MATCHBox API Utils Documentation

Release 3.5.20190221

Dave Sims

Feb 21, 2019

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(matchbox='adult-matchbox')
>>> 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
```

Attention: While throughout these docs the need to always make sure one specifies which MATHCBox they intend on connecting to and pulling data from is mentioned frequently, at the time of this publication (4/20/2018), the established methods have not been verified to work on other versions of MATCHBox. Those components will be added and available soon, which is why getting into the habit of specifying MATCHBox systems is important (especially in the event of writing code using these methods, where changes will be necessary later).

1.2 Installation

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

```
python3 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.

Note: Alternatively, one can install with Python `easy_install`, but there may be some issues with the postinstaller script, and so this method is not preferred.

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 require Python >=3.5 to run. In addition to the above, the package requires the installation of:

- requests
- termcolor

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.

Note: Live queries to MATCHBox can take a long time to perform due to network traffic, and the very large size of the database, and is not the preferred way to work with the data. Especially since most of the data is in a final, locked state, it's recommended to use a JSON snapshot of data when possible.

Attention: Now that we have multiple MATCHBoxes in production for different studies, one will always need to specify which MATCHBox they are intending to connect to when initializing any of these classes. By default, the Adult MATCHBox is loaded since this is the most frequently used. But, that is likely to change without warning, and you are advised to explicityly specify which system you are interested in calling.

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(matchbox='adult', method='mongo', mongo_collection='patient',
...     make_raw='mb')
```

This will create a complete raw 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, as one can see with regard to how many required arguments there are to configure the API call, most of which are set up in the configuration file generated upon installation, `class MatchData()` is a much better entry point, and only requires one argument to generate the same file, as shown below.

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(matchbox='adult')
>>> data.get_biopsy_summary(category='progression', ret_type='counts')
{'progression': 19}
```

Note: The resultant data above may be different than your result depending on when the dataset was generated!

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 `sys_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. In this case passing a PSN to the `patient` argument will allow one to filter data to only that patient when reading the `sys_default` JSON file, or, and arguably better, getting live and up to date data for only that patient from MATCHBox.

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:

```
.. note::
    I am listing all of the options to call these methods, even when some
    of them are default, just to give a sense of what is possible.  See the
    detailed docs for a list of default args.

>>> from matchbox_api_utils import MatchData
>>> from pprint import pprint as pp
>>> data = MatchData(matchbox='adult', method='mongo', json_db=None,
...    quiet=False)
>>> 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 unfortu-

nately, 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. This class has a set of useful functions to generate arm related data, including mapping patients to arms and variants to arms.

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(matchbox='adult', method='mongo', json_db=None,
...     quiet=False)
```

Or, 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`.

Once 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 some NCI-MATCH level variant data (typically from Ion Reporter / OVAT) 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. 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(matchbox='adult', method='mongo', json_db=None,
...   quiet=False)
>>> 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'))
['Papillary thyroid carcinoma',
 'Melanoma',
 'Malignant Melanoma of sites other than skin or eye',
 'Acral Lentiginous Melanoma',
 'Adenocarcinoma of the colon',
 'Adenocarcinoma of the rectum',
 'Colorectal cancer, NOS',
 'Bronchioloalveolar carcinoma',
 'Lung adenocarcinoma',
 'Lung adenocarcinoma with bronchioloalveolar features',
 'Non-small cell lung cancer, NOS',
 'Squamous cell lung carcinoma']
```

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 NCI-MATCH Arm.

Note: This mapping functionality is very simple and only relies on the reported histology and the input biomarker. There are many other NCI-MATCH trial factors that determine eligibility, which are way outside the scope of this utility. In essence, this is not meant to be a treatment assignment utility and is only meant to help classify variants.

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

Included in the package are a few pre-made helper scripts for routine work. In general you'll get more mileage out of just loading the modules and rolling your own. However, there are some very frequent use cases where one of these pre-made scripts might be helpful.

3.1 MATCHBox JSON Dump (matchbox_json_dump.py)

Get a 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.

Note: The older JSON database files are retained with the new one, and as such can be loaded into the API at any time, in the event that one requires older data.

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>] [-c <connection_method>] [-v]
                             <matchbox>

positional arguments:
  <matchbox>            Name of MATCHBox to which we make the file. Valid
                        systems are: "adult", "adult-uat", "ped".
```

(continues on next page)

