False    False      False     True      False    False
20               False    False      False    False      False    False
30               False    False      False     True       True    False
40               False    False      False     True       True    False

ligand         TYR40.A                        GLN41.A  ...   HSD381.B  \
protein        TYR38.A   HSE347.B ASP352.B    LYS49.A  ...   PHE380.B
interaction PiStacking PiStacking  HBDonor HBAcceptor  ... PiStacking
Frame                                                  ...
0                False      False     True      False  ...       True
10               False      False     True      False  ...      False
20               False      False     True      False  ...      False
30               False      False     True      False  ...       True
40               False      False     True      False  ...      False

ligand                                            PHE386.B           \
```

(continues on next page)

(continued from previous page)

```
protein     PHE386.B                       LYS387.B   PHE380.B HSD381.B
interaction  HBDonor HBAcceptor PiStacking  HBDonor PiStacking  HBDonor
Frame
0              False       True       True    False      False     True
10             False       True       True     True      False     True
20             False       True      False    False      False     True
30              True       True      False    False      False     True
40             False       True      False    False      False     True

ligand                             LYS387.B
protein                            HSD381.B
interaction HBAcceptor PiStacking HBAcceptor
Frame
0                False       True      False
10               False       True       True
20               False      False      False
30                True      False      False
40               False      False      False

[5 rows x 416 columns]
```

[16]:
```
data = (df.groupby(level=["ligand", "protein"], axis=1, sort=False)
          .sum()
          .astype(bool)
          .mean())

G = make_graph(data, df, width_multiplier=5)

# color each node based on its degree
max_nbr = len(max(G.adj.values(), key=lambda x: len(x)))
palette = cm.get_cmap('YlGnBu', max_nbr)
for n, d in G.nodes(data=True):
    n_neighbors = len(G.adj[n])
    d["color"] = colors.to_hex( palette(n_neighbors / max_nbr) )

# convert to pyvis network
net = Network(width=640, height=500,  notebook=True, heading="")
net.from_nx(G)

# use specific layout
layout = nx.circular_layout(G)
for node in net.nodes:
    node["x"] = layout[node["id"]][0] * 1000
    node["y"] = layout[node["id"]][1] * 1000
net.toggle_physics(False)

net.write_html("residue-network_graph.html")
IFrame("residue-network_graph.html", width=650, height=510)
```

[16]:
```
<IPython.lib.display.IFrame at 0x7f75221f5130>
```

End of this notebook. If you have other suggestions for displaying interaction fingerprints, please create a new Discussion on GitHub

List of files to be automatically copied to the docs:

- lig-prot_graph.html

- lig-prot_hydrophobic_graph.html
- prot-prot_graph.html
- residue-network_graph.html

1.6.8 Summary

Input data

Currently, ProLIF only accepts RDKit molecules as input and relies on MDAnalysis to convert input files for Molecular Dynamics simulations and docking experiments to this RDKit object. The documentation can be found in the *Molecules* section.

Residues

To decompose interactions by residue, ProLIF needs to split the *prolif.molecule.Molecule* component in residues. Information on how these residues are stored and accessed can be found in the *Residues* section.

Interaction fingerprint

An interaction fingerprint decomposes the interactions between two molecules in a binary vector. The interactions are detected by looking up for predefined molecular patterns (like SMARTS queries) that satisfy geometrical constraints (distance, angle, dihedral...). To learn more on how these interactions are defined and how to use prolif to extract fingerprints, please refer to the *Interaction fingerprint* section.

Helper functions

ProLIF comes with a set of functions that should help users analyse the results produced by the package more easily. The documentation for these functions can be found in the *Helper functions* section.

1.6.9 Molecules

Reading RDKit molecules — `prolif.rdkitmol`

class prolif.rdkitmol.**BaseRDKitMol**

Bases: rdkit.Chem.rdchem.Mol

Base molecular class that behaves like an RDKit `Mol` with extra attributes (see below). The sole purpose of this class is to define the common API between the *Molecule* and *Residue* classes. This class should not be instantiated by users.

Parameters **mol** (*rdkit.Chem.rdchem.Mol*) – A molecule (protein, ligand, or residue) with a single conformer

centroid

XYZ coordinates of the centroid of the molecule

Type numpy.ndarray

xyz

XYZ coordinates of all atoms in the molecule

Type numpy.ndarray

Reading proteins and ligands — `prolif.molecule`

class prolif.molecule.**Molecule**(*mol*)
Bases: *prolif.rdkitmol.BaseRDKitMol*

Main molecule class that behaves like an RDKit `Mol` with extra attributes (see examples below). The main purpose of this class is to access residues as fragments of the molecule.

Parameters **mol** (*rdkit.Chem.rdchem.Mol*) – A ligand or protein with a single conformer

residues
A dictionnary storing one/many *Residue* indexed by *ResidueId*. The residue list is sorted.

Type *prolif.residue.ResidueGroup*

n_residues
Number of residues

Type int

Examples

