SMITER Documentation

Release 0.1.0

Manuel Kösters

Mar 04, 2021

CONTENTS:

CHAPTER

ONE

SMITER

1.1 Summary

Python library to create synthetic mzMLs file based on chemical formulas. All molecules can be simulated du to abstraction to chemical formulas.

1.2 Abstract

SMITER (Synthetic mzML writer) is a python-based command-line tool designed to simulate LC-MS/MS runs. It enables the simulation of any biomolecule since all calculations are based on the chemical formulas. As SMITER features a modular design, noise and fragmentation models can easily be implemented or adapted. By default, SMITER uses an established noise model and offers several methods for peptide fragmentation or two models for nucleoside fragmentation. Due to the rich python ecosystem, other modules, e.g. for retention time prediction, can easily be implemented for the tailored simulation of any molecule of choice. This allows for the facile creation of defined gold-standard-LC-MS/MS datasets for any type of experiment. Such gold standards, where the ground truth is known, are required in computational mass spectrometry to test new algorithms and to improve parameters for existing ones. Similarly, gold-standard datasets can be used to evaluate analytical hurdles e.g. by predicting co-elution and co-fragmentation of molecules. As these challenges hinder the detection or quantification of co-eluents, a comprehensive simulation can identify and thus prevent such difficulties before performing actual MS experiments. SMITER allows to create such datasets easily, fast and efficiently

1.3 Features

- simulate mass spectrometry data for any biomolecule
- usage of highly-accurate isotopic patterns enabled by pyQms
- feature scaling by gauss-, gamma- and exponentially-modified gaussian distributions
- m/z-and intensity noise injection (uniform noise or a noise model that combines general noise with intensity-specific noise)

- MS2 fragmentation for peptides and modified nucleosides.
- Free software: MIT license
- Documentation: https://smiter.readthedocs.io.

1.4 Download and Installation

SMITER requires Python 3.6 or higher.

There are two recommended ways for installing SMITER

- Installation via pip
- Installation from the source (GitHub)

1.5 Installation via pip

Execute the following command from your command line:

```
user@localhost:~$ pip install smiter
```

1.6 Installation from source

Clone the GitHub repo GitHub:

```
user@localhost:~$ git clone https://github.com/LeidelLab/SMITER.git
```

Install the requirements and SMITER:

```
user@localhost:~$ cd smiter
user@localhost:~/smiter$ pip install -r requirements.txt
user@localhost:~/smiter$ python setup.py install
```

Note: We recommend using an virtual environment when using SMITER

1.7 Testing

To test the package and correct installation:

```
user@localhost:~/smiter$ tox
```

1.8 Copyrights

1.9 Contact

Prof. Dr. Sebastian Leidel University of Bern Department of Chemistry, Biochemistry and Pharmaceutical Sciences Freiestrasse 3 3012 Bern Switzerland

1.10 Citation

Please do not forget to cite SMITER:

<ref>

1.11 Credits

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

CHAPTER

TWO

INSTALLATION

2.1 Stable release

To install SMITER, run this command in your terminal:

```
$ pip install smiter
```

This is the preferred method to install SMITER, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

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

You can either clone the public repository:

```
$ git clone git://github.com/MKoesters/smiter
```

Or download the tarball:

```
$ curl -OJL https://github.com/MKoesters/smiter/tarball/master
```

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

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

CHAPTER

THREE

USAGE

To use SMITER in a project:

```
import smiter
```

CHAPTER

FOUR

SMITER

4.1 smiter package

4.1.1 Subpackages

smiter.ext package

Submodules

smiter.ext.nucleoside_fragment_kb module

Mapping nucleoside names to its fragments.

smiter.ext.trim_lines module

`smiter.ext.trim_lines.`**`main`**`()`

Module contents

External resources.

4.1.2 Submodules

4.1.3 smiter.cli module

4.1.4 smiter.fragmentation_functions module

Callables for fragmenting molecules.

Upon calling the callabe, a list/np.array of mz and intensities should be returned. Arguments should be passed via *args and **kwargs

`class` `smiter.fragmentation_functions.`**`AbstractFragmentor`**

Bases: `abc.ABC`

Summary.

`abstract fragment`(*entity*)

Summary.

