MATCHBox API Utils Documentation

Release 1.0.0

Dave Sims

Mar 01, 2018

Contents

CHAPTER 1

MATCHBox API Utils

1.1 Introduction

Retrieving results from the MATCHBox, an NCI derived system to collect, collate, and report data from the NCI-MATCH study can be a difficult task. While there is a rich API and set of tools underlying the user interfact (UI), it can be very difficult at times to quickly gleen the information one is looking for from clicking through all of the web pages. This is where MATCHBox API Utils comes in.

The system is designed to directly pull data from a live MATCHBox instance, and build a simple JSON file, from which a whole host of methods have been created to access, filter, interrogate, report, etc. the data. For example, if you wanted to know which MSNs were associated with patient identifier (PSN) 13070, you could click through the GUI, and find the correct data. Or, with this tool, you can simply run:

```
>>> from matchbox_api_utils import MatchData
>>> data = MatchData()
>>> data.get_msn(psn=13070)
[u'MSN31054', u'MSN59774']
```

More on the specifics of how to use these modules and methods a little later.

Also, along with the set of Python modules and methods, included is a set of *helper* scripts that will act as nice wrappers around some of these basic methods. For example, running the `map_msn_psn.py` script that's a part of this package will generate really nice tables of data, allow for batch processing from either a list input on the commandline or a file containing a set of identifiers of interest.

```
$ map_msn_psn.py -t MSN MSN31054,MSN12724,MSN37895,MSN12104

Getting MSN / PSN mapping data...

PSN,BSN,MSN
PSN10818,T-16-000029,MSN12104
PSN10955,T-16-000213,MSN12724
PSN13070,T-16-002251,MSN31054
PSN13948,T-16-003010,MSN37895
```

1.2 Installation

At this time, the most simple installation is to unzip the provided tarball, and run:

```
python setup.py install
```

From here, the installer will add the library modules to your standard Python library location, put the helper scripts into the standard binary direcotry in your `$PATH`, and run a post installation script that will set up the appropriate URLs, username, and password for your system and user. At the end, a new set of JSON files will be created in `$HOME/.mb_utils/` which will contain the initial MATCH dataset for use.

We recommend re-building this dataset on occassion using the `matchbox_json_dump.py` program included in the package for up to date data, especially as the dataset is not yet locked down. Additionally, one can keep old sets of these JSON databases for use at any time (e.g comparing version data), by inputting the custom JSON into the MatchData object (see: `matchbox_api_utils` API documentation for details).

1.3 Requirements

These modules are run under Python 2.7. However, all coding (where possible) is done with Python 3 in mind, and should be easily portable. This is especially the case with the main library modules and classes. Your mileage may vary with the helper scripts somewhat.

In addition to the above, the package requires the installation of:

- requests

The setup script should take care of this dependency upon installation.

1.4 Using MATCHBox API Utils

Once the installation is run, one can either use the pre-built scripts available in your systems binary directory (see Helper Scripts documentation), or write your own scripts using any one of the included methods (see the MATCHBox API Utils API documentation).

CHAPTER 2

MATCHBox API Utils Tutorial

Working with MATCHBox API Utils is very easy. It is simply a matter of loading one (or all) of the available modules:

1. `Matchbox()`
2. `MatchData()`
3. `TreatmentArms()`

Most often one will be using the `MatchData()` module, as that deals directly with the majority of the MATCH data that needs to be dealt with. It directly pulls data from `Matchbox` and `TreatmentArms` where need be, and it allows for the most flexible parsing and reporting of data. That's not to say that the other modules are not useful. Far from it! As we'll see a little later down, there are a number of useful things that one can do with those as well.

By default modules will load the most recent dataset available in `$HOME/.mb_utils` (see API documentation for specifics), and be ready for use. However, if one wanted to get a live dataset, or one wanted to get data from an older snapshot of the database, that is all possible too.

2.1 Module: Matchbox

The `Matchbox` module is a basic data connector to the live instance of the MATCHBox system. Apart from a URL to query and credentials, there are not a whole lot of options.

In general this module is called by MatchData and TreatmentArms to make the connection to the live system and pull data. I've included some documentation of this module, though, in the event that one needs to call on it at some point.

For example, on occassion one might want to get a complete raw MATCHBox dataset, which has not been parse and filtered, and can be accepted into the other modules as if they were making a live call. In this case, one could run:

```
>>> from matchbox_api_utils import Matchbox
>>> Matchbox(make_raw='raw_mb_dataset.json')
```

This will create a raw complete JSON dump of MATCHBox.

Without dumping that data to a JSON file, one could assign a variable to the Matchbox obj, and then pass it around. However, MatchData is better for an entry point here.

2.2 Module: MatchData

This is by far the workhorse module of the package. In its most basic form, one needs only load the object into a variable, and then run the host of methods available:

```
>>> from matchbox_api_utils import MatchData
>>> data = MatchData()
>>> data.get_biopsy_summary(category='progression', ret_type='counts')
{'progression': 16}
```

This simply works by loading a processed version of MATCHBox, which has been optimized to be smaller and more efficient than the raw MATCHBox dataset, and then having a set of methods to extract portions of the data for cohort type analysis.

As mentioned above, one can either choose to use the *system_default* JSON option for `MatchData`, in which case the latest version of the MATCHBox JSON dump will be loaded, or one can load their own custom file (e.g. if one wanted to load an old version of the data to see how things changed). Alternatively (and with the same warning as above), one can make a live query at the cost of a long load time.

There is also the possibility of loading just one or more patient IDs into the API and limiting data to that cohort. However, this is usually only meant for debugging purposes and filtering on a patient identifier is usually best done at each method call.

2.2.1 Toy Example #1

You would like to know the disease state of the progression biopsies indicated above. You don't have any identifiers or other information to go on.

