BIOINFORMATICS APPLICATIONS NOTE
Vol. 28 no. 7 2012, pages 1052–1053
doi:10.1093/bioinformatics/bts066

Data and text mining

Advance Access publication February 2, 2012

pymzML—Python module for high-throughput bioinformatics on mass spectrometry data

Till Bald†, Johannes Barth†, Anna Niehues, Michael Specht, Michael Hippler and Christian Fufezan*

Institute of Plant Biology and Biotechnology, University of Muenster, 48143 Muenster, Germany

Associate Editor: Olga Troyanskaya

ABSTRACT

Summary: pymzML is an extension to Python that offers (i) an easy access to mass spectrometry (MS) data that allows the rapid development of tools, (ii) a very fast parser for mzML data, the standard data format in MS and (iii) a set of functions to compare or handle spectra.

Availability and implementation: pymzML requires Python2.6.5+ and is fully compatible with Python3. The module is freely available on http://pymzml.github.com or pypi, is published under LGPL license and requires no additional modules to be installed.

Contact: christian@fufezan.net

Received on October 12, 2011; revised on January 11, 2012; accepted on January 30, 2012

1 INTRODUCTION

Mass spectrometry (MS) has evolved into a very diverse field that relies heavily on high-throughput bioinformatic tools. Due to the increasing complexity of the questions asked and biological problems addressed, standard tools might not be sufficient and tailored tools still have to be developed. For this a rapid development cycle and efficient data handling are required.

The development of such tools has been hindered by proprietary data formats. To overcome this problem there are currently two approaches to develop a universal access to MS: (i) the mzAPI (Askenazi *et al.*, 2009) project seeks to develop a direct interface to raw proprietary data and (ii) mzML standard by the HUPO Proteomics Standards Initiative (Martens *et al.*, 2011) that allows to convert the raw data into a standardized xml format.

In order to rapidly develop bioinformatic tools that can explore MS data one needs a portable, robust, yet quick interface. pymzML is such an interface to mzML files based on the Python scripting language, which is very suitable for such a task.

Scripting languages have several advantages compared with compiled programs and although compiled programs tend to run faster, scripting languages can already compete successfully in some tasks. For example, XML parsing is extremely optimized in Python due to the cElementTree module, which allows XML parsing in a fraction of time compared to classical C/C++ XML libraries, such as libxml2 or sgmlop. Therefore, it seems natural that a well designed Python mzML parser can be competitive while additionally offering the advantages of a scripting language.

2 RESULTS

Basic usage: pymzML offers an interface between the Python scripting language and MS data. pymzML has several distinctive advantages over other currently available approaches since it is based on a scripting language that offers rapid evaluation, prototyping, portability and even complex evaluation of large MS datasets due to its efficient XML parsing engine.

The following examples are executed within the Python console (indicated by '>>>') but can equally be incorporated in standalone scripts. A detailed documentation and example scripts can be found on the pymzML website and within the pymzML package. The pymzML file object is declared as follows:

```
>>> import pymzml
>>> msrun = pymzml.run.Reader("big-1.0.0.mzML")
```

The input files can be plain or compressed mzML files. Initialization of the Run class accepts additional keywords, e.g. precision of MS1 or MSn spectra and extra accessions. Please refer to the manual for a more detailed description of the optional keywords. The run object returns an iterator, hence one can loop over all spectra and/or chromatograms using classic Python syntax.

Additionally, one can retrieve a specific spectrum by its nativeID via random access[1] using `msrun[1]`, `msrun['TIC']` to access the chromatogram or in case of MRM experiments as e.g. `msrun['transition_445-672']`.

The pymzML spectrum object offers basic information on the spectrum that can be accessed like a Python dictionary. The keys in this dictionary are the accession numbers (e.g. 'MS:1000511') or the name of the accession (e.g. 'ms level'), as they are defined by the HUPO Proteomics Standards Initiative in the mzML vocabulary, i.e. in the OBO files. Both keys of the spectrum dictionary point to the value that has been extracted from the mzML file. Association of MS tag to name is done on the basis of the OBO files that are supplied in the pymzML package. The pymzML package offers a script queryOBO.py that can be used to translate between MS tags and names. It is worth noting that the definition of MS tags and their attribute that the user wants to be associated with the tag and the trivial name is a feature. For example, the scan time is

*To whom correspondence should be addressed.

†The authors wish it to known that, in their opinion, the first two authors should be regarded as joint First Authors.

[1]This requires the mzML file not to be compressed or truncated by a conversion program.

