opentapioca Documentation

Release stable

Nov 10, 2022

Contents:

This explains how to install and configure OpenTapioca.

CHAPTER 1

Installing OpenTapioca

This software is a Python web service that requires Solr.

1.1 Installing Solr

It relies on a recent new feature of Solr (since 7.4.0), which was previously available as an external plugin, SolrText-Tagger. If you cannot use a recent Solr version, it is possible to use older versions with the plugin installed: this will require changing the class names in the Solr configs (in the `configset` directory).

Install Solr 7.4.0 or above.

OpenTapioca requires that Solr runs in Cloud mode, so you can start it as follows:

```
bin/solr start -c -m 4g
```

The memory available to Solr (here 4 GB) will determine how many indexing operations you can run in parallel (searching is cheap).

In its Cloud mode, Solr reads the configuration for its indices from so-called "configsets" which gouvern the configuration of multiple collections. OpenTapioca comes with the appropriate configsets for its collections and the default one is called "tapioca". You need to upload it to Solr before indexing any data, as follows:

```
bin/solr zk -upconfig -z localhost:9983 -n tapioca -d configsets/tapioca
```

1.1.1 Custom analyzers

Some profiles require custom Solr analyzers and tokenizers. For instance the Twitter profile can be used to index Twitter usernames and hashtags as labels, which is useful to annotate mentions in Twitter feeds. This requires a special tokenizer which handles these tokens appropriately. This tokenizer is provided as a Solr plugin in the `plugins` directory. It can be installed by adding this jar in the `server/solr/lib` directory of your Solr instance (the `lib` subfolder needs to be created first).

1.2 Installing Python dependencies

OpenTapioca is known to work with Python 3.6, and offers a command-line interface to manipulate Wikidata dumps and train classifiers from datasets.

In a Virtualenv, do `pip install -r requirements.txt` to install the Python dependencies, and `python setup.py install` to install the CLI in your PATH.

When developing OpenTapioca, you can use *pip install -e .* to install the CLI from the local files, so that your changes on the source code are directly reflected in the CLI, without the need to run `python setup.py install` every time you change something.

CHAPTER 2

Dump preprocessing and indexing

Various components need to be trained in order to obtain a functional tagger.

First, download a Wikidata JSON dump compressed in `.bz2` format:

```
wget https://dumps.wikimedia.org/wikidatawiki/entities/latest-all.json.bz2
```

2.1 Language model

We will first use this dump to train a bag of words language model:

```
tapioca train-bow latest-all.json.bz2
```

This will create a `bow.pkl` file which counts the number of occurences of words in Wikidata labels.

2.2 PageRank computation

Second, we will use the dump to extract a more compact graph of entities that can be stored in memory. This will be used to compute the pagerank of items in this graph. We convert a Wikidata dump into an adjacency matrix and a pagerank vector in four steps:

1. preprocess the dump, only extracting the information we need: this creates a TSV file containing on each line the item id (without leading Q), the list of ids this item points to, and the number of occurences of such links. There will be an output for every 10'000 items that have been processed. For a rough estimate about the total number of pages please consult the "Content pages" figure on https://www.wikidata.org/wiki/Special:Statistics

   ```
   tapioca preprocess latest-all.json.bz2
   ```

2. this dump must be externally sorted (for instance with GNU sort). Doing the sorting externally is more efficient than doing it inside Python itself.

```
sort -n -k 1 latest-all.unsorted.tsv > wikidata_graph.tsv
```

3. the sorted dump is converted into a Numpy sparse adjacency matrix `wikidata_graph.npz`

```
tapioca compile wikidata_graph.tsv
```

4. we can compute the pagerank from the Numpy sparse matrix and store it as a dense matrix `wikidata_graph.pgrank.npy`

```
tapioca compute-pagerank wikidata_graph.npz
```

This slightly convoluted setup makes it possible to compute the adjacency matrix and pagerank from entire dumps on a machine with little memory (8GB).

2.3 Indexing for tagging

We then need to index the Wikidata dump in a Solr collection. This uses the JSON dump only. This also requires creating an indexing profile, which defines which items will be indexed and how. A sample profile is provided to index people, organizations and places at `profiles/human_organization_place.json`:

```
{
    "language": "en", # The preferred language
    "name": "human_organization_location", # An identifier for the profile
    "solrconfig": "tapioca", # the name of the Solr pipeline to index the dumps
    "restrict_properties": [
        "P2427", "P1566", "P496", # Include all items bearing any of these properties
    ],
    "restrict_types": [
        # Include all items with any of these types, or subclasses of them
        {"type": "Q43229", "property": "P31"},
        {"type": "Q618123", "property": "P31"},
        {"type": "Q5", "property": "P31"}
    ],
    "alias_properties": [
        # Add as alias the values of these properties
        {"property": "P496", "prefix": null},
        {"property": "P2002", "prefix": "@"},
        {"property": "P4550", "prefix": null}
    ]
}
```

Of course the comments in the sample above should not be included: the raw JSON file can be found here.