Here's a way one might work through it:

```
>>> from matchbox_api_utils import MatchData
>>> from pprint import pprint as pp
>>> data = MatchData()
>>> results = {}  # Let's create a dict to put the results in.
>>> biopsies = data.get_biopsy_summary(category='progression', ret_type='ids')
>>> pp(biopsies)
{'progression': [u'T-17-002064',
                 u'T-17-002755',
                 u'T-18-000071',
                 u'T-17-002680',
                 u'T-17-000787',
                 u'T-17-002621',
                 u'T-17-002657',
                 u'T-17-001275',
                 u'T-17-001175',
                 u'T-17-002564',
                 u'T-17-002556',
                 u'T-17-002730',
                 u'T-17-002600',
                 u'T-18-000005',
                 u'T-17-000333',
                 u'T-18-000031']}
```

Notice that we changed the return type of the `get_biopsy_summary()` call to *ids*, which allows us to get ids rather than counts. Now that we have those biopsy IDs, we can get some PSNs , which will be helpful in getting the disease data ultimately:

```
>>> for bsn in biopsies['progression']:
...     psn = data.get_psn(bsn=bsn)
...     msn = data.get_msn(bsn=bsn)
...     print('%s: %s' % (psn, msn))
PSN13070: [u'MSN59774']
PSN13670: [u'MSN62646']
PSN15436: None
PSN15362: [u'MSN62489']
PSN10955: [u'MSN46367']
PSN13948: [u'MSN62208']
PSN11347: [u'MSN62398']
PSN10818: [u'MSN51268']
PSN11083: [u'MSN50799']
PSN11707: [u'MSN62042']
PSN11769: [u'MSN62018']
PSN11127: [u'MSN62547']
PSN12471: [u'MSN62109']
PSN14705: [u'MSN62687']
PSN11583: [u'MSN41897']
PSN15971: None
```

Looks like there are a couple issues here.

1. First, results from the `get_msn()` method are always lists. We can have multiple MSNs per BSN unfortunately, and so we need to output more than one on occassion. In this case, for what we want to do, we can just output them all as a comma separated list.
2. Second some results to not have a MSN returned! This can happen. In this case, there was a progression biopsy collected, but no valid MSN was yet generated for the case. Since we would prefer to only work with complete data for now, we'll skip those cases.

So, now that we know which are progression cases from the whole dataset, and know their PSN, BSN, and MSN identifiers, let's get the disease for each, and store it in our `results` dict above. We'll rewrite a little bit of the code above to help with some of the processing:

```
>>> for bsn in biopsies['progression']:
...     psn = data.get_psn(bsn=bsn)
...     msn = data.get_msn(bsn=bsn)
...     if msn is not None:
...         results[psn] = [bsn, ','.join(msn)]
...
>>> pp(results)
{'PSN10818': [u'T-17-001275', u'MSN51268'],
 'PSN10955': [u'T-17-000787', u'MSN46367'],
 'PSN11083': [u'T-17-001175', u'MSN50799'],
 'PSN11127': [u'T-17-002730', u'MSN62547'],
 'PSN11347': [u'T-17-002657', u'MSN62398'],
 'PSN11583': [u'T-17-000333', u'MSN41897'],
 'PSN11707': [u'T-17-002564', u'MSN62042'],
 'PSN11769': [u'T-17-002556', u'MSN62018'],
 'PSN12471': [u'T-17-002600', u'MSN62109'],
 'PSN13070': [u'T-17-002064', u'MSN59774'],
 'PSN13670': [u'T-17-002755', u'MSN62646'],
 'PSN13948': [u'T-17-002621', u'MSN62208'],
 'PSN14705': [u'T-18-000005', u'MSN62687'],
 'PSN15362': [u'T-17-002680', u'MSN62489']}
```

Much better! Now, let's leverage another method `get_hisology()` to get the patient's disease and add it to the

data:

```
>>> for p in results:
...     print(data.get_histology(psn=p))
{'PSN15362': u'Salivary gland cancer'}
{'PSN11583': u'Salivary gland cancer'}
{'PSN13070': u'Cholangiocarcinoma, intrahepatic and extrahepatic bile ducts␣
↪(adenocarcinoma)'}
{'PSN10955': u'Squamous cell carcinoma of the anus'}
{'PSN12471': u'Carcinosarcoma of the uterus'}
{'PSN10818': u'Colorectal cancer, NOS'}
{'PSN11769': u'Renal cell carcinoma, clear cell adenocarcinoma'}
{'PSN11347': u'Salivary gland cancer'}
{'PSN13948': u'Adenocarcinoma of the rectum'}
{'PSN13670': u'Ovarian epithelial cancer'}
{'PSN11707': u'Cholangiocarcinoma, intrahepatic and extrahepatic bile ducts␣
↪(adenocarcinoma)'}
{'PSN11083': u'Adenocarcinoma of the colon'}
{'PSN14705': u'Laryngeal squamous cell carcinoma'}
{'PSN11127': u'Invasive breast carcinoma'}
```

