Knowledge Graph Exchange Documentation

Chris Mungall et al.

Jan 02, 2020

Contents

A utility library and set of command line tools for exchanging data in knowledge graphs.

The tooling here is partly generic but intended primarily for building the translator-knowledge-graph.

CHAPTER 1

Installation

```
pip3 install -r requirements.txt
python3 setup.py install
```

The installation requires Python 3.

For convenience, make use of the venv module in Python 3 to create a lightweight virtual environment:

```
python3 -m venv env
source env/bin/activate

pip install -r requirements.txt
python setup.py install
```

CHAPTER 2

Documentation

2.1 Command Line Usage

Use the `--help` flag with any command to view documentation.

See the Makefile for examples,

```
make examples
```

To run these examples use,

```
make run_examples
```

2.1.1 Summary

The `node-summary` and `edge-summary` commands takes the address, username, and password of a Neo4j instance, and returns a summary of the data.

2.1.2 Neo4j Upload

The `neo4j-upload` command takes any number of input files, builds a networkx graph from them, and uploads that graph to a neo4j database. To do this it of course needs the database address, username, and password. This will only work through bolt. By default you can access a local neo4j instance at the address `bolt://localhost:7687`.

```
Usage: kgx neo4j-upload [OPTIONS] ADDRESS USERNAME PASSWORD INPUTS...
```

The `--input-type` option can be used to specify the format of these files: csv, ttl, json, txt, graphml, rq, tsv.

2.1.3 Neo4j Download

The `neo4j-download` command downloads a neo4j instance, builds a networkx graph from it, and saves it to the specified file. Like the upload command, this will only work through bolt.

```
Usage: kgx neo4j-download [OPTIONS] ADDRESS USERNAME PASSWORD OUTPUT
```

The `--output-type` option can be used to specify the format of these files: csv, ttl, json, txt, graphml, rq, tsv. The `--labels` and `--properties` options allow for filtering on node and edge labels and properties.

The labels filter takes two inputs. The first input is a choice of where to apply the filter: subject, object, edge, node. The second is the label to apply.

```
--labels edge causes
```

This will result in searching for triples of the form: `(s)-[r:causes]-(o)`

The properties filter takes three inputs: the first being a choice of where to apply the filter, the second being the name of the property, and the third being the value of the property.

```
--properties subject name FANC
```

This will result in searching for triples of the form: `(s {name: "FANC"})-[r]-(o)`. These filter options can be given multiple times.

The `--directed` flag enforces the subject -> object edge direction.

The batch options allow you to download into multiple files. The `--batch-size` option determines the number of entries in each file, and the `--batch-start` determines which batch to start on.

2.1.4 Validate

The `validate` command loads any number of files into a graph and checks that they adhere to the TKG standard.

```
Usage: kgx validate [OPTIONS] INPUTS...
```

The `--input-type` option can be used to specify the format of these files: csv, ttl, json, txt, graphml, rq, tsv.

2.1.5 Dump

The `dump` command takes any number of input file paths (all with the same file format), and outputs a file in the desired format.

```
Usage: kgx dump [OPTIONS] INPUTS... OUTPUT
```

The format will be inferred from the file extention. But if this cannot be done then the `--input-type` and `--output-type` flags are useful to enforce a particular format. The following formats are supported: csv, tsv, txt (pipe delimited text), json, rq, graphml, ttl.

> *Note:* CSV/TSV representation require two files, one that represents the vertex set and one for the edge set. JSON, TTL, and GRAPHML files represent a whole graph in a single file. For this reason when creating CSV/TSV representation we will zip the resulting files in a .tar file.

The `dump` command can also be used to relabel nodes. This is particularly useful for ensuring that the CURIE identifier of each node reflects its category (e.g. genes having NCBIGene identifiers, proteins having UNIPROT identifiers, and so on). The `--mapping` option can be used to apply a pre-loaded mapping to the output as it gets transformed. If the

`--preserve` flag is used then the old labels will be preserved under a modified name. Mappings are loaded with the load-mapping command.

