pyRDF2Vec: A Python Implementation and Extension of RDF2Vec

Gilles Vandewiele, Bram Steenwinckel, Terencio Agozzino, and Femke Ongenae

IDLab, Ghent University - imec, 9000 Gent, Belgium

Abstract. This paper introduces pyRDF2Vec¹, a Python software package that reimplements the well-known RDF2Vec algorithm along with several of its extensions. By making the algorithm available in the most popular data science language, and by bundling all extensions into a single place, the use of RDF2Vec is simplified for data scientists. The package is released under a MIT license and structured in such a way to foster further research into sampling, walking, and embedding strategies, which are vital components of the RDF2Vec algorithm. Several optimisations have been implemented in pyRDF2Vec that allow for more efficient walk extraction than the original algorithm. Furthermore, best practices in terms of code styling, testing, and documentation were applied such that the package is future-proof as well as to facilitate external contributions.

Keywords: RDF2Vec · walk-based embeddings · open source

1 Introduction

Knowledge Graphs (KGs) are an ideal candidate to perform hybrid Machine Learning (ML) where both background and observational knowledge are taken into account to construct predictive models. However, since KGs are symbolic data structures, they cannot be fed to ML algorithms directly and first require a non-trivial transformation step in which symbolic substructures of the graph are converted into numerical representations. These transformation techniques can typically be classified as being feature-based or embedding-based [24]. Featurebased approaches are often interpretable, but require domain knowledge about the task at hand and are effort-intensive. Embedding-based approaches, on the other hand, are typically agnostic to the task and are usually able to outperform their feature-based counterparts. Resource Description Framework To Vector (RDF2Vec) [16] is an unsupervised, task-agnostic, and embedding-based approach that has gained significant popularity over the past few years. RDF2Vec builds on the popular Natural Language Processing (NLP) technique Word2Vec. The latter generates embeddings for different tokens present in a corpus, by training a neural network in an unsupervised way that must predict either a token based on its context (Continuous Bag of Words) or the context based on a token

¹ https://github.com/IBCNServices/pyRDF2Vec

(Skip-Gram). The corpus, fed to Word2Vec, is constructed by extracting a large number of walks from the KG. A walk is a sequence of entities obtained from the KG by starting at a certain entity and traversing the directed edges.

Since its initial publication, in 2017, many extensions to the algorithm have been proposed. However, each of these extensions are individual implementations, which complicates combining several of them. Moreover, the original code for RDF2Vec was written in Java, which is significantly less popular than Python for data science, according to the Kaggle Survey 2021¹. In Figure 1, the answers to the question "What programming languages do you use on a regular basis?", where multiple answers were possible, are depicted. It should be noted that among the 4769 people who selected Java as being used regularly, only 598 did not pick Python. This makes it difficult to integrate the original RDF2Vec implementation into a data science pipeline, which is typically written in Python.

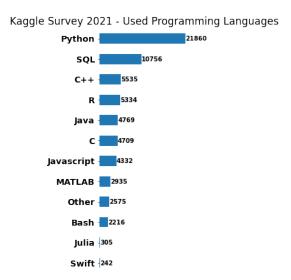


Fig. 1: Programming languages used by data scientists according to the Kaggle Survey 2021.

In this paper, we present pyRDF2Vec, a Python implementation of the original algorithm and many of its extensions. Moreover, various mechanisms are built in which allows to better handle large KGs. The code is released under an open-source license and is written in a way to facilitate further research into the different components of the RDF2Vec algorithm. The remainder of this paper is structured as follows. In Section 2, we provide background on representation learning for KGs, followed by an in-depth discussion of RDF2Vec and its exten-

¹ https://www.kaggle.com/c/kaggle-survey-2021

sions. Then, in Section 3, we present the architecture of our pyRDF2Vec package and the mechanisms set in-place to easily allow for contributions by others. In Section 4, we discuss some studies and other software packages that have already made use of pyRDF2Vec. Finally, we conclude our paper in Section 5. In Appendix A, we provide a code snippet that shows how pyRDF2Vec can be used.

