Step by Step Guide

For more information contact scnunes@fc.ul.pt

1 Document Structure

This guide has three main phases and a section dedicated to the baseline of our paper.

1. Obtain two embeddings for the pairs you desire to predict an association;
2. Obtain a final embedding for each pair through vector operations OR calculate cosine similarity between embeddings;
3. Association prediction with machine learning algorithms;

Attention: All the phases are prepared for a 70-30 split, in case you want to use a 10-fold cross validation you need to access the *10-fold* folder with files and scripts prepared for that effect. If you choose to use the 10-fold we also provide a Median-Calculation.py to calculate the median result of the 10 partitions generated.

2 Phase 1 (Embedding Calculation)

For this phase, you have the folder ***1.Embeddings_calculation***[1] with each possible embedding method (files that start with *Run_*): RDF2Vec, OPA2Vec, OWL2Vec and OpenKE (TransE and Distmult).

2.1 RDF2Vec

In the folder ***DEMO*** we provide an example of entities file called *rdf2vec_entities* and annotations file *rdf2vec_openke_annotations*.

- You need the *Run_RDF2VEC_embeddings.py* and folder *pyrdf2vec*

Instructions
Open python file and run it:

- Running 1 ontology: Function *run_embedddings_1kg* that needs 1 ontology file, 1 entities file, 1 annotations file, the vector size, the type of word2vec, the number of walks and the name of the output file or path for it.

[1] Important: In the 'README.md' file are the GitHub links for all the original implementations and the needed requirements.

- Running 2 ontologies: Function *run_embedddings_2kg* that needs 2 ontology files, 1 entities file, 2 annotations files, the vector size, the type of word2vec, the number of walks and the name of the output file or path for it.

- Some of the variables have already example values in the script such as the vector size, the type of wor2vec and number of walks but that can be changed.

- The entities file is one entity per line with the full url (example on folder *DEMO*).

- The annotations files are designed to be in the format 'url_entity *tab* list with annotations' (example on folder *DEMO* that also apply for the OpenKE embeddings)

Attention: In case of problems with the libraries versions for rdf2vec here are some versions that work: numpy==1.18.1; scipy==1.4.1; rdflib==4.2.2; gensim==3.8.1; tqdm==4.40.0

2.2 OPA2Vec

In the folder ***DEMO*** we provide an example of entities file *opa2vec_entities* and annotations file *opa2vec_annotations*.

- You need the *Run_OPA2Vec_embeddings.txt* and folder *opa2vec*

Instructions

You need to run in the command line so the text file only has information for you to do that:

- Download the default pre-trained pubmed model (two files called *RepresentationModel_pubmed.txt.wv.syn0.npy* and *RepresentationModel_pubmed.txt.wv.syn1neg.npy*) and put inside the *opa2vec* folder. The files can be downloaded from the original opa2vec GitHub or from this drive. This model is pretrained on vectors with 200 features so to change the vector size you need to change the main code inside word2vec.py (*opa2vec* folder) and pre train on other model or without pre train (we provide a modified word2vec python file without pre training available inside the ***DEMO*** folder).

- This embedding method only accepts one ontology file in owl format even if you have more ontologies, so you need to merge them into a single file (for instance use the ROBOT Tool.

- Only accepts one annotation file in the format 'entity *tab* <url>' (example on folder ***DEMO***)

- Run in command line inside the *opa2vec* folder:

  python3 runOPA2Vec.py -ontology "ontology_file_path.owl" -associations "AnnotationsFile_path.tsv" -outfile "Opa2vec_embeddings.txt"

Attention: This method does not need an entities file, but you need to have your all your entities in the annotations file provided.

2.3 OWL2Vec

In the folder ***DEMO*** we provide an example of entities file *owl2vec_entities*.

- You need the *Run_OWL2Vec_embeddings.py* and folder *owl2vec*

Instructions
Open python file and run it:

- The OWL2Vec method only accepts one single file with the ontologies and annotations file in. So in case of more than one ontology you need to merge them but before you need to add the annotations. We recommend the use of the OWLReady2 library to add the annotations.
- Use function *write_embeddings* that will get the embeddings from the function *run_OWL2Vec*. The first function needs the name or path for the output file and the second one needs the ontology file with the ontologies and annotations (in owl format) and the entities file.
- Some of the variables have already example values in the script such as the vector size and number of walks but that can be changed.
- The entities file is one entity per line with the full url (example on folder ***DEMO***).

Attention: To familiarize with OWLReady2, inside the folder ***DEMO*** we provide a folder *Add_annots_owlready2* with an example script called *owlready2_HPannots.py* that adds the HP annotations to the Human Phenotype Ontology, so only needs the annotations file (example in folder) and the needed ontology in owl format. We also give a script *owlready2_GOannots.py* that adds GO annotations to the Gene Ontology and the difference is that besides the ontology in owl format, it is also required the ontology in obo format.