(continued from previous page)

```
optional arguments:
  -h, --help            show this help message and exit
  -d <raw_mb_datafile.json>, --data <raw_mb_datafile.json>
                        Load a raw MATCHBox database file (usually after
                        running with the -r option).
  -r, --raw             Generate a raw dump of MATCHbox for debugging and dev
                        purposes.
  -p <psn>, --patient <psn>
                        Patient sequence number used to limit output for
                        testing and dev purposes
  -t <ta_obj.json>, --ta_json <ta_obj.json>
                        Treatment Arms obj JSON filename. DEFAULT:
                        ta_obj_<datestring>.json
  -a <amoi_obj.json>, --amoi_json <amoi_obj.json>
                        aMOIs lookup filename. DEFAULT:
                        "amois_lookup_<datestring>.json".
  -m <mb_obj.json>, --mb_json <mb_obj.json>
                        Name of Match Data obj JSON file. DEFAULT:
                        "mb_obj_<datestring>.json".
  -c <connection_method>, --connection <connection_method>
                        Connection method used to access MATCHBox data. Choose
                        from either "api" or "mongo". DEFAULT: mongo
  -v, --version         show program\'s version number and exit
```

Note that we can either retrieve a small, parsed JSON object of data, or we can get an entire dump of the raw MATCHBox for things like development and testing, or troubleshooting.

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.

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] -t {psn,msn,bsn} [-l] [-f <input_file>]
                      [-o <outfile>] [-v]
                      <matchbox> [<IDs>]

positional arguments:
  <matchbox>            Name of MATCHBox to which we make the connection.
                        Valid systems are: "adult", "adult-uat", "ped".
  <IDs>                 MATCH IDs to query. Can be single or comma separated
                        list. Must be used with PSN or MSN option.

optional arguments:
  -h, --help            show this help message and exit
  -t {psn,msn,bsn}, --type {psn,msn,bsn}
```

(continues on next page)

(continued from previous page)

```
                        Type of query string input. Can be MSN, PSN, or BSN
  -l, --live            Make a live call to MATCHbox instead of relying on
                        local JSON database. This is especially helpful for
                        newly sequenced patients since the last dump.
  -f <input_file>, --file <input_file>
                        Load a batch file of all MSNs or PSNs to proc
  -o <outfile>, --outfile <outfile>
                        File to which output should be written. Default:
                        STDOUT.
  -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. 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.

3.3.1 MATCH Variant Frequency Help Docs

```
usage: match_variant_frequency.py [-h] [-l] [-p <PSN>] [-s <gene_list>]
                                  [-c <gene_list>] [-f <gene_list>]
                                  [-i <gene_list>] [-a <all_types>]
                                  [--style <pp,csv,tsv>] [-o <output_file>]
                                  [-v]
                                  <matchbox>
positional arguments:
  <matchbox>            Name of MATCHBox system to which the connection should
                        be made. Valid names are "adult", "ped", "adult-uat".

optional arguments:
  -h, --help            show this help message and exit
  -l, --live            Get a live MATCHBox query instead of loading a local
                        JSON filederived from "matchbox_json_dump.py"
  -p <PSN>, --psn <PSN>
                        Only output data for a specific patient or comma
                        separated list of patients
  -s <gene_list>, --snv <gene_list>
                        Comma separated list of SNVs to look up in MATCHBox
                        data.
  -c <gene_list>, --cnv <gene_list>
                        Comma separated list of CNVs to look up in MATCHBox
                        data.
  -f <gene_list>, --fusion <gene_list>
                        Comma separated list of Fusions to look up in MATCHBox
                        data.
  -i <gene_list>, --indel <gene_list>
                        Comma separated list of Fusions to look up in MATCHBox
                        data.
  -a <all_types>, --all <all_types>
                        Query variants across all variant types for a set of
                        genes, rather than one by one. Helpful if one wants to
                        find any BRAF MOIs, no matter what type, for example.
```

(continues on next page)

(continued from previous page)

```
  --style <pp,csv,tsv>  Format for output. Can choose pretty print (pp), CSV,
                        or TSV
  -o <output_file>, --output <output_file>
                        Output file to which to write data Default is stdout
  -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. Choosing the `patient` option will allow one to get a listing of patients in the study and their respective disease. One can also filter that list down by specifying a PSN (or comma separated list of PSNs) of interest Choosing the `disease` option will give a summary of the types and counts of each disease in the study. Similar to the patients query, one can filter the list down by inputting MEDDRA codes or tumor hisologies.

Note: Note that you must quote tumor names with spaces in them, and they must exactly match the string indicated in MATCHBox. The use of MEDDRA codes is recommended and preferred.

3.4.1 MATCHBox Patient Summary Help Docs

