MutationInfo Documentation

Release 1.3.0

Alexandros Kanterakis

Mar 14, 2017

Contents

MutationInfo is a python package to extract the position, the reference and the alternative sequence of a genomic variant. It accepts variants in dbSNP rs format or in HGVS format.

Source: https://github.com/kantale/MutationInfo/ License: MIT License

Contents:

CHAPTER 1

Introduction

The main purpose of MutationInfo is to simplify the process of locating a variant in a dataset (i.e. of sequences or variants) that is aligned in a human reference genome (for example hg19 or hg38). It mainly wraps a collection of existing tools with a simple interface.

Example:

```
from MutationInfo import MutationInfo
mi = MutationInfo()

# RS variant
print mi.get_info('rs53576')
{'chrom': '3', 'notes': '', 'source': 'UCSC',
'genome': 'hg19', 'offset': 8804371L, 'alt': 'G', 'ref': 'A'}

# HGVS variant
print mi.get_info('NM_000367.2:c.-178C>T')
{'chrom': '6', 'notes': '', 'source': 'counsyl_hgvs_to_vcf',
'genome': 'hg19', 'offset': 18155397, 'alt': 'A', 'ref': 'G'}
```

How it works

MutationInfo tries to infer the position, reference and alternative of a variant through the following pipeline:

- **If the variant is in rs format, then**
  - Try the Variant Effect Predictor through the pyVEP package.
  - If this fails, try the MyVariant.info service.
  - If this fails, access the UCSC tables <https://genome.ucsc.edu/cgi-bin/hgTables>_ through the 'cruzdb package.
- **If the variant is in HGVS then:**
  - Try to parse the variant with the biocommon/hgvs parser.

- If the parse fails then look if the variant contains some common mistakes in HGVS formatting. Correct if possible and then try again. For example remove parenthesis in the following variant: `NM_001042351.1:-1923(A>C)`
- If parse still fails then make a request to the mutalyzer.nl . For example `NT_005120.15:c.IVS1-72T>G` is parsed only from mutalyzer but not from biocommons/hgvs
- If neither of these methods are able to parse the variant then return `None`.
- If biocommons/hgvs parses the variant then use the variantmapper method to locate the location of the variant in the reference assembly.
- If this method fails then use the pyhgvs package and the *hgvs_to_vcf* method to convert the variant in a VCF entry.
- If this method fails then use Mutalyzer's Position Converter
- if this method fails then use the Mutalyzer's Name Checker which generates a genomic description of the variant. Then perform a blat search) with this variant (see below).
- **If both methods from Mutalyzer fail (for example `M61857.1:c.121A>G` crashes mutalyzer!) then:**
  - Download the FASTA sequence of the trascript of the variant from NCBI database.
  - If the position of the variant is in coding (c.) coordinates then convert to genomic (g.) coordinates. To do that, we use the Coordinate mapper addition of biopython.
  - Perform a blat search) from UCSC. This methods performs an alignment search of the fasta sequence in the reference assembly. In case this succeeds then report the location of the variant in the reference genome.
- If this method fails then search the LOVD database.
- If all the aforementioned methods fail then return `None`

CHAPTER 2

Installation

Note: Important! Requires 13 GB of disk space.

To install MutationInfo, download the latest release from https://github.com/kantale/MutationInfo/releases , uncompress and run:

```
pip install --upgrade setuptools
python setup.py install
```

Then the first time you instantiate the MutationInfo class, it installs all required datasets:

```
from MutationInfo import MutationInfo
mi = MutationInfo()
```

Installation in Ubuntu

Before installing in Ubuntu Linux, make sure that the following packages / tools are installed:

```
sudo apt-get update
sudo apt-get install git
sudo apt-get install gcc python-dev libpq-dev python-pip python-mysqldb-dbg

wget https://bootstrap.pypa.io/ez_setup.py -O - | sudo python
```

Test Installation

To verify that everything works fine run: `python test.py` in `test/` directory. The output after the long log messages should be:

```
----------------------------------------------------------------------
Ran 6 tests in 21.923s

OK
```

Troubleshooting

Possible problems from installing / running MutationInfo are:

- **Exception:** `psycopg2.OperationalError: invalid connection option "application_name"` See also: https://github.com/kantale/MutationInfo/issues/16 . Most likely, the version of PostgreSQL in your system is too old.
- **Exception:** `ImportError: cannot import name ExtendedInterpolation` See also: https://github.com/kantale/MutationInfo/issues/9 . One solution is to downgrade the `future` package. In that case, it is a good practice to run MutationInfo in a virtualenv so that the whole system is not affected.
- **Exception:** `ImportError: No module named MySQLdb` See also: https://github.com/kantale/MutationInfo/issues/7 . mysql is not installed in the system.
- **Error Message:** `Library not loaded: libssl.1.0.0.dylib` See: https://github.com/kantale/MutationInfo/issues/5 .

