OnToma

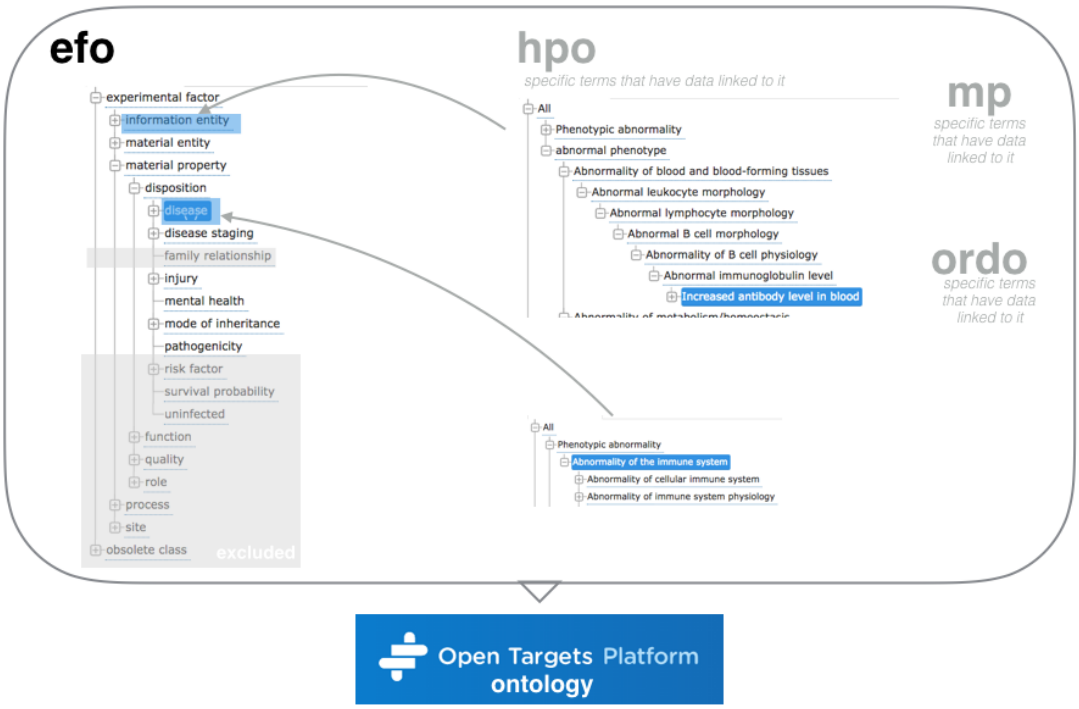
Release 0.0.17

Oct 26, 2020

Contents

OnToma is a python module that helps you map your disease/phenotype terms to the ontology we use in the Open Targets platform.

The ontology we use in the Open Targets platform is a subset (aka. _slim_) of the EFO ontology _plus_ any HPO terms for which a valid EFO mapping could not be found.



This package tries to take the final structure into account and avoids mapping to terms that are not currently in the ontology

Contents:

OnToma is a python module that helps you map your disease/phenotype terms to the ontology we use in the Open Targets platform.

The ontology we use in the Open Targets platform is a subset (aka. *slim*) of the EFO ontology *plus* any HPO terms for which a valid EFO mapping could not be found.

features

- Wrap OLS, OXO, Zooma in a pythonic API
- Always tries to output full URI
- Tries to find mappings iteratively using the faster methods first
- Checks if mapping is in the subset of EFO that gets included in the Open Targets platform
- *tries to* follow the procedure highlighted in https://github.com/opentargets/data_release/blob/master/diagram.jpg

CHAPTER 1

Usage

1.1 Installing

```
pip install ontoma
```

1.2 Quickstart

Looking for a disease or phenotype string is simple:

```
from ontoma import OnToma

otmap = OnToma()
print(otmap.find_term('asthma'))

#outputs:
'http://www.ebi.ac.uk/efo/EFO_0000270'
```

you can obtain more details using the `verbose` flag:

```
print(otmap.find_term('asthma',verbose=True))

#outputs something similar to:
{'term':'http://www.ebi.ac.uk/efo/EFO_0000270','label':'asthma',
'source':'EFO OBO','quality':'match' ...}
```

From the command line

The command line version can be invoked with `ontoma` (type `ontoma --help` to find out about the usage):

```
ontoma <input_file> <output_file>
```

where input file can be replaced with − to read from stdin and write to stdout.

