BigWorldGraph Documentation

Release 0.9

Malabalistalicious Unipessoal Lda.

Jul 28, 2017

Contents

Revealing the connections that shape the world we live in.

DISCLAIMER: **This project is still in development and hasn't produced a stable version yet**.

CHAPTER 1

BigWorldGraph

This project is dedicated to make today's political sphere more transparent, visualizing the links between entities in power. This is achieved by automatically extracting those links from texts, utilizing techniques from Natural Language Processing and enriching those results with Wikidata.

The data can then be inspected afterwards using an interactive graph.

General information

The prototype of this project was developed during an internship at MAJ // Digital in Lisbon in 2017. It is open-source (see `LICENSE.md`) and hoped to be improved upon by other volunteers (see section Contributing for more information).

The project is intended to work with all kinds of texts in different languages; however, the prototype was developed to work with a corpus composed of Wikipedia articles of (political) affairs in France since 1996.

To see what future features are planned to be included in this project, check `TODO.md`.

Project description

The project consists of two main parts: The NLP pipeline, extracting links between entities from text and storing them in a graph database as well as the front end using a graph visualization library.

Contributing

Contributions of all forms, be it code contributions, bug reports or suggestions are welcome. Please read the `CONTRIBUTE.md` file for more information or visit the project's GitHub page.

Documentation

Documentation is hosted on *Readthedocs* and can be found here. There you can find more information about how the pipeline is used, how the data in your input corpus is supposed to look and what to expect as intermediate results from the pipeline tasks.

Usage

Installation

Testing

To test the project, execute the following commands in the project's root directory

```
docker-compose -f docker-compose-test.yml build --no-cache
docker-compose -f docker-compose-test.yml up
```

Quickstart

There are two things you can do right after cloning the repository:

1. Running a demo pipeline to see how the pipeline in this project is supposed to work.
2. Building the project and playing around with some toy data on the frontend.

Running the demo pipeline

To run the demo pipeline, execute

```
python3 backend/bwg/demo_pipeline.py
```

Your terminal should show you the following on successful execution:

```
===== Luigi Execution Summary =====

Scheduled 4 tasks of which:
* 4 ran successfully:
    - 1 DemoTask1(...)
    - 1 DemoTask2(...)
    - 1 DemoTask3(...)
    - 1 SimpleReadingTask(...)

This progress looks :) because there were no failed tasks or missing external␣
↪dependencies

===== Luigi Execution Summary =====
```

To see the results of your pipeline, go to `backend/data/pipeline_demo`: There you can the output of different pipeline tasks. The output of the final task, `demo_corpus_replaced.json`, is written in prettified `JSON` to enhance readability.

If you want to know more about how the demo pipeline works, visit this site in the project's documentation.

Building the project with toy data

Building is handled by Docker, so make sure to have it installed beforehand.

Afterwards, the project setup is fairly simple. Just go the root directory of the project and execute the following command:

```
docker-compose build && docker-compose up
```

The building of the docker images in this project might take a while, especially during the first time you're using this project. requests to the API using port `6050` by default (see the documentation for ``bwg/run_api.py` <http://bigworldgraph.readthedocs.io/bwg.run_api.html>`__ for more information).

Now you can play around on the frontend by visiting 127.0.0.1:8080 on your browser!

Getting serious: Running the real pipeline

First of all, make sure to have the corpus file in its designated directory (`backend/data/corpora_french/` by default).

Set up the project like in the previous step with

```
docker-compose build && docker-compose up
```

After all containers are running, you can run the pipeline by executing the following:

```
cd ./pipeline/
docker build . -t pipeline
docker run -v /var/run/docker.sock:/var/run/docker.sock -v `pwd`/stanford/models/:/
↪stanford_models/ --name pipeline pipeline
```

If you are on a Windows system, replace `pwd` inside the `-v` flag with the **absolute** path to the `stanford/models` directory.

First of all, all the necessary Stanford models will be downloaded from a MAJ server to `/pipeline/stanford/models` if necessary. This might take a while. Afterwards, the pipeline will be started. Depending on the size of the corpus file and the tasks in the pipeline, run time can also vary heavily.