```
In [1]: import MDAnalysis as mda

In [2]: import prolif

In [3]: u = mda.Universe(prolif.datafiles.TOP, prolif.datafiles.TRAJ)

In [4]: mol = u.select_atoms("protein").convert_to("RDKIT")

In [5]: mol = prolif.Molecule(mol)

In [6]: mol
Out[6]: <prolif.molecule.Molecule with 302 residues and 4988 atoms at␣
↪0x7f00b5a07e00>
```

You can also create a Molecule directly from a `Universe`:

```
In [7]: mol = prolif.Molecule.from_mda(u, "protein")

In [8]: mol
Out[8]: <prolif.molecule.Molecule with 302 residues and 4988 atoms at␣
↪0x7f00b5da9ea0>
```

Notes

Residues can be accessed easily in different ways:

```
In [9]: mol["TYR38.A"] # by resid string (residue name + number + chain)
Out[9]: <prolif.residue.Residue TYR38.A at 0x7f00b55d2e50>

In [10]: mol[42] # by index (from 0 to n_residues-1)
Out[10]: <prolif.residue.Residue LEU80.A at 0x7f00b55c0310>

In [11]: mol[prolif.ResidueId("TYR", 38, "A")] # by ResidueId
Out[11]: <prolif.residue.Residue TYR38.A at 0x7f00b55d2e50>
```

See *prolif.residue* for more information on residues

classmethod from_mda(*obj*, *selection=None*, ***kwargs*)
Creates a Molecule from an MDAnalysis object

Parameters

- **obj** (*MDAnalysis.core.universe.Universe or MDAnalysis.core.groups.AtomGroup*) – The MDAnalysis object to convert
- **selection** (*None or str*) – Apply a selection to *obj* to create an AtomGroup. Uses all atoms in *obj* if `selection=None`
- ****kwargs** (*object*) – Other arguments passed to the RDKitConverter of MDAnalysis

Example

```
In [1]: mol = prolif.Molecule.from_mda(u, "protein")

In [2]: mol
Out[2]: <prolif.molecule.Molecule with 302 residues and 4988 atoms at␣
↪0x7f00b5a25770>
```

Which is equivalent to:

```
In [3]: protein = u.select_atoms("protein")

In [4]: mol = prolif.Molecule.from_mda(protein)

In [5]: mol
Out[5]: <prolif.molecule.Molecule with 302 residues and 4988 atoms at␣
↪0x7f00b56760e0>
```

classmethod from_rdkit(*mol*, *resname='UNL'*, *resnumber=1*, *chain=''*)
Creates a Molecule from an RDKit molecule

While directly instantiating a molecule with `prolif.Molecule(mol)` would also work, this method insures that every atom is linked to an AtomPDBResidueInfo which is required by ProLIF

Parameters

- **mol** (*rdkit.Chem.rdchem.Mol*) – The input RDKit molecule
- **resname** (*str*) – The default residue name that is used if none was found
- **resnumber** (*int*) – The default residue number that is used if none was found
- **chain** (*str*) – The default chain Id that is used if none was found

Notes

This method only checks for an existing AtomPDBResidueInfo in the first atom. If none was found, it will patch all atoms with the one created from the method's arguments (resname, resnumber, chain).

prolif.molecule.**mol2_supplier**(*path*, ***kwargs*)
Generates prolif.Molecule objects from a MOL2 file

Parameters

- **path** (*str*) – A path to the .mol2 file

- **resname** (*str*) – Residue name for every ligand
- **resnumber** (*int*) – Residue number for every ligand
- **chain** (*str*) – Chain ID for every ligand

Returns **suppl** – A generator object that will provide the *Molecule* object as you iterate over it.

Return type generator

Example

The supplier is typically used like this:

```
>>> lig_suppl = mol2_supplier("docking/output.mol2")
>>> for lig in lig_suppl:
...     # do something with each ligand
```

prolif.molecule.**pdbqt_supplier**(*paths*, *template*)
Supplies molecules, given a path to PDBQT files

Parameters

- **paths** (*list*) – A list (or any iterable) of PDBQT files
- **template** (*rdkit.Chem.rdchem.Mol*) – A template molecule with the correct bond orders and charges. It must match exactly the molecule inside the PDBQT file.

Returns **suppl** – A generator object that will provide the *Molecule* object as you iterate over it.

Return type generator

Example

The supplier is typically used like this:

```
>>> import glob
>>> pdbqts = glob.glob("docking/ligand1/*.pdbqt")
>>> lig_suppl = pdbqt_supplier(pdbqts, template)
>>> for lig in lig_suppl:
...     # do something with each ligand
```

prolif.molecule.**sdf_supplier**(*path*, ***kwargs*)
Supplies molecules, given a path to an SDFile

Parameters

- **path** (*str*) – A path to the .sdf file
- **resname** (*str*) – Residue name for every ligand
- **resnumber** (*int*) – Residue number for every ligand
- **chain** (*str*) – Chain ID for every ligand

Returns **suppl** – A generator object that will provide the *Molecule* object as you iterate over it.

Return type generator

Example

The supplier is typically used like this:

```
>>> lig_suppl = sdf_supplier("docking/output.sdf")
>>> for lig in lig_suppl:
...     # do something with each ligand
```

class prolif.residue.**Residue**(*mol*)

Bases: *prolif.rdkitmol.BaseRDKitMol*

A class for residues as RDKit molecules

Parameters **mol** (*rdkit.Chem.rdchem.Mol*) – The residue as an RDKit molecule

resid

The residue identifier

Type *prolif.residue.ResidueId*

Notes

The name of the residue can be converted to a string by using `str(Residue)`

1.6.10 Interaction fingerprint

Calculate a Protein-Ligand Interaction Fingerprint — `prolif.fingerprint`

```
In [1]: import MDAnalysis as mda

In [2]: from rdkit.DataStructs import TanimotoSimilarity

In [3]: import prolif

In [4]: u = mda.Universe(prolif.datafiles.TOP, prolif.datafiles.TRAJ)

In [5]: prot = u.select_atoms("protein")

In [6]: lig = u.select_atoms("resname LIG")

In [7]: fp = prolif.Fingerprint(["HBDonor", "HBAcceptor", "PiStacking", "CationPi",
 ↪"Cationic"])

In [8]: fp.run(u.trajectory[::10], lig, prot)
Out[8]: <prolif.fingerprint.Fingerprint: 5 interactions: ['HBDonor', 'HBAcceptor',
 ↪'Cationic', 'CationPi', 'PiStacking'] at 0x7f00b55979d0>

In [9]: df = fp.to_dataframe()

In [10]: df
Out[10]:
ligand        LIG1.G                    ...
protein     ASP129.A          THR134.A  ...   PHE330.B   PHE331.B   PHE351.B
interaction  HBDonor Cationic  HBDonor  ... PiStacking PiStacking PiStacking
Frame                                   ...
0               True     True    False  ...      False       True      False
```

(continues on next page)

(continued from previous page)

```
10              True     True    False  ...       True       True       True
20              True     True    False  ...      False       True      False
30              True     True    False  ...      False      False       True
40              True     True    False  ...      False       True       True
50              True     True    False  ...      False       True       True
60              True     True    False  ...      False       True      False
70              True     True    False  ...      False       True      False
80              True     True    False  ...       True       True      False
90              True     True    False  ...      False      False       True
100             True     True    False  ...      False      False      False
110             True     True    False  ...       True       True      False
120             True     True    False  ...       True       True       True
130             True     True     True  ...      False      False      False
140             True     True    False  ...      False      False      False
150             True     True    False  ...      False      False      False
160             True     True    False  ...      False       True      False
170             True     True    False  ...      False       True      False
180             True     True    False  ...      False       True      False
190             True     True    False  ...      False       True      False
200             True     True    False  ...       True      False       True
210             True     True    False  ...       True       True       True
220             True     True    False  ...      False       True      False
230             True     True    False  ...      False       True       True
240             True     True    False  ...       True       True       True

[25 rows x 9 columns]

In [11]: bv = fp.to_bitvectors()

In [12]: TanimotoSimilarity(bv[0], bv[1])
Out[12]: 0.7142857142857143
```

class prolif.fingerprint.**Fingerprint**(*interactions=['Hydrophobic', 'HBDonor', 'HBAcceptor', 'PiStacking', 'Anionic', 'Cationic', 'CationPi', 'PiCation']*)

Class that generates an interaction fingerprint between two molecules

While in most cases the fingerprint will be generated between a ligand and a protein, it is also possible to use this class for protein-protein interactions, or host-guest systems.

Parameters **interactions** (*list*) – List of names (str) of interaction classes as found in the *prolif.interactions* module

interactions

Dictionnary of interaction functions indexed by class name. For more details, see *prolif.interactions*

Type dict

n_interactions

Number of interaction functions registered by the fingerprint

Type int

ifp

List of interactions fingerprints for the given trajectory.

Type list, optionnal

Notes

You can use the fingerprint generator in multiple ways:

- On a trajectory directly from MDAnalysis objects:

```
In [1]: prot = u.select_atoms("protein")

In [2]: lig = u.select_atoms("resname LIG")