Which means that to read from a previous command, using pipes:

```
echo 'asthma' | ontoma - <output_file>
```

will output a file `test.txt` containing the result, where it came from and the degree of confidence of the match (one of {match, fuzzy, check}):

```
http://www.ebi.ac.uk/efo/EFO_0000270    asthma  EFO OBO     match
```

Piping also works for the output. If you want to find the string "mymatch" from the results, you can:

```
ontoma <input_file> - | grep "mymatch"
```

More detailed documentation is at Documentation Status http://ontoma.readthedocs.io/en/stable/

CHAPTER 2

Developing

2.1 set up your environment

First clone this repo

```
git clone https://github.com/opentargets/OnToma.git
```

Install pipenv and then run

```
pipenv install --dev
```

to get all development dependencies installed.

Test everything is working:

```
pipenv run pytest
```

if you don't like pipenv you can stick with the more traditional setuptools/virtualenv setup:

```
git clone https://github.com/opentargets/OnToma.git
virtualenv -p python3 venv
source venv/bin/activate
pip install --editable .
```

2.2 How to add a dependency

Add to both pipenv AND setup.py

To add a dep for a library, add it by hand to `setup.py`, then add it separately to `Pipfile`, so that it shows up both as a transitive dependency and in your locked dev environment

2.3 Release to PyPi

Simply run `./bumpversion.sh`

The script will tag, push and trigger a new CI run. The package will be automatically uploaded to pypi.

CHAPTER 3

Ontoma Interface

main interface class

class ontoma.interface.**OnToma**(*exclude=[]*)

Open Targets ontology mapping wrapper

Parameters **exclude** (*str or [str]*) – Excludes 'zooma','ols_hp' or 'ols_ordo' API calls from the search, to speed things up.

Example

Initialize the class (which will download EFO,OBO and others):

```
>>> t=OnToma()
```

We can now lookup "asthma" and get:

```
>>> t.efo_lookup('asthma')
'http://www.ebi.ac.uk/efo/EFO_0000270'
```

Notice we always tend to return a full IRI

Search by synonyms coming from the OBO file is also supported

```
>>> t.efo_lookup('Asthma unspecified')
'http://www.ebi.ac.uk/efo/EFO_0000270'
```

Reverse lookups uses the get_efo_label() method

```
>>> t.get_efo_label('EFO_0000270')
'asthma'
>>> t.get_efo_label('EFO:0000270')
'asthma'
>>> t.get_efo_label('http://www.ebi.ac.uk/efo/EFO_0000270')
'asthma'
```

Similarly, we can now lookup "Phenotypic abnormality" on HP OBO:

```
>>> t.hp_lookup('Phenotypic abnormality')
'http://purl.obolibrary.org/obo/HP_0000118'
>>> t.hp_lookup('Narrow nasal tip')
'http://purl.obolibrary.org/obo/HP_0011832'
```

OMIM code lookup

```
>>> t.omim_lookup('230650')
'http://www.orpha.net/ORDO/Orphanet_79257'
```

```
>>> t.zooma_lookup('asthma')
'http://www.ebi.ac.uk/efo/EFO_0000270'
```

MONDO lookup >>> t.mondo_lookup('asthma') 'http://purl.obolibrary.org/obo/MONDO_0004979'