The final output of the pipeline should look something like this:

```
===== Luigi Execution Summary =====

Scheduled 4 tasks of which:
* 3 present dependencies were encountered:
    - 1 FrenchPipelineRunInfoGenerationTask(...)
    - 1 FrenchServerPropertiesCompletionTask(...)
    - 1 FrenchServerRelationMergingTask(...)
* 1 ran successfully:
    - 1 FrenchRelationsDatabaseWritingTask(...)

This progress looks :) because there were no failed tasks or missing external␣
↪dependencies

===== Luigi Execution Summary =====
```

Now go to 127.0.0.1:8080 again and marvel at your graph!

CHAPTER 2

Main project contents

The project contains the following modules:

- *bwg.additional_tasks*: Tasks that can be helpful to include in the pipeline, but are not obligatory.
- *bwg.config_management*: Functions used to create the configuration for the pipeline.
- *bwg.corenlp_server_tasks*: NLP pipeline tasks that use the *Stanford CoreNLP Server‘*.
- *bwg.french_wikipedia*: Example pipeline tasks to specifically process the french wikipedia.
- *bwg.helpers*: Helper functions used throughout the project.
- *bwg.mixins*: Mixin classes used in the projects.
- *neo4j_extensions*: Extensions to the `Neo4j` database, to make it compatible to `Luigi` and `Eve`.
- *bwg.standard_tasks*: Standard tasks often uses in pipelines.
- *bwg.utilities*: Mostly serializing functions used in pipeline tasks.
- *bwg.wikidata*: Wrappers for Wikidata, using `pywikibot`.
- *bwg.wikipedia_tasks*: Tasks involving the Wikidata or processing a Wikipedia corpus.
- *bwg.run_api*: Module used to run the API.
- *bwg.demo_pipeline.py*: Simple example pipeline to process a minimal corpus.

Modules

bwg.additional_tasks

Module contents

bwg.config_management

Creating your own pipeline configuration

Every pipeline requires a configuration file with certain parameters. It contains two parts:

1. A kind of "meta" configuration, specifying the parameters a task requires. This way is it secured that the pipeline is run with all necessary configuration parameters. Otherwise a special exception is thrown. Its is stored in a single parameter, called `CONFIG_DEPENDENCIES`.
2. The "real" configuration, just how you know it - config parameters and their corresponding values.

If you add a new kind of task to the pipeline, make sure to include a description of its necessary parameters in your config file's (e.g. `my_pipeline_config.py`) meta config:

```
CONFIG_DEPENDENCIES = {
    ...
    # Your task
    "my_new_task": [
         "{language}_SPECIFIC_PARAMETER",
         "LANGUAGE_INDEPENDENT_PARAMETER"
    ],
    ...
}
```

Then you have to include those declared parameters somewhere in your config file:

```
# My config parameters
ENGLISH_SPECIFIC_PARAMETER = 42
LANGUAGE_INDPENENDENT_PARAMETER = "yada yada"
```

If you implement tasks that extend the pipeline to support other language, please add it to the following list:

```
SUPPORTED_LANGUAGES = ["FRENCH", "ENGLISH"]
```

Finally, create a module for your own pipeline (e.g. `my_pipeline.py`) and build the configuration before running the pipeline, using the pre-defined task names in `my_pipeline_config.py`:

```
import luigi
from bwg.nlp.config_management import build_task_config_for_language

class MyNewTask(luigi.Task):
    def requires():
        # Define task input here

    def output():
        # Define task output here

    def run():
        # Define what to do during the task here


if __name__ == "__main__":
    task_config = build_task_config_for_language(
        tasks=[
            "my_new_task"
        ],
        language="english",
        config_file_path="path/to/my_pipeline_config.py"
    )
    # MyNewTask is the last task of the pipeline
    luigi.build(
        [MyNewTask(task_config=task_config)],
        local_scheduler=True, workers=1, los
```

In case you are writing the data into a `Neo4j` database, make sure to include the following parameters

```
# Neo4j
NEO4J_USER = "neo4j"
NEO4J_PASSWORD = "neo4j"
NEO4J_NETAG2MODEL = {
    "I-PER": "Person",
    "I-LOC": "Location",
    "I-ORG": "Organization",
    "DATE": "Date",
    "I-MISC": "Miscellaneous"
}
```

Module contents

bwg.corenlp_server_tasks

Using `CoreNLP Server` tasks

To use `CoreNLP Server` tasks, appropriate Stanford NLP models are also required. For french, those are downloaded from a MAJ // Digital server when `docker-compose.yml` is called.

