Walk Extraction Strategies for Node Embeddings with RDF2Vec in Knowledge Graphs

Bram Steenwinckel[1], Gilles Vandewiele[1], Pieter Bonte[1], Michael Weyns[1], Heiko Paulheim[2], Petar Ristoski[3], Filip De Turck[1], and Femke Ongenae[1]

[1] IDLab, Ghent University – imec, Belgium
{firstname.lastname}@ugent.be
[2] Data and Web Science Group, University of Mannheim, Germany
heiko@informatik.uni-mannheim.de
[3] eBay Research, United States of America
pristoski@ebay.com

Abstract. As Knowledge Graphs are symbolic constructs, specialized techniques have to be applied in order to make them compatible with data mining techniques. RDF2Vec is an unsupervised technique that can create task-agnostic numerical representations of the nodes in a KG by extending successful language modeling techniques. The original work proposed the Weisfeiler-Lehman kernel to improve the quality of the representations. However, in this work, we show that the Weisfeiler-Lehman kernel does little to improve walk embeddings in the context of a single Knowledge Graph. As an alternative, we examined five alternative strategies to extract information complementary to basic random walks and compare them on several benchmark datasets to show that research within this field is still relevant for node classification tasks.

Keywords: Knowledge graphs · Embeddings · Representation Learning

1 Introduction

As a result of the recent data deluge, the Semantic Web's (SW) Linked Open Data (LOD) initiative is used more and more to interlink various data sources and unite them under a common queryable interface. The product of such a consolidation effort is often called a Knowledge Graph (KG). In addition to unifying information from various sources, KGs are able to enrich classical data formats by explicitly encoding relations between different data points in the form of edges.

Using these KGs to enhance traditional data mining techniques with background knowledge is a relatively recent endeavour [28]. Because KGs are symbolic constructs, they provide the background information in a more graphical representation. Data mining techniques usually require inputs to be presented as numerical feature vectors and are thus unable to process KGs directly. With this in mind, some of the earliest knowledge-enhanced data mining approaches proceeded by extracting custom features from specific and generic relations inside the graph [16]. These approaches produce human-interpretable variables, but they have to be tailored to the task at hand and therefore require extensive effort. As an alternative, techniques can be applied to learn vector representations, called embeddings, for each of the entities inside a graph based on a limited set of global latent features [12,6]. These techniques are task-agnostic, allowing them to be used for different downstream tasks, such as predicting missing links inside a graph or categorizing different nodes [17].

Natural language and graphs often share similarities. Node2Vec [5] and other related techniques were among the first to leverage these similarities, by extending successful language modeling techniques, such as Word2Vec [11], to deal with graph-based data. Their proposed techniques rely on the extraction of sequences of graph vertices, which are then fed as sentences to language models. Similarly, work on (deep) graph kernels also relies on language modeling to learn the latent representations of graph substructures [24,29,8]. RDF2Vec is a technique that builds on the progress made by these previous two types of techniques by adapting random walks and the Weisfeiler-Lehman (WL) subtree kernel to directed graphs with labeled edges, i.e. KGs [18].

In this work, we show that this WL kernel, while effective for measuring similarities between nodes or when working with regular graphs, offers little improvements in the context of a single KG with respect to walk embeddings. In response to this observation, we broadened our search and examined alternative walk strategies for RDF data. Some were designed for regular graphs but in this paper, we show their applicability on KGs and compare them against the random and WL strategies on different benchmark datasets.

The remainder of this paper is structured as follows. In Section 2, background information is provided on KGs and walk embeddings. Next, Section 3 discusses five possible alternative walk strategies, including pseudo-code listings for each algorithm. Section 4 then describes the datasets used to evaluate these alternative strategies and lists the corresponding results. These results are subsequently discussed in Section 5. Finally, in Section 6 we conclude this work with a general reflection.

2 Background

A knowledge graph is a multi-relational directed graph, $\mathbb{G} = (\mathbb{V}, \mathbb{E}, \ell)$, where $\mathbb{V}$ are the vertices or entities in our graph, $\mathbb{E}$ the edges or predicates and ℓ a labeling function that maps each vertex or edge onto its corresponding label. We can simplify any further analysis by applying a transformation to the knowledge graph which removes the multi-relational aspect, as done by de Vries et al. [26].