As we can see the results for this method call are all dicts of *PSN : Disease* mappings. So, we can use the PSN to pull the disease and add it to the results:

```
>>> for p in results:
...     results[p].append(data.get_histology(psn=p)[p])
>>> pp(results)
{'PSN10818': [u'T-17-001275', u'MSN51268', u'Colorectal cancer, NOS'],
 'PSN10955': [u'T-17-000787',
              u'MSN46367',
              u'Squamous cell carcinoma of the anus'],
'PSN11083': [u'T-17-001175', u'MSN50799', u'Adenocarcinoma of the colon'],
'PSN11127': [u'T-17-002730', u'MSN62547', u'Invasive breast carcinoma'],
'PSN11347': [u'T-17-002657', u'MSN62398', u'Salivary gland cancer'],
'PSN11583': [u'T-17-000333', u'MSN41897', u'Salivary gland cancer'],
'PSN11707': [u'T-17-002564',
             u'MSN62042',
             u'Cholangiocarcinoma, intrahepatic and extrahepatic bile ducts␣
↪(adenocarcinoma)'],
'PSN11769': [u'T-17-002556',
             u'MSN62018',
             u'Renal cell carcinoma, clear cell adenocarcinoma'],
'PSN12471': [u'T-17-002600', u'MSN62109', u'Carcinosarcoma of the uterus'],
'PSN13070': [u'T-17-002064',
             u'MSN59774',
             u'Cholangiocarcinoma, intrahepatic and extrahepatic bile ducts␣
↪(adenocarcinoma)'],
'PSN13670': [u'T-17-002755', u'MSN62646', u'Ovarian epithelial cancer'],
'PSN13948': [u'T-17-002621', u'MSN62208', u'Adenocarcinoma of the rectum'],
'PSN14705': [u'T-18-000005',
             u'MSN62687',
             u'Laryngeal squamous cell carcinoma'],
'PSN15362': [u'T-17-002680', u'MSN62489', u'Salivary gland cancer']}
```

And finally we have a nice list of collected data for each progression case, which is ready to print out for downstream use:

```
>>> for patient in results:
...     print('\t'.join([patient] + results[patient]))
PSN15362    T-17-002680    MSN62489    Salivary gland cancer
PSN11583    T-17-000333    MSN41897    Salivary gland cancer
PSN13070    T-17-002064    MSN59774    Cholangiocarcinoma, intrahepatic and␣
↪extrahepatic bile ducts (adenocarcinoma)
PSN10955    T-17-000787    MSN46367    Squamous cell carcinoma of the anus
PSN12471    T-17-002600    MSN62109    Carcinosarcoma of the uterus
PSN10818    T-17-001275    MSN51268    Colorectal cancer, NOS
PSN11769    T-17-002556    MSN62018    Renal cell carcinoma, clear cell adenocarcinoma
PSN11347    T-17-002657    MSN62398    Salivary gland cancer
PSN13948    T-17-002621    MSN62208    Adenocarcinoma of the rectum
PSN13670    T-17-002755    MSN62646    Ovarian epithelial cancer
PSN11707    T-17-002564    MSN62042    Cholangiocarcinoma, intrahepatic and␣
↪extrahepatic bile ducts (adenocarcinoma)
PSN11083    T-17-001175    MSN50799    Adenocarcinoma of the colon
PSN14705    T-18-000005    MSN62687    Laryngeal squamous cell carcinoma
PSN11127    T-17-002730    MSN62547    Invasive breast carcinoma
```

So, there you have it. Very simple toy case, but hopefully one that highlights some of the features of the MatchData module. See the API documentation section for more information about the included modules and their usage.

2.3 Module: TreatmentArms

The `TreatmentArms` module will handle all NCI-MATCH treatment arm related data, including the handling of a "rules engine" to categorize mutations of interest (MOIs) as being actionable (aMOIs) or not. At the time of this writing, there are not a lot of public methods available for the module, and it's main use will be directly (really behind the scenes) from `MatchData`. However, there are a few methods that one will (hopefully) find handy.

As with the other modules, one can either make a live query to MATCHBox to generate a dataset:

```
>>> from matchbox_api_utils import TreatmentArms
>>> ta_data = TreatmentArms()
```

As with `MatchData`, not specifying a JSON database will result in loading the *sys_default* database which is built at the same time as the `MatchData` JSON database. You'll see this file in `$HOME/.mb_utils/ta_obj_<date>.json`.

One you have object loaded, then you can run one of the public methods available, including `map_amoi()`, `map_drug_arm()`, or `get_exclusion_disease()`.

2.3.1 Toy Example #2

Let's say you have a *'BRAF p.V600E'* mutation that you discovered in a patient diagnosed with *'Melanoma'* , but you are not sure whether or not any arms cover the patient, and if there is a qualifying arm, whether or not the patient has an exclusionary disease (i.e. a histological subtype that is excluded from arm eligibility).

The first step is to try to map that aMOI to the study arms. You need to have a of aMOI level data since there are some extra rules to map. We always need to know the following (dict_key: accepted_values):

Variant Key	Acceptable Values
type	{ snvs_indels, cnvs, fusions }
oncomineVariantClass	{ Hotspot, Deleterious }
gene	Any acceptable HUGO gene name (e.g. BRAF)
identifier	Any variant identifier, usually from COSMIC (e.g. COSM476)
exon	The numeric value for the exon in which the variant is found. For example, if the variant is in Exon 15, you would indicate 15 in this field.
function	{'missense', 'nonsense', 'frameshiftInsertion', 'frameshiftDeletion', 'nonframeshiftDeletion', 'nonframeshiftInsertion', 'frameshiftBlockSubstitution'}

In the case of a typical BRAF p.V600E variant, we would set up our environment as follows:

```
>>> from matchbox_api_utils import TreatmentArms
>>> from pprint import pprint as pp
>>> ta_data = TreatmentArms()
>>> variant = {
...     'type' : 'snvs_indels',
...     'gene' : 'BRAF',
...     'identifier' : 'COSM476',
...     'exon' : '15',
...     'function' : 'missense',
...     'oncominevariantclass' : 'Hotspot' }
```

Now to find out if our variant would qualify for any arms, we'll run `map_amoi()` to check:

```
>>> v600e_arms = ta_data.map_amoi(variant)
>>> pp(v600e_arms)
['EAY131-Y(e)', 'EAY131-P(e)', 'EAY131-N(e)', 'EAY131-H(i)']
```

So we see that there are 4 arms that have identified this variant as being an aMOI. But, based on the notation in parenthesis (e.g. *'(e)'*), we can see that Arms Y, P, and N consider this aMOI to be exclusionary, while arm H consider this aMOI to be inclusionary. So, it looks like so far the patient is a potential match for Arm H only. Now, let's see if

their disease would qualify them for this arm:

```
>>> pp(ta_data.get_exclusion_disease('EAY131-H'))
[u'Papillary thyroid carcinoma',
 u'Melanoma',
 u'Acral Lentiginous Melanoma',
 u'Adenocarcinoma - colon',
 u'Malignant Melanoma of sites other than skin or eye',
 u'Adenocarcinoma - rectum',
 u'Colorectal cancer, NOS']
```

Well, that's bad news! Based on the fact that the patient has *Melanoma* and it is an exclusionary disease for Arm H, the patient would not currently qualify for any MATCH Arm.