```
usage: matchbox_patient_summary.py [-h] [-l] [-p PSN] [-t <tumor_type>]
                                   [-m <meddra_code>] [-O] [-o <output csv>]
                                   [-v]
                                   <matchbox> {patient,disease}
positional arguments:
  <matchbox>            Name of MATCHBox system to which the connection will
                        be made. Valid systems are "adult", "ped", "adult-
                        uat".
  {patient,disease}     Category of data to output. Can either be patient or
                        disease level.

optional arguments:
  -h, --help            show this help message and exit
  -l, --live            Make a live call to MATCHBox rather than loading a
                        local JSON containing patient data, usually from
                        matchbox_json_dump.py
  -p PSN, --psn PSN     Filter patient summary to only these patients. Can be
                        a comma separated list
  -t <tumor_type>, --tumor <tumor_type>
                        Retrieve data for only this tumor type or comma
                        separate list of tumors. Note that you must quote
                        tumors with names containing spaces.
  -m <meddra_code>, --meddra <meddra_code>
                        MEDDRA Code or comma separated list of codes to
                        search.
  -O, --Outside         Include Outside Assay study data (DEFAULT: False).
  -o <output csv>, --outfile <output csv>
                        Name of output file. Output will be in CSV format.
                        DEFAULT: STDOUT.
  -v, --version         show program\'s version number and exit
```

3.5 MATCHBox Arm Enrollment Summary (matchbox_arm_enrollment_summary.py)

Script to output NCI-MATCH arm enrollment data similar to a Treatment Arms page of MATCHBox, outlining which patients were eligible (based on the presence of an appropriate aMOI), and their status.

3.5.1 MATCHBox Arm Enrollment Summary Help Docs

```
usage: match_arm_enrollment_summary.py [-h] [-a] [-O] [-o <outfile>] [-v]
                                       <ARM ID>

Input a valid NCI-MATCH Arm ID and get a list of patients and trial status
information. Output intended to be similar to Treatment arms page of MATCHbox.

positional arguments:
  <ARM ID>              Valid NCI-MATCH study arm, or comma separated list of
                        study arms, in the format of EAY131-*.

optional arguments:
  -h, --help            show this help message and exit
  -a, --all             Output all patients and do not filter out those that
                        received compassionate care or other outcomes. Only
                        output patients that were enrolled.
  -O, --Outside         Include Outside Assays results in output. This may
                        cause some problems with mapping and whatnot as the
                        data are a bit scattershot. You have been warned!
  -o <outfile>, --outfile <outfile>
                        Write results to a file instead of stdout.
  -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**(*method*, *config*, *params={}*, *mongo_collection=None*, *make_raw=None*, *quiet=False*)

Bases: object

MATCHBox API Connector Class

Basic connector class to make a call to the API and load the raw data. Used for calling to the MATCHBox API, loading data, and pass along to the appropriate calling classes for make 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

- **method** (*str*) – API call method to use. Can only choose from `api`, the conventional way to make the call to MATCHBox using the actual API, or `mongo`, the new, preferred way, connecting directly to the MongoDB.
- **config** (*dict*) – Dictionary of config variables to pass along to this object. These are generated by parsing the MATCHBox API Utils config file, and usually contains things like the username, password, URL, etc.
- **mongo_collection** (*str*) – This is the name of the MongoDB database collection for which you want to get data. For now, we are only using the `patients` and `treatmentArms` tables, and so these are the only accepted values. In the near future, though, all tables will be incorporated.
- **params** (*dict*) – Parameters to pass along to the API in the request. This is onyl when using `api` with the `method` arg. For example, if you wanted to add "is_oa=True" to the API URL, you can add:

```
params={'is_oa' : True}
```

and this will be passed along to the request.

- **make_raw** (*str*) – Make a raw, unprocessed MATCHBox API JSON file. Default filename will be `raw_mb_obj` for the raw MATCHBox patient dataset, or `raw_ta_obj` for the raw treatment arm dataset. Each will also contain the date string of generation. Inputting a string will save the file with requested filename.
- **quiet** (*bool*) – Suppress debug and information messages.