2 Background

In this section, we describe the necessary background to elaborate upon pyRDF2Vec. First, we will discuss related work regarding the transformation of a KG into numerical representations. Afterwards, we outline an in-depth overview of how RDF2Vec works and its extensions released over the past few years.

2.1 Representation Learning

As mentioned in the introduction, a feature-based or embedding-based transformation step is required that converts the symbolic KGs into numerical vectors before they can be used in ML models. Especially embedding-based approaches, which make use of Deep Learning techniques, have gained increasing popularity over the past few years as these can be applied out-of-the-box and can run efficiently on Graphical Processing Units (GPUs), which are quite commonly available today. Moreover, the largest advantage of embedding-based techniques is that they are typically task-agnostic and as such do not require extensive domain knowledge and/or significant effort, as opposed to feature-based approaches. A further distinction can be made between embedding-based techniques. A first category consists of techniques that learn embeddings either through tensor factorisation or through negative sampling [13,2,24], e.g. TransE [?]. A second category consists of Deep Learning architectures that make use of parameterised transformations, based on information from the neighbourhood of a node that is collected through message passing [17], e.g. Relational Graph Convolutional Networks (R-GCN). The parameters of this transformation are learned through back-propagation in a supervised fashion. A third, and final, category adapts existing NLP techniques, such as Word2Vec [11], to work on graph structures. RDF2Vec belongs to this final category [16].

2.2 RDF2Vec

RDF2Vec is an unsupervised, task-agnostic algorithm that achieves state-of-theart performances on many benchmark datasets [16]. It extends Word2Vec to work on graph structures by first extracting walks that serve as corpus. Each walk can be seen as a sentence of a corpus and each hop within such walks corresponds to a token. Word2Vec will then learn embeddings for each of these tokens in an unsupervised matter by learning to predict either a token based on its context (Continuous Bag of Words), or the context based on a token (Skip-Gram). Over the past few years, several extensions to RDF2Vec have been suggested, which we

4 G. Vandewiele et al.

will discuss subsequently. A good up-to-date overview on how RDF2Vec works, which extensions have been proposed over the last few years, and of applications that make use of RDF2Vec can be found on a website hosted by the original authors².

The number of walks that can be extracted quickly grows, depending on the depth of those walks and the size of the KG. As such, exhaustively extracting every possible walk becomes infeasible rather quickly. As a solution, Cochez et al. [3] proposed several sampling, or biased walking, techniques which enable to only extract a subset of walks that still capture most of the information. Recently, more sampling strategies have been proposed: (i) utilising page transition probabilities [22], (ii) using Metropolis-Hastings sampling [25], or (iii) other forms of prior knowledge [12].

Originally, the RDF2Vec algorithm used random walking and the Weisfeiler-Lehman paradigm to extract the corpus of walks for Word2Vec. However, within the domain of graph-based ML, walking techniques that are more advanced than random sampling have been suggested over the past few years. In addition, it has been shown that the Weisfeiler-Lehman paradigm introduces little to no extra information in the extracted walks. As such, Vandewiele et al. evaluated different walking strategies on several benchmark datasets to show that there is no one-size-fits-all strategy, and that tuning the strategy for the task at hand can result in increased performances [23].

Finally, Portisch et al. [15] applied an order-aware variant of Word2Vec to the corpus extracted by the walking and sampling strategies, which resulted in significant increased predictive performances on multiple benchmark datasets.

3 pyRDF2Vec

In this section, we elaborate upon our pyRDF2Vec package. We first present its architecture, then give an overview of all the extensions available today and finally discuss the different mechanisms implemented to facilitate external contributions.

3.1 Architecture

In Figure 2, an overview of the pyRDF2Vec workflow is provided. Seven main modules are used, which we now discuss subsequently.

