prunito Documentation

kp14

Feb 15, 2020

Contents:

A package providing tools for accessing and working with protein sequences and associated data. The focus is on data from the UniProtKB. This is reflected in the package name which is an anagram of UniProt. UniProt has two sets of REST services, uniprot.org and the newer Proteins API. Additionally, a few tools for accessing services or working with data from the following resources are provided:

- EuropePMC
- InterPro
- ENA
- EBI web services

CHAPTER 1

Installing prunito

Prunito is packaged as a wheel and can be installed using pip. This will also install all the (mandatory) dependencies. Python >=3.6 is required. As usual, it probably best to install prunito and its dependencies into a dedicated virtual environment.

1. Create a virtual environment

   Using `conda` from the Anaconda Python distribution :

   - Create a new environment (e.g. called *myenv*) which runs Python 3.6 and has `pip` installed:

     ```
     conda create -n myenv python=3.6 pip
     ```

   - Activate the env:

     ```
     conda activate myenv
     ```

   Using *venv* in a regular Python installation. Python is usually available out of the box on Linux:

   - Ensure that the version of Python used is 3.6 or higher:

     ```
     python --version
     ```

   - Create a new environment (e.g. called *myenv*). `ensurepip` will bootstrap pip into the env:

     ```
     pyvenv /path/to/new/virtual/environment/myenv
     ```

2. Install prunito with its dependencies:

   ```
   pip install prunito-<version-py3-none-any>.whl
   ```

CHAPTER 2

Dependencies

All of the following packages have to be installed some of which come with their own dependencies but those should be taken care of by running the usual pip command:

- requests
- lxml
- pandas (optional)
- venndy (optional)

CHAPTER 3

Quick start

First, import the relevant package:

```
from prunito import uniprot as up
```

A simple search for all reviewed UniProtKB entries with *tax* in their names.

```
result = up.search_reviewed('name:tax')
```

Check how many hits this search retrieved. As results are sequence-like, they support `__len__`.

```
result.size()
# or
#len(result)
```

What would happen if a given query had many hits? By default, the maximum number of hits retrieved is 2000. This can be changed using the parameter `limit`.

Let's re-run the previous search but this time not just for reviewed entries but all of UniProtKB:

```
huge = up.search('name:tax', limit=1000)
```

```
Partial dataset retrieved. Size: 1890. Retrieved: 1000.
Consider increasing the limit and/or using offset.
```

If the set limit is lower than the actual number of search hits, the above hint is printed. One could then set `limit=2000`.

Back to the initial result. What does `size` mean? UniProtKB entries. And these entries we would like to parse. As prunito provides functionality for both searching and parsing UniProt, one can directly iterate over the entries in a search result for convenience:

```
entries = list(result)
# or
# for entry in result: ...
```

Iterate over the entries, printing out primary accessions and recommended full names. Both fields are provided for convenience.

```
for entry in entries:
    print(entry.primary_accession, entry.recommended_full_name)
```

```
Q06507 Cyclic AMP-dependent transcription factor ATF-4
P18848 Cyclic AMP-dependent transcription factor ATF-4
Q9Y6D9 Mitotic spindle assembly checkpoint protein MAD1
P47911 60S ribosomal protein L6
P10070 Zinc finger protein GLI2
Q0VGT2 Zinc finger protein GLI2
...
```

Which methods and fields are available on a Record object? Basically, all the ones Biopython's REcord objects provide plus as few more for convenience. See `prunito.uniprot.parsers.parser_knowledgebase_txt.Record`.

Get isoforms for those entries that have them. We use the presence of a keyword, *Alternative splicing*, as a filter here.

```
for e in entries:
    if 'Alternative splicing' in e.keywords:
        for i in e.isoforms():
            print(i)
```

```
>sp|Q9Y6D9-2|MD1L1_HUMAN Isoform 2 of Mitotic spindle assembly checkpoint protein␣
↪MAD1 OS=Homo sapiens (Human). OX=['9606']
MLPARGCVRKRTVWPRLARVLIVTLLTLELSYAPLPCQLSGVPYNTGDPVGRWARPCIWP
CPWHTTINALKGRISELQWSVMDQEMRVKRLESEKQELQEQLDLQHKKCQEANQKIQELQ
...
>sp|P10070-1|GLI2_HUMAN Isoform 1 of Zinc finger protein GLI2 OS=Homo sapiens (Human).
↪ OX=['9606']
MALTSINATPTQLSSSSNCLSDTNQNKQSSESAVSSTVNPVAIHKRSKVKTEPEGLRPAS
PLALTQGQVSGHGSCGCALPLSQEQLADLKEDLDRDDCKQEAEVVIYETNCHWEDCTKEY
...
>sp|P10070-2|GLI2_HUMAN Isoform 2 of Zinc finger protein GLI2 OS=Homo sapiens (Human).
↪ OX=['9606']
MALTSINATPTQLSSSSNCLSDTNQNKQSSESAVSSTVNPVAIHKRSKVKTEPEGLRPAS
PLALTQEQLADLKEDLDRDDCKQEAEVVIYETNCHWEDCTKEYDTQEQLVHHINNEHIHG
...
```