There is also a semi-intelligent wrapper, which tries to guess the best matching strategy:

```
>>> t.find_term('asthma')
'http://www.ebi.ac.uk/efo/EFO_0000270'
>>> t.find_term('615877',code='OMIM')
'http://www.orpha.net/ORDO/Orphanet_202948'
```

It returns *None* if it cannot find an EFO id:

```
>>> t.find_term('notadisease') is None
True
```

efo_lookup(*name*)
Searches the EFO OBO file for a direct match

efo_to_name

exclude = None
Initialize API clients

find_term(*query*, *code=None*, *suggest=False*, *verbose=False*)
Finds the most likely EFO code for a given string or ontology code.

If the code argument is passed, it will attempt to perform an exact match amongst the mappings available.

If only a string is passed, it will attempt to match it against mappings, but will try using the EBI SPOT APIs if no match is found, until a likely code is identified.

Operations roughly ordered from least expensive to most expensive and also from most authorative to least authorative

1. EFO OBO lookup
2. Zooma mappings lookup
3. Zooma API high confidence lookup

4. OLS API EFO lookup - exact match — below this line we might not have a term in the platform — 5. HP OBO lookup 6. OLS API HP lookup - exact match 7. OLS API ORDO lookup - exact match 8. OLS API EFO lookup - not exact 9. OLS API HP+ORDO lookup - not exact

Parameters

- **query** (*str*) – the disease/phenotype to be matched to an EFO code

- **code** – accepts one of "ICD9CM", "OMIM" **TODO** expand to more ontologies If a code is passed, it will attempt to find the code in one of our curated mapping datasources. Defaults to None.
- **suggest** (*boolean*) – if True the OLS API will be queried for any match in HP, ORDO and EFO, whether or not these terms are already included in the Open Targets platform ontology.
- **verbose** (*bool*) – if True returns a dictionary containing {term, label, source, quality, action}

Returns A valid OT ontology URI. *None* if no EFO code was found

get_efo_from_xref(*efocode*)
Given an short disease id, returns the id and label of equivalent term(s) in EFO as defined by xrefs

get_efo_label(*efocode*)
Given an EFO short form code, returns the label as taken from the OBO

get_hp_label(*hpcode*)
Given an HP short form code, returns the label as taken from the OBO

get_mondo_label(*mondocode*)
Given an MONDO short form code, returns the label as taken from the OBO

hp_lookup(*name*)
Searches the HP OBO file for a direct match

hp_to_name

icd9_lookup(*icd9code*)
Searches the ICD9CM <=> EFO mappings returned from the OXO API #FIXME Results don't seem to be deterministic, some mappings appear and disappear between calls, e.g. t.icd9_lookup('696')

mondo_lookup(*name*)
Searches the mondo OBO file for a direct match

mondo_to_name

name_to_efo

name_to_hp

name_to_mondo

omim_lookup(*omimcode*)
Searches our own curated OMIM <=> EFO mappings #FIXME assumes the first is the best hit. is this ok?

otzooma_map_lookup(*name*)
Searches against the curated OpenTargets mapping we submitted to zooma.

These mappings are usually stored on github. NOTE: this is not a lookup to zooma API

oxo_lookup(*other_ontology_id*, *input_source='ICD9CM'*)
Searches in the mappings returned from the EBI OXO API.

The function should return an EFO code for any given xref, if one exists.

Parameters

- **other_ontology_id** – the code that should be mapped to EFO
- **input_source** – an ontology code. Defaults to 'ICD9CM'. Available ontologies are listed at https://www.ebi.ac.uk/spot/oxo/api/datasources?fields=preferredPrefix

Returns the EFO code

Return type str

xref_to_efo

zooma_lookup(*name*)
Searches against the EBI Zooma service for an high confidence mapping

ontoma.interface.**make_uri**(*ontology_short_form*)
Transform a short form ontology code in a full URI. Currently works for EFO, HPO, ORDO and MP.