2.4 OpenKE(DistMult and TransE)

Both DistMult and TransE are from the OpenKE library and the implementation is similar to the RDF2Vec. In the folder ***DEMO*** we provide an example of entities file called *openke_entities* and annotations file *rdf2vec_openke_annotations*.

- You need the *_Embeddings.pyRun_OpenKE* and *Process_KG_TransE-DistMult.py'OpenKE'*

Instructions
Open *Run_OpenKE_Embeddings.py* python file and run it:

- Running 1 ontology: Function *construct_embeddings_1onto* that needs 1 ontology file, 1 entities file, 1 annotations file, the vector size, the name of the output file or path for it and the model embedding you want to use (model_embedding = "distMult" or model_embedding = "TransE")
- Running 2 ontologies: Function *run_embedddings_2kg* that needs 2 ontology files, 1 entities file, 2 annotations files, the vector size, the name of the output file and the model embedding you want to use (model_embedding = "distMult" or model_embedding = "TransE")
- Some of the variables have already example values in the script such as the vector size but that can be changed.
- The entities file is a csv with the correspondent entity pairs. The format of each line of the dataset files is "Ent1 Ent2" (example on folder ***DEMO***);
- The annotations files are designed to be in the format 'url_entity *tab* list with annotations' like the RDF2Vec method (example on folder textbftextitDEMO)

Attention: The python file *Process_KG_TransE-DistMult* does not to be changed unless you intend to give a different entities file.

3 Phase 2 (Vector Operations)

For this phase, you have the folder ***2. Vector Operations an CS*** with two python files: *Vector_operations.py* and *CS_threshold.py*. Besides the cosine similarity we use five other vector operations: Hadamard, concatenation, average, Weighted-L1 and Weighted-L2

We provide an example *output_operator.csv* that shows the final result output of the vector operations, in this case the cosine similarity, and the performance of the cosine similarity in the file *example_Performance_Baseline_Cosine_Similarity.txt*.

Additionally there is a folder *Indexes_split* containing the indexes for our dataset using a 70-30 split as well as the used script for that effect.

1. **Instructions for vector operations**
   Open *Vector_operations.py* python file and run it:

- Inside the script there are two functions: Function *run_opa2vec_operations*, ready to read the opa2vec output result, and the function *run_other_operations* that reads all the other embedding outputs.
- You have available vector operations and the cosine similarity calculations. Each of these has a final CSV for output on the format 'Entity 1; Entity 2; Operation vector ;Label'.

- To run this script you need to provide the embeddings file (ex. opa2vec), a dataset and a path or name for the output file.
- The dataset file is in the format 'Nr;Entity A;Entity B;Label' (we provide the dataset used *Dataset_Pairs_Label.csv* in the main root of the GitHub).

After running the file *Vector_operations.py* for a specific embedding method you will arrive to 6 final files. The vector operations will be used in phase 3. The file called *Data_Cosine_Similarity.csv* will be used when running the *CS_threshold.py* to obtain a performance of the cosine similarity.

Attention: For the 10fold cross validation access the ***10-fold*** **folder** with files and scripts prepared for that effect.

2. Instructions for cosine similarity performance calculation
Open *CS_threshold* python file and run it:

- This file is prepared for a 70-30 split, and it will need the cosine similarity file (example in folder) and the indexes file splinted in train and test (the script is ready to get the indexes we used in the folder *Indexes_split*).
- This script is automatic without any changes needed. Example of a performance output file in the folder.

Attention: For the 10fold cross validation access the ***10-fold*** **folder** with files and scripts prepared for that effect.

4 Phase 3 (ML Performance)

For this phase, you have the folder ***3. Performance ML***. It uses four diferent ML algorithms: Random Forest, XGB, Nayve Nayes and MLP.
Instructions Open *Performance_ML_70-30split* python file and run it:

- This file is prepared for a 70-30 split, and it will need the vector operations output files and the indexes file splinted in train and test (the script is ready to get the indexes we used in the folder *Indexes_split*).
- This script is automatic without any changes needed. Example of a output file with the performance of random forest and hadamard in the folder.

Attention: For the 10fold cross validation access the ***10-fold*** **folder** with files and scripts prepared for that effect.

5 Baseline

For the baseline you have the folder ***SS_Baseline***. The baseline uses the SSMC tool and six different semantic similarity measures.

Instructions for SSMC tool
Run the *SSMC-script.py*:

- SSMC accepts as input a JSON file with a series of mandatory (and optional) user defined configurations (example in *Configuration.json*).

Instructions for baseline performance
Run the *Threshold_Baseline.py*:

- This file is prepared for a 70-30 split, and it will need the *SSMC_output.csv* file (example in folder) and the indexes file splinted in train and test. For a 10-fold cross validation, access the *10-fold* folder with files and scripts prepared for that effect.

- This script is automatic without any changes needed except if you want other measures. Example of a performance output file for one of the measures is in the folder.

Attention: For the 10fold option access the *10-fold* folder with files and scripts prepared for that effect.