In [3]: fp = prolif.Fingerprint(["HBDonor", "HBAcceptor", "PiStacking",
↪"Hydrophobic"])

In [4]: fp.run(u.trajectory[:5], lig, prot)
Out[4]: <prolif.fingerprint.Fingerprint: 4 interactions: ['Hydrophobic', 'HBDonor
↪', 'HBAcceptor', 'PiStacking'] at 0x7f00b51dad30>

In [5]: fp.to_dataframe()
Out[5]:
ligand           LIG1.G                ...
protein         TYR38.A    TYR109.A   ...    THR355.B    TYR359.B
interaction Hydrophobic Hydrophobic   ... Hydrophobic Hydrophobic
Frame                                 ...
0                 False        True   ...        True        True
1                 False       False   ...        True        True
2                 False        True   ...        True        True
3                  True       False   ...        True        True
4                  True        True   ...        True        True

[5 rows x 31 columns]
```

- On a specific frame and a specific pair of residues:

```
In [6]: u.trajectory[0] # use the first frame
Out[6]: < Timestep 0 with unit cell dimensions [ 89.6909   87.95726 102.02041  90.
↪       90.       90.     ] >

In [7]: prot = prolif.Molecule.from_mda(prot)

In [8]: lig = prolif.Molecule.from_mda(lig)

In [9]: fp.bitvector(lig, prot["ASP129.A"])
Out[9]: array([ True,  True, False, False])
```

- On a specific pair of residues for a specific interaction:

```
In [10]: fp.hbdonor(lig, prot["ASP129.A"]) # ligand-protein
Out[10]: True

In [11]: fp.hbacceptor(prot["ASP129.A"], prot["CYS133.A"]) # protein-protein␣
↪(alpha helix)
Out[11]: True
```

You can also obtain the indices of atoms responsible for the interaction:

```
In [1]: fp.bitvector_atoms(lig, prot["ASP129.A"])
Out[1]:
(array([ True,  True, False, False]),
 [[8, 9], [52, 10], [None, None], [None, None]])

In [2]: fp.hbdonor.__wrapped__(lig, prot["ASP129.A"])
Out[2]: (True, 52, 10)
```

bitvector(*res1*, *res2*)
Generates the complete bitvector for the interactions between two residues. To get the indices of atoms responsible for each interaction, see *bitvector_atoms()*

Parameters

- **res1** (prolif.residue.Residue *or* prolif.molecule.Molecule) – A residue, usually from a ligand
- **res2** (prolif.residue.Residue *or* prolif.molecule.Molecule) – A residue, usually from a protein

Returns **bitvector** – An array storing the encoded interactions between res1 and res2

Return type numpy.ndarray

bitvector_atoms(*res1*, *res2*)
Generates the complete bitvector for the interactions between two residues, and returns the indices of atoms responsible for these interactions

Parameters

- **res1** (prolif.residue.Residue *or* prolif.molecule.Molecule) – A residue, usually from a ligand
- **res2** (prolif.residue.Residue *or* prolif.molecule.Molecule) – A residue, usually from a protein

Returns

- **bitvector** (*numpy.ndarray*) – An array storing the encoded interactions between res1 and res2
- **atoms** (*list*) – A list containing tuples of (res1_atom_index, res2_atom_index) for each interaction

static list_available(*show_hidden=False*)
List interactions available to the Fingerprint class.

Parameters **show_hidden** (*bool*) – Show hidden classes (usually base classes whose name starts with an underscore _. Those are not supposed to be called directly)

run(*traj*, *lig*, *prot*, *residues=None*, *return_atoms=False*, *progress=True*)
Generates the fingerprint on a trajectory for a ligand and a protein

Parameters

- **traj** (*MDAnalysis.coordinates.base.ProtoReader or MDAnalysis.coordinates.base.FrameIteratorSliced*) – Iterate over this Universe trajectory or sliced trajectory object to extract the frames used for the fingerprint extraction
- **lig** (*MDAnalysis.core.groups.AtomGroup*) – An MDAnalysis AtomGroup for the ligand

- **prot** (*MDAnalysis.core.groups.AtomGroup*) – An MDAnalysis AtomGroup for the protein (with multiple residues)
- **residues** (*list or "all" or None*) – A list of protein residues (str, int or *ResidueId*) to take into account for the fingerprint extraction. If "all", all residues will be used. If None, at each frame the *get_residues_near_ligand()* function is used to automatically use protein residues that are distant of 6.0 Å or less from each ligand residue.
- **return_atoms** (*bool*) – For each residue pair and interaction, return indices of atoms responsible for the interaction instead of bits.
- **progress** (*bool*) – Use the tqdm package to display a progressbar while running the calculation

Returns The Fingerprint instance that generated the fingerprint

Return type *prolif.fingerprint.Fingerprint*

Example