Todo: Fix this arg and make it something like `verbosity`, which will link to the utils.msg() function and allow output at certain levels.

4.2 matchbox_api_utils.match_data module

class matchbox_api_utils.match_data.**MatchData**(*matchbox='adult'*, *method='mongo'*, *config_file=None*, *username=None*, *password=None*, *patient=None*, *json_db='sys_default'*, *load_raw=None*, *make_raw=None*, *quiet=False*)

Bases: `object`

NCI-MATCH MATCHBox Data class

Parsed MATCHBox Data from the API as collected from the Matchbox class. 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

- **matchbox** (*str*) – Name of the MATCHBox system from which we want to get data. This is required now that we have several systems to choose from. Valid names are `adult`, `ped`, and `adult-uat` for those that have access to the adult MATCHBox test system. **DEFAULT:** `adult`.
- **method** (*str*) – MATCHBox connection method. Can choose from `api` or `mongo` if one wants to use the old API method or the new MongoDB connection method.

Note: The API method is to be deprecated and using the MongoDB method is much preferred.

- **config_file** (*file*) – Custom config file to use if not using system default.
- **username** (*str*) – Username required for access to the MATCHBox. Typically this is already present in the config file made during setup, but in cases where needed, it can be explicitly defined here.

- **password** (*str*) – Password associated with the user. As with the above username argument, this is typically indicated in the config file generated during setup.
- **patient** (*str*) – Limit data to a specific PSN.

Note: When trying to limit the database data to only one patient, this only works for the API call method. However, limiting data to one patient is capable in most of the package methods and the `mongo` method is fast, so this is not an issue.

- **json_db** (*file*) – MATCHbox processed JSON file containing the whole dataset. This is usually generated from 'matchbox_json_dump.py'. By default the package JSON file, located in `mb_root` is loaded if no value passed to this argument. 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 may be disabled later.
- **make_raw** (*bool*) – Make a raw API JSON dataset for dev purposes only. This will be the file used with the `load_raw` option.
- **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 nothing 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:
  - patients - Any patient that has ever registered.
  - failed_biopsy - Number of failed biopsies.

- no_biopsy - Number of patient for which no biopsy collected.
- pass - Number of biopsies that passed.
- sequenced - Number of sequencing results available.
- outside - Number of outside assay cases.
- confirmation - Number of confirmation sequences for outside assay results.
- progression - Number of progression biopsy cases.
- initial - Number of non-outside assay cases.

- **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 Dictionary of whole set of `{category : count}` or `{single category : count}` data.

Return type dict

Examples

```
>>> print(data.get_biopsy_summary())
{'patients': 6560, 'failed_biopsy': 576, 'no_biopsy': 465,
 'pass': 5776, 'initial': 5563, 'sequenced': 5748,
 'progression': 19, 'outside': 130, 'confirmation': 64}
```

```
>>> print(data.get_biopsy_summary(category='progression'))
{'progression': 19}
```

```
>>> print(data.get_biopsy_summary(category='progression',
...     ret_type='ids')
{'progression': ['T-17-001275',
     'T-17-000787',
     'T-17-001175',
     'T-17-002730',
     'T-17-002657',
     'T-17-000333',
     'T-17-002564',
     'T-17-002556',
     'T-17-002600',
     'T-17-002064',
     'T-18-000113',
     'T-17-002755',
     'T-17-002621',
     'T-18-000005',
     'T-17-002680',
     'T-18-000071',
     'T-18-000171',
     'T-18-000123',
     'T-18-000031']}
```

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
```

```
>>> print(get_psn(msn='57471'))
PSN15971
```

Warning: I have found at least one example where there was a duplicate BSN used for a patient, and so great care must be used if trying to map this BSN to other data (see PSNs 12913 and 12850)!

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']
```

```
>>> print(get_bsn(psn='11586'))
None
```

get_disease_summary (*query_disease=None*, *query_meddra=None*, *outside=False*)
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 meddra codes and get counts for those only.