We would like to run a FASTA similarity search against Swiss-Prot for one of the sequences. Let's take the canonical sequence of the first entry in *entries*.

Here we use the `ebiwebservices` module from `prunito`. The EBI web services require an email address to be set.

```
from prunito import ebiwebservices as ews

ews.set_email('some@gmx.de')

first_entry = entries[0]
similar = ews.fasta_search(first_entry.as_fasta())

print(similar.text[:600])
```

```
# /nfs/public/release/wp-jdispatcher/latest/appbin/linux-x86_64/fasta-36.3.7b/fasta36
↪-l /nfs/public/ro/es/data/idata/latest/fastacfg/fasta3db -L -T 8 -p -m "F9 fasta-
↪R20180501-155642-0060-16766253-p1m.m9" @:1- +uniprotkb_swissprot+
FASTA searches a protein or DNA sequence data bank
 version 36.3.7b Jun, 2015(preload9)
Please cite:
 W.R. Pearson & D.J. Lipman PNAS (1988) 85:2444-2448

Query: @
  1>>>sp|Q06507|ATF4_MOUSE Cyclic AMP-dependent transcription factor ATF-4 OS=Mus
↪musculus (Mouse). OX=['10090'] - 349 aa
Library: UniProtKB/Swiss-Prot
  199856860 residues in 557275 sequences

Statistic...
```

How about using InterPro's HMMER search instead of FASTA?

```
from prunito import interpro as ip

ip_similar = ip.search_phmmer(first_entry.as_fasta())
print(ip_similar.summary())
```

```
acc2        acc     desc    species kg      evalue
Q06507      ATF4_MOUSE      Cyclic AMP-dependent transcription factor ATF-4 Mus
↪musculus    Eukaryota       1.0e-232
Q9ES19      ATF4_RAT        Cyclic AMP-dependent transcription factor ATF-4 Rattus
↪norvegicus       Eukaryota       2.6e-216
P18848      ATF4_HUMAN      Cyclic AMP-dependent transcription factor ATF-4 Homo
↪sapiens    Eukaryota       2.4e-195
Q3ZCH6      ATF4_BOVIN      Cyclic AMP-dependent transcription factor ATF-4 Bos
↪taurus      Eukaryota       1.9e-169
Q6NW59      ATF4_DANRE      Cyclic AMP-dependent transcription factor ATF-4 Danio
↪rerio      Eukaryota       5.0e-34
Q9Y2D1      ATF5_HUMAN      Cyclic AMP-dependent transcription factor ATF-5 Homo
↪sapiens    Eukaryota       6.3e-20
Q6P788      ATF5_RAT        Cyclic AMP-dependent transcription factor ATF-5 Rattus
↪norvegicus       Eukaryota       5.8e-18
Q9GPH3      ATFC_BOMMO      Activating transcription factor of chaperone    Bombyx
↪mori     Eukaryota       2.4e-16
O70191      ATF5_MOUSE      Cyclic AMP-dependent transcription factor ATF-5 Mus
↪musculus    Eukaryota       3.5e-13
Q8TFF3      HAC1_HYPJE      Transcriptional activator hac1  Hypocrea jecorina (strain
↪QM6a) Eukaryota       5.4e-05
```

The result summary is also available as a dataframe if `pandas` is.

```
df_hmmer = ip_similar.as_dataframe()
```

Do some of the entries contain the same PubMed IDs? Let's find the 5 most common ones.

```
from collections import Counter

c = Counter()
for e in entries:
    c.update(e.all_pubmed_ids)
print(c.most_common(5))
```

```
[('15489334', 24), ('20068231', 9), ('14702039', 8), ('23186163', 8), ('21269460', 7)]
```

Which are the accession numbers and species of those 24 entries containing the most common one (15489334)?

```
for e in entries:
    if '15489334' in e.all_pubmed_ids:
        print(e.primary_accession, e.organism)
```

```
Q06507 Mus musculus (Mouse).
P18848 Homo sapiens (Human).
Q9Y6D9 Homo sapiens (Human).
P47911 Mus musculus (Mouse).
Q0VGT2 Mus musculus (Mouse).
...
```

So, which paper is hiding behind this PMID 15489334? Here we use another module for accessing *EuropePMC <https://europepmc.org>* from `prunito`. EuropePMC returns data for example in JSON format. We can iterate over the results.

```
from prunito import europepmc as epmc

paper = epmc.get_pmid_metadata('15489334')
for p in paper:
    print(p['title'])
    print(p['abstractText'])
```

```
The status, quality, and expansion of the NIH full-length cDNA project: the Mammalian␣
↪Gene Collection (MGC).

"The National Institutes of Health's Mammalian Gene Collection (MGC) project was␣
↪designed to generate and
sequence a publicly accessible cDNA resource containing a complete open reading frame␣
↪(ORF) for every human
and mouse gene. The project initially used a random strategy to select clones from a␣
↪large number of cDNA
libraries from diverse tissues. Candidate clones were chosen based on 5'-EST␣
↪sequences, and then fully sequenced
to high accuracy and analyzed by algorithms developed for this project. Currently,␣
↪more than 11,000 human and
10,000 mouse genes are represented in MGC by at least one clone with a full ORF. The␣
↪random selection approach
is now reaching a saturation point, and a transition to protocols targeted at the␣
↪missing transcripts is now
required to complete the mouse and human collections. Comparison of the sequence of␣
↪the MGC clones to reference
genome sequences reveals that most cDNA clones are of very high sequence quality,␣
↪although it is likely that some
cDNAs may carry missense variants as a consequence of experimental artifact, such as␣
↪PCR, cloning, or reverse
transcriptase errors. Recently, a rat cDNA component was added to the project, and␣
↪ongoing frog (Xenopus) and
zebrafish (Danio) cDNA projects were expanded to take advantage of the high-
↪throughput MGC pipeline."
```

The paper mentions the Mammalian Gene Collection. Why not search EuropePMC for articles mentioning the collection in their abstracts?

```
mgc_papers = epmc.search('abstract:"Mammalian Gene Collection"')
mgc_papers.size()
#
# len(mgc_papers)
for idx, hit in enumerate(mgc_papers):
    print(idx, hit['title'])
```

```
0 Identification of candidate transcription factor binding sites in the cattle genome.
1 Selenoproteins in bladder cancer.
2 NSrp70 is a novel nuclear speckle-related protein that modulates alternative pre-
↪mRNA splicing in vivo.
3 Generation of a genome scale lentiviral vector library for EF1Î± promoter-driven␣
↪expression of human ORFs ...
4 The completion of the Mammalian Gene Collection (MGC).
5 A high-throughput platform for lentiviral overexpression screening of the human␣
↪ORFeome.
6 PRFdb: a database of computationally predicted eukaryotic programmed -1 ribosomal␣
↪frameshift signals.
7 Transcriptome analysis of a cDNA library from adult human epididymis.
...
```

Each hit/paper has many extra data fields including DOI, PubMed ID etc. If the abstract is needed, `resulttype='core'` has to be specified as a search parameter.

```
for k, v in list(mgc_papers)[3].items():
    print(k + ':\t' + str(v))
```

```
id: 23251614
source:     MED
pmid:       23251614
pmcid:      PMC3520899
doi:        10.1371/journal.pone.0051733
title:      Generation of a genome scale lentiviral vector library for EF1Î± promoter-
↪driven expression of human ORFs and identification of human genes affecting viral␣
↪titer.
authorString:       Å kalamera D, Dahmer M, Purdon AS, Wilson BM, Ranall MV,␣
↪Blumenthal A, Gabrielli B, Gonda TJ.
journalTitle:       PLoS One
issue:      12
journalVolume:      7
pubYear:    2012
journalIssn:        1932-6203
pageInfo:   e51733
pubType:    research support, non-u.s. gov't; research-article; journal article;
isOpenAccess:       Y
inEPMC:     Y
inPMC:      Y
hasPDF:     Y
hasBook:    N
hasSuppl:   Y
citedByCount:       8
hasReferences:      Y
hasTextMinedTerms:  Y
hasDbCrossReferences:       Y
dbCrossReferenceList:       {'dbName': ['EMBL']}
hasLabsLinks:       Y
```

(continues on next page)

(continued from previous page)

```
hasTMAccessionNumbers:      Y
tmAccessionTypeList:        {'accessionType': ['gen']}
firstPublicationDate:       2012-12-12
```

CHAPTER 4

Dealing with results from web services

Internally, `prunito` uses the wonderful requests package and that provides a sophisticated `Response` object. To avoid losing the power of that object, `prunito` provides a wrapper around it, the `WSResponse` class (where WS stands for web service). Or course, format and content of results returned by calls to web services depend on the service and so there are custom subclasses of `WSResponse` for various web services.

4.1 What does WSResponse (sub)classes do?

They provide helper methods. For example, `size()` returns the number of hits if that particular web service gives that information or it can be quickly determined. `as_file_obejct()` wraps the result's text representation in `io.StringIO`. As many results are sequences-like–we do expect one or more hits for a call after all– special methods for sequences are provided wherever possible to allow iteration or slicing. Obviously, this depends on the service and kind of results.

Any attribute they don't recognize is passed on to the wrapped Response object. This means that things like the the Response's attributes can be accessed via as expected: `text`, `url`, `status`. Methods like `json()` are available, too.

4.2 Results from searching UniProtKB

In addition to what the parent class has, this one also:

- Has a method, `release()`, for getting the current release. Releases are specified as year_number, e.g. 2017_10.
- Has a method, `date()`, to get the date of the current release. This can be returned as a string or a `datetime.datetime` obejct.
- Allows iterating over results for some formats.

  When full UniProtKB text entries are retrieved, they are parsed and the iterator returns the `Record` object. For tabular data, like *list*, *gff* or *tab*, lines are iterated over.

- Tabular data can also be returned as a pandas dataframe.

See also `WSResponseUniprot`.

4.3 Results from identifiers mappings

Results for mapping calls are returned as a table with two columns, *From* and *To*. This table can be accessed as text via `<obj>.text`, the lines can be iterated over but, for convenience, a dictionary of the results is prepared as `<obj>.as_dict()`. A list of mapped-to target IDs is `<obj>.target_ids()`. Identifiers that could not be mapped will be silently ignored, i.e., there won't be any mappings in the result set.

See also `WSResponseUniprotMapping`.

4.4 Results from HMMER searches

todo See also `WSResponseHmmer`.

4.5 Results from searching EuropePMC

todo See also `WSResponseEPMC`.

4.6 Taxonomy results

todo See also `WSResponseTax`.

CHAPTER 5

UniProt REST services

Using UniProt REST services is about retrieving data, for example based on a query. By carefully choosing their format, retrieved data might constitute the true end result, they might be input for another procedure (e.g. FASTA sequences feeding into a similarity search), but often results have to processed further. This is called *parsing*. Another section of the help will take a closer look at this.

5.1 UniProt vs. Proteins API

Services via UniProt allow the following:

- *Searching UniProtKB*
- *Mapping identifiers*
- *Retrieving batches of entries*
- *Converting between different UniProt formats*

The Proteins API provides access data (sets) that often go beyond what you would find in UniProtKB. Many have not been included in `prunito` yet.

- Protein and isoform sequences
- Residue-specific annotations (also called *features*)
- Variation data
- Proteomics data
- Antigen binding sites
- Proteome data
- *Retrieving taxonomy data*
- Genomic coordinates
- Access to UniParc

Common result formats for the REST services are:

- TXT, also called flat file version. The classic format. Contains the most important information. Hard to parse though.
- TAB, tabular format. Data fields (columns) can be specified.
- GFF, for residue-specific annotations, i.e., anything that can be linked to a (range of) amino acid(s).
- XML
- LIST, just accession numbers.
- FASTA, sequences only, in FASTA format.
- JSON

Note: Note that while the classic UniProt REST API does not provide data in JSON format the Proteins API does. On the other hand, Proteins API does not provide GFF or tabular output.

5.2 Searching UniProtKB

The UniProt website has powerful search functionality supporting both free-text and field-based queries. To get the most concise and relevant data set, field-based queries (also called advanced search) should be used. As UniProt's advanced search is RESTful anyway, searching via `prunito` can be done using queries copied directly from the website. This means that complicated queries can be developed and tested on the website first which might come in handy, especially when learning. There is a help page listing all the possible fields for the advanced search.

There is really only one function for searching, `search()`. A query string is the only mandatory parameter. In that case full UniProtKB entries in text (flat file) format are retrieved. The default limit is 2000 entries; this can be changed, of course.

For convenience, methods limiting results to reviewed (i.e. Swiss-Prot) or unreviewed (i.e. TrEMBL) entries can be used. Unsurprisingly, these are `search_reviewed()` and `search_unreviewed()`. Refer to the *API for uniprot.org* docs for more information.

A few example queries:

```
from prunito import uniprot as up

result = up.search('name:laccase AND reviewed:yes')
# or using the convenience function
result = up.search_reviewed('name:laccase')
# getting sequences only
result = up.search_reviewed('name:laccase', frmt='fasta')
# As laccases are enzyme, get relevant enzyme data
result = up.search_reviewed('name:laccase', frmt='tab', columns='id,entry name,
↪comment(FUNCTION),ec')
```

5.3 Looking at results

The section on *dealing with results* explains the basics.

5.4 Mapping identifiers

A paper might contain a list of identifiers for 3D protein structures. A repository for such structure is PDB and their IDs look like *1ABC*. Say we wanted to map those PDB IDs to Ensembl ones–this is what the mapping does. As mappings always have to include UniProt accessions as either source or target, mapping from PDB to Ensembl is s two-step process.

```
# Map PDB -> UniProt
up_from_pdb = up.map_to_or_from_uniprot(['1YWT', '3SMN', '4F3L', '1ES7', '2KDD'],
↪'PDB_ID', 'ACC')
# Map UniProt -> Ensembl
ensembl = up.map_to_or_from_uniprot(up_from_pdb.target_ids(), 'ACC', 'ENSEMBL_ID')
```

From the result, the target IDs that the original set has been mapped to, are available as a list via `up_from_pdb.target_ids()`.

A full list of sources, targets and their abbreviations can be found here. Refer to the *API for uniprot.org* docs for more information.

5.5 Converting between different UniProt formats

I don't think this is used much. One could, for example, convert the text version of a UniProt entry into XML. The text entry would have to be without any errors though for this to work. Refer to the *API for uniprot.org* docs for more information.

5.6 Retrieving batches of entries

If one already has a list of UniProt accessions these can be retrieved using the batch functionality. Refer to the *API for uniprot.org* docs for more information.

```
result = up.retrieve_batch(['P12345', 'P12344'], frmt='txt')
```

5.7 Retrieving taxonomy data

Although UniProt entries contain taxonomy data–an NCBI taxonomy ID, a species name and an abbreviated lineage–extracting the information from, say, the text version is cumbersome. In addition, the information will be incomplete (abbreviated lineage) and for nodes in the lineage no taxonomy IDs are given. Here, the Proteins API comes to the rescue, allowing e.g. retrieval of information on particular nodes or entire lineages of a given taxonomy node, including IDs. Results from Proteins API are always retreived in JSON format and taxonomy nodes can be iterated over. Refer to the *UniProt Proteins API* docs for more information.

```
from prunito import uniprot as up

result = up.get_lineage_for_taxID('9606'):
for node in result:
    print(node)
```

```
{'taxonomyId': 9606, 'scientificName': 'Homo sapiens'}
{'taxonomyId': 9605, 'scientificName': 'Homo'}
{'taxonomyId': 207598, 'scientificName': 'Homininae'}
{'taxonomyId': 9604, 'scientificName': 'Hominidae'}
...
```

Details of a single taxonomy ID can also be retrieved:

```
hs = up.get_info_on_taxID('9606')
print(hs.json())
```

```
{'childrenLinks': ['https://www.ebi.ac.uk/proteins/api/taxonomy/id/741158',
                   'https://www.ebi.ac.uk/proteins/api/taxonomy/id/63221'],
 'commonName': 'Human',
 'mnemonic': 'HUMAN',
 'parentLink': 'https://www.ebi.ac.uk/proteins/api/taxonomy/id/9605',
 'rank': 'species',
 'scientificName': 'Homo sapiens',
 'siblingsLinks': ['https://www.ebi.ac.uk/proteins/api/taxonomy/id/1425170'],
 'superregnum': 'E',
 'taxonomyId': 9606}
```

If the same is needed for several IDs:

```
several_nodes = up.get_info_on_taxIDs(['9606', '6237', '83333'])
for node in several:
    print(node['scientificName'], node['taxonomyId'], len(node['childrenLinks']))
```

```
Homo sapiens 9606 2
Caenorhabditis 6237 51
Escherichia coli (strain K12) 83333 13
```

CHAPTER 6

Parsers for UniProt data

`Prunito` provides parsers for the text (flat file) version of UniProtKB entries and the UniRule XML format. Support for the latter is not up-to-date.

6.1 UniProtKB text parser

CHAPTER 7

API for uniprot.org

CHAPTER 8

UniProt Proteins API

CHAPTER 9

Utilities used in the package

CHAPTER 10

Indices and tables

- genindex
- modindex
- search