Table 1. Parsing time of BSA1.mzML

Decoding of m/z & i data	OpenMS	ProteoWizard	pymzML
No	1.9s	**0.7s**	1.3s
Yes	**3.1s**	5.9s	3.8s

Benchmarks run on a Xeon 2.8 ghz, executables used: *OpenMS (1.8)*: FileInfo, DTAExtractor, *ProteoWizard (2.2)*: msbenchmark, mscat and *pymzML*: simpleParser.py, simpleParser_d.py, for 'no decoding' and 'decoding' of m/z & intensity arrays, respectively. Fastest parser time shown in bold.

associated with two values, the actual time and its unit. From a programming point of view, access to the time as float by calling `spec['scan time']` is desirable. Nevertheless, pymzML also offers an iterator over the original Python xmltree object that has been used to initialize the spectrum via `spectrum.xmlTree`, hence all MS-tags, their values and associated tags can be retrieved.

Data that is associated with the spectrum object, i.e. mass over charge (m/z) or intensity values can be accessed by the `.mz` or `.intensity` properties, respectively, which are iterators themselves. The `.peaks` property also offers an iterator that returns mz and intensity as a tuple for each peak.

Since mass spectrometry data can be measured in profile mode, the `.centroidedPeak` property offers an iterator that performs a simple Gauss fit on the profiled data, thereby returning the fitted m/z and maximum intensity of the bell shape curve as tuple. Basic visualization of spectra using XHTML and SVG can be done using the `pymzml.plot` submodule.

Advanced usage: the spectrum class also offers other functions, such as deconvolution, estimation of similarity of spectra, simple noise reduction, addition of spectra and simple arithmetics on the intensity values by multiplication or division. An averaged spectrum can be created during parsing as follows:

```
>>> t = pymzml.spec.Spectrum()
>>> for s in msrun:
>>>     t += s / s['total ion current']
```

The addition of spectra is done by creating Gaussian distributions around the centroided peak. This reprofiling is done internally as part of the addition and its values can be accessed by the `.reprofiledPeaks` property of the spectrum. To reduce the computational overhead one can remove noise by calling the `.removeNoise()` method of the spectrum class. This method accepts different modes of noise reduction. Deconvoluted peaks can be accessed for high precision spectra by calling the `.deconvolutedPeaks` property. The pymzML package contains detailed example scripts for all these methods.

Code design: pymzML gains its parsing efficiency from the intrinsically efficient Python XML parser and two code design decisions:

(i) Only one temporary spectrum is used during parsing. This reduces the memory and time that is required for each new spectrum initialization. This restriction however requires a function call `spectrum.deRef()` onto the temporary spectrum if the user plans to store the spectrum object for further analysis.

(ii) The data of the spectrum, i.e. m/z and intensity are decoded only on demand, i.e. if the user evokes one of the iterators of the spectrum data.

Benchmark: pymzML was benchmarked against the C/C++ libraries, i.e. the OpenMS tool box (Sturm *et al.*, 2008) and the ProteoWizard (Kessner *et al.*, 2008). BSA1.mzml, which is part of the OpenMS tool box, was used as a test file. Table 1 lists the parsing times of the test file. Clearly from the parsing efficiency point of view the pymzML scripting approach can compete very well with the C/C++ implementations.

Parsing efficiency is, however, not the only important benchmark that should be taken into account. The time needed to code, e.g. simpleParser_d.py stands in no relation to the C/C++ alternatives. SimpleParser_d.py, which in principle converts the complete mzML file into DTA files can be written with seven lines of code.

3 DISCUSSION

pymzML is an object-oriented Python module that offers an interface between the mzML file format and the Python interpreter. The scripting approach, i.e. using pymzML in combination with Python brings several distinct advantages over compiled languages. Scripts that analyze mzML data can be written with a few lines of code in a short time, i.e. fast prototyping is possible. The scripts are very portable, i.e. executable on every platform that supports Python. The source code is readable and no additional compiling or other libraries are required. The scripting approach also allows to dig into a vast amount of third party packages that can be integrated seamlessly. Another unmatched advantage is that pymzML scripts can be wrapped with an additional 100 lines accessing the Python multiprocessing module to take full advantage of multiprocessor systems, thereby reducing the execution time to a fraction of time. This advantage is even less matched using the C or C++ libraries.

Given all these advantages, pymzML is the optimal tool to develop high-throughput programs/scripts that analyze MS data.

Funding: BMBF (0315265C to M.H.) DFG (to C.F., Fu-780/2).

Conflict of Interest: none declared.

REFERENCES

Askenazi,M. *et al.* (2009) mzAPI: a new strategy for efficiently sharing mass spectrometry data. *Nat. Methods*, **6**, 240–241.

Kessner,D. *et al.* (2008) ProteoWizard: open source software for rapid proteomics tools development. *Bioinformatics*, **24**, 2534–2536.

Martens,L. *et al.* (2011) mzML–a community standard for mass spectrometry data. *Mol. Cell. Proteomics : MCP*, **10**, R110.000133.

Sturm,M. *et al.* (2008) OpenMS An open-source software framework for mass spectrometry. *BMC Bioinformatics*, **9**, 163.