For more detailed descriptions of the methods in the module and their use, see the TreatmentArms API documention section.

CHAPTER 3

MATCHBox API Utils Helper Scripts

3.1 MATCHBox JSON Dump (matchbox_json_dump.py)

Get parsed dataset from MATCHBox and dump as a JSON object that we can use later on to speed up development and periodic searching for data.

This is the script that is run during the post installer, and something that should be run on a periodic basis to take advantage of new MATCHBox data entered into the system (at least until lockdown occurs). This will put the resultant JSON file into the current directory, where it should probably be migrated into `$HOME/.mb_utils/` to be picked up by the rest of the system.

3.1.1 MATCHBox JSON Dump Help Doc

```
Usage:
    matchbox_json_dump.py [-h] [-d <raw_mb_datafile.json>] [-r] [-p <psn>]
    [-t <ta_obj.json>] [-a <amoi_obj.json>]
    [-m <mb_obj.json>] [-v]

Optional Arguments:
    -r, --raw         Generate a raw dump of MATCHBox for debugging and dev
                      purposes.
                      No filtering or condensing of data at all.
    -d, --data        Load a raw MATCHBox database file (usually after running
                      with the -r option.
    -p, --patient     Patient sequence number used to limit output for testing
                      and dev purposes
    -t, --ta_json     Treatment Arms obj JSON filename. DEFAULT:
                      ta_obj_<datestring>.json
    -a, --amoi_json   aMOIs lookup filename. DEFAULT:
                      "amois_lookup_<datestring>.json".
    -m, --mb_json     Name of Match Data obj JSON file. DEFAULT:
                      "mb_obj_<datestring>.json".
```

(continues on next page)

(continued from previous page)

```
-h, --help            Show this help message and exit
-v, --version         Show program's version number and exit
```

3.2 MAP MSN PSN (map_msn_psn.py)

Input a MSN, BSN, or PSN, and return the other identifiers. Useful when trying to retrieve the correct dataset and you only know one piece of information.

Note: We are only working with internal BSN, MSN, and PSN numbers for now and can not return Outside Assay identifiers at this time.

3.2.1 MAP MSN PSN Help Doc

```
Usage:
    map_msn_psn.py [-h] [-j <mb_json_file>] -t {psn,msn,bsn} [-f <input_file>]
    [-o <outfile>] [-v] [<IDs>]

Positional Arguments:
IDs     MATCH IDs to query. Can be single or comma separated list. Must be used
        with PSN or MSN option.

Optional Arguments:
-j, --json      Load a MATCHBox JSON file derived from "matchbox_json_dump.py"
                instead of a live query. By default will load the "sys_default"
                created during package installation. If you wish to do a live
                query (i.e. not load a previously downloaded JSON dump), set
                -j to "None".
-t, --type      Type of query string input. Can only be MSN, PSN, or BSN
-f, --file      Load a batch file of all MSNs or PSNs to proc
-o, --outfile   File to which output should be written. Default: STDOUT.
-h, --help      Show this help message and exit
-v, --version   Show program's version number and exit
```

3.3 MATCH Variant Frequency (match_variant_frequency.py)

Input a list of genes by variant type and get back a table of NCI-MATCH hits that can be further analyzed in Excel or some other tool. Can either input a patient (or comma separated list of patients) to query, or query the entire dataset. Will limit the patient set to the non-outside assay results only, as the Outside Assay data is very unreliable.

3.3.1 MATCH Variant Frequency Help Docs

```
Usage:
match_variant_frequency.py [-h] [-j <mb_json_file>] [-p <PSN>] [-s <gene_list>]
[-c <gene_list>] [-f <gene_list>] [-i <gene_list>] [--style <pp,csv,tsv>]
[-o <output_file>] [-v]
```

(continues on next page)

(continued from previous page)

```
Optional Arguments:
-j, --json      Load a MATCHBox JSON file derived from "matchbox_json_dump.py"
                instead of a live query
-p, --psn       Only output data for a specific patient or comma separated list
                of patients
-s, --snv       Comma separated list of SNVs to look up in MATCHBox data.
-c, --cnv       Comma separated list of CNVs to look up in MATCHBox data.
-f, --fusion    Comma separated list of Fusions to look up in MATCHBox data.
-i, --indel     Comma separated list of Fusions to look up in MATCHBox data.

--style         Format for output. Can choose pretty print (pp), CSV, or TSV
-o, --output    Output file to which to write data. Default is stdout
-h, --help      Show this help message and exit
-v, --version   Show program's version number and exit
```

3.4 MATCHBox Patient Summary (matchbox_patient_summary.py)

Get patient or disease summary statistics and data from the MATCH dataset.