```
>>> u = mda.Universe("top.pdb", "traj.nc")
>>> lig = u.select_atoms("resname LIG")
>>> prot = u.select_atoms("protein")
>>> fp = prolif.Fingerprint().run(u.trajectory[:10], lig, prot)
```

run_from_iterable(*lig_iterable*, *prot_mol*, *residues=None*, *return_atoms=False*, *progress=True*)

Generates the fingerprint between a list of ligands and a protein

Parameters

- **lig_iterable** (*list or generator*) – An iterable yielding ligands as *Molecule* objects
- **prot_mol** (prolif.molecule.Molecule) – The protein
- **residues** (*list or "all" or None*) – A list of protein residues (str, int or *ResidueId*) to take into account for the fingerprint extraction. If "all", all residues will be used. If None, at each frame the *get_residues_near_ligand()* function is used to automatically use protein residues that are distant of 6.0 Å or less from each ligand residue.
- **return_atoms** (*bool*) – For each residue pair and interaction, return indices of atoms responsible for the interaction instead of bits.
- **progress** (*bool*) – Use the tqdm package to display a progressbar while running the calculation

Returns The Fingerprint instance that generated the fingerprint

Return type *prolif.fingerprint.Fingerprint*

Example

```
>>> prot = mda.Universe("protein.pdb")
>>> prot = plf.Molecule.from_mda(prot)
>>> lig_iter = plf.mol2_supplier("docking_output.mol2")
>>> fp = plf.Fingerprint()
>>> fp.run_from_iterable(lig_iter, prot)
```

to_bitvectors()

Converts fingerprints to a list of RDKit ExplicitBitVector

Returns **bvs** – A list of `ExplicitBitVect` for each frame

Return type list

Raises **AttributeError** – If the *run()* method hasn't been used

Example

```
>>> from rdkit.DataStructs import TanimotoSimilarity
>>> bv = fp.to_bitvectors()
>>> TanimotoSimilarity(bv[0], bv[1])
0.42
```

to_dataframe(***kwargs*)

Converts fingerprints to a pandas DataFrame

Parameters

- **dtype** (*object or None*) – Cast the input of each bit in the bitvector to this type. If None, keep the data as is
- **drop_empty** (*bool*) – Drop columns with only empty values

Returns **df** – A DataFrame storing the results frame by frame and residue by residue. See *to_dataframe()* for more information

Return type pandas.DataFrame

Raises **AttributeError** – If the *run()* method hasn't been used

Example

```
>>> df = fp.to_dataframe(dtype=np.uint8)
>>> print(df)
ligand             LIG1.G
protein             ILE59                  ILE55       TYR93
interaction   Hydrophobic HBAcceptor Hydrophobic Hydrophobic PiStacking
Frame
0                       0          1           0           0          0
...
```

Detecting interactions between residues — `prolif.interactions`

You can declare your own interaction class like this:

```
from scipy.spatial import distance_matrix

class CloseContact(prolif.Interaction):
    def detect(self, res1, res2, threshold=2.0):
        dist_matrix = distance_matrix(res1.xyz, res2.xyz)
        if (dist_matrix <= threshold).any():
            return True
```

Warning: Your custom class must inherit from *prolif.interactions.Interaction*

The new "CloseContact" class is then automatically added to the list of interactions available to the fingerprint generator:

```
>>> u = mda.Universe(prolif.datafiles.TOP, prolif.datafiles.TRAJ)
>>> prot = u.select_atoms("protein")
>>> lig = u.select_atoms("resname LIG")
>>> fp = prolif.Fingerprint(interactions="all")
>>> df = fp.run(u.trajectory[0:1], lig, prot).to_dataframe()
>>> df.xs("CloseContact", level=1, axis=1)
   ASP129.A  VAL201.A
0         1         1
>>> lmol = prolif.Molecule.from_mda(lig)
>>> pmol = prolif.Molecule.from_mda(prot)
>>> fp.closecontact(lmol, pmol["ASP129.A"])
True
```

class prolif.interactions.**Anionic**(*cation='[+{1-}]'*, *anion='[-{1-}]'*, *distance=4.5*)

Bases: *prolif.interactions._BaseIonic*

Ionic interaction between a ligand (anion) and a residue (cation)

class prolif.interactions.**CationPi**(*cation='[+{1-}]'*, *pi_ring=('a1:a:a:a:a:a:1', 'a1:a:a:a:a:1')*, *distance=4.5*, *angles=(0, 30)*)

Bases: *prolif.interactions._BaseCationPi*

Cation-Pi interaction between a ligand (cation) and a residue (aromatic ring)

class prolif.interactions.**Cationic**(*cation='[+{1-}]'*, *anion='[-{1-}]'*, *distance=4.5*)

Bases: *prolif.interactions._BaseIonic*

Ionic interaction between a ligand (cation) and a residue (anion)

class prolif.interactions.**EdgeToFace**(*centroid_distance=6.0*, *plane_angles=(50, 90)*, ***kwargs*)

Bases: *prolif.interactions.PiStacking*

Edge-to-face Pi-Stacking interaction between a ligand and a residue

class prolif.interactions.**FaceToFace**(*centroid_distance=4.5*, *plane_angles=(0, 40)*, ***kwargs*)

Bases: *prolif.interactions.PiStacking*

Face-to-face Pi-Stacking interaction between a ligand and a residue

class prolif.interactions.**HBAcceptor**(*donor='[#7,#8,#16][H]'*, *acceptor='[N,O,F,-{1-};!+{1-}]'*, *distance=3.5*, *angles=(130, 180)*)

Bases: *prolif.interactions._BaseHBond*

Hbond interaction between a ligand (acceptor) and a residue (donor)

class prolif.interactions.**HBDonor**(*donor='[#7,#8,#16][H]'*, *acceptor='[N,O,F,-{1-};!+{1-}]'*, *distance=3.5*, *angles=(130, 180)*)

Bases: *prolif.interactions._BaseHBond*

Hbond interaction between a ligand (donor) and a residue (acceptor)

class prolif.interactions.**Hydrophobic**(*hydrophobic='[#6,#16,F,Cl,Br,I,At;!$([+{1-},-{1-}])]'*, *distance=4.5*)

Bases: *prolif.interactions._Distance*

Hydrophobic interaction

Parameters

- **hydrophobic** (*str*) – SMARTS query for hydrophobic atoms
- **distance** (*float*) – Cutoff distance for the interaction

class prolif.interactions.**Interaction**

Bases: abc.ABC

Abstract class for interactions

All interaction classes must inherit this class and define a `detect()` method

class prolif.interactions.**MetalAcceptor**(*metal='[Ca,Cd,Co,Cu,Fe,Mg,Mn,Ni,Zn]'*, *ligand='[O,N,-{1-};!+{1-}]'*, *distance=2.8*)

Bases: *prolif.interactions._BaseMetallic*

Metallic interaction between a ligand (chelated) and a residue (metal)

class prolif.interactions.**MetalDonor**(*metal='[Ca,Cd,Co,Cu,Fe,Mg,Mn,Ni,Zn]'*, *ligand='[O,N,-{1-};!+{1-}]'*, *distance=2.8*)

Bases: *prolif.interactions._BaseMetallic*

Metallic interaction between a ligand (metal) and a residue (chelated)

class prolif.interactions.**PiCation**(*cation='[+{1-}]'*, *pi_ring=('a1:a:a:a:a:a:1'*, *'a1:a:a:a:a:1')*, *distance=4.5*, *angles=(0, 30)*)

Bases: *prolif.interactions._BaseCationPi*

Cation-Pi interaction between a ligand (aromatic ring) and a residue (cation)

class prolif.interactions.**PiStacking**(*centroid_distance=6.0*, *shortest_distance=3.8*, *plane_angles=(0, 90)*, *pi_ring=('a1:a:a:a:a:a:1'*, *'a1:a:a:a:a:1')*)

Bases: *prolif.interactions.Interaction*

Pi-Stacking interaction between a ligand and a residue

Parameters

- **centroid_distance** (*float*) – Cutoff distance between each rings centroid
- **shortest_distance** (*float*) – Shortest distance allowed between the closest atoms of both rings
- **plane_angles** (*tuple*) – Min and max values for the angle between the ring planes
- **pi_ring** (*list*) – List of SMARTS for aromatic rings

class prolif.interactions.**XBAcceptor**(*donor='[#6,#7,Si,F,Cl,Br,I]-[Cl,Br,I,At]'*, *acceptor='[#7,#8,P,S,Se,Te,a;!+{1-}][*]'*, *distance=3.5*, *axd_angles=(130, 180)*, *xar_angles=(80, 140)*)

Bases: *prolif.interactions._BaseXBond*

Halogen bonding between a ligand (acceptor) and a residue (donor)

class prolif.interactions.**XBDonor**(*donor='[#6,#7,Si,F,Cl,Br,I]-[Cl,Br,I,At]'*, *acceptor='[#7,#8,P,S,Se,Te,a;!+{1-}][*]'*, *distance=3.5*, *axd_angles=(130, 180)*, *xar_angles=(80, 140)*)

Bases: *prolif.interactions._BaseXBond*

Halogen bonding between a ligand (donor) and a residue (acceptor)

class prolif.interactions.**_BaseCationPi**(*cation='[+{1-}]'*, *pi_ring=('a1:a:a:a:a:a:1'*, *'a1:a:a:a:a:1')*, *distance=4.5*, *angles=(0, 30)*)

Bases: *prolif.interactions.Interaction*

Base class for cation-pi interactions

Parameters

- **cation** (*str*) – SMARTS for cation
- **pi_ring** (*tuple*) – SMARTS for aromatic rings (5 and 6 membered rings only)
- **distance** (*float*) – Cutoff distance between the centroid and the cation
- **angles** (*tuple*) – Min and max values for the angle between the vector normal to the ring plane and the vector going from the centroid to the cation

class prolif.interactions.**_BaseHBond**(*donor='[#7,#8,#16][H]'*, *acceptor='[N,O,F,-{1-};!+{1-}]'*, *distance=3.5*, *angles=(130, 180)*)

Bases: *prolif.interactions.Interaction*

Base class for Hydrogen bond interactions

Parameters

- **donor** (*str*) – SMARTS for `[Donor]-[Hydrogen]`
- **acceptor** (*str*) – SMARTS for `[Acceptor]`
- **distance** (*float*) – Cutoff distance between the donor and acceptor atoms
- **angles** (*tuple*) – Min and max values for the `[Donor]-[Hydrogen]...[Acceptor]` angle

class prolif.interactions.**_BaseIonic**(*cation='[+{1-}]'*, *anion='[-{1-}]'*, *distance=4.5*)

Bases: *prolif.interactions._Distance*

Base class for ionic interactions

class prolif.interactions.**_BaseMetallic**(*metal='[Ca,Cd,Co,Cu,Fe,Mg,Mn,Ni,Zn]'*, *ligand='[O,N,-{1-};!+{1-}]'*, *distance=2.8*)

Bases: *prolif.interactions._Distance*

Base class for metal complexation

Parameters

- **metal** (*str*) – SMARTS for a transition metal
- **ligand** (*str*) – SMARTS for a ligand
- **distance** (*float*) – Cutoff distance

class prolif.interactions.**_BaseXBond**(*donor='[#6,#7,Si,F,Cl,Br,I]-[Cl,Br,I,At]'*, *acceptor='[#7,#8,P,S,Se,Te,a;!+{1-}][*]'*, *distance=3.5*, *axd_angles=(130, 180)*, *xar_angles=(80, 140)*)

Bases: *prolif.interactions.Interaction*

Base class for Halogen bond interactions

Parameters

- **donor** (*str*) – SMARTS for `[Donor]-[Halogen]`
- **acceptor** (*str*) – SMARTS for `[Acceptor]-[R]`
- **distance** (*float*) – Cutoff distance between the halogen and acceptor atoms
- **axd_angles** (*tuple*) – Min and max values for the `[Acceptor]...[Halogen]-[Donor]` angle
- **xar_angles** (*tuple*) – Min and max values for the `[R]-[Acceptor]...[Halogen]` angle

Notes

Distance and angle adapted from Auffinger et al. PNAS 2004

class prolif.interactions.**_Distance**(*lig_pattern*, *prot_pattern*, *distance*)

Bases: *prolif.interactions.Interaction*

Generic class for distance-based interactions

Parameters

- **lig_pattern** (*str*) – SMARTS pattern for atoms in ligand residues
- **prot_pattern** (*str*) – SMARTS pattern for atoms in protein residues
- **distance** (*float*) – Cutoff distance, measured between the first atom of each pattern

class prolif.interactions.**_InteractionMeta**(*name*, *bases*, *namespace*, ***kwargs*)

Bases: abc.ABCMeta

Metaclass to register interactions automatically

prolif.interactions.**get_mapindex**(*res*, *index*)

Get the index of the atom in the original molecule

Parameters

- **res** (prolif.residue.Residue) – The residue in the protein or ligand
- **index** (*int*) – The index of the atom in the *Residue*

Returns **mapindex** – The index of the atom in the *Molecule*

Return type int

1.6.11 Residues

Residue-related classes — `prolif.residue`

ResidueId

class prolif.residue.**ResidueId**(*name: Optional[str] = None*, *number: Optional[int] = None*, *chain: Optional[str] = None*)

A unique residue identifier

Parameters

- **name** (*str or None, optionnal*) – 3-letter residue name
- **number** (*int or None, optionnal*) – residue number
- **chain** (*str or None, optionnal*) – 1-letter protein chain

classmethod from_atom(*atom*)

Creates a ResidueId from an RDKit atom

Parameters **atom** (*rdkit.Chem.rdchem.Atom*) – An atom that contains an RDKit `AtomMonomerInfo`

classmethod from_string(*resid_str*)

Creates a ResidueId from a string

Parameters **resid_str** (*str*) – A string in the format `<3-letter code><residue number>.<chain>` All arguments are optionnal, and the dot should be present only if the chain identifier is also present

Examples

string	Corresponding ResidueId
"ALA10.A"	`ResidueId("ALA", 10, "A")`
"GLU33"	`ResidueId("GLU", 33, None)`
"LYS.B"	`ResidueId("LYS", None, "B")`
"ARG"	`ResidueId("ARG", None, None)`
"5.C"	`ResidueId(None, 5, "C")`
"42"	`ResidueId(None, 42, None)`
".D"	`ResidueId(None, None, "D")`
""	`ResidueId(None, None, None)`

ResidueGroup

class prolif.residue.**ResidueGroup**(*residues: List[prolif.residue.Residue]*)

A container to store and retrieve Residue instances easily

Parameters **residues** (*list*) – A list of *Residue*

n_residues

Number of residues in the ResidueGroup

Type int

Notes

Residues in the group can be accessed by *ResidueId*, string, or index. See the *Molecule* class for an example. You can also use the *select()* method to access a subset of a ResidueGroup.

select(*mask*)
Locate a subset of a ResidueGroup based on a boolean mask

Parameters **mask** (*numpy.ndarray*) – A 1D array of `dtype=bool` with the same length as the number of residues in the ResidueGroup. The mask should be constructed by using conditions on the "name", "number", and "chain" residue attributes as defined in the *ResidueId* class

Returns **rg** – A subset of the original ResidueGroup

Return type *prolif.residue.ResidueGroup*

Examples