2.1.6 Load Mapping

A mapping is just a python dict object. The `load-mapping` command builds a mapping out of the given CSV file, and saves it with the given name. That name can then be used with the `dump` commands `--mapping` option to apply the mapping.

```
Usage: kgx load-mapping [OPTIONS] NAME CSV
```

By default the command will treat the first and second columns as the input and output values for the mapping. But you can use the `--columns` option to specify which columns to use. The first and second integers provided will be the indexes of the input and output columns.

> **Note:** the columns are zero indexed, so the first is 0 and the second is 1, and so on.

The `--show` flag can be used to display a slice of the mapping when it is loaded so that the user can see which columns have been used. The `--no-header` flag can be used to indicate that the given CSV file does not have a header. If this flag is used then the first row will be used, otherwise it will be ignored.

Example:

First we load a mapping from a CSV file.

```
$ kgx load-mapping --show --columns 0 1 a_to_b_mapping tests/resources/mapping/
↪mapping.csv
a58 : b58
a77 : b77
a17 : b17
a28 : b28
a92 : b92
Mapping 'a_to_b_mapping' saved at /home/user/.config/translator_kgx/a_to_b_mapping.pkl
```

Then we apply this mapping with the dump command.

```
kgx dump --mapping a_to_b_mapping tests/resources/mapping/nodes.csv target/mapping-
↪out.json
Performing mapping: a_to_b_mapping
File created at: target/mapping-out.json
```

2.1.7 Load and Merge

The `load-and-merge` command loads nodes and edges from knowledge graphs as defined in a config YAML, and merges them into a single graph. The destination URI, username, and password can be set with the `--destination-uri`, `--destination-username`, `--destination-password` options.

CHAPTER 3

Examples

3.1 Command Line Usage

Here we will walk through the basic work flow of using KGX from the command line. The files can be found in the github repo.

We'll assume that we're using a local instance of neo4j at `bolt://localhost:7687`, with the username `neo4j` and the password `password`.

3.1.1 Validate

```
$ kgx validate tests/resources/semmed/cell.json tests/resources/semmed/gene.json␣
↪tests/resources/semmed/protein.json
|Nodes|=395
|Edges|=300
ERROR:root:Item: umls_type Message: no such short form
ERROR:root:Item: labels Message: no such short form
ERROR:root:Item: xrefs Message: no such short form
ERROR:root:Item: pmids Message: no such short form
ERROR:root:Item: predicate Message: no such short form
ERROR:root:Item: n_pmids Message: no such short form
ERROR:root:Item: is_defined_by Message: no such short form
```

3.1.2 Dump

Combine three files into one

```
$ kgx dump tests/resources/semmed/cell.json tests/resources/semmed/gene.json tests/
↪resources/semmed/protein.json target/combined.json
|Nodes|=395
```

(continues on next page)

(continued from previous page)

```
|Edges|=300
File created at: target/combined.json
```

3.1.3 Neo4j

Neo4j Upload operation

Uploading `combined.json` to a local neo4j instance

```
kgx neo4j-upload bolt://localhost:7687 neo4j password target/combined.json
|Nodes|=395
|Edges|=300
```

Neo4j Download operation

Downloading a subset of what we had uploaded. Running in debug mode so we can see the cypher queries

```
kgx --debug neo4j-download --properties object id UMLS:C1290952 --labels subject␣
↪disease_or_phenotypic_feature bolt://localhost:7687 neo4j password target/neo4j-
↪download.json
```

Downloading another subset, this time filtering on the edge label

```
kgx --debug neo4j-download --labels edge predisposes bolt://localhost:7687 neo4j␣
↪password target/predisposes.json
```

Merge operation

Load nodes and edges from multiple Knowledge Graphs, as defined in a config YAML, merge them into a single graph and save the merged graph into a Neo4j database.

```
kgx --debug neo4j load-and-merge --destination-uri bolt://localhost:7687 neo4j neo4j␣
↪config.yml
```

where the `config.yml` is similar to sample-merge-config.yml.