Parameters

- **query_disease** (*list*) – List of diseases to filter on.
- **query_meddra** (*list*) – List of MEDDRA codes to filter on.
- **outside** (*bool*) – Include patients registered under outside assay initiative in counts. DEFAULT: False

Returns

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

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

Return type dict

Examples

```
>>> data.get_disease_summary(query_meddra=['10006190'])
{'10006190': (u'Invasive breast carcinoma', 605)}
```

```
>>> data.get_disease_summary(query_disease=['Invasive breast carcinoma'])
{u'10006190': ('Invasive breast carcinoma', 605)}
```

```
>>> data.get_disease_summary(query_meddra=[10006190,10024193,10014735])
{'10006190': (u'Invasive breast carcinoma', 605),
 '10014735': (u'Endometrioid endometrial adenocarcinoma', 111),
  '10024193': (u'Leiomyosarcoma (excluding uterine leiomyosarcoma)', 55)}
```

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

Return dict of PSN:Disease for valid biopsies. Valid biopsies 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`
- **ret_type** (*str*) – Type of data to return. Can only either be one of 'ctep_term' or 'meddra_code'

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

Return type dict

Examples

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

```
>>> data.get_histology(psn='12104,12724,12948,13367,15784')
WARN: The following specimens were filtered from the output due to
either the "outside" or "no_disease" filters:
     12724,15784
{'PSN12104': 'CNS primary tumor, NOS',
```

(continues on next page)

(continued from previous page)

```
 'PSN12948': 'Cholangiocarcinoma, intrahepatic and extrahepatic '
             'bile ducts (adenocarcinoma)',
 'PSN13367': 'Adenocarcinoma of the pancreas'}
```

```
>>> data.get_histology(msn='12104,12724,12948,13367,15784')
{
    'MSN15748': None,
    'MSN12104': 'Colorectal cancer, NOS',
    'MSN12724': 'Squamous cell carcinoma of the anus',
    'MSN12948': 'Adenocarcinoma of the colon',
    'MSN13367': 'Invasive breast carcinoma'
}
```

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

```
>>> data.get_histology(psn='11352', ret_type='meddra_code')
{'PSN11352' : '10033700'}
```

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 prevent 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)
>>> # Note the result list is too long to print here.
```

```
>>> query = {'snvs':['EGFR'], 'indels':['EGFR']}
>>> data.find_variant_frequency(query, [15232])[0]
{'15232': {'bsns': ['T-17-001423'],
'disease': 'Lung adenocarcinoma',
'mois': [{'alleleFrequency': 0.191596,
          'alternative': 'T',
          'amoi': ['EAY131-A(e)', 'EAY131-E(i)'],
          'chromosome': 'chr7',
          'confirmed': True,
          'exon': '20',
          'function': 'missense',
          'gene': 'EGFR',
          'hgvs': 'c.2369C>T',
          'identifier': 'COSM6240',
          'oncominevariantclass': 'Hotspot',
          'position': '55249071',
          'protein': 'p.Thr790Met',
          'reference': 'C',
          'transcript': 'NM_005228.3',
          'type': 'snvs_indels'},
         {'alleleFrequency': 0.748107,
          'alternative': '-',
          'amoi': ['EAY131-A(i)'],
          'chromosome': 'chr7',
          'confirmed': True,
          'exon': '19',
          'function': 'nonframeshiftDeletion',
          'gene': 'EGFR',
          'hgvs': 'c.2240_2257delTAAGAGAAGCAACATCTC',
          'identifier': 'COSM12370',
          'oncominevariantclass': 'Hotspot',
          'position': '55242470',
          'protein': 'p.Leu747_Pro753delinsSer',
          'reference': 'TAAGAGAAGCAACATCTC',
          'transcript': 'NM_005228.3',
          'type': 'snvs_indels'}],
'msns': ['MSN52258'],
'psn': '15232'}}
```

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}],
```

(continues on next page)

(continued from previous page)

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

Return type dict

Examples

```
>>> data.get_variant_report(psn=10005)
{'MSN3111': {'unifiedGeneFusions': [{'amoi': None,
                         'annotation': 'COSF1232',
                         'confirmed': True,
                         'driverGene': 'RET',
                         'driverReadCount': 7121,
                         'gene': 'RET',
                         'identifier': 'KIF5B-RET.K15R12.COSF1232',
                         'partnerGene': 'KIF5B',
                         'type': 'fusions'}]}}
```

```
>>> data.get_variant_report(msn=35733)
{
    'MSN35733': {
        'singleNucleotideVariants': [
            {
                'alleleFrequency': 0.570856,
                'alternative': 'T',
                'alternativeAlleleObservationCount': 3190,
                'amoi': ['EAY131-Z1I(i)'],
                'chromosome': 'chr13',
                'confirmed': True,
                'exon': '25',
                'flowAlternativeAlleleObservationCount': '1140',
                'flowReferenceAlleleObservations': '177',
                'function': 'nonsense',
                'gene': 'BRCA2',
                'hgvs': 'c.9382C>T',
                'identifier': '.',
                'oncominevariantclass': 'Deleterious',
                'position': '32968951',
                'protein': 'p.Arg3128Ter',
                'readDepth': 5551,
                'reference': 'C',
                'referenceAlleleObservations': 456,
                'transcript': 'NM_000059.3',
                'type': 'snvs_indels'
            }
        ]
    }
}
```