 Connector: coordinates the interaction with a local or remote graph. For KGs located on hard disk, pyRDF2Vec uses rdflib to load the graph into memory. If required, walk extraction from remote graphs is also possible through a SPARQL endpoint. Additional connectors can be implemented based on the provided Connector base class.

www.rdf2vec.org

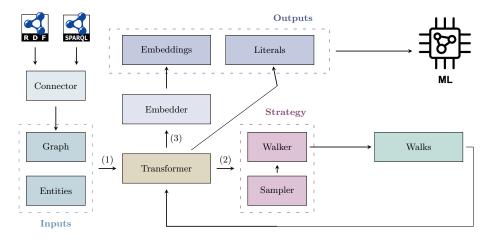


Fig. 2: Workflow of pyRDF2Vec. A Graph and collection of Entities are provided by the user to the Transformer (1), which is instantiated with a list of different strategies consisting of a Walker and Sampler (2). The latter are responsible for extracting walks from the Graph which are, in turn, fed to the embedder to calculate Embeddings (3). In addition, the Transformer also extracts Literals by following paths specified by the user.

- 2. **Graph**: is the internal representation of the KG. It is used to efficiently traverse the graph and to store additional information regarding nodes and edges.
- 3. **Entities**: is the set of nodes within the graph for which we want to generate embeddings. These entities will serve as the starting points for the walk extraction and need to be provided by the user. It should however be noted that in fact all of the entities that appear in these extracted walks will have an associated embedding.
- 4. **Transformer**: the main interface for users that combines all other components
- 5. **Sampler**: prioritises the use of some edges in the graph over others using a weight allocation strategy. The current pyRDF2Vec version implemented each of the sampling techniques described by Cochez et al. [3]. Additional sampling techniques can easily be implemented, according to the provided Sampler base class.
- 6. Walker: responsible for extracting walks from the KG. Different walking strategies, proposed by Vandewiele et al. [23] are incorporated in the current pyRDF2Vec version. New walking strategies can be implemented using the Walker base class.
- 7. **Embedder**: is in charge of transforming the extracted walks into embeddings, based on a trained model. By default, Word2Vec is used within this embedder code to generate these embeddings. A fastText [7] embedder is also

made available in the current pyRDF2Vec version and additional embedding techniques can be added by using the Embedder base class.

It is important to Connector, Sampler, Walker, and Embedder expose interfaces that can be implemented by users. That way, we hope to both facilitate and stimulate further research into these components of the RDF2Vec algorithm.

3.2 Optimizations and extensions

The pyRDF2Vec implementation has several extensions, that speed up walk extraction and which provide information in addition to the embeddings based on walks.

First, the Transformer takes a list of Walker strategies, with optionally associated Sampler strategies, which enables to combine several strategies. This allows for further research into techniques similar to ensembling, where the information obtained from several strategies is combined. This combination can be done either (i) on corpus-level, by concatenating the walks extracted by the different strategies together before feeding them to the Embedder, (ii) on embedding level, where embeddings are learned on the corpora of each strategy individually and then aggregated, or (iii) on prediction level, where the embeddings learned on each corpora are fed to a classifier to make predictions for the downstream task and then aggregated. The combination of different strategies is illustrated in the example code provided in Appendix A.

A second extension in the pyRDF2Vec allows to extract literal information in addition to the embeddings learned, based on the graph structure surrounding entities of interest. To achieve this, the user can specify a set of paths, starting from the nodes provided in Entities, for which literal information can be found. pyRDF2Vec will then traverse these paths and return (i) NaN if the literal cannot be found, (ii) a scalar in case exactly one literal can be found, and (iii) a list of literals in case the path to a literal can be found multiple times. From then on, the user can process this information and concatenate this to the provided embeddings.

pyRDF2Vec enables reverse walking by traversing across incoming edges as opposed to outgoing edges. This is due to the fact that the direction of certain predicates is chosen rather arbitrarily [e.g., (Brussels, isCapitalOf, Belgium) vs. (Belgium, hasCapital, Brussels)]. This also allows for nodes from Entities to be in positions different from the starting position within walks.

Several mechanisms are implemented to speed up the walk extraction: (i) SPARQL requests to find the next hop in walks can be bundled together to reduce overhead introduced by HTTP when a remote KG is used, (ii) multi-threading is enabled to parallelize the extraction of walks, and (iii) caching is implemented to avoid redundant requests.