In case you are running the project on a non-french corpus, you can adjust the command that runs the server in its corresponding dockerfile at `backend/data/stanford/Dockerfile` to fit the current language. Do this by adjusting the `-serverProperties` argument appropriately (consult the official documentation for more help).

Module contents

bwg.demo_pipeline

This module illustrates how pipelines in the context of this project are used. For this purpose, the pipeline in this module comprises three very simple tasks:

- `DemoTask1` simply replaces characters defined in `demo_pipeline_config.py` with "x".
- `DemoTask2` duplicates characters defined in `demo_pipeline_config.py`.
- `DemoTask3` removes characters defined in `demo_pipeline_config.py`.

Therefore, the original corpus, that should look like this (see `data/corpora_demo/demo_corpus.xml`):

```
<doc id="12345" url="https://github.com/majdigital/bigworldgraph" title="Demo corpus">
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor␣
↪incididunt ut labore et dolore magna aliqua.
Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex␣
↪ea commodo consequat.
Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu␣
↪fugiat nulla pariatur.
Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt␣
↪mollit anim id est laborum.
```

```
</doc>
```

Should result in `data/pipeline_demo/demo_corpus_removed.json` looking like this

```
{
    "meta": {
        "id": "12345",
        "url": "https://github.com/majdigital/bigworldgraph",
        "title": "Demo corpus",
        "type": "article",
        "state": "removed"
    },
    "data": {
        "12345/00001": {
            "meta": {
                "id": "12345/00001",
                "state": "removed",
                "type": "sentence"
            },
            "data": "Looex isux doooo sit xxet, coonnsectetu xdiiscinng eit, sed doo␣
↪eiusxood texoo inncididunnt ut xbbooe et dooooe xxgnnx xiqux."
        },
        "12345/00002": {
            "meta": {
                "id": "12345/00002",
                "state": "removed",
                "type": "sentence"
            },
            "data": "Ut ennix xd xinnix vennixx, quis nnoostud executxtioonn uxxcoo␣
↪xbboois nnisi ut xiqui ex ex cooxxoodoo coonnsequxt."
        },
        "12345/00003": {
            "meta": {
                "id": "12345/00003",
                "state": "removed",
                "type": "sentence"
            },
            "data": "Duis xute iue doooo inn eehenndeit inn vooutxte veit esse ciux␣
↪dooooe eu fugixt nnux xixtu."
        },
        "12345/00004": {
            "meta": {
                "id": "12345/00004",
                "state": "removed",
                "type": "sentence"
            },
            "data": "Exceteu sinnt ooccxecxt cuidxtxt nnoonn ooidennt, sunnt inn cux␣
↪qui oofficix deseunnt xooit xnnix id est xbbooux."
        }
    }
}
```

The pipeline is simply run by running the module:

```
python3 bwg/demo_pipeline.py
```

Adjusting your pipeline configuration

If you add a new kind of task to the pipeline, make sure to include a description of its necessary parameters in your pipeline configuration file. You can use `bwg/raw_pipeline_config.py` as a template, which provides a minimal example.

```
CONFIG_DEPENDENCIES = {
    ...
    # Your task
    "my_new_task": [
         "{language}_SPECIFIC_PARAMETER",
         "LANGUAGE_INDEPENDENT_PARAMETER"
    ],
    ...
}
```

Then you have to include those declared parameters somewhere in your config file:

```
# My config parameters
ENGLISH_SPECIFIC_PARAMETER = 42
LANGUAGE_INDPENENDENT_PARAMETER = "yada yada"
```

If you implement tasks that extend the pipeline to support other language, please add it to the following list:

```
SUPPORTED_LANGUAGES = ["FRENCH", "ENGLISH"]
```

Finally, create a module for your own pipeline (e.g. `bwg/my_pipeline.py`) and build the configuration before running the pipeline, using the pre-defined task names in your pipeline file:

```
import luigi
from bwg.nlp.config_management import build_task_config_for_language

class MyNewTask(luigi.Task):
    def requires():
        # Define task input here

    def output():
        # Define task output here

    def run():
        # Define what to do during the task here


if __name__ == "__main__":
    task_config = build_task_config_for_language(
        tasks=[
            "my_new_task"
        ],
        language="english",
        config_file_path="path/to/pipeline_config.py"
    )

    # MyNewTask is the last task of the pipeline
    luigi.build(
        [MyNewTask(task_config=task_config)],
        local_scheduler=True, workers=1, log_level="INFO"
    )
```