3.4.1 MATCHBox Patient Summary Help Docs

```
Usage:
matchbox_patient_summary.py [-h] [-j <mb_json_file>] [-p PSN] [-t <tumor_type>]
[-m <medra_code>] [-O] [-o <results.txt>] [-v] {patient,disease}

Positional Arguments:
patient, disease    Category of data to output. Can either be patient or disease
level.

Optional Arguments:
-j, --json      MATCHBox JSON file containing patient data, usually from
                matchbox_json_dump.py
-p, --psn       Filter patient summary to only these patients. Can be a comma
                separated list
-t, --tumor     Retrieve data for only this tumor type or comma separate list of
                tumors. Note that you must quote tumors with names containing
                spaces.
-m, --medra     MEDRA Code or comma separated list of codes to search.
-O, --Outside   Include Outside Assay study data (DEFAULT: False).
-o, --outfile   Name of output file. DEFAULT: STDOUT.

-h, --help      Show this help message and exit
-v, --version   Show program's version number and exit
```

CHAPTER 4

MATCHBox API Utils API Documentation

4.1 matchbox_api_utils.matchbox module

class matchbox_api_utils.matchbox.**Matchbox**(*url*, *creds*, *make_raw=None*)

Bases: object

MATCHBox API Connector Class

Basic connector class to make a call to the API and load the raw data. From here we pass data, current MatchData or TreatmentArm data to appropiate calling classes.

Used for calling to the MATCHBox API, loading data and, creating a basic data structure. Can load a raw MATCHBox API dataset JSON file, or create one. Requires credentials, generally acquired from the config file generated upon package setup.

Parameters

- **url** (*str*) – API URL for MATCHbox. Generally only using one at the moment, but possible to add others later.
- **creds** (*dict*) – Username and Password credentials obtained from the config file generated upon setup. Can also just input a dict in the form of:

```
{ 'username' : <username>, 'password' : <password> }
```

- **make_raw** (*str*) – Make a raw, unprocessed MATCHBox API JSON file. Default filename will be raw_mb_obj (raw MB patient dataset) or raw_ta_obj (raw treatment arm dataset) followed by a datestring. Inputting a a string will save the file with the requested filename.

Returns MATCHBox API dataset, used in `MatchData` or `TreatmentArm` classes.

4.2 matchbox_api_utils.match_data module

class matchbox_api_utils.match_data.**MatchData**(*config_file=None*, *url=None*, *creds=None*, *patient=None*, *json_db='sys_default'*, *load_raw=None*, *make_raw=None*, *quiet=True*)

Bases: object

MatchboxData class

Parsed MATCHBox Data from the API as collected from the Matchbox class above. This class has methods to generate queries, further filtering, and heuristics on the dataset.

Generate a MATCHBox data object that can be parsed and queried downstream with some methods.

Can instantiate with either a config JSON file, which contains the url, username, and password information needed to access the resource, or by supplying the individual arguments to make the connection. This class will call the Matchbox class in order to make the connection and deploy the data.

Can do a live query to get data in real time, or load a MATCHBox JSON file derived from the `matchbox_json_dump.py` script that is a part of the package. Since data in MATCHBox is relatively static these days, it's preferred to use an existing JSON DB and only periodically update the DB with a call to the aforementioned script.

Parameters

- **config_file** (*file*) – Custom config file to use if not using system default.
- **url** (*str*) – MATCHBox API URL to use if not using a config file.
- **creds** (*dict*) – MATCHBox credentials to use if not using a config file. Needs to be in the form of:

  `{'username':<username>, 'password':<password>}`
- **patient** (*str*) – Limit data to a specific PSN.
- **json_db** (*file*) – MATCHbox processed JSON file containing the whole dataset. This is usually generated from 'matchbox_json_dump.py'. The default value is `'sys_default'` which loads the default package data. If you wish you get a live call, set this variable to `None`.
- **load_raw** (*file*) – Load a raw API dataset rather than making a fresh call to the API. This is intended for dev purpose only and will be disabled.
- **make_raw** (*bool*) – Make a raw API JSON dataset for dev purposes only.
- **quiet** (*bool*) – If True, suppress module output debug, information, etc. messages.

get_patient_meta(*psn*, *val=None*)

Return data for a patient based on a metadata field name. Sometimes we may want to just get a quick bit of data or field for a patient record rather than a whole analysis, and this can be a convenient way to just check a component rather than writing a method to get each and every bit out. If no metaval is entered, will return the whole patient dict.

Note: This function is more for debugging and troubleshooting that for real functionality.

Parameters

- **psn** (*str*) – PSN of patient for which we want to receive data.

- **val** (*str*) – Optional metaval of data we want. If no entered, will return the entire patient record.

Returns Either return dict of data if a metaval entered, or whole patient record. Returns `None` if no data for a particular record and raises and error if the metaval is not valid for the dataset.

Return type dict

get_biopsy_summary(*category=None*, *ret_type='counts'*)

Return dict of patients registered in MATCHBox with biopsy and sequencing information.

Categories returned are total PSNs issued (including outside assay patients), total passed biopsies, total failed biopsies (per MDACC message), total MSNs (only counting latest MSN if more than one issued to a biopsy due to a failure) as a method of figuring out how many NAs were prepared, and total with sequencing data.

Can filter output based on any one criteria by leveraging the `category` variable

Parameters

- **catetory** (*str*) – biopsy category to return. Valid categories are:
  - pass
  - failed_biopsy
  - sequenced
  - outside
  - outside_confirmation
  - progression
  - initial
- **ret_type** (*str*) – Data type to return. Valid types are "counts" and "ids", where counts is the total number in that category, and ids are the BSNs for the category. Default is "counts"

Returns Whole set of category : count or single category : count data.

Return type dict

Todo:

- Fix examples.

Examples

```
>>> print(data.get_biopsy_summary())
{u'sequenced': 5620, u'msns': 5620, u'progression': 9,
    u'initial': 5563, u'patients': 6491, u'outside': 61,
    u'no_biopsy': 465, u'failed_biopsy': 574, u'pass': 5654,
    u'outside_confirmation': 21}
```

matchbox_dump(*filename=None*)

Dump a parsed MATCHBox dataset.