3.3 CI/CT/CD and Documentation

To facilitate contributions by the open-source community to our code repository, multiple mechanisms have been set up. First, Continuous Integration (CI), through the use of Github Actions³, is implemented which makes sure that the merge of the work of several developers does not impact the release of a project. With each push to one of the branches, several checks are performed, such as checking whether any styling guidelines have been violated. Second, Continuous Delivery (CD) is guaranteed as the main is always supposed to be the stable branch for which the checks performed by the CI pass. Added to that, the use of poetry ⁴ as dependency manager helps to facilitate future releases of pyRDF2Vec to the PyPI platform. Finally, a Continuous Testing (CT) mechanism executes a battery of unit tests, using pytest ⁵, for every push to the code repository. Afterwards, a coverage report is generated. With the help of these continuous methods, pyRDF2Vec has been able to release several new features and fix bugs to increase its stability, popularity, and notoriety.

Having an updated and clear documentation is essential for the proper use of a library and its evolution. Good documentation will make it easier to use and contribute to a library. To improve the clarity of the documentation in Python, mypy ⁶, an optional static type checker, can also be used in addition to PyDoc. While Python is natively a dynamically typed language, the use of such a static type checker requires that consistent types are filled in, which improved documentation. Finally, this documentation generation is done with Sphinx ⁷ and is automatically updated on the online website hosted by Read the Docs, at each commit on the main branch.

4 Package Usage

At the time of writing, pyRDF2Vec has amassed 146 stars on Github and 17,500 downloads according to PePy⁸. An overview of the number of downloads for the latest six months can be found in Figure 3.

pyRDF2Vec has been used in several research projects and practical use cases. As of today, pyRDF2Vec appears in 31 studies published on Google Scholar⁹. We now give a brief overview of these studies. Ontowalk2vec [6] and Owl2Vec* [1] extend pyRDF2Vec to embed concepts by extracting walks from ontology information. Iana et al. [9] showed that applying reasoning to infer extra information

```
3 https://github.com/features/actions
4 https://python-poetry.org/
5 www.pytest.org
6 http://mypy-lang.org/
7 https://www.sphinx-doc.org/
8 https://pepy.tech/project/pyRDF2Vec
9 https://scholar.google.com/scholar?q="pyRDF2Vec"
```

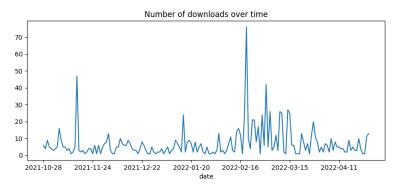


Fig. 3: The number of downloads of the last 180 days of our pyRDF2Vec package.

in the KG before extracting walks results in little to no increased predictive performance. Portisch et al. [14] compared embedding techniques suited for link prediction and suited for data mining on both link prediction and data mining tasks. pyRDF2Vec was used as one of the data mining techniques during evaluation. In [10], pyRDF2Vec among many other embedding techniques, has been compared to non-embedding methods to better understand their semantic capabilities. In Sousa et al. [20] pyRDF2Vec is used to tailor aspect-oriented semantic similarity measures to fit a particular view on biological similarity or relatedness in protein-protein, protein function similarity, protein sequence similarity and phenotype-based gene similarity tasks. Engleitner et al. [5] compare pyRDF2Vec with other embedding techniques for news article tag recommendation. Shi et al. [19,18] use pyRDF2Vec to calculate semantic similarity between concepts in several datasets. Gurbuz et al. [8] evaluate many different techniques, including pyRDF2Vec, for explainable target-disease link prediction. Steenwinckel et al. [21] compare their newly proposed technique, INK, to state-of-the-art techniques such as pyRDF2Vec. Finally, Degraeve et al. [4] qualitatively compare embeddings produced by pyRDF2Vec with embeddings produced by their proposed RR-GCN through a t-SNE plot.

