MyVariant.py Documentation

Release v1.0.0

Chunlei Wu

Dec 18, 2018

Contents

MyVariant.Info provides simple-to-use REST web services to query/retrieve variant annotation data. It's designed with simplicity and performance emphasized. *myvariant*, is an easy-to-use Python wrapper to access MyVariant.Info services.

Note: As of v1.0.0, myvariant Python package is now a thin wrapper of underlying biothings_client package, a universal Python client for all BioThings APIs, including MyVariant.info. The installation of myvariant will install biothings_client automatically. The following code snippets are essentially equivalent:

- Continue using myvariant package

```
In [1]: import myvariant
In [2]: mv = myvariant.MyVariantInfo()
```

- Use biothings_client package directly

```
In [1]: from biothings_client import get_client
In [2]: mv = get_client('variant')
```

After that, the use of `mv` instance is exactly the same.

CHAPTER 1

Requirements

python >=2.7 (including python3)

(Python 2.6 might still work, not it's not supported any more since v1.0.0)

biothings_client (>=0.2.0, install using "pip install biothings_client")

CHAPTER 2

Optional dependencies

pandas (install using “pip install pandas”) is required for returning a list of gene objects as DataFrame.

CHAPTER 3

Installation

Option 1

```
pip install myvariant
```

Option 2 download/extract the source code and run:

```
python setup.py install
```

Option 3 install the latest code directly from the repository:

```
pip install -e git+https://github.com/biothings/myvariant.py
```

CHAPTER 4

Version history

CHANGES.txt

CHAPTER 5

Tutorial

Access ClinVar Data from MyVariant.info Services (the raw ipynb file is here)

CHAPTER 6

API

myvariant.**get_hgvs_from_vcf**(*input_vcf*)
: *get_hgvs_from_vcf()* helper function is now moved into *MyVariantInfo* class as a method: *get_hgvs_from_vcf()*.

myvariant.**format_hgvs**(*chrom*, *pos*, *ref*, *alt*)
: *format_hgvs()* helper function is now moved into *MyVariantInfo* class as a method: *format_hgvs()*.

class myvariant.**MyVariantInfo**(*url=None*)
: This is the client for MyVariant.info web services. Example:

```
>>> mv = MyVariantInfo()
```

clear_cache()
: Clear the globally installed cache.

format_hgvs(*chrom*, *pos*, *ref*, *alt*)
: get a valid hgvs name from VCF-style "chrom, pos, ref, alt" data. Example:

```
>>> utils.variant.format_hgvs("1", 35366, "C", "T")
>>> utils.variant.format_hgvs("2", 17142, "G", "GA")
>>> utils.variant.format_hgvs("MT", 8270, "CACCCCCTCT", "C")
>>> utils.variant.format_hgvs("X", 107930849, "GGA", "C")
```

get_fields(*search_term=None*, *verbose=True*)
: Wrapper for http://myvariant.info/v1/metadata/fields

Parameters

- **search_term** – a case insensitive string to search for in available field names. If not provided, all available fields will be returned.
- **assembly** – return the metadata for either hg19 or hg38 variants, "hg19" (default) or "hg38".

Example:

```
>>> mv.get_fields()
>>> mv.get_fields("rsid")
>>> mv.get_fields("sift")
```

Hint: This is useful to find out the field names you need to pass to **fields** parameter of other methods.

get_hgvs_from_vcf(*input_vcf*)
From the input VCF file (filename or file handle), return a generator of genomic based HGVS ids. :param input_vcf: input VCF file, can be a filename or a file handle :returns: a generator of genomic based HGVS ids. To get back a list instead,

using `list(get_hgvs_from_vcf("your_vcf_file"))`

Note: This is a lightweight VCF parser to return valid genomic-based HGVS ids from the *input_vcf* file. For more sophisticated VCF parser, consider using PyVCF module.

getvariant(*_id*, *fields=None*, ***kwargs*)
Return the variant object for the give HGVS-based variant id. This is a wrapper for GET query of "/variant/<hgvsid>" service.

Parameters

- **vid** – an HGVS-based variant id. More about HGVS id.
- **fields** – fields to return, a list or a comma-separated string. If not provided or **fields="all"**, all available fields are returned. See here for all available fields.
- **assembly** – specify the human genome assembly used in HGVS-based variant id, "hg19" (default) or "hg38".