Call to the API and make a JSON file that can later be loaded in, rather than making an API call and reprocessing. Useful for quicker look ups as the API call can be very, very slow with such a large DB.

Note: This is a different dataset than the raw dump.

Parameters **filename** (*str*) – Filename to use for output. Default filename is:

```
mb_obj_<date_generated>.json
```

Returns MATCHBox API JSON file.

Return type json

get_psn(*msn=None*, *bsn=None*)

Retrieve a patient PSN from either an input MSN or BSN.

Parameters

- **msn** (*str*) – A MSN number to query.
- **bsn** (*str*) – A BSN number to query.

Returns A PSN that maps to the MSN or BSN input.

Return type str

Examples

```
>>> print(get_psn(bsn='T-17-000550'))
PSN14420
```

get_msn(*psn=None*, *bsn=None*)

Retrieve a patient MSN from either an input PSN or BSN.

Parameters

- **psn** (*str*) – A MSN number to query.
- **bsn** (*str*) – A BSN number to query.

Note: There can always be more than 1 MSN per patient, but can only ever be 1 MSN per biopsy at a time.

Returns A list of MSNs that correspond with the input PSN or BSN.

Return type list

Examples

```
>>> print(get_msn(bsn='T-17-000550'))
[u'MSN44180']
```

```
>>> print(get_msn(bsn='T-16-000811'))
[u'MSN18184']
```

```
>>> print(get_msn(psn='11583'))
[u'MSN18184', u'MSN41897']
```

get_bsn(*psn=None*, *msn=None*)
Retrieve a patient BSN from either an input PSN or MSN.

Parameters

- **psn** (*str*) – A PSN number to query.
- **msn** (*str*) – A MSN number to query.

Note: Can have more than one BSN per PSN, but can only ever have one BSN / MSN.

Returns A list BSNs that correspond to the PSN or MSN input.

Return type list

Examples

```
>>> print(get_bsn(psn='14420'))
[u'T-17-000550']
```

```
>>> print(get_bsn(psn='11583'))
[u'T-16-000811', u'T-17-000333']
```

```
>>> print(get_bsn(msn='18184'))
[u'T-16-000811']
```

get_disease_summary(*query_disease=None*, *query_medra=None*)
Return a summary of registered diseases and counts. With no args, will return a list of all diseases and counts as a dict. One can also limit output to a list of diseases or medra codes and get counts for those only.

Parameters

- **query_disease** (*list*) – List of diseases to filter on.
- **query_medra** (*list*) – List of MEDRA codes to filter on.

Returns

Dictionary of disease(s) and counts in the form of:

```
{medra_code : (ctep_term, count)}
```

Return type dict

Todo:

- Data is good but should try to filter outside assay data, failed specimens, and progression biopsies (collapsed into one count) to get a value that is closer to the accepted final count.

Examples

```
>>> data.get_disease_summary(query_medra=['10006190'])
{'10006190': (u'Invasive breast carcinoma', 641)}
```

```
>>> data.get_disease_summary(query_disease=['Invasive breast carcinoma'])
{u'10006190': ('Invasive breast carcinoma', 641)}
```

```
>>> data.get_disease_summary(query_medra=[10006190,10024193,10014735])
{'10006190': (u'Invasive breast carcinoma', 641),
 '10014735': (u'Endometrioid endometrial adenocarcinoma', 124),
  '10024193': (u'Leiomyosarcoma (excluding uterine leiomyosarcoma)', 57)}
```

get_histology (*psn=None*, *msn=None*, *bsn=None*, *outside=False*, *no_disease=False*)

Return dict of PSN:Disease for valid biopsies. Valid biopsies can are defined as being only *Passed* and can not be *Failed*, *No Biopsy* or outside assay biopsies at this time.

Parameters

- **psn** (*str*) – Optional PSN or comma separated list of PSNs on which to filter data.
- **bsn** (*str*) – Optional BSN or comma separated list of BSNs on which to filter data.
- **msn** (*str*) – Optional MSN or comma separated list of MSNs on which to filter data.
- **outside** (*bool*) – Also include outside assay data. Default: `False`
- **no_disease** (*bool*) – Return all data, even if there is no disease indicated for the patient specimen. Default: `False`

Returns Dict of ID : Disease mappings. If no match for input ID, returns `None`.

Return type dict

Examples

```
>>> data.get_histology(psn='11352')
{'PSN11352': u'Serous endometrial adenocarcinoma'}
```

```
>>> data.get_histology(msn=3060)
No result for id MSN3060
{'MSN3060': None}
```

find_variant_frequency (*query*, *query_patients=None*)

Find and return variant hit rates.

Based on an input query in the form of a *variant_type : gene* dict, where the gene value can be a list of genes, output a list of patients that had hits in those gene with some disease and variant information.

The return val will be unique to a patient. So, in the cases where we have multiple biopsies from the same patient (an intitial and progression re-biopsy for example), we will only get the union of the two sets, and duplicate variants will not be output. This will preven the hit rate from getting over inflated. Also, there is no sequence specific information output in this version (i.e. no VAF, Coverage, etc.). Sequence level information for a call can be obtained from the method *get_variant_report*

Parameters

- **query** (*dict*) – Dictionary of variant_type: gene mappings where:

```
-  variant type is one or more of 'snvs', 'indels', 'fusions',
    'cnvs'
-  gene is a list of genes to query.
```

- **query_patients** (*list*) – List of patients for which we want to obtain data.

Returns Return a dict of matching data with disease and MOI information, along with a count of the number of patients queried and the number of biopsies queried.