Machine learning algorithms cannot work directly on this graph-based data, as they require numerical vectors as input. RDF2Vec is an unsupervised, task-agnostic approach that solves this problem by transforming the information of the nodes in the graph into numerical data, which are called latent representations or embeddings [18]. The goal is to capture as much of the semantics as possible in the numerical representation, e.g. entities that are semantically related should be close to each other in the embedded space. RDF2Vec builds on word embedding techniques, which have shown great success in the domain of natural language processing. These word embedding techniques take a corpus of sentences as input, and learn a latent representation for each of the unique words within the corpus. Learning this latent representation can be done, for example, by learning to predict a word based on its context (continuous bag-of-words) or predicting the context based on a target word (skip-gram) [4,11].

In the context of (knowledge) graphs, we can construct an input corpus by extracting walks. A walk is a sequence of vertices that can be found in the graph by traversing the directed links. We can notate a walk of length n and the labeling function to create a sentence as follows:

$$v_0 \rightarrow v_1 \rightarrow \ldots \rightarrow v_{n-1} \tag{1}$$

$$\ell(v_0) \rightarrow \ell(v_1) \rightarrow \ldots \rightarrow \ell(v_{n-1}) \tag{2}$$

The most straightforward strategy to extract walks is by doing a breadth-first traversal of the graph starting from the nodes of interest. Since the total number of walks that can be extracted grows exponentially in function of the depth, sampling can be applied after each iteration of breadth-first traversal. This sampling can either be guided by some metric, resulting in a collection of biased walks [2], or can be performed at random which results in random walks.

2.1 Weisfeiler-Lehman kernel for Knowledge Graphs

Ristoski et al. proposed to use the WL kernel in order to relabel nodes as an alternative to extracting random walks [18]. The WL kernel was proposed as an extension to the labelling function and is originally an algorithm to test whether two graphs were isomorphic in polynomial time [27]. The intuition behind the algorithm was to assign new labels to each of the nodes, where each of the newly assigned labels captured the information of an entire subgraph up to a certain depth. This algorithm was later adapted to serve as a kernel, or similarity measure, between graphs [25,22], by counting the number of WL labels two graphs had in common. However, we argue that the WL kernel provides no additional information with respect to entity representations when extracting a fixed number of random walks from a knowledge graph. Entities in RDF are represented by Uniform Resource Identifiers (URI), which need to be unique[1]. As such:

$$\ell(x) = \ell(y) \iff x = y \tag{3}$$

[1] `https://www.w3.org/DesignIssues/Axioms.html`

Due to this property, WL relabeling, when applied on RDF data, is nothing more than a bijection from the hops in random walks to the hops in the walks obtained through WL relabeling. This relabelling task compresses a subtree into a new string label. There are no situations where a certain fixed label, present in the random walks, is mapped onto different labels in the WL walks or vice versa, multiple labels within the random walks that get mapped onto the same single label in the WL walks. As Word2Vec simply uses a bag-of-words representation internally, it does not make any difference if the original labels or the compressed WL labels were used. This means that WL relabeling does not add any useful additional information in the context of RDF data.

3 Custom walk extraction strategies

Five different strategies were adapted to work with RDF data:
Community hops: As opposed to iteratively extending the walk with neighbors of a vertex, we could, with a certain probability, allow for teleportation to a node that has properties similar to a certain neighbor. The idea of introducing community hops is to capture implicit relations between nodes that are not explicitly modeled in the KG, and to allow for including related pieces of knowledge in the walks which are otherwise out of reach. In order to group nodes with similar properties together, unsupervised community detection can be applied [3]. In this work, we use the Louvain method [1] due to its excellent trade-off between speed and clustering quality. We provide pseudo-code for this strategy in Algorithm 1. We will refer to this strategy as *community*.