Parameters **ontology_short_form** – An ontology code in the short format, like 'EFO:0000270'. The function won't break if a valid URI is passed.

Returns A full URI. Raises a *ValueError* if the short form code was not one of the supported ones.

Example

```
>>> make_uri('EFO:0000270')
'http://www.ebi.ac.uk/efo/EFO_0000270'
```

```
>>> make_uri('HP_0000270')
'http://purl.obolibrary.org/obo/HP_0000270'
```

```
>>> make_uri('http://purl.obolibrary.org/obo/HP_0000270')
'http://purl.obolibrary.org/obo/HP_0000270'
```

```
>>> make_uri('TEST_0000270')
Traceback (most recent call last):
    ...
ValueError: Could not build an URI. Short form: TEST_0000270 not recognized
```

CHAPTER 4

OLS wrapper

OLS API wrapper

Original code borrowed from https://github.com/cthoyt/ols-client/blob/master/src/ols_client/client.py

- Removed ontology and term methods.
- Added details/parameters for all search methods

TODO: check input parameters are valid TODO: handle requests.exceptions.ConnectionError when traffic is too high and API goes down

class ontoma.ols.**OlsClient**(*ols_base=None*, *ontology=None*, *field_list=None*, *query_fields=None*)

Wraps the functions to query the Ontology Lookup Service.

```
>>> ols = OlsClient()
>>> ols.search('asthma')[0]['iri']
'http://purl.obolibrary.org/obo/NCIT_C28397'
```

You can search in other ontologies and pass all other parameters accepted by OLS

```
>>> ols.search('lung',ontology='uberon')[0]['iri']
'http://purl.obolibrary.org/obo/UBERON_0002048'
```

besthit() simply returns the first element:

```
>>> ols.besthit('lung',ontology='uberon')['iri']
'http://purl.obolibrary.org/obo/UBERON_0002048'
```

exact=True forces an exact match:

```
>>> ols.besthit('hypogammaglobulinemia',ontology='efo')['label']
'Osteopetrosis - hypogammaglobulinemia'
```

```
>>> ols.besthit('hypogammaglobulinemia',ontology='efo',exact=True)['label']
'Agammaglobulinemia'
```

```
>>> r = ols.search('asthma',ontology=['efo'],query_fields=['synonym'],field_list=[
↪'iri','label'])
>>> 'http://www.ebi.ac.uk/efo/EFO_0004591' in [syn['iri'] for syn in r]
True
```

Find the label of its first ancestor:

```
>>> a = ols.get_ancestors('efo',r[0]['iri'])
>>> a[0]['label']
'asthma'
```

```
>>> r = ols.suggest('asthma', ontology=['efo','ordo','hpo'])
>>> r[0]['autosuggest']
'asthma'
```

```
>>> r= ols.select('asthma', ontology=['efo','ordo','hpo'],field_list=['iri'])
>>> r[0]['iri']
'http://www.ebi.ac.uk/efo/EFO_0000270'
```

```
>>> ols.search('Myofascial Pain Syndrome',ontology=['efo'])[0]['short_form']
'EFO_1001054'
```

```
>>> [x['short_form'] for x in ols.select('alzheimer')[:2]]
['PW_0000015', 'DOID_0080348']
```

You can also pass your favourite parameters at class instantiation:

```
>>> ot_ols = OlsClient(ontology=['efo'],field_list=['short_form'])
>>> ot_ols.search('asthma')[0]['short_form']
'EFO_0000270'
>>> ot_ols.besthit('asthma')['short_form']
'EFO_0000270'
```

besthit (*name*, ***kwargs*)
select first element of the /search API response

get_ancestors (*ont*, *iri*)
Gets the data for a given term

Parameters

- **ontology** – The name of the ontology
- **iri** – The IRI of a term

get_term (*ontology*, *iri*)
Gets the data for a given term

Parameters

- **ontology** – The name of the ontology
- **iri** – The IRI of a term

search (*name*, *query_fields=None*, *ontology=None*, *field_list=None*, *exact=None*, *bytype='class'*)
Searches the OLS with the given term