Returns a variant object as a dictionary, or None if vid is not found.

Example:

```
>>> mv.getvariant('chr9:g.107620835G>A')
>>> mv.getvariant('chr9:g.107620835G>A', fields='dbnsfp.genename')
>>> mv.getvariant('chr9:g.107620835G>A', fields=['dbnsfp.genename', 'cadd.
↪phred'])
>>> mv.getvariant('chr9:g.107620835G>A', fields='all')
>>> mv.getvariant('chr1:g.161362951G>A', assembly='hg38')
```

Hint: The supported field names passed to **fields** parameter can be found from any full variant object (without **fields**, or **fields="all"**). Note that field name supports dot notation for nested data structure as well, e.g. you can pass "dbnsfp.genename" or "cadd.phred".

getvariants(*ids*, *fields=None*, ***kwargs*)
Return the list of variant annotation objects for the given list of hgvs-base varaint ids. This is a wrapper for POST query of "/variant" service.

Parameters

- **ids** – a list/tuple/iterable or a string of comma-separated HGVS ids. More about hgvs id.
- **fields** – fields to return, a list or a comma-separated string. If not provided or **fields="all"**, all available fields are returned. See here for all available fields.

- **assembly** – specify the human genome assembly used in HGVS-based variant id, "hg19" (default) or "hg38".
- **as_generator** – if True, will yield the results in a generator.
- **as_dataframe** – if True or 1 or 2, return object as DataFrame (requires Pandas). True or 1: using json_normalize 2 : using DataFrame.from_dict otherwise: return original json
- **df_index** – if True (default), index returned DataFrame by 'query', otherwise, index by number. Only applicable if as_dataframe=True.

Returns a list of variant objects or a pandas DataFrame object (when **as_dataframe** is True)

Ref http://docs.myvariant.info/en/latest/doc/variant_annotation_service.html.

Example:

```
>>> vars = ['chr1:g.866422C>T',
...         'chr1:g.876664G>A',
...         'chr1:g.69635G>C',
...         'chr1:g.69869T>A',
...         'chr1:g.881918G>A',
...         'chr1:g.865625G>A',
...         'chr1:g.69892T>C',
...         'chr1:g.879381C>T',
...         'chr1:g.878330C>G']
>>> mv.getvariants(vars, fields="cadd.phred")
>>> mv.getvariants('chr1:g.876664G>A,chr1:g.881918G>A', fields="all")
>>> mv.getvariants(['chr1:g.876664G>A', 'chr1:g.881918G>A'], as_
↪dataframe=True)
>>> mv.getvariants(['chr1:g.161362951G>A', 'chr2:g.51032181G>A'], assembly=
↪'hg38')
```

Hint: A large list of more than 1000 input ids will be sent to the backend web service in batches (1000 at a time), and then the results will be concatenated together. So, from the user-end, it's exactly the same as passing a shorter list. You don't need to worry about saturating our backend servers.

Hint: If you need to pass a very large list of input ids, you can pass a generator instead of a full list, which is more memory efficient.

---

metadata(*verbose=True*, ***kwargs*)

Return a dictionary of MyVariant.info metadata.

Parameters **assembly** – return the metadata for either hg19 or hg38 variants, "hg19" (default) or "hg38".

Example:

```
>>> metadata = mv.metadata()
>>> metadata = mv.metadata(assembly='hg38')
```

query(*q*, ***kwargs*)

Return the query result. This is a wrapper for GET query of "/query?q=<query>" service.

Parameters

- **q** – a query string, detailed query syntax here.

- **fields** – fields to return, a list or a comma-separated string. If not provided or **fields="all"**, all available fields are returned. See here for all available fields.
- **assembly** – specify the human genome assembly used for the query, "hg19" (default) or "hg38".
- **size** – the maximum number of results to return (with a cap of 1000 at the moment). Default: 10.
- **skip** – the number of results to skip. Default: 0.
- **sort** – Prefix with "-" for descending order, otherwise in ascending order. Default: sort by matching scores in decending order.
- **as_dataframe** – if True or 1 or 2, return object as DataFrame (requires Pandas). True or 1: using json_normalize 2 : using DataFrame.from_dict otherwise: return original json
- **fetch_all** – if True, return a generator to all query results (unsorted). This can provide a very fast return of all hits from a large query. Server requests are done in blocks of 1000 and yielded individually. Each 1000 block of results must be yielded within 1 minute, otherwise the request will expire at server side.