Alg. 1: COMMUNITY_WALK($\mathbb{G}$, v, depth, p, hop_prob)

```
# List of comm and vertex->comm dict
com, com_map = COM_DETECTION($\mathbb{G}$)
walks = { (v) }
for $d$ in RANGE($depth$):
    new = set()
    for walk in walks:
        for n in GET_NEIGHBORS($\mathbb{G}$, v):
            # Sample neighborhood
            if RANDOM() < p:
                new.ADD(walk + (n))

            # Hop to community
            if RANDOM() < hop_prob:
                c_n = com[com_map[n]]
                hop = CHOICE(c_n)
                new.ADD(walk + (hop))
    walks = new
return walks
```

Hierarchical random walk (HALK): The frequency of entities in a knowledge graph often follows a long-tailed distribution, similar to natural language. Entities rarely occurring often carry little information, and increase the number of hops between the root and potentially more interesting entities. As such, the removal of rare entities from the random walks can increase the quality of the generated embeddings while decreasing the memory usage [20]. Pseudo-code for this strategy is provided in Algorithm 2. We will refer to this strategy as *HALK*.
N-grams: Another approach that defines a one-to-many mapping is relabeling

n-grams in the random walks. The intuition behind this is that the predecessor nodes two different walks have in common can be different. Additionally, we can inject wildcards into the walk before relabeling n-grams [23]. This injection allows subsequences with small differences to be mapped onto the same label. Pseudo-code for this strategy is provided in Algorithm 3. We will refer to this strategy as *n-gram*.

Alg. 2: HALK(walks, thresholds)

```
# Count nr. of walks a hop occurs
counts = { }
for i in RANGE(|walk|):
    for hop in walks[i]:
        if hop not in counts:
            counts[hop] = [i]
        else:
            counts[hop].APPEND(i)

# Skip rare hops
halk_walks = [ ]
for thresh in thresholds:
    for walk in walks:
        new = [ walk[0] ]
        for hop in walk[1:]:
            if $\frac{|counts[hop]|}{|walks|} \geq$ thresh:
                new.APPEND(hop)
        halk_walks.APPEND(new)
return halk_walks
```

Alg. 3: NGRAM(walks, n, n_wild)

```
# Introduce wildcards in the walks
extended_walks = walks
for walk in walks:
    idx = RANGE(1, |walk|)
    combs = COMBINATIONS(idx, n_wild)
    for comb in combs:
        new = walk
        for i in comb:
            new[i] = '*'
        extended_walks.APPEND(new)

# Relabel ngrams in the walk
ngram_walks = [ ]
map = { }
for walk in extended_walks:
    new = walk[:n]
    for i in RANGE(n, |walk|):
        ngram = walk[i-n:i]
        if ngram not in map:
            map[ngram] = |map|
        new.APPEND(map[ngram])
    ngram_walks.APPEND(new)
return ngram_walks
```

Alg. 4: ANONYMIZE(walks)

```
anon_walks = [ ]
for walk in walks:
    new = [ walk[0] ]
    for hop in walk[1:]:
        new.APPEND(walk.INDEX(hop))
    anon_walks.APPEND(new)
return anon_walks
```

Alg. 5: WALKLETS(walks)

```
walklets = set()
for walk in walks:
    for i in RANGE(1, |walk|):
        walklets.ADD((walk[0], walk[i]))
return walklets
```

Anonymous walks: Random walks can be anonymized by transforming the label information into positional information. More formally, a walk $w = v_0 \rightarrow v_1 \rightarrow \ldots \rightarrow v_n$, is transformed into $f(v_0) \rightarrow f(v_1) \rightarrow \ldots \rightarrow f(v_n)$ with $f(v_i) = \min(\{i \mid w[i] = v_i\})$, which corresponds to the first index where v_i can be found in the walk w [7]. Local graph structures often bear enough information for encoding and reconstructing a graph, even when anonymizing the node labels. Ignoring the labels, on the other hand, allows for computationally efficient generation of the walks. We present pseudo-code for this transformation in Algorithm 4. We will refer to this strategy as *anonymous*.

Walklets: Walks can be transformed into walklets, which are walks of length two consisting of the root of the original walk and one of the hops. Provided a walk $w = v_0 \rightarrow v_1 \rightarrow \ldots \rightarrow v_n$, we can construct sets of walklets $\{(v_0, v_i) \mid 1 \leq i \leq n\}$ [14]. While standard RDF2vec does not consider the distance between two nodes in a walk, walklets are explicitly created for different scales. Hence, they allow for such a distinction between a direct neighbor and a node which is further away. Pseudocode for this approach is provided in Algorithm 5. We will refer to this strategy as *walklet*.

