CITY
UNIVERSITY OF LONDON
EST 1894

IN3067/INM713 Semantic Web Technologies and Knowledge Graphs

Laboratory 9: Ontology Embeddings with OWL2Vec*

Ernesto Jiménez-Ruiz

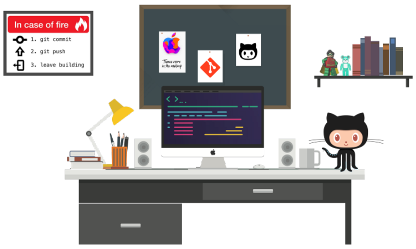
Academic course: 2023-2024
Updated: April 1, 2024

Contents

1 Git Repositories

Support codes for the laboratory sessions are available in *GitHub*. There are two repositories, one in Python and another in Java:

`https://github.com/city-knowledge-graphs`



2 Module evaluation

Your (constructive) feedback is very important for the future evolution of the module. More information on `https://city.surveys.evasysplus.co.uk/` and the **Student Hub**. Evaluations are anonymous.

3 Using OWL2Vec*

In this laboratory session we are using OWL2Vec*, a system that embeds the semantics of an OWL ontology (`https://github.com/KRR-Oxford/OWL2Vec-Star`). OWL2Vec* currently relies on random walks and word embedding and learns a (numerical) vector representation for each URI and word in the ontology. OWL2Vec* generates a document of sentences from the ontology (using this RDF2vec implementation: `https://github.com/IBCNServices/pyRDF2Vec/`) and then applies Word2vec as neural language model (see details here: `https://radimrehurek.com/gensim/models/word2vec.html`).

3.1 Installation

Please use the the OWL2Vec* version (zip file) from the GitHub repositories (`OWL2Vec-Star-IN3067-INM713.zip`). Unfortunately for the Java developers, OWL2Vec* is only available in Python. Nevertheless, running OWL2Vec* is very easy. Before you start, download OWL2Vec* (`OWL2Vec-Star-IN3067-INM713.zip`), unzip the file, and open the folder where the codes are (*e.g.*, location of `setup.py`). ***Tip:*** *To run the commands in Options 1 and 2 below make sure you are located in that directory, also to run the jupyter notebook that is provided within the zip file.*

Option 1 (running OWL2Vec* from the terminal):

1. Install Python 3 (if not done before): `https://www.python.org/downloads/`
2. Install setuptools: `https://pypi.org/project/setuptools/`
3. Run this command in the terminal:[1] `make install` or `python setup.py install` (in Windows and other distributions). You may need Root privileges in Linux.
4. Run OWL2Vec* in the terminal:
   `owl2vec_star standalone --config_file default.cfg`

Option 2 (running OWL2Vec* from a notebook or a python script):

1. Install Python 3 (if not done before): `https://www.python.org/downloads/`
2. Install `pip` (if not done before): (`https://pip.pypa.io/en/stable/installation/`)
3. Install the library dependencies in the file `requirements_owl2vec.txt` (use of environments is recommended):
   e.g., `pip3 install -r requirements_owl2vec.txt`
4. Run the notebook: `jupyter_notebook_owl2vec.ipynb`

3.2 Configuration

In the file `default.cfg` we can indicate the following configuration parameters for OWL2Vec* (see lecture slides or the OWL2Vec* paper for more details):

- `ontology_file:` input OWL ontology.
- `embedding_dir:` path and file name for the output embeddings.
- `cache_dir:` path to store intermediate files.

[1] For Windows user you need to use the Windows Command Prompt. I can help on this: `https://www.makeuseof.com/tag/a-beginners-guide-to-the-windows-command-line/`

- `ontology_projection`: if the graph projection of the ontology is enabled.
- `projection_only_taxonomy`: if only rdfs:subClassOf is taken into account.
- `multiple_labels`: if more than the main label is considered.
- `avoid_owl_constructs`: if OWL 2 constructs are avoided in the generated document.
- **Walker parameters:** to build the document sentences.
  - `walker`: random walks or Weisfeiler-Lehman (wl) subtree kernel.
  - `walk_depth`: depth of each of the performed walks to create each sentence.
- **OWL2Vec* parameters:** type of document of sentences to build. One could create a document with only words (*i.e.*, Lit_Doc)
  - `URI_Doc`: sentences with only entity URIs.
  - `Lit_Doc`: sentences with only words.
  - `Mix_Doc`: mixing words and URIs in the sentences.
  - `Mix_Type`: the type for generating the mixture document (all or random).
- `pre_train_model`: optional path to a pre-trained Word2vec model.
- **Word2vec parameters:**
  - `embed_size`: the size for embedding.
  - `iteration`: number of iterations in training the language model.
  - `min_count`: minimum word occurrence to create an embedding.
  - `window`, `negative` and `seed`: other Word2vec training paramaters.