Parameters

- **query_fields** – By default the search is performed over term labels, synonyms, descriptions, identifiers and annotation properties. This option allows to specify the fields to query, the defaults are *{label, synonym, description, short_form, obo_id, annotations, logical_description, iri}*
- **exact** – Forces exact match if not *None*
- **bytype** – restrict to terms one of {class,property,individual,ontology}

select(*name*, *ontology=None*, *field_list=None*)

Select terms, Tuned specifically to support applications such as autocomplete.

See also:

https://www.ebi.ac.uk/ols/docs/api#_select

suggest(*name*, *ontology=None*)

Suggest terms from an optional list of ontologies

See also:

https://www.ebi.ac.uk/ols/docs/api#_suggest_term

CHAPTER 5

Zooma wrapper

ZOOMA api wrapper

class ontoma.zooma.**ZoomaClient**(*zooma_base=None*, *required=None*, *preferred=None*, *ontologies='none'*)

Simple client to query zooma

By default (specifying nothing), Zooma will search its available databases containing curated mappings (and that do not include ontology sources), and if nothing is found it will look in the Ontology Lookup Service (OLS) to predict ontology annotations.

Example

```
>>> z = ZoomaClient()
>>> r = z.annotate("mus musculus")
>>> r[0]['semanticTags']
['http://purl.obolibrary.org/obo/NCBITaxon_10090']
```

```
>>> r[0]['confidence']
'HIGH'
```

```
>>> z.besthit("mus musculus")['iri']
'http://purl.obolibrary.org/obo/NCBITaxon_10090'
```

annotate(*name*, *property_type=None*, *required=None*, *preferred=None*, *ontologies='none'*)

besthit(*name*)

highconfhits(*name*)

CHAPTER 6

OXO wrapper

wrapper for the OXO api (reverse engineered! argh!)

class ontoma.oxo.**OxoClient**(*base_url='https://www.ebi.ac.uk/spot/oxo/api'*)

OXO wrapper class FIXME Results don't seem to be deterministic, some mappings appear and disappear between calls, e.g. icd9s = oxo.make_mappings(input_source="ICD9CM", distance=2); icd9s['733.09']

get_data_sources()

make_mappings(*input_source='ICD9CM'*, ***kwargs*)

search(*ids=None*, *input_source=None*, *mapping_target='EFO'*, *distance=1*, *size=500*)