4 Results

To evaluate the impact of custom walking strategies, we measure the predictive performance on different datasets and various tasks.

4.1 Datasets

Four datasets, each describing knowledge graphs, serve as benchmarks for node classification and are available from a public repository set up by Ristoski et al. [15]. The names of these benchmark datasets are AIFB, MUTAG, BGS and AM. For each of these data sets, we remove triples with specific predicates that would leak the target from our knowledge graph, as provided by the original authors. Moreover, a predefined split into train and test set, with the corresponding ground truth, is provided by the authors, which we used in our experiments. Three citation networks [21] were converted to knowledge graphs. These citation networks describe scientific papers and the goal is to categorize each of the papers into the correct research domain. Finally, the English version of the 2016-10 DBpedia dataset [10] was used to obtain embeddings for multiple different downstream tasks: 5 different classification tasks (AAUP, Cities, Forbes, Albums and Movies), document similarity and entity relatedness. More details on each of these tasks can be found in the original RDF2Vec paper [18].

4.2 Setup

For each of the entities in all of the datasets, walks of depth 4 are extracted. Only for the entities of DBpedia, the maximum number of walks per entity is limited to 500. These walks are then provided to a Word2Vec model to create 500-dimensional embeddings. Skip-Gram is used with a window size equal to 5 and the maximum number of iterations is set to 10 with negative sampling set to 25. These configurations are identical to the original RDF2Vec study. For node classification tasks, embeddings are fed to a Support Vector Machine (SVM) classifier with Radial Basis Function (RBF) kernel. The regularization strength of the SVM is tuned to be one of 0.001, 0.01, 0.1, 1.0, 10.0, 100.0, 1000.0. For tasks other than node classification, an evaluation framework is used [13]. For document similarity, we measure the Pearson's linear correlation coefficient, Spearman's rank correlation and their harmonic mean. For entity relatedness, we measure the Kendall's rank correlation coefficient. For the benchmark datasets and citation networks, a pre-defined train/test split is used and experiments are repeated 5 times in order to report a corresponding standard deviation. For the tasks involving DBpedia data, 10-fold cross-validation is used and experiments are only repeated once for timing reasons. Moreover, the community strategy was excluded from the DBpedia experiments, as it cannot be efficiently performed on large knowledge graphs. For each of the walking strategies, we used the following configurations:

- The *random*, *anonymous* and *walklet* walkers are parameter-free.
- For the *n-gram* walker, we tune $n \in [1, 2, 3]$ and the number of introduced wildcards to be either 0 or 1.
- For the *community* walker, we set the resolution of the Louvain algorithm to 1.0 [9] and the probability to teleport to a community node to 10%.
- For the *WL* walker, we use the original algorithm used by Ristoski et al. [18]. We set the number of iterations of the WL kernel to 4 and extract walks of fixed depth for each of the iterations, including zero.
- For the *HALK* strategy, we extract sets of walks using different thresholds: $[0.0, 0.1, 0.05, 0.01, 0.005, 0.001, 0.0005, 0.0001]$.

4.3 Evaluation Results

The results for the various classification tasks are provided in Table 1. The results for the document similarity and entity relatedness task are provided in Table 2.

Table 1: The accuracy scores obtained by various techniques on different datasets.

	Random	**WL**	**Walkets**	**Anonym.**	**HALK**	**N-Gram**	**Commun.**
AIFB	86.11±2.48	**91.67±0.00**	63.89±0.00	41.67±0.00	86.11±0.00	88.33±1.11	88.89±1.76
MUTAG	76.76±0.59	75.00±2.46	72.06±0.00	66.18±0.00	75.00±0.00	**77.65±2.85**	74.71±3.99
BGS	79.31±0.00	80.69±6.40	65.52±0.00	65.52±0.00	80.00±4.57	83.45±4.02	**84.14±3.52**
AM	75.56±2.70	82.53±1.68	47.47±0.00	34.85±0.00	80.10±0.88	**84.44±2.22**	73.94±2.70
CORA	**77.20±0.00**	74.32±1.56	58.20±0.00	14.30±0.00	76.62±0.36	76.46±0.78	67.92±1.22
CITESEER	64.68±1.58	64.02±1.46	38.40±0.00	16.00±0.00	**66.90±0.00**	65.38±1.22	58.66±0.50
PUBMED	75.66±1.36	73.70±2.87	68.30±0.00	24.20±0.00	75.56±0.08	**78.48±0.35**	54.64±2.40
DB:AAUP	67.94	**69.88**	69.27	54.73	60.08	66.96	/
DB:Cities	79.07	79.12	79.08	55.34	73.34	**79.79**	/
DB:Forbes	63.73	**64.60**	62.28	55.16	60.98	63.65	/
DB:Albums	75.24	79.31	**79.99**	54.45	66.89	79.38	/
DB:Movies	80.06	**80.48**	78.89	59.40	68.11	78.84	/