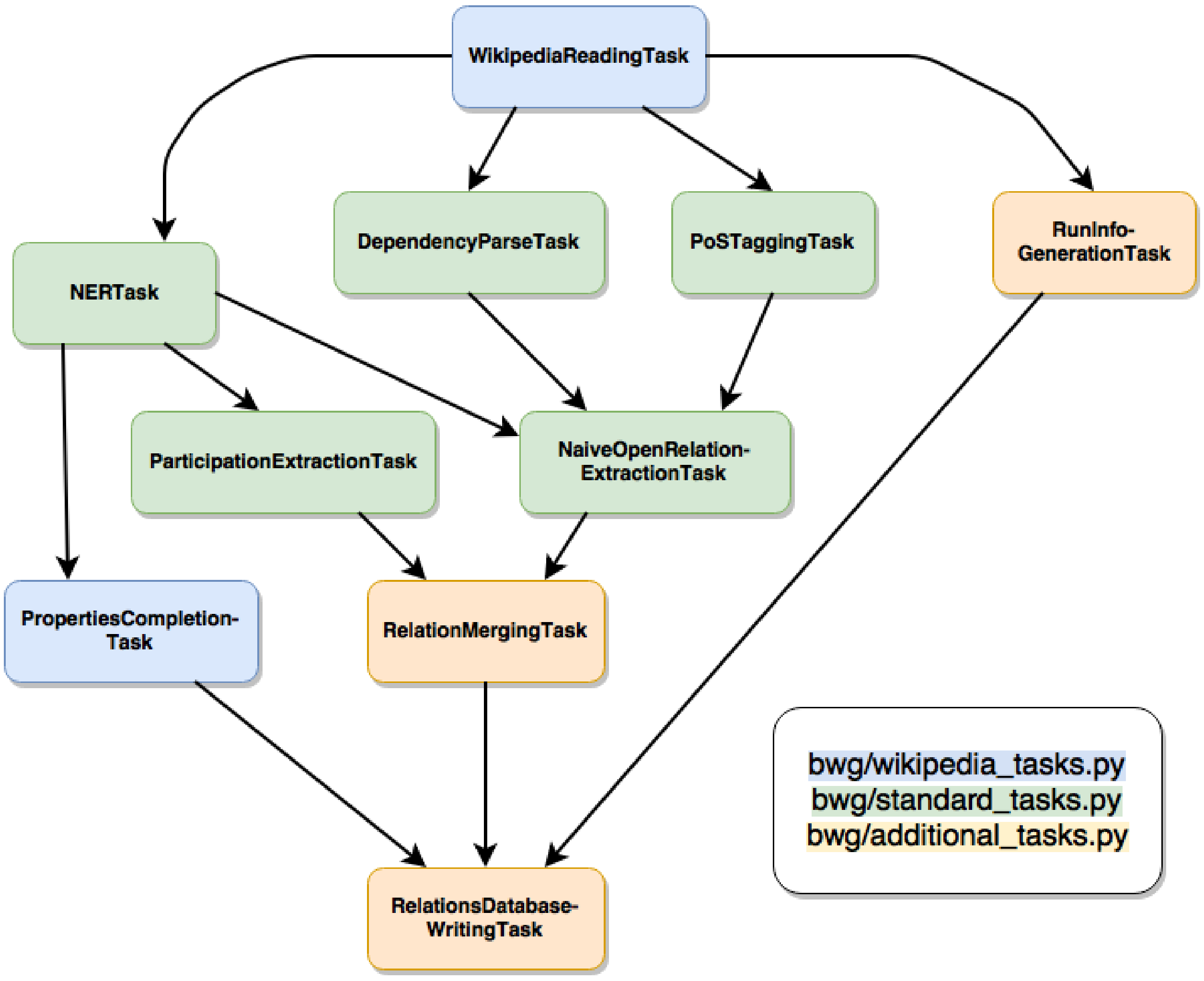
Module contents

bwg.french_wikipedia

Backstory

This is sample pipeline to illustrate the project. It uses a Wikipedia dump from the French Wikipedia as a basis, filtering all entries about affairs afterwards. The final corpus was then compiled manually to make sure to only include relevant articles. It was created around the time of the French elections in 2017.

