OnToma

Aug 04, 2021

Contents

CHAPTER 1

Introduction

OnToma is a Python module which maps the disease or phenotype terms to EFO, the ontology used in the Open Targets platform.

Note: More precisely, only a subset of EFO is used in the Open Targets platform, called *EFO slim*. It is released alongside with every new version of EFO.

OnToma supports two kinds of inputs: either identifiers from other ontologies (e.g. `OMIM:102900`), or strings (e.g. `pyruvate kinase hyperactivity`).

The way OnToma operates is by trying a series of lookups in EFO and also querying tools from the EBI ontology stack such as OxO and ZOOMA.

For each input, it will return anywhere from zero to multiple matches in EFO. This version of OnToma only returns mappings which are considered of high quality.

For each input type, you can run OnToma from command line or directly from other Python code.

CHAPTER 2

Installing OnToma

```
pip install --upgrade ontoma
```

CHAPTER 3

Running OnToma from CLI

In this mode, the input is a file or STDIN, with one input per line:

```
echo -e 'asthma\npyruvate kinase hyperactivity' | ontoma --input-type string
echo -e 'OMIM:102900\nOMIM:104310' | ontoma --input-type ontology
```

The output format is a TSV file, containing one query-to-EFO mapping per line. The columns to include are configurable via the `--columns` flag and can include the following:

- `query`: The original query, e.g. *asthma*
- `id_normalised`: The EFO term ID in the internal normalised CURIE representation as used by OnToma, e.g. *EFO:0000270.*
- `id_ot_schema`: The EFO term ID as supported by the Open Targets JSON schema, e.g. *EFO_0000270.*
- `id_full_uri`: The full EFO term URI, e.g. **http://www.ebi.ac.uk/efo/EFO_0000270.**
- `label`: The normalised (lower case) label as specified in EFO, e.g. *asthma*.

By default, two columns are included: `query` and `id_ot_schema`.

In case no results were found, the query will be missing from the output. In case multiple results were found, the query will appear multiple times in the output.

You can read about additional flags by running `ontoma --help`.

CHAPTER 4

Running OnToma from Python code

```
from ontoma import OnToma
otmap = OnToma()
result_code = otmap.find_term('OMIM:102900', code=True)
result_string = otmap.find_term('asthma')
# Columns are available as attributes.
print(result.id_ot_schema)  # Prints 'EFO_0000270'.
```

CHAPTER 5

Speeding up subsequent OnToma runs

When you initialise an OnToma client, it needs to download and parse the latest EFO OT slim release. Depending on your internet connection, it may take anywhere from 10 seconds to a few minutes.

To speed up subsequent OnToma runs, you can specify a cache directory to avoid doing this every time. This can be supplied by a `--cache-dir` option in the CLI or by a `cache_dir` parameter in the Python interface.

CHAPTER 6

Contents

6.1 Development

6.1.1 Set up environment

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

6.1.2 Install development packages

```
python3 -m pip install --upgrade \
  build pip twine \
  pytest \
  sphinx sphinx-rtd-theme
```

6.1.3 Adding a dependency

Installation dependencies are stored in the `setup.py` file, in the `install_requires` section.

6.1.4 Releasing a new version

1. Modify the version in the `VERSION` file.
2. Add a tag: `git tag $(cat VERSION) && git push origin --tags`.
3. Create a release on GitHub.
4. Generate distribution archives: `python3 -m build`.

5. Upload the release: `python3 -m twine upload dist/*`. Use the usual login and password for PyPi.

6.1.5 Performing a comparison benchmark

This can be used in case of major updates to OnToma algorithm to make sure they don't break things significantly.

1. Call `benchmark/sample.sh` to create a random sample of input strings. This will fetch 100 random records each from ClinVar, PhenoDigm, and gene2phenotype sources. Together they are thought to cover a wide range of use case well. Due to deduplication, the final file `sample.txt` may contain slightly less than 300 records.
2. Optionally, to add some additional samples from PanelApp JSON file, you can do: `jq 'keys[]' diseaseToEfo_results.json | tr -d '"' | shuf -n 100 >> sample.txt; sort -uf -o sample.txt sample.txt`.
3. **Process the file so that the final output format is a two column TSV file with the following columns: original query; full UI**