Table 2: Document similarity and entity relatedness results

Strategy	Pears. r	Spear. ρ	μ
Random	**0.578**	0.390	0.466
Anonymous	0.321	0.324	0.322
Walklets	0.528	0.372	0.437
HALK	0.455	0.376	0.412
N-grams	0.551	0.353	0.431
WL	0.576	**0.412**	**0.480**

Strategy	Kendall τ
Random	**0.523**
Anonymous	0.243
Walklets	0.520
HALK	0.424
N-grams	0.483
WL	0.516

5 Discussion

Based on the provided results, several observations can be made. The random and WL strategies were already evaluated in the original RDF2Vec study [18]. As such, the results reported in this study can be seen as a reproduction of those results. It is important to note here that the only reason why the results obtained by the WL and random strategy differ in this and the original work, is because originally the walks are extracted after each iteration of the WL relabelling algorithm. This results in k times as many walks, with k the number of iterations in the relabelling algorithm. If walks from only one of the iterations would be used, the results would be identical to those of the random strategy. We hypothesize that this is due to more weight being given, internally in Word2Vec, to the entities where many walks can be extracted from. While the original WL and random strategies result in very strong performances, especially on all downstream tasks of DBpedia, they are in this evaluation often outperformed by custom strategies proposed in this work.

The results indicate that there is currently no one-size-fits-all walking strategy for all tasks and datasets. It seems that the n-gram strategy results in the best predictive performances on average for node classification tasks. The average rank of the n-gram strategy on the four node classification and three citation network datasets, using all seven techniques, is equal to 1.86, followed by 3 of the HALK strategy and 3.07 of both the random and WL strategy. An average rank of 1 would mean that the technique outperforms all others on each dataset. The average rank of the n-gram strategy on all the node classification tasks, excluding the community strategy, is equal to 2.08, followed by 2.375, 2.875 and 3.67 by random, WL and HALK respectively. The performance of the community strategy varies a lot. On some datasets, such as AIFB and BGS, its performance is among the best while it performs a lot worse than random walks on others. This is due to the fact that the quality of the walks is highly dependent on the quality of the community detection. If the groups of nodes, clustered by the community detection, do not align well with the downstream task, the performance worsens.

Some limitations of this study can be identified. Firstly, no comparisons with other techniques are performed. RDF2Vec is an unsupervised and task-agnostic technique. As such, comparisons with supervised techniques, specifically tailored to the task, such as Relational Graph Convolutional Networks [19] can hold unfair results. In the original work of Ristoski et al. [18] it was already shown that RDF2Vec outperforms other unsupervised variants such as TransE, TransH and TransR. Second, a fixed depth and fixed hyper-parameters for the Word2Vec model were used within this study. While tuning these hyper-parameters could possibly result in increased predictive performances, it should be noted that the number of hyper-parameters and the range of a Word2Vec model are very large and that the time required to generate the embedding is significant.

6 Conclusion and future work

In this work, five walk strategies that can serve as an alternative to the basic random walk approach are evaluated as a response to the observation that the WL kernel offers little improvement in the context of a single KG. Results indicate that there is no *one-size-fits-all* strategy for all datasets and tasks, and that tuning the strategy to use, as opposed to simple using the random walk approach, can result in increased predictive performances.

There are several future directions that we deem interesting. First, a formal proof is required to show the non-applicability of the Weisfeiler-Lehman kernel on KGs. Second, it would be interesting to study what the impact on the performance is when the strategies are combined