5 Conclusion and Future Work

This paper presented the pyRDF2Vec software package. It reimplements the well-known RDF2Vec algorithm in Python, as this language is several significantly more popular in the data science community than Java, in which RDF2Vec was originally implemented. This reimplementation allows for data scientists to integrate RDF2Vec immediately into their pipeline. In addition to the original algorithm, pyRDF2Vec implements many extensions that have already been published, provides additional information and speeds up the walk extraction. The fact that these extensions are bundled in a single place could facilitate future research. The pyRDF2Vec architecture is set up in such a way, in combination with

automatic styling, testing, and documentation to foster future external contributions. Several research projects and use cases have already used pyRDF2Vec in their experimentation or as a basis for their code, which we discuss in this paper.

Resource Availability Statement: pyRDF2Vec is available under a MIT license on Github¹⁰.

Acknowledgements

Bram Steenwinckel (1SA0219N) is funded by a strategic base research Grant of the Fund for Scientific Research Flanders (FWO).

References

- Chen, J., Hu, P., Jimenez-Ruiz, E., Holter, O.M., Antonyrajah, D., Horrocks, I.: Owl2vec*: Embedding of owl ontologies. Machine Learning 110(7), 1813–1845 (2021)
- 2. Choudhary, S., Luthra, T., Mittal, A., Singh, R.: A survey of knowledge graph embedding and their applications. arXiv preprint arXiv:2107.07842 (2021)
- 3. Cochez, M., Ristoski, P., Ponzetto, S.P., Paulheim, H.: Biased graph walks for rdf graph embeddings. In: Proceedings of the 7th International Conference on Web Intelligence, Mining and Semantics. pp. 1–12 (2017)
- 4. Degraeve, V., Vandewiele, G., Ongenae, F., Van Hoecke, S.: R-gcn: The r could stand for random. arXiv preprint arXiv:2203.02424 (2022)
- Engleitner, N., Kreiner, W., Schwarz, N., Kopetzky, T., Ehrlinger, L.: Knowledge graph embeddings for news article tag recommendation. In: Joint Proceedings of the Semantics co-located events: Poster\&Demo track and Workshop on Ontology-Driven Conceptual Modelling of Digital Twins co-located with Semantics 2021, Amsterdam and Online, September 6-9, 2021. CEUR-WS. org (2021)
- 6. Gkotse, B., Jouvelot, P., Ravotti, F.: Ontology Embeddings with ontowalk2vec: an Application to UI Personalisation. Ph.D. thesis, MINES ParisTech-PSL Research University; CERN-Suisse (2022)
- Grave, E., Bojanowski, P., Gupta, P., Joulin, A., Mikolov, T.: Learning word vectors for 157 languages. In: Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018) (2018)
- 8. Gurbuz, O., Alanis-Lobato, G., Picart-Armada, S., Sun, M., Haslinger, C., Lawless, N., Fernandez-Albert, F.: Knowledge graphs for indication expansion: An explainable target-disease prediction method. Frontiers in genetics 13, 814093–814093 (2022)
- 9. Iana, A., Paulheim, H.: More is not always better: The negative impact of a-box materialization on rdf2vec knowledge graph embeddings. arXiv preprint arXiv:2009.00318 (2020)
- 10. Jain, N., Kalo, J.C., Balke, W.T., Krestel, R.: Do embeddings actually capture knowledge graph semantics? In: European Semantic Web Conference. pp. 143–159. Springer (2021)