Return type dict

Examples

```
>>> query={'snvs' : ['BRAF','MTOR'], 'indels' : ['BRAF', 'MTOR']}
find_variant_frequency(query)
```

```
>>> pprint(data.find_variant_frequency({'snvs':['EGFR'], 'indels':['EGFR']},␣
↪[15232]))
({'15232': {'bsns': [u'T-17-001423'],
            'disease': u'Lung adenocarcinoma',
            'mois': [{'alternative': u'T',
                      'amoi': [u'EAY131-E(i)', u'EAY131-A(e)'],
                      'chromosome': u'chr7',
                      'confirmed': True,
                      'exon': u'20',
                      'function': u'missense',
                      'gene': u'EGFR',
                      'hgvs': u'c.2369C>T',
                      'identifier': u'COSM6240',
                      'oncominevariantclass': u'Hotspot',
                      'position': u'55249071',
                      'protein': u'p.Thr790Met',
                      'reference': u'C',
                      'transcript': u'NM_005228.3',
                      'type': u'snvs_indels'},
                     {'alternative': u'-',
                      'amoi': [u'EAY131-A(i)'],
                      'chromosome': u'chr7',
                      'confirmed': True,
                      'exon': u'19',
                      'function': u'nonframeshiftDeletion',
                      'gene': u'EGFR',
                      'hgvs': u'c.2240_2257delTAAGAGAAGCAACATCTC',
                      'identifier': u'COSM12370',
                      'oncominevariantclass': u'Hotspot',
                      'position': u'55242470',
                      'protein': u'p.Leu747_Pro753delinsSer',
                      'reference': u'TAAGAGAAGCAACATCTC',
                      'transcript': u'NM_005228.3',
                      'type': u'snvs_indels'}],
            'msns': [u'MSN52258'],
            'psn': u'15232'}},
1)
```

get_variant_report (*psn=None*, *msn=None*)

Input a PSN or MSN **(preferred!)** and return a list of dicts of variant call data.

Since there can be more than one MSN per patient, one will get a more robust result by querying on a MSN. That is, only one variant report per MSN can be generated and the results, then, will be clear. In the case of querying by PSN, a variant report for each MSN under that PSN, assuming that the MSN is associated with a variant report, will be returned.

Parameters

- **msn** (*str*) – MSN for which a variant report should be returned.
- **psn** (*str*) – PSN for which the variant reports should be returned.

Returns

List of dicts of variant results.

```
msn: {
    'singleNucleotideVariants' : [{var_data}],
    'copyNumberVariants' : [{var_data},{var_data}], etc.
}
```

Return type dict

Examples:

Todo: Add examples here.

---

get_patient_ta_status(*psn=None*)

Input a list of PSNs and return information about the treatment arm(s) to which they were assigned, if they were assigned to any arms. If no PSN list is passed to the function, return results for every PSN in the study.

Parameters **psn** (*str*) – PSN string to query.

Returns Dict of Arm IDs with last status.

Return type dict

Examples

```
>>> data.get_patient_ta_status(psn=10837)
{u'EAY131-Z1A': u'ON_TREATMENT_ARM'}
```

```
>>> data.get_patient_ta_status(psn=11889)
{u'EAY131-IX1': u'FORMERLY_ON_ARM_OFF_TRIAL',
        u'EAY131-I': u'COMPASSIONATE_CARE'}
```

```
>>> data.get_patient_ta_status(psn=10003)
{}
```

get_patients_by_disease(*histology=None*, *medra_code=None*)

Input a disease and return a list of patients that were registered with that disease type. For histology query, we can do partial matching based on the python `in` function. So, if one were to query *Lung Adenocarcinoma*, *Lung* , *Lung Adeno*, or *Adeno*, all Lung Adenocarinoma cases would be returned.

Note: Simply inputting *Lung*, would also return *Non-small Cell Lung Adenocarinoma*, *Squamous Cell Lung Adenocarcinoma*, etc, and querying *Adeno* would return anything that had *adeno*. So, care must

be taken with the query, and secondary filtering may be necessary. Querying based on MEDRA codes is specific and only an exact match will return results; **This is the preferred method.**

Parameters

- **histology** (*str*) – One of the CTEP shotname disease codes.
- **medra_code** (*str*) – A MEDRA code to query rather than histology.

Returns Dict of Patient : Histology Mapping

Return type dict

Examples

```
>>> <put example here>
```

get_patients_by_arm(*arm*)

Input an official NCI-MATCH arm identifier (e.g. *EAY131-A*) and return a set of patients that have ever qualified for the arm based on variant level data. This not only includes patients *ON_TREATMENT_ARM*, but also *FORMERLY_ON_ARM_OFF_TRIAL* and even *COMPASSIONATE_CARE*.

Parameters **arm** (*str*) – One of the official NCI-MATCH arm identifiers.

Returns List of tuples of patient, arm, and arm_status.

Return type list

Examples

```
>>> print(self.get_patients_by_arm(arm='EAY131-E'))
[
    (u'11476', 'EAY131-E', u'OFF_TRIAL_DECEASED'),
    (u'14343', 'EAY131-E', u'FORMERLY_ON_ARM_OFF_TRIAL'),
    (u'10626', 'EAY131-E', u'ON_TREATMENT_ARM'),
    (u'14256', 'EAY131-E', u'ON_TREATMENT_ARM'),
    (u'16472', 'EAY131-E', u'FORMERLY_ON_ARM_OFF_TRIAL')
]
```

get_ihc_results(*psn=None*, *msn=None*, *bsn=None*, *assays=None*)

Get the IHC results for a patient.

Input a PSN, MSN, or BSN and / or a set of IHC assays, and return a dict of data.

Parameters

- **psn** (*str*) – Query the data by PSN.
- **msn** (*str*) – Query the data by MSN.
- **bsn** (*str*) – Query the data by BSN.
- **assay** (*list*) – IHC assay for which we want to return results. If no assay is passed, will return all IHC assay results.

Note: Multiple results will be returned since there can be more than one MSN / PSN.