Returns a dictionary with returned variant hits or a pandas DataFrame object (when **as_dataframe** is True) or a generator of all hits (when **fetch_all** is True)

Ref http://docs.myvariant.info/en/latest/doc/variant_query_service.html.

Example:

```
>>> mv.query('_exists_:dbsnp AND _exists_:cosmic')
>>> mv.query('dbnsfp.polyphen2.hdiv.score:>0.99 AND chrom:1')
>>> mv.query('cadd.phred:>50')
>>> mv.query('dbnsfp.genename:CDK2', size=5)
>>> mv.query('dbnsfp.genename:CDK2', size=5, assembly='hg38')
>>> mv.query('dbnsfp.genename:CDK2', fetch_all=True)
>>> mv.query('chrX:151073054-151383976')
```

Hint: By default, **query** method returns the first 10 hits if the matched hits are >10. If the total number of hits are less than 1000, you can increase the value for **size** parameter. For a query returns more than 1000 hits, you can pass "fetch_all=True" to return a generator of all matching hits (internally, those hits are requested from the server-side in blocks of 1000).

querymany(*qterms*, *scopes=None*, ***kwargs*)

Return the batch query result. This is a wrapper for POST query of "/query" service.

Parameters

- **qterms** – a list/tuple/iterable of query terms, or a string of comma-separated query terms.
- **scopes** – specify the type (or types) of identifiers passed to **qterms**, either a list or a comma-separated fields to specify type of input qterms, e.g. "dbsnp.rsid", "clinvar.rcv_accession", ["dbsnp.rsid", "cosmic.cosmic_id"]. See here for full list of supported fields.
- **fields** – fields to return, a list or a comma-separated string. If not provided or **fields="all"**, all available fields are returned. See here for all available fields.
- **assembly** – specify the human genome assembly used for the query, "hg19" (default) or "hg38".
- **returnall** – if True, return a dict of all related data, including dup. and missing qterms

- **verbose** – if True (default), print out information about dup and missing qterms
- **as_dataframe** – if True or 1 or 2, return object as DataFrame (requires Pandas). True or 1: using json_normalize 2 : using DataFrame.from_dict otherwise: return original json
- **df_index** – if True (default), index returned DataFrame by 'query', otherwise, index by number. Only applicable if as_dataframe=True.

Returns a list of matching variant objects or a pandas DataFrame object.

Ref http://docs.myvariant.info/en/latest/doc/variant_query_service.html for available fields, extra *kwargs* and more.

Example:

```
>>> mv.querymany(['rs58991260', 'rs2500'], scopes='dbsnp.rsid')
>>> mv.querymany(['rs58991260', 'rs2500'], scopes='dbsnp.rsid', assembly='hg38
↪')
>>> mv.querymany(['RCV000083620', 'RCV000083611', 'RCV000083584'], scopes=
↪'clinvar.rcv_accession')
>>> mv.querymany(['COSM1362966', 'COSM990046', 'COSM1392449'], scopes='cosmic.
↪cosmic_id', fields='cosmic')
>>> mv.querymany(['COSM1362966', 'COSM990046', 'COSM1392449'], scopes='cosmic.
↪cosmic_id',
...              fields='cosmic.tumor_site', as_dataframe=True)
```

Hint: *querymany()* is perfect for query variants based different ids, e.g. rsid, clinvar ids, etc.

Hint: Just like *getvariants()*, passing a large list of ids (>1000) to *querymany()* is perfectly fine.

Hint: If you need to pass a very large list of input qterms, you can pass a generator instead of a full list, which is more memory efficient.

set_caching(*cache_db=None*, *verbose=True*, ***kwargs*)
Installs a local cache for all requests.

cache_db is the path to the local sqlite cache database.

stop_caching()
Stop caching.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

m

Index