```
>>> rg
<prolif.residue.ResidueGroup with 200 residues at 0x7f9a68719ac0>
>>> rg.select(rg.chain == "A")
<prolif.residue.ResidueGroup with 42 residues at 0x7fe3fdb86ca0>
>>> rg.select((10 <= rg.number) & (rg.number < 30))
<prolif.residue.ResidueGroup with 20 residues at 0x7f5f3c69aaf0>
>>> rg.select((rg.chain == "B") & (np.isin(rg.name, ["ASP", "GLU"])))
<prolif.residue.ResidueGroup with 3 residues at 0x7f5f3c510c70>
```

As seen in these examples, you can combine masks with different operators, similarly to numpy boolean indexing or pandas `loc()` method

- AND –> `&`
- OR –> `|`
- XOR –> `^`
- NOT –> `~`

1.6.12 Helper functions

Helper functions — `prolif.utils`

prolif.utils.**get_residues_near_ligand**(*lig*, *prot*, *cutoff=6.0*)
Detects residues close to a reference ligand

Parameters

- **lig** (prolif.molecule.Molecule) – Select residues that are near this ligand
- **prot** (prolif.molecule.Molecule) – Protein containing the residues
- **cutoff** (*float*) – If any interatomic distance between the ligand reference points and a residue is below or equal to this cutoff, the residue will be selected

Returns **residues** – A list of unique *ResidueId* that are close to the ligand

Return type list

prolif.utils.**to_bitvectors**(*df*)
Converts an interaction DataFrame to a list of RDKit ExplicitBitVector

Parameters **df** (*pandas.DataFrame*) – A DataFrame where each column corresponds to an interaction between two residues

Returns **bv** – A list of `ExplicitBitVect` for each frame

Return type list

Example