CHAPTER 3

How to

The MutationInfo package contains one class: `MutationInfo` which offers a single method: `get_info`.

The `MutationInfo` class

class `MutationInfo.`**`MutationInfo`**(*local_directory=None*, *email=None*, *genome='hg19'*, *dbsnp_version='snp146'*, ***kwargs*)

The MutationInfo class handles all necessary connections to various sources in order to assess the chromosomal position of a variant. The first time that this class is instantiated it downloads the reference genome in fasta format and splits it per chromosome. This might take approximately 13GB of disc space.

MutationInfo offers a single method for accessing the complete functionality of the module: `get_info()`.

This class does not have any required arguments for initialization. Nevertheless the following optional arguments are supported:

Parameters

- **`local_directory`** – The local directory where the fasta files will be stored. By default MutationInfo uses the appdirs module in order to create a platform specifc local directory. This directory is also used as a cache. Whenever there is a succesful attempt to access an external service, the acquired object is saved to local_directory for future reference.
- **`email`** – An email is required to connect with Entrez through biopython (see also this: http://biopython.org/DIST/docs/api/Bio.Entrez-module.html). If not set, MutationInfo looks for an email entry in the file stored in `<local_directory>/properties.json`. If this file does not exist (for example when the class is instantiated for the first time), then it requests one email from the user and stores it in the `properties.json` file.
- **`genome`** – The version of the **preferred** human genome assembly that will be used for reporting chromosomal positions. Accepted values should have the `hgXX` format. Default value is *hg19*.

> **Warning:** MutationInfo does not guarantee that the returned position is aligned according to the `genome` parameter since certain tools work only with specific genome assemblies. For this reason always check the `genome` key of the returned item after calling the `get_info()` method.

- **`ucsc_genome`** – Set the version of human genome assembly explicitly for the CruzDB tool (UCSC). Default: Same as the `genome` parameter.
- **`dbsnp_version`** – The version of dbsnp for rs variants. Default value is *snp146*.

The `get_info` method

`MutationInfo.`**`get_info`**(*variant*, *empty_current_fatal_error=True*, ***kwargs*)

Gets the chromosome, position, reference and alternative of a dbsnp or HGVS variant. If the `method` parameter is not specified, by default it will go through the following pipeline:

Parameters **`variant`** – A variant (in str or unicode) or list of variants. Both rs (i.e. `rs56404215`) or HGVS (i.e. `NM_006446.4:c.1198T>G`) are accepted.

Optional arguments:

Parameters **`method`** – Instead of the default pipeline, use a specific tool. Accepted values are:

- `UCSC` : Use CruzDB (only for dbsnp variants)
- `VEP` : Use Variant Effect Predictor (only for dbsnp variants)
- `MYVARIANTINFO` : Use MyVariant.info (only for dbsnp variants)
- `BIOCOMMONS` : Use Biocommons HGVS (only for HGVS variants)
- `COUNSYL` : Use Counsyl HGVS (only for HGVS variants)
- `MUTALYZER` : Use Mutalyzer (only for HGVS variants)
- `BLAT` : Perform a BLAT search (only for HGVS variants)
- `LOVD` Search LOVD database (only for HGVS variants)
- `VARIATION_REPORTER` Search Variation Reported
- `TRANSVAR` Search Transvar (Experimental, requires installation of TRANSVAR CLI)

Returns If the pipeline or the selected method fails then the return value is `None`. Otherwise it returns a dictionary with the following keys:

- `chrom` : The chromosome where this variant is located. The type of this value is *str* in order to have a universal type for all possible chromosome values (including X and Y).
- `offset` : The nucleotide position of the variant.
- `ref` : The reference sequence of the variant. In case of insertions this value is an empty string.
- `alt` : The alternative sequence of the variant. In case of deletions this value is an empty string.
- `genome` : The version of the human genome assembly for this position.
- `source` : The name of the tool that was used to locate the position.
- `notes` : Possible warnings, errors and notes that the tools generated during the conversion.

An example of output is the following:

Example

```
>>> from MutationInfo import MutationInfo
>>> mi = MutationInfo()
>>> info = mi.get_info('NM_000367.2:c.-178C>T')
>>> print info
{'chrom': '6', 'notes': '', 'source': 'counsyl_hgvs_to_vcf', 'genome': 'hg19',
↪'offset': 18155397, 'alt': 'A', 'ref': 'G'}
```

CHAPTER 4

License

MIT License (MIT)

CHAPTER 5

Contact

Alexandros Kanterakis

CHAPTER 6

Indices and tables

- genindex
- modindex
- search

Index

G

M