get_patient_ta_status(*psn=None*)

Input a PSN and return information about the treatment arm(s) to which they were assigned, if they were assigned to any arms. If no PSN 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)
{'EAY131-Z1A': 'FORMERLY_ON_ARM_OFF_TRIAL'}
```

```
>>> 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*, *meddra_code=None*, *outside=False*)

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. It may be more robust to just stick with MEDDRA codes as they would to be more precise

---

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 MEDDRA 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.
- **meddra_code** (*str*) – A MEDDRA code to query rather than histology.
- **outside** (*bool*) – Include Outside Assays patients in the results.

Returns Dict of Patient : Histology Mapping

Return type dict

Examples

```
>>> data.get_patients_by_disease(histology='glioma')
{'10512': 'Oligodendroglioma, NOS',
 '13496': 'Anaplastic oligodendroglioma',
 '13511': 'Anaplastic oligodendroglioma',
 '16124': 'Anaplastic oligodendroglioma',
 '16160': 'Anaplastic oligodendroglioma',
 '16248': 'Anaplastic oligodendroglioma'}
```

We see here that Small Cell and Non-small cell get combined.

```
>>> ret_list = data.get_patients_by_disease(
...     histology='Small cell lung cancer').values()
>>> print('Total returned: {}'.format(len(ret_list)))
102
```

(continues on next page)

(continued from previous page)

```
>>> print(set(ret_list))
{'Small cell lung cancer', 'Non-small cell lung cancer, NOS'}
```

Here we distinguish and only get the Non-small cell cases, by using a MEDDRA code

```
>>> meddra = utils.map_histology(
...     self._disease_db,
...     histology='Non-small cell lung cancer, NOS')
```

```
>>> data.get_patients_by_disease(meddra_code=meddra)
{'10196': 'Non-small cell lung cancer, NOS',
 '10312': 'Non-small cell lung cancer, NOS',
 '10540': 'Non-small cell lung cancer, NOS',
 '11850': 'Non-small cell lung cancer, NOS',
 '11929': 'Non-small cell lung cancer, NOS',
 '12541': 'Non-small cell lung cancer, NOS',
 '12577': 'Non-small cell lung cancer, NOS',
 '12790': 'Non-small cell lung cancer, NOS',
 '12916': 'Non-small cell lung cancer, NOS',
 '13187': 'Non-small cell lung cancer, NOS',
 '13242': 'Non-small cell lung cancer, NOS',
 '13620': 'Non-small cell lung cancer, NOS',
 '14256': 'Non-small cell lung cancer, NOS',
 '15097': 'Non-small cell lung cancer, NOS',
 '15114': 'Non-small cell lung cancer, NOS',
 '16095': 'Non-small cell lung cancer, NOS',
 '16212': 'Non-small cell lung cancer, NOS',
 '16412': 'Non-small cell lung cancer, NOS',
 '16467': 'Non-small cell lung cancer, NOS',
 '16469': 'Non-small cell lung cancer, NOS',
 '16498': 'Non-small cell lung cancer, NOS',
 '16554': 'Non-small cell lung cancer, NOS'}
```

get_patients_by_arm(*arm*, *outside=False*)

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*. Also, since the variant call data is not always confirmable by the NCI-MATCH assay, and only those that are confirmed are "evaluable", all outside assay cases are removed from this list as well, unless explicity requested at your own peril.

Parameters

- **arm** (*str*) – One of the official NCI-MATCH arm identifiers.
- **outside** (*bool*) – If set to `True`, will also output outside assay cases in the cohort. DEFAULT: `False`.