```
>>> from rdkit.DataStructs import TanimotoSimilarity
>>> bv = prolif.to_bitvectors(df)
>>> TanimotoSimilarity(bv[0], bv[1])
0.42
```

prolif.utils.**to_dataframe**(*ifp*, *interactions*, *index_col='Frame'*, *dtype=None*, *drop_empty=True*)
Converts IFPs to a pandas DataFrame

Parameters

- **ifp** (*list*) – A list of dict in the format {key: bitvector} where "bitvector" is a numpy.ndarray obtained by running the *bitvector()* method of a *Fingerprint*, and "key" is a tuple of ligand and protein ResidueId. Each dictionnary must also contain an entry that will be used as an index, typically a frame number.
- **interactions** (*list*) – A list of interactions, in the same order as the bitvector.
- **index_col** (*str*) – The dictionnary key that will be used as an index in the DataFrame
- **dtype** (*object or None*) – Cast the input of each bit in the bitvector to this type. If None, keep the data as is.
- **drop_empty** (*bool*) – Drop columns with only empty values

Returns **df** – A 3-levels DataFrame where each ligand residue, protein residue, and interaction type are in separate columns

Return type pandas.DataFrame

Example

```
>>> df = prolif.to_dataframe(results, fp.interactions.keys(), dtype=int)
>>> print(df)
ligand             LIG1.G
protein             ILE59                  ILE55       TYR93
interaction   Hydrophobic HBAcceptor Hydrophobic Hydrophobic PiStacking
Frame
0                       0          1           0           0          0
...
```

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

p

INDEX

Symbols

A

B

C

E

F

G

H

I

L

M

N

P

R

S

T

X