 $^{^{10}~{\}tt https://github.com/IBCNServices/pyRDF2Vec}$

- 11. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space (2013)
- 12. Mukherjee, S., Oates, T., Wright, R.: Graph node embeddings using domain-aware biased random walks. arXiv preprint arXiv:1908.02947 (2019)
- 13. Nickel, M., Murphy, K., Tresp, V., Gabrilovich, E.: A review of relational machine learning for knowledge graphs. Proceedings of the IEEE **104**(1), 11–33 (2015)
- 14. Portisch, J., Heist, N., Paulheim, H.: Knowledge graph embedding for data mining vs. knowledge graph embedding for link prediction—two sides of the same coin? Semantic Web (Preprint), 1–24
- 15. Portisch, J., Paulheim, H.: Putting rdf2vec in order. arXiv preprint arXiv:2108.05280 (2021)
- Ristoski, P., Rosati, J., Di Noia, T., De Leone, R., Paulheim, H.: Rdf2vec: Rdf graph embeddings and their applications. Semantic Web 10(4), 721–752 (2019)
- 17. Schlichtkrull, M., Kipf, T.N., Bloem, P., Van Den Berg, R., Titov, I., Welling, M.: Modeling relational data with graph convolutional networks. In: European semantic web conference. pp. 593–607. Springer (2018)
- Shi, Y., Cheng, G., Tran, T.K., Kharlamov, E., Shen, Y.: Efficient computation of semantically cohesive subgraphs for keyword-based knowledge graph exploration. In: Proceedings of the Web Conference 2021. pp. 1410–1421 (2021)
- 19. Shi, Y., Cheng, G., Tran, T.K., Tang, J., Kharlamov, E.: Keyword-based knowledge graph exploration based on quadratic group steiner trees. IJCAI (2021)
- 20. Sousa, R.T., Silva, S., Pesquita, C.: Supervised semantic similarity. bioRxiv (2021)
- Steenwinckel, B., Vandewiele, G., Weyns, M., Agozzino, T., Turck, F.D., Ongenae, F.: Ink: knowledge graph embeddings for node classification. Data Mining and Knowledge Discovery pp. 1–48 (2022)
- Taweel, A.A., Paulheim, H.: Towards exploiting implicit human feedback for improving rdf2vec embeddings. arXiv preprint arXiv:2004.04423 (2020)
- Vandewiele, G., Steenwinckel, B., Bonte, P., Weyns, M., Paulheim, H., Ristoski, P., De Turck, F., Ongenae, F.: Walk extraction strategies for node embeddings with rdf2vec in knowledge graphs. arXiv preprint arXiv:2009.04404 (2020)
- 24. Wang, Q., Mao, Z., Wang, B., Guo, L.: Knowledge graph embedding: A survey of approaches and applications. IEEE Transactions on Knowledge and Data Engineering 29(12), 2724–2743 (2017)
- 25. Zhang, S., Lin, X., Zhang, X.: Discovering dti and ddi by knowledge graph with mhrw and improved neural network. In: 2021 IEEE International Conference on Bioinformatics and Biomedicine (BIBM). pp. 588–593. IEEE (2021)

A Appendix: Example Usage

We now provide a simple code snippet in Listing 1 that demonstrates how a user can generate embeddings for nodes of interest in his/her KG with just a few lines of code.

Listing 1: Example usage of pyRDF2Vec

```
1 # entities is a list of URIs which we want to embed
 _2 entities = [ ... ]
4 # Loads a KG object from hard disk removes triples with
5 # "dl#isMutagenic" as predicate and specifies the paths
 6 # where literals can be found.
 7 dl = "http://dl-learner.org/carcinogenesis"
 8 \text{ kg} = \text{KG}(
       "mutag.owl",
       skip_predicates={dl + "#isMutagenic"},
       literals=[
11
           Γ
12
               dl + "#hasBond",
13
               dl + "#inBond",
14
           ],
           Ε
17
               dl + "#hasAtom",
               dl + "#charge",
18
           ],
19
      ]
20
21 )
23 # Create a Word2Vec embedder that trains for 10 epochs
24 embedder = Word2Vec(workers=1, epochs=10)
26 # Create a Sampler that uses PageRank (damping 0.85)
27 sampler = PageRankSampler(alpha=0.85)
29 # Use HALK strategy to extract all walks of depth 2
30 walker1 = HALKWalker(2, None, n_jobs=4, sampler=None)
32 # Create walker that samples 100 walks per entity
33 walker2 = RandomWalker(2, 100, n_jobs=4, sampler=sampler)
35 # Create our transformer object
36 transformer = RDF2VecTransformer(
       embedder,
37
       walkers=[walker1, walker2]
38
39 )
41 # Extract the embeddings and literals
42 embeddings, literals = transformer.fit_transform(kg, entities)
```