Args: entity (TYPE): Description

class smiter.fragmentation_functions.**LipidFragmentor**(*lipid_input_csv: Optional[str] = None*, *raise_error_for_non_existing_fragments=True*)

Bases: *smiter.fragmentation_functions.AbstractFragmentor*

Summary.

fragment(*entities: Union[list, str]*, *raise_error_for_non_existing_fragments=False*)

Summary.

Args: entity (TYPE): Description

class smiter.fragmentation_functions.**NucleosideFragmentor**(*nucleotide_fragment_kb: Optional[Dict[str, dict]] = None*, *raise_error_for_non_existing_fragments=True*)

Bases: *smiter.fragmentation_functions.AbstractFragmentor*

Summary.

fragment(*entities: Union[list, str]*, *raise_error_for_non_existing_fragments=False*)

Summary.

Args: entity (TYPE): Description

class smiter.fragmentation_functions.**PeptideFragmentor**(**args*, ***kwargs*)

Bases: *smiter.fragmentation_functions.AbstractFragmentor*

Summary.

fragment(*entities*)

Summary.

Args: entity (TYPE): Description

class smiter.fragmentation_functions.**PeptideFragmentorPyteomics**(**args*, ***kwargs*)

Bases: *smiter.fragmentation_functions.AbstractFragmentor*

fragment(*entities*)

Summary.

Args: entity (TYPE): Description

4.1.5 smiter.lib module

Core functionality.

smiter.lib.**calc_mz**(*mass: float*, *charge: int*)

Calculate m/z.

Args: mass (TYPE): Description charge (TYPE): Description

smiter.lib.**check_mzml_params**(*mzml_params: dict*) → dict

Summary.

Args: mzml_params (dict): Description

Returns: dict: Description

Raises: Exception: Description

smiter.lib.**check_peak_properties**(*peak_properties: dict*) → dict
Summary.

Args: peak_properties (dict): Description

Returns: dict: Description

Raises: Exception: Description

smiter.lib.**csv_to_peak_properties**(*csv_file*)

smiter.lib.**peak_properties_to_csv**(*peak_properties*, *csv_file*)

4.1.6 smiter.noise_functions module

Callables for injection noise into scans.

Upon calling the callabe, a list/np.array of mz and intensities should be returned. Arguments should be passed via *args and **kwargs

class smiter.noise_functions.**AbstractNoiseInjector**(**args*, ***kwargs*)
Bases: abc.ABC

Summary.

abstract inject_noise(*scan*, **args*, ***kwargs*)
Main noise injection method.

Args: scan (Scan): Scan object *args: Description **kwargs: Description

class smiter.noise_functions.**GaussNoiseInjector**(**args*, ***kwargs*)
Bases: *smiter.noise_functions.AbstractNoiseInjector*

inject_noise(*scan*, **args*, ***kwargs*)
Main noise injection method.

Args: scan (Scan): Scan object *args: Description **kwargs: Description

class smiter.noise_functions.**JamssNoiseInjector**(**args*, ***kwargs*)
Bases: *smiter.noise_functions.AbstractNoiseInjector*

inject_noise(*scan*, **args*, ***kwargs*)
Main noise injection method.

Args: scan (Scan): Scan object *args: Description **kwargs: Description

class smiter.noise_functions.**PPMShiftInjector**(**args*, ***kwargs*)
Bases: *smiter.noise_functions.AbstractNoiseInjector*

inject_noise(*scan*, **args*, ***kwargs*)
Main noise injection method.

Args: scan (Scan): Scan object *args: Description **kwargs: Description

class smiter.noise_functions.**UniformNoiseInjector**(**args*, ***kwargs*)
Bases: *smiter.noise_functions.AbstractNoiseInjector*

inject_noise(*scan*, **args*, ***kwargs*)
Main noise injection method.

Args: scan (Scan): Scan object *args: Description **kwargs: Description

4.1.7 smiter.peak_distribution module

Distribution funtion for chromo peaks.

Attributes: distributions (dict): mapping distribution name to distribution function

smiter.peak_distribution.**gamma_dist**(*x: float*, *a: float = 5*, *scale: float = 0.33*)
Calc gamma distribution.

Args: x (float): Description a (float, optional): Description scale (float, optional): Description

Returns: float: y

smiter.peak_distribution.**gauss_dist**(*x: float*, *sigma: float = 1*, *mu: float = 0*)
Calc Gauss distribution.