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

Return type list

Examples

```
>>> data.get_patients_by_arm(arm='EAY131-E', outside=True)
[
    (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')
]
```

Note: we use default outside assay filter in this case.

```
>>> data.get_patients_by_arm(arm='EAY131-E')
[
    ('10626', 'EAY131-E', 'ON_TREATMENT_ARM'),
    ('11476', 'EAY131-E', 'OFF_TRIAL_DECEASED'),
    ('14256', 'EAY131-E', 'ON_TREATMENT_ARM'),
    ('14343', 'EAY131-E', 'ON_TREATMENT_ARM')
]
```

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.

Note: Each MSN or BSN will have only one set of IHC results typically, since newer results would overwrite the other ones. However, there can be more than one BSN or MSN for a PSN, and so using a PSN for this query is discouraged since you can get multiple results.

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.

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

Return type dict

Examples

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

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

```
>>> data.get_ihc_results(psn=10818)
{'MSN12104': {'MLH1': 'POSITIVE',
              'MSH2': 'POSITIVE',
              'PTEN': 'POSITIVE',
              'RB': 'ND'},
 'MSN51268': {'MLH1': 'POSITIVE',
              'MSH2': 'POSITIVE',
              'PTEN': 'POSITIVE',
              'RB': 'ND'}}
```

get_biopsy_info(*, *bsn*, *term=None*)

Collect some metadata info about a biopsy and return a dict of all data, or just a single term:value entry.

Parameters

- **bsn** (*str*) – BSN for which we want to get data.
- **term** (*str*) – Optional metadata value to filter on.

Returns Dict of results matching the BSN and term (if entered).

Return type dict

Examples

```
>>> self.get_biopsy_info(bsn='T-16-000029')
{'T-16-000029': {
    'biopsy_source': 'Initial',
    'biopsy_status': 'Pass'
    }
}
```

```
>>> self.get_biopsy_info(bsn='T-16-000029', term='biopsy_source')
{'T-16-000029': {'biopsy_source': 'Initial'}}
```

```
>>> self.get_biopsy_info(bsn='T-16-000029', term='foo')
None
ERROR: No such term 'foo' in data structure!
```

get_patient_demographics(*psn*)

Input a PSN, and get meta information about a patient, including gender, ethnicity, any arms they've been on, etc.

Args psn (str): Patient Sequencing Number (PSN) for the query patient

Returns dict: Dictionary of patient metadata

Examples

```
>>> self.get_patient_demographics(psn=13070)
# Add result here
```

4.3 matchbox_api_utils.match_arms module

class matchbox_api_utils.match_arms.**TreatmentArms**(*matchbox='adult'*, *method='mongo'*, *config_file=None*, *username=None*, *password=None*, *json_db='sys_default'*, *load_raw=None*, *make_raw=None*, *quiet=True*)

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

- **matchbox** (*str*) – Name of the MATCHBox system from which we want to get data. This is required now that we have several systems to choose from. Valid names are `adult`, `ped`, and `adult-uat` for those that have access to the adult MATCHBox test system. **DEFUALT:** `adult`.
- **config_file** (*file*) – Custom config file to use if not using system default.
- **username** (*str*) – Username required for access to MATCHBox. Typically this is already present in the config file made during setup, but in cases where needed, it can be explicitly defined here.
- **password** (*str*) – Password associated with the user. As with the above username argument, this is typically indicated in the config file generated during setup.
- **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 may be disabled later.
- **make_raw** (*bool*) – Make a raw API JSON dataset for dev purposes only. This file will be used with the `load_raw` option.
- **quiet** (*bool*) – etc. messages.

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*, *status=None*, *outside=False*)

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.).

- **status** (*str*) – Only output arms that contain this status. Valid statuses are 'OPEN', 'SUSPENDED', 'CLOSED'. If no value input, all arms will be output.
- **outside** (*bool*) – If *True*, only output arms that are open for the Designated (AKA 'Outside') Labs program. Otherwise will list all. DEFAULT: *False*.

Returns list: Arm ID(s) with (i)nclusion or (e)xclusion information, or `None` if the variant is not an aMOI.