3.2 Workflows

KGX provides workflows in Jupyter notebooks that demonstrates how to make use of various components of KGX.

Currently there are 4 workflows:

1. Read From a Remote Knowledge Graph
2. Remap SemMedDB Node ID from UMLS to HGNC
3. Merge SemMedDB with Monarch-Lite
4. Merge RTX with Monarch-Lite

3.3 Examples scripts

3.3.1 SemMedDB Knowledge Graph

1. Get all genomic entities that interacts with one or more genomic entities

```
python examples/scripts/read_from_neo4j.py --host localhost --bolt_port 7687 \
    --username <username> \
    --password <password> \
    --filter subject_category=genomic_entity \
    --filter edge_label=interacts_with \
    --filter object_category=genomic_entity
```

- Total number of nodes loaded: 20656
- Total number of edges loaded: 171811

2. Get all chemical substances and what diseases they treat

```
python examples/scripts/read_from_neo4j.py --host localhost --bolt_port 7687 \
    --username <username> \
    --password <password> \
    --filter subject_category=chemical_substance \
    --filter edge_label=treats \
    --filter object_category=disease
```

- Total number of nodes loaded: 49713
- Total number of edges loaded: 516334

3. Get all chemical substances and the biological process that they affect

```
python examples/scripts/read_from_neo4j.py --host localhost --bolt_port 7687 \
    --username <username> \
    --password <password> \
    --filter subject_category=chemical_substance \
    --filter edge_label=affects \
    --filter object_category=biological_process
```

- Total number of nodes loaded: 43587
- Total number of edges loaded: 658355

4. Get all anatomical entities and the disease that they are associated with

```
python examples/scripts/read_from_neo4j.py --host localhost --bolt_port 7687 \
    --username <username> \
    --password <password> \
    --filter subject_category=anatomical_entity \
    --filter edge_label=location_of \
    --filter object_category=disease
```

- Total number of nodes loaded: 36179

- Total number of edges loaded: 510709

5. Get all activities and behaviors and the diseases that they predispose to

```
python examples/scripts/read_from_neo4j.py --host localhost --bolt_port 7687 \
    --username <username> \
    --password <password> \
    --filter subject_category=activity_and_behavior \
    --filter edge_label=predisposes \
    --filter object_category=disease
```

- Total number of nodes loaded: 1960
- Total number of edges loaded: 5493

6. Get all chemical substances and the diseases that they cause

```
python examples/scripts/read_from_neo4j.py --host localhost --bolt_port 7687 \
    --username neo4j \
    --password <password> \
    --filter subject_category=chemical_substance \
    --filter edge_label=causes \
    --filter object_category=disease
```

- Total number of nodes loaded: 37977
- Total number of edges loaded: 299806

CHAPTER 4

Internal Representation

Internal representation is networkx MultiDiGraph which is a property graph.

The structure of this graph is expected to conform to the tr-kg standard, briefly summarized here:

Nodes

- *id* : required
- *name* : string
- *category* : string. broad high level type. Corresponds to label in neo4j
- extensible other properties

Edges

- *subject* : required
- *predicate* : required
- *object* : required
- extensible other fields

CHAPTER 5

Serialization/Deserialization

Intended to support,

- Generic Graph Formats
- **local or remote files**
  - CSV
  - TSV (such as the RKB adapted data loading formats)
  - RDF (Monarch/OBAN style, . . .)
  - GraphML
  - CX
- remote store via query API
- Neo4j/bolt
- RDF

CHAPTER 6

RDF

CHAPTER 7

Neo4j

Neo4j implements property graphs out the box. However, some implementations use reification nodes. The transform should allow for de-reification.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search