Args: x (float): x sigma (float, optional): standard deviation mu (float, optional): mean

Returns: float: y

smiter.peak_distribution.**gauss_tail**(*x: float*, *mu: float*, *sigma: float*, *scan_start_time: float*, *h: float = 1*, *t: float = 0.2*, *f: float = 0.01*) → float

4.1.8 smiter.synthetic_mzml module

Main module.

class smiter.synthetic_mzml.**Scan**(*data: Optional[dict] = None*)
Bases: dict

Summary.

property i
Summary.

property id
Summary.

property ms_level
Summary.

Returns: TYPE: Description

property mz
Summary.

property precursor_charge
Summary.

property precursor_i
Summary.

property precursor_mz
Summary.

property retention_time
Summary.

smiter.synthetic_mzml.**generate_interval_tree**(*peak_properties*)
Conctruct an interval tree containing the elution windows of the analytes.

Args: peak_properties (dict): Description

Returns: IntervalTree: Description

smiter.synthetic_mzml.**generate_molecule_isotopologue_lib**(*peak_properties: Dict[str, dict], charges: Optional[List[int]] = None, trivial_names: Optional[Dict[str, str]] = None*)

Summary.

Args: molecules (TYPE): Description

smiter.synthetic_mzml.**generate_scans**(*isotopologue_lib: dict, peak_properties: dict, interval_tree: intervaltree.intervaltree.IntervalTree, fragmentor:* smiter.fragmentation_functions.AbstractFragmentor, *noise_injector:* smiter.noise_functions.AbstractNoiseInjector, *mzml_params: dict*)

Summary.

Args: isotopologue_lib (TYPE): Description peak_properties (TYPE): Description fragmentation_function (A): Description mzml_params (TYPE): Description

smiter.synthetic_mzml.**rescale_intensity**(*i: float, rt: float, molecule: str, peak_properties: dict, isotopologue_lib: dict*)

Rescale intensity value for a given molecule according to scale factor and distribution function.

Args: i (TYPE): Description rt (TYPE): Description molecule (TYPE): Description peak_properties (TYPE): Description isotopologue_lib (TYPE): Description

Returns: TYPE: Description

smiter.synthetic_mzml.**write_mzml**(*file: Union[str, _io.TextIOWrapper], peak_properties: Dict[str, dict], fragmentor:* smiter.fragmentation_functions.AbstractFragmentor, *noise_injector:* smiter.noise_functions.AbstractNoiseInjector, *mzml_params: Dict[str, Union[int, float, str]]*) → str

Write mzML file with chromatographic peaks and fragment spectra for the given molecules.

Args: file (Union[str, io.TextIOWrapper]): Description molecules (List[str]): Description fragmentation_function (Callable[[str], List[Tuple[float, float]]], optional): Description peak_properties (Dict[str, dict], optional): Description

smiter.synthetic_mzml.**write_scans**(*file: Union[str, _io.TextIOWrapper], scans: List[Tuple[*smiter.synthetic_mzml.Scan, *List[*smiter.synthetic_mzml.Scan*]]]*) → None

Generate given scans to mzML file.

Args: file (Union[str, io.TextIOWrapper]): Description scans (List[Tuple[Scan, List[Scan]]]): Description

Returns: None: Description

4.1.9 Module contents

Top-level package for SMITER.

CHAPTER

FIVE

CONTRIBUTING

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://github.com/MKoesters/smiter/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.1.2 Fix Bugs

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

5.1.3 Implement Features

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

5.1.4 Write Documentation

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

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/MKoesters/smiter/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

5.2 Get Started!

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

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

```
$ git clone git@github.com:your_name_here/smiter.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv smiter
$ cd smiter/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 smiter tests
$ python setup.py test or pytest
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.5, 3.6, 3.7 and 3.8, and for PyPy. Check https://travis-ci.com/MKoesters/smiter/pull_requests and make sure that the tests pass for all supported Python versions.

5.4 Tips

To run a subset of tests:

```
$ pytest tests.test_smiter
```

5.5 Deploying

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

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

Travis will then deploy to PyPI if tests pass.

CHAPTER

SIX

CREDITS

6.1 Development Lead

- Manuel Kösters <manuel.koesters@dcb.unibe.ch>

6.2 Contributors

None yet. Why not be the first?

CHAPTER

SEVEN

HISTORY

7.1 0.1.0 (2020-03-16)

- First release on PyPI.

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

S

INDEX

P

R

S

U

W