The pipeline comprises the following tasks:



If you want to work with the original data, get in touch with MAJ // Digital or the creator of the dataset.

Module contents

bwg.helpers

Module contents

bwg.mixins

Module contents

bwg.neo4j_extensions

Motivation

This module was created for several reasons: First of all, `Neo4jDatabase` was created to provide a tailored wrapper to a `Neo4j` database with extended and only relevant functionalities, including `Entity` and `Relation`.

Other classes like `EveCompabilityMixin`, `Neo4jLayer`, `Neo4jLayer` were written to make the database work with the `RESTful API` Eve. Although there is a library already bridging this gap, which is called eve-neo4j, it was causing problems, namely this one. No viable solution could be found, thereby these classes were created.

Finally, `Luigi` defines an interface for possible task targets (= output destinations). To write data directly into a database within a `Luigi` task, `Neo4jTarget` was created.

Module contents

bwg.run_api

Usage

This module contains the API. To run the API, just this command in the source root folder:

```
docker-compose build && docker-compose up
```

So fill the database connected to the API with data, use

```
sh ./run_pipeline.sh
```

This requires you to provide a corpus file in a directory specified in your pipeline's config file.

Features

So far, you can use the API to do the following things:

Get the current version of the API by using the ``/version/‘ endpoint:

```
curl -gX GET http://127.0.0.1:5000/version/
```

will return

```
{"version": "0.7.0", "license": "Copyright (c) 2017 Malabaristalicious Unipessoal,␣
↪Lda.\nFor more information read LICENSE.md on https://github.com/majdigital/
↪bigworldgraph"}
```

You can also query for entities and their relationships in the database using the `/entities/` endpoints (not recommended because the result will probably be very big):

```
curl -gX GET http://127.0.0.1:5000/entities/
```

which returns

```
{"_items": [{"nodes": [{"uid": "42904ca7f582449eafb98c1e94cd8b0d", "category": "Person
↪", "label": "Dreyfus", "data": {"wikidata_last_modified": "2015-12-31T03:11:20Z",
↪"wikidata_id": "Q3039421", "label": "Dreyfus", "description": "pi\u00e8ce de␣
↪th\u00e9\u00e2tre de Jean-Claude Grumberg", "type": "I-PER", "claims": {}}, "id": 0}
↪, ...}
```

To be more specific, you can query special kinds of entities like /people/, /locations/, /organizations/, /companies/, /parties/, /affairs/, /politicians/, /businesspeople/, /media/, /misc/.

To make responses more readable for humans, use the ?pretty query parameter:

```
curl -gX GET http://127.0.0.1:5000/people?pretty
```

which in turn returns

```
{
    "_items": [
        {
            "nodes": [
                {
                    "uid": "42904ca7f582449eafb98c1e94cd8b0d",
                    "category": "Person",
                    "label": "Dreyfus",
                    "data": {
                        "wikidata_last_modified": "2015-12-31T03:11:20Z",
                        "wikidata_id": "Q3039421",
                        "label": "Dreyfus",
                        "description": "pi\u00e8ce de th\u00e9\u00e2tre de Jean-Claude␣
↪Grumberg",
                        "type": "I-PER",
                        "claims": {}
                    },
                    "id": 0
                },
                ...
            ],
            "links": [
                ...
            ]
        }
    ],
    ...
}
```

Warning:

As of version 0.7.0, `BigWorldGraph` only comprises the following query parameters:

- ?pretty
- Filtering like /entities?uid=42904ca7f582449eafb98c1e94cd8b0d

Any other functionalities presented by Eve in its documentation are not implemented yet.

To only get a subset of a graph, specify an identifier in for the target node in the request:

```
curl -gX GET http://127.0.0.1:5000/people?uid=42904ca7f582449eafb98c1e94cd8b0d
```

This will return the target node as well as its **friends** and its **friends of friends**.

Module contents

bwg.standard_tasks

Module contents

bwg.utilities

Module contents

bwg.wikidata

Module contents

bwg.wikipedia_tasks

Module contents

CHAPTER 3

Indices and tables

- genindex
- modindex
- search