BioTK Documentation

Release

BioTK contributors

March 11, 2014

Contents

CHAPTER 1

Overview

BioTK is a Python toolkit, containing a library and scripts, for various bioinformatics tasks:

- Differential expression analysis on microarray and RNA-seq data
- Functional and downstream analysis of DE gene lists
- Ontology handling
- Text mining, ranging from shallow NLP to relation extraction
- Efficient storage and querying of sets of genomic intervals (similar to BEDTools, GenomicRanges, etc.)

CHAPTER 2

Tutorial

2.1 Expression analysis

BioTK aims to provide an alternative to the standard R/Bioconductor environment to perform run-of-the-mill differential expression analyses. Thus, BioTK has the ability to perform all the standard steps in a differential expression analysis pipeline:

1. Loading raw or preprocessed data
2. Preprocessing and normalizing the data
3. Finding differentially expressed probes/genes between conditions
4. Analyses of DE gene lists: - Performing enrichment analyses against ontologies - Visualizing expression or DE results as heatmaps or networks

There are also features for downstream analyses of and methods to take large collections of expression data, from GEO, in-house data, or a combination thereof, and use these collections for large-scale meta-analysis.

Todo

- put a simple example of a complete-ish analysis here
- possibly explain important data structures?

2.1.1 Loading expression data

From Affymetrix CEL files

From GEO

From RNA-seq aligned reads

2.1.2 Normalizing expression data

Quantile normalization

2.1.3 Differential expression

Currently, the available differential expression algorithms are:

- t-test
- ANOVA
- SAM

In the future, we plan to provide either a port or a simplified Python interface to the R package limma, which is one of the most popular tools for finding DE genes.

T-test

ANOVA

SAM

2.1.4 Visualization

Heatmap

2.1.5 Enrichment analysis

2.1.6 Meta-analysis

BioTK can store large amounts of expression data from multiple experiments and even multiple organisms and efficiently perform meta-analyses on this data. Please see *Transcript expression meta-analysis*.

2.2 Transcript expression meta-analysis

BioTK provides a way to store up to millions of expression vectors, along with associated probe mappings and phenotype/sample data, in HDF5 format on your hard drive for efficient querying and meta-analysis. HDF5 has many benefits, including query efficiency and the ability to store all your expression data in a single file that can be easily backed up or transferred to different computers.

If you like, you can easily store and query all available GEO samples for your model organism(s) of choice.

2.2.1 Importing data

2.2.2 Performing a meta-analysis

2.2.3 Finding coexpressed genes

2.2.4 Inferring categories for genes or samples

CHAPTER 3

Development information

3.1 General development information

3.1.1 Contributors

- Cory Giles
- Mikhail Dozmorov

3.1.2 License

BioTK is licensed under the GNU Affero General Public License version 3 (or any later version thereof, at your option).

3.2 How to contribute

If you'd like to contribute to BioTK (yay!), please follow the guidelines below. In summary:

1. Write the code
2. Write documentation
3. Write tests (that pass)
4. Submit a pull request on GitHub

Although we use PEP8 for variable names and Numpy docstring format (with a few exceptions), don't obsess over following every detail of these format guides, especially if you are new to them. Just do what you can, and we can fix minor errors later.

3.2.1 Writing code

Naming conventions

Code mostly follows standard Python naming conventions from PEP8 (lower_case_with_underscore for variables and functions, CamelCase for classes, private variables or methods with leading underscore, etc.).

There is one exception: modules and packages. Because of the huge preponderance of acronyms in biomedicine, modules are CamelCase with acronyms all uppercase if they describe a file format, external program, or well-known algorithm. Otherwise, they are lower-case.

Examples:

- BioTK.io.BEDGraph
- BioTK.io.Aspera
- BioTK.text.parse
- BioTK.text.AhoCorasick

3.2.2 Documentation

If you want to contribute some code, the most important kind of documentation to provide is docstrings. Sphinx documentation and in-code comments are nice to have, but not crucial.

Sphinx documentation

High level information and tutorials are written in the doc/ directory in reStructuredText format, and built into HTML and other formats using Sphinx. These docs are automatically mirrored to http://BioTK.readthedocs.org/ .

A useful reStructuredText primer can be found at http://docutils.sourceforge.net/docs/user/rst/quickref.html .

Docstrings

Modules, and public classes and functions inside them, need docstrings. Keep it high level, explaining *what* the module/function and the parameters are doing, not *how* they are doing it. Provide citations to the algorithm's paper if appropriate. Generally speaking, the more "public" a function/class is, the more documentation it needs. If it should be rarely or never directly called by a user, it may only need one line. Conversely, a large class with many methods may need quite extensive documentation.

The docstrings are written in Numpy format, with one exception: in BioTK, there is always a leading newline on the first line. Thus, instead of:

```
def add(a, b):
    """The sum of two numbers.

    (rest of docstring ...)
    """
```

we use:

```
def add(a, b):
    """
    The sum of two numbers.

    (rest of docstring ...)
    """
```

The numpy docstring format is described here:

- https://github.com/numpy/numpy/blob/master/doc/HOWTO_DOCUMENT.rst.txt

And many good examples are here:

- http://sphinx-doc.org/latest/ext/example_numpy.html

Internal code comments

Functions and class methods should be short enough, and the code should be clear enough, that code comments should mostly be unnecessary. Use good judgment: if a particularly tricky method is being used, it may need some explanation, but in general keep comments high level.

You can mark "wishlist" items with a TODO comment, and items that are actually broken or need urgent attention with "FIXME" (obviously the latter should be done sparingly).

3.2.3 Unit tests

They are written using the py.test framework, and are placed in the test/ directory, with a directory structure that mirrors the structure of BioTK.

If possible, avoid tests that take a long time to run or require network access.

CHAPTER 4

Indices and tables

- *genindex*
- *modindex*
- *search*