Returns Dict of lists containing the MSN and all IHC assays available for the specimen.

Return type dict

Examples

```
>>> self.get_ihc_results(msn='MSN30791')
{'MSN30791': {'MLH1': u'POSITIVE',
              'MSH2': u'POSITIVE',
              'PTEN': u'POSITIVE',
              'RB': u'ND'}}
```

```
>>> get_ihc_results(bsn='T-16-002222', assays=['PTEN'])
{u'MSN30791': {'PTEN': u'POSITIVE'}}
```

4.3 matchbox_api_utils.match_arms module

class matchbox_api_utils.match_arms.**TreatmentArms**(*config_file=None*, *url=None*, *creds=None*, *json_db='sys_default'*, *load_raw=None*, *make_raw=False*)

Bases: object

NCI-MATCH Treatment Arms and aMOIs Class

Generate a MATCHBox treatment arms object that can be parsed and queried downstream.

Can instantiate with either a config JSON file, which contains the url, username, and password information needed to access the resource, or by supplying the individual arguments to make the connection. This class will call the Matchbox class in order to make the connection and deploy the data.

Can do a live query to get data in real time, or load a MATCHBox JSON file derived from the matchbox_json_dump.py script that is a part of the package. Since data in MATCHBox is relatively static these days, it's preferred to use an existing JSON DB and only periodically update the DB with a call to the aforementioned script.

Parameters

- **config_file** (*file*) – Custom config file to use if not using system default.
- **url** (*str*) – MATCHBox API URL to use.
- **creds** (*dict*) – MATCHBox credentials to use. Needs to be in the form of:

```
{ 'username':<username>, 'password':<password> }
```

- **json_db** (*file*) – MATCHbox processed JSON file containing the whole dataset. This is usually generated from `'matchbox_json_dump.py'`. The default value is `'sys_default'` which loads the default package data. If you wish you get a live call, set this variable to *"None"*.
- **load_raw** (*file*) – Load a raw API dataset rather than making a fresh call to the API. This is intended for dev purpose only and will be disabled in production.
- **make_raw** (*bool*) – Make a raw API JSON dataset for dev purposes only.

Returns MATCH Treatment Arms object of data and methods.

ta_json_dump(*amois_filename=None*, *ta_filename=None*)
Dump the TreatmentArms data to a JSON file that can be easily loaded downstream. We will make both the treatment arms object, as well as the amois lookup table object.

Parameters

- **amois_filename** (*str*) – Name of aMOI lookup JSON file. **Default:** *amoi_lookup_<datestring>.json*
- **ta_filename** (*str*) – Name of TA object JSON file **Default:** *ta_obj_<datestring>.json*

Returns ta_obj_<date>.json amois_lookup_<date>.json

Return type json

make_match_arms_db(*api_data*)
Make a database of MATCH Treatment Arms

Read in raw API data and create pared down JSON structure that can be easily parsed later one.

Parameters **api_data** (*json*) – Entire raw MATCHBox API retrieved dataset.

Returns All Arm data.

Return type json

map_amoi(*variant*)
Input a variant dict derived from some kind and return either an aMOI id in the form of Arm(ile). If variant is not an aMOI, returns `'None'`.

Parameters **variant** (*dict*) – Variant dict to annotate. Dict must have the following keys in order to be valid:

```
- type : [snvs_indels, cnvs, fusions]
- oncominevariantclass
- gene
- identifier (i.e. variant ID (COSM476))
- exon
- function
```

Not all variant types will have meaningful data for these fields, and so fields may be padded with a null char (e.g. '.', '-', 'NA', etc.).

Returns list: Arm ID(s) with (i)nclusion or (e)xclusion information.

Examples

```
>>> variant = {
    'type' : 'snvs_indels',
    'gene' : 'BRAF',
    'identifier' : 'COSM476',
    'exon' : '15',
    'function' : 'missense' ,
    'oncominevariantclass' : 'Hotspot'
}
self.map_amoi(variant)
['EAY131-Y(e)', 'EAY131-P(e)', 'EAY131-N(e)', 'EAY131-H(i)']
```

Todo: Check the examples work.

map_drug_arm(*armid=None*, *drugname=None*)

Input an Arm ID or a drug name, and retun a tuple of arm, drugname, and ID. If no arm ID or drug name is input, will return a whole table of all arm data.

Parameters

- **armid** (*str*) – Offcial NCI-MATCH Arm ID in the form of *EAY131-xxx* (e.g. 'EAY131-Z1A').
- **drugname** (*str*) – Drug name as registered in the NCI-MATCH subprotocols. Right now, required to have the full string (e.g. 'MLN0128(TAK-228)' or, unfortunately, 'Sunitinib malate (SU011248 L-malate)'), but will work on a regex to help make this easier later on.

Returns List of tuples or `None`.

Return type list

Examples

```
>>> map_drug_arm(armid='EAY131-Z1A')
(u'EAY131-Z1A', 'Binimetinib', u'788187')
```

Todo: Write regex for drug name mapping so that we don't need to deal with long, cryptic strings.

get_exclusion_disease(*armid*)

Input an arm ID and return a list of exclusionary diseases for the arm, if there are any. Otherwise return `None`.

Parameters **armid** (*str*) – Full identifier of the arm to be queried.

Returns List of exclusionary diseases for the arm, or `None` if there aren't any.

Return type list

Example

```
>>> get_exclusion_disease('EAY131-Z1A')
[u'Melanoma', u'Colorectal Cancer']
```

```
>>> get_exclusion_disease('EAY131-Y')
None
```

get_amois_by_arm(*arm*)

Input an arm identifier and return a list of aMOIs for the arm broken down by category.

Parameters **arm** (*str*) – Arm identifier to query

Returns:

Todo: Need to finish implementing this method. Currently only dumping a dict.

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

m

Index