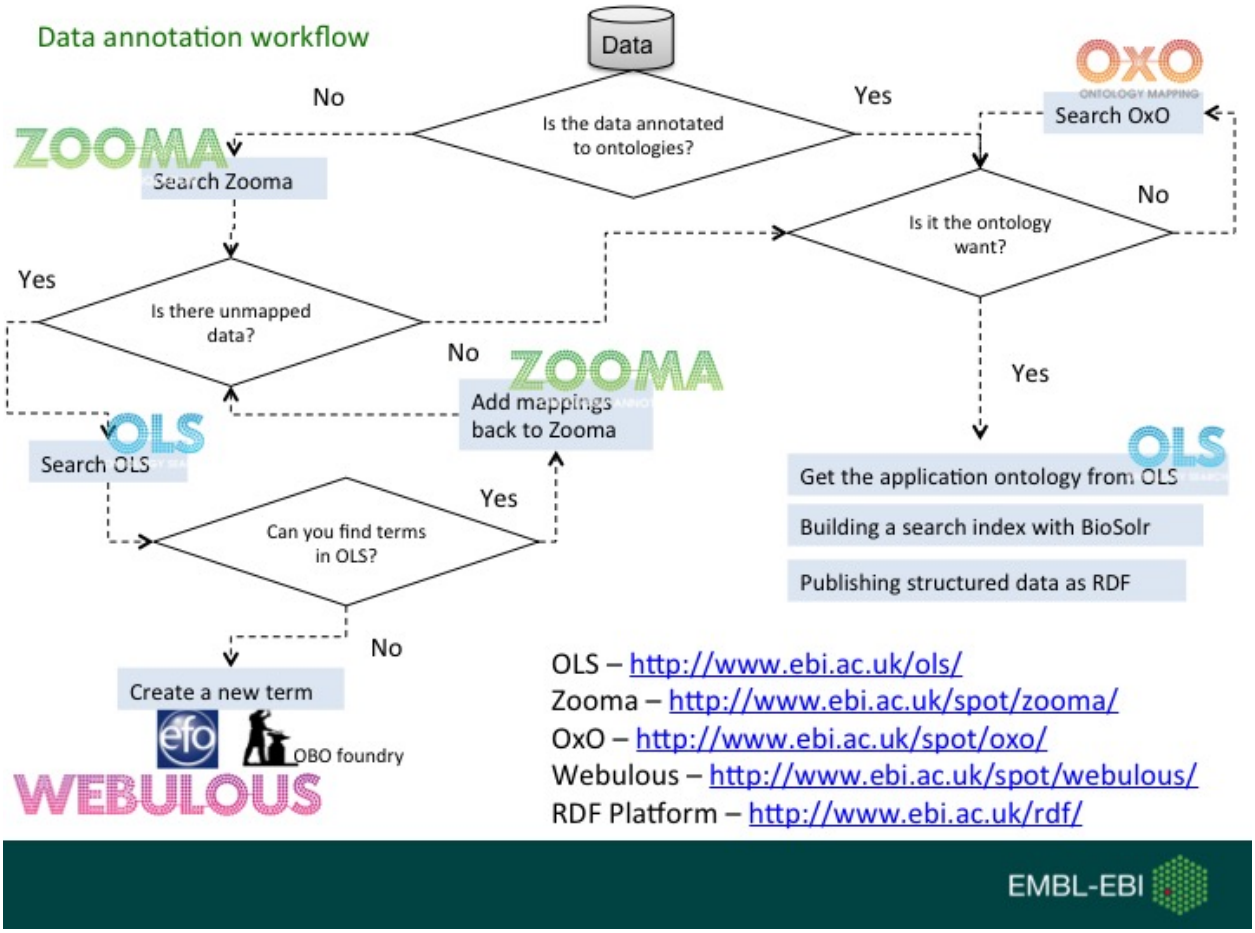
iterates over the mappings, each being a dict with the following keys: ["_links", "curie", "label", "mappingResponseList", "queryId", "querySource"]

CHAPTER 7

Features

- Wrap OLS, OXO, Zooma in a pythonic API
- Tries to find mappings iteratively using the faster methods first
- Checks if mapping is in the subset of EFO that gets included in the

Open Targets platform - *tries to* follow the procedure highlighted in https://github.com/opentargets/data_release/wiki/EFO-Ontology-Annotation-Process


Data annotation workflow
Data
Is the data annotated to ontologies?
No
Yes
ZOOMA
Search Zooma
OxO
ONTOLOGY MAPPING
Search OxO
Is it the ontology want?
No
Yes
Is there unmapped data?
Yes
No
ZOOMA
Add mappings back to Zooma
OLS
Search OLS
Get the application ontology from OLS
Building a search index with BioSolr
Publishing structured data as RDF
Can you find terms in OLS?
Yes
No
Create a new term
efo
OBO foundry
WEBULOUS
OLS – http://www.ebi.ac.uk/ols/
Zooma – http://www.ebi.ac.uk/spot/zooma/
OxO – http://www.ebi.ac.uk/spot/oxo/
Webulous – http://www.ebi.ac.uk/spot/webulous/
RDF Platform – http://www.ebi.ac.uk/rdf/
EMBL-EBI

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

o

Index

Z