    - For OnToma pre-v1.0.0, use: `time ontoma sample.txt - | awk -F$'\t' '{if (length($2) != 0) {print $1 "\t" $2}}' | tail -n+2 > ontoma_old.txt`.
    - For OnToma v1.0.0+, use: `time ontoma --cache-dir EFO_CACHE --columns query,id_full_uri <sample.txt | tail -n+2 >ontoma_new.txt`.
4. Collate the results by using `benchmark/collate.py --old ontoma_old.txt --new ontoma_new.txt > benchmark_comparison.txt`. Load the resulting file into Google Sheets and manually mark the mappings as good/bad/uncertain. Exact, letter to letter matches are marked as good automatically.

6.2 OnToma interface

Main interface class.

class ontoma.interface.**OnToma**(*cache_dir=None*, *efo_release='latest'*)

Open Targets ontology mapping wrapper. Please refer to documentation for usage details.

filter_identifiers_by_efo_current(*normalised_identifiers*)

Returns a subset of the idenfitiers which are in EFO and not marked as obsolete.

find_term(*query: str*, *code: bool = False*, *suggest: bool = False*) → list

For a given query (an ontology identifier or a string), find matches in EFO Open Targets slim.

The algorithm operates in a series of steps. If a given step is successful, the result is returned immediately, and the remaining steps are not executed. If all steps fail to provide a match, None is returned. Note that in general more than one mapping can be returned. This can happen for complex traits which require more than one ontology term to represent them.

If the *code* flag is specified, it is assumed that the query is an ontology identifier, such as 'OMIM:615632', and the following steps are attempted: 1. See if the term is already in EFO OT slim OWL file. 2. Match terms by cross-references (hasDbXref) from the OWL file. 3. Mapping from the manual cross-reference database. 4. Request through OxO with a distance of 2.

If the query is a string, the following steps are attempted: 5. Exact name match from EFO OT slim OWL file. 6. Exact synonym (hasExactSynonym) from the OWL file. 7. Mapping from the manual string-to-ontology database. 8. High confidence mapping from ZOOMA with default parameters.

The following functionality is planned, but not yet implemented. — If the query is a string, and additionally the *suggest* flag is specified, additional steps are attempted: 9. Inexact synonyms (hasRelatedSynonym) from the OWL file. 10. Any confidence mapping from ZOOMA with default parameters.

Args: query: Either an ontology identifier, or the disease/phenotype string to be matched to an EFO code. code: Whether to treat the query as an ontology identifier. suggest: Whether to report low quality mappings which are not guaranteed to be contained in EFO OT slim.

Returns: A list of values dependent on the *verbose* flag (either strings with ontology identifiers, or a dictionary of additional information). The list will be empty if no hits were identified.

get_label_from_efo(*normalised_identifier*)

step01_owl_identifier_match(*normalised_identifier*)
If the term is already present in EFO, return it as is.

step02_owl_db_xref(*normalised_identifier*)
If there are terms in EFO referenced by the *hasDbXref* field to the query, return them.

step03_manual_xref(*normalised_identifier*)
Look for the queried term in the manual ontology-to-ontology mapping list.

step04_oxo_query(*normalised_identifier*)
Find cross-references using OxO.

step05_owl_name_match(*normalised_string*)
Find EFO terms which match the string query exactly.

step06_owl_exact_synonym(*normalised_string*)
Find EFO terms which have the query as an exact synonym.

step07_manual_mapping(*normalised_string*)
Find the query in the manual string-to-ontology mapping database.

step08_zooma_high_confidence(*normalised_string*)

step09_owl_related_synonym(*normalised_string*)

step10_zooma_any(*normalised_string*)

6.3 ZOOMA wrapper

ZOOMA API wrapper.

class ontoma.zooma.**ZoomaClient**(*zooma_base='https://www.ebi.ac.uk/spot/zooma/v2/api'*)
Simple client to query ZOOMA. Will look in all curated datasources and perform a fuzzy search in EFO. Only HIGH quality mappings are considered.

search(*query_string*)
Query ZOOMA and return all high confidence mappings.

6.4 OXO wrapper

Wrapper for the OxO API.

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

search(*ids=None*, *mapping_target='EFO'*, *distance=1*)
Query OxO with given parameters and yield the found mappings one by one.

Python Module Index

o

Index