Examples

```
>>> # Variant that maps to a study arm (i.e. aMOI).
>>> 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)']
```

```
>>> # Variant that does not map to a study arm.
>>> variant = {
    'type' : 'snvs_indels',
    'gene' : 'TP53',
    'identifier' : 'COSM10660',
    'exon' : '-',
    'function' : 'missense' ,
    'oncominevariantclass' : '-'
}
>>> self.map_amoi(variant)
None
```

```
>>> # Only output arms that are open to outside labs.
>>> variant = {
    'type' : 'snvs_indels',
    'gene' : 'PIK3CA',
    'identifier' : 'COSM775',
    'exon' : '21',
    'function' : 'missense',
    'oncominevariantclass' : 'Hotspot'
}
>>> self.map_amoi(variant, outside=True)
['EAY131-Z1F(i)', 'EAY131-Z1G(e)', 'EAY131-Z1H(e)']
```

```
>>> # Only output arms that are *both* open to outside labs, and open.
>>> variant = {
    'type' : 'snvs_indels',
    'gene' : 'PIK3CA',
    'identifier' : 'COSM775',
    'exon' : '21',
    'function' : 'missense',
    'oncominevariantclass' : 'Hotspot'
}
>>> self.map_amoi(variant, status='OPEN', outside=True)
['EAY131-Z1G(e)', 'EAY131-Z1H(e)']
```

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

Input an Arm ID or a drug name, and return a tuple of arm, drugname, and ID.

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.
- **drugcode** (*str*) – Use the 6-digit drug code to pull results.

Note: Note that using the `drugname` or `drugcode` option may return more than one result as we can have more than one arm per drug.

Returns List of tuples or `None`.

Return type list

Examples

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

```
>>> map_drug_arm(drugname='Afatinib')
[('EAY131-A', 'Afatinib', '750691'),
 ('EAY131-B', 'Afatinib', '750691'),
 ('EAY131-BX1', 'Afatinib', '750691')]
```

```
>>> map_drug_arm(drugcode='750691')
[('EAY131-A', 'Afatinib', '750691'),
 ('EAY131-B', 'Afatinib', '750691'),
 ('EAY131-BX1', 'Afatinib', '750691')]
```

```
>>> map_drug_arm(drugname='Tylenol')
None
```

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

```
>>> self.get_exclusion_disease('EAY131-Z1A')
['Melanoma']
```

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

```
>>> self.get_exclusion_disease('EAY131-A')
['Bronchioloalveolar carcinoma',
 'Lung adenocar. w/ bronch. feat.',
 'Lung adenocarcinoma',
 'Non-small cell lung cancer, NOS',
 'Small Cell Lung Cancer',
 'Squamous cell lung carcinoma']
```

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 All aMOIs indicated for an arm.

Return type dict

get_match_arm_info(*armid=None*)

Get information about NCI-MATCH study arms.

This method will either return a full dictionary of study arms, with drug, target, short description information for the whole study, or filter out the data by arm using the `armid` argument.

Parameters **armid** (*list*) – List of arms that you would like to filter on.

Returns Dict of study arm data.

Return type dict

Example

```
>>> # Need to add example here.
```

get_arm_by_amoi(*gene=None*, *hotspot=None*)

Query study arms by gene or hotspot ID

Input either a HUGO gene name or a hotspot ID as it is represented in the hotspots BED file (e.g. COSM476, MCH12, etc.), and return a list of arms for which that variant is a part, along with the type of variant represented by the identifier. If one were to enter BRAF, then all arms that contain BRAF mutations, along with the categories of *Hotspot* and *Fusion* would be indicated, as BRAF can be activating in either of those categories.

Parameters

- **gene** (*str*) – HUGO genename to use for querying the database.
- **hotspot** (*str*) – NCI-MATCH assay hotspots BED file identifier to use query the database.

Returns List of study arms for which the query variants are aMOIs.

Return type list

Examples

```
>>> # Need to put an example here.
```

Attention: This method is not yet functional and is only a placeholder for now. Intend to code and implement soon!

arm_summary(*arm*)

Output summary information about a particular arm

Input a valid MATCH study arm identifier, in the format 'EAY131-<arm>', and return a dict of summary metrics that can be used for other reports.

Parameters **arm** (*str*) – Valid NCI-MATCH arm name, in the format EAY131-<arm>

Returns Summary of arm details.

Return type dict

Examples

```
>>> print(self.arm_summary('EAY131-A'))
{'arm_id': 'EAY131-A',
 'assigned': 2,
 'drug_name': 'Afatinib',
 'name': 'Afatinib in EGFR activating',
 'status': 'OPEN',
 'outside_open': True,
 'version': '2018-11-19',
 'cnv': 0,
 'fusion': 1,
 'hotspot': 66,
 'non_hs': 2}
```

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

Python Module Index

m

Index

T