3.3 Output

OWL2Vec* produces two embedding files as output (in `embedding_dir`):

- Gensim model (`.embedding` file).
- Word2vec text format (`.txt` file). This file is readable with a text editor. Each line is composed by a key (*i.e.*, word) and a value (*i.e.*, vector).

The generated sentence document is available in the `cache` output folder (*i.e.*, `document_sentences.txt`). This file is interesting to see how the ontology has been transformed into a text document. It is also very relevant for the Java developers (see below).

4 Using the Embeddings

The computed embeddings can be used to calculate (cosine) similarities, perform clustering of entities, and use them in subsequent machine learning tasks.

Python. In the GitHub repository there is an example script (`load_embeddings.py`) that loads pre-computed embeddings, in this case by OWL2Vec*. In python we use the gensim `keyedvectors` library: `https://radimrehurek.com/gensim/models/keyedvectors.html`. Once the model has been loaded one can calculate (cosine) similarities among ontology entities/words (*i.e.*, using their associated vectors) and get their closest entities/words. A jupyter notebook is also provided.

Java. In the GitHub repository, there is a class `WordEmbeddings.java`, implemented over the Deeplearning4j library (`https://deeplearning4j.konduit.ai`). Although it is in principle possible to load in Java embedding models pre-computed in Python, this did not seem to work. `WordEmbeddings.java` creates embeddings from the document of sentences (*i.e.*, `document_sentences.txt`). As in Python, with this class one can also load a pre-computed model and calculate similarities among ontology entities/words (*i.e.*, using their associated vectors) and get their closest entities/words. The results with the java version of `Word2vec` differ with respect to those in Python. Alternatively, one could also try to load the vectors directly reading the text file computed by OWL2Vec* (*e.g.*, `ontology.embeddings.txt`).

5 Embedding the Pizza ontology

The ontology is available in the GitHub repositories.

Task 9.1 Run OWL2Vec* over the `pizza.owl` ontology.

Task 9.2 Compute the similarity between the following elements.

- `http://www.co-ode.org/ontologies/pizza/pizza.owl#Margherita` and `margherita`
- `http://www.co-ode.org/ontologies/pizza/pizza.owl#Hot` and `http://www.co-ode.org/ontologies/pizza/pizza.owl#Medium`
- `CheesyPizza` and `CheeseTopping`

Task 9.3 Get the most similar entities/words for the following elements:

- `Hot`
- `http://www.co-ode.org/ontologies/pizza/pizza.owl#CheesyPizza`
- `http://www.co-ode.org/ontologies/pizza/pizza.owl#Fiorentina`
- `Soho`

Task 9.4 (Optional) Perform clustering using the K-means algorithm for example and visually represent the results in 2-dimensions using PCA.

Python, relevant references:

- `https://jakevdp.github.io/PythonDataScienceHandbook/05.11-k-means.html`
- `https://jakevdp.github.io/PythonDataScienceHandbook/05.09-principal-component-analysis.html`

Java, relevant references:

- `https://spark.apache.org/docs/2.1.0/mllib-dimensionality-reduction.html#principal-component-analysis-pca`
- `https://spark.apache.org/docs/2.1.0/mllib-clustering.html#k-means`

6 Embedding the FoodOn ontology

In the following tasks we will use teh `FoodOn` ontology (`https://foodon.org/`). The ontology is also available in the GitHub repositories.

Task 9.5 Run OWL2Vec* over the `foodon-merged.owl` ontology. This ontology is larger and computing the embeddings will take much longer. The embeddings will also be richer as the generated document is rather large (>1Gb).

Task 9.6 Compute the similarity between the following elements.

- `http://purl.obolibrary.org/obo/FOODON_00002434` (mushroom food product) and `mushroom`
- `http://purl.obolibrary.org/obo/FOODON_00002434` (mushroom food product) and `http://purl.obolibrary.org/obo/FOODON_00001287` (mushroom food source)
- `http://purl.obolibrary.org/obo/FOODON_03304544` (frozen chicken) and `http://purl.obolibrary.org/obo/FOODON_03311876` (baked chicken)

Task 9.7 Get the most similar entities/words for the following elements:

- `mushrooms`
- `chicken`
- `http://purl.obolibrary.org/obo/FOODON_03411323` (rabbit)
- `http://purl.obolibrary.org/obo/FOODON_03411129` (trout and salmon family)

Task 9.8 (Optional) Perform clustering using the K-means algorithm for example and visually represent the results in 2-dimensions.