Pick a Solr collection name (without creating the collection in advance) and run:

```
tapioca index-dump my_collection_name latest-all.json.bz2 --profile profiles/human_
↪organization_place.json
```

Tapioca will create the collection for you, using the appropriate configuration. Note that if you have multiple cores available, you might want to run decompression as a separate process, given that it is generally the bottleneck:

```
bunzip2 < latest-all.json.bz2 | tapioca index-dump my_collection_name - --profile␣
↪profiles/human_organization_place.json
```

2.4 Indexing via SPARQL

If the collection of items to ingest is small enough, it can be fetched by a SPARQL query. In that case we can avoid processing an entire dump and selectively index the items which are returned by the SPARQL query. The SPARQL query is written in a file:

```
tapioca index-sparql my_collection_name my_sparql_query_file --profile profiles/human_
↪organization_place.json
```

The SPARQL query is required to have a variable *item* which ranges over the items to index. It is recommended that the query returns distinct items.

CHAPTER 3

Training a classifier

Once a Wikidata dump is preprocessed and indexed, we can train a classifier to predict matches in text.

3.1 Getting a NIF dataset

Training requires access to a dataset encoded in NIF (Natural Language Interchange Format). Various such datasets can be found at the NLP2RDF dashboard (archived version). The NIF dataset is required to use Wikidata entity URIs for its annotations. Here is an example of what it looks like in the flesh:

```
<https://zenodo.org/wd_affiliations/4> a nif:Context,
      nif:OffsetBasedString ;
   nif:beginIndex "0"^^xsd:nonNegativeInteger ;
   nif:endIndex "67"^^xsd:nonNegativeInteger ;
   nif:isString "Konarka Technologies, 116 John St., Suite 12, Lowell, MA 01852, USA"␣
↪;
   nif:sourceUrl <https://doi.org/10.1002/aenm.201100390> .

<https://zenodo.org/wd_affiliations/4#offset_64_67> a nif:OffsetBasedString,
      nif:Phrase ;
   nif:anchorOf "USA" ;
   nif:beginIndex "64"^^xsd:nonNegativeInteger ;
   nif:endIndex "67"^^xsd:nonNegativeInteger ;
   nif:referenceContext <https://zenodo.org/wd_affiliations/4> ;
   itsrdf:taIdentRef <http://www.wikidata.org/entity/Q30> .
```

Converting an existing dataset from a custom format to NIF can be done using the pynif Python library. This library can be used to generate and parse NIF datasets with a simple API.

3.1.1 Annotating your own dataset

If you want to annotate your own dataset, you could use an existing annotator such as NIFIFY (although it currently does not seem to handle large datasets very well).

3.1.2 Converting an existing NIF dataset to Wikidata

If you have an existing dataset with URIs pointing to another knowledge base, such as DBpedia, you can convert it to Wikidata. This will first require translating existing annotations, which can be done automatically with tools such as nifconverter. Then comes the harder part: you need to annotate any mention of an entity which is not covered by the original knowledge base, but is included in Wikidata. If out-of-KB mentions are already annotated in your dataset, then you can extract these and use tools such as OpenRefine to match their phrases to Wikidata. Otherwise, you can extract them with a named entity recognition tool, or annotate them manually.

3.2 Training with cross-validation

Training a classifier on a dataset is done via the CLI, as follows:

```
tapioca train-classifier -c my_solr_collection -b my_language_model.pkl -p my_
↪pagerank.npy -d my_dataset.ttl -o my_classifier.pkl
```

This will save the classifier as `my_classifier.pkl`, which can then be used to tag text in the web app.

CHAPTER 4

Running the web app

Once a classifier is trained, you can run the web app. This requires supplying the filenames to the various pre-processed files and the Solr collection name in the `settings.py` file. A template for this file is provided as `settings_template.py`. You can then run the application locally for development as follows:

```
python app.py
```

This will expose a development web server at http://localhost:8457/.

For production deployment, you should use a proper web server with WSGI support.

4.1 Keeping in sync with Wikidata

You can keep up to date with Wikidata by listening to its stream of edits, which will keep your Solr collection in sync:

```
tapioca index-stream -p profiles/human_organization_location.json my_solr_collection
```

This command has other options, use *tapioca index-stream –help* for a description of those. This will not update the PageRank and the language model, which are not expected to evolve quickly. You can refresh those from time to time with fresh dumps.

CHAPTER 5

Testing OpenTapioca

OpenTapioca comes with a test suite that can be run with `pytest`. This requires a Solr server to be running on `localhost:8983`, in Cloud mode.

CHAPTER 6

Indices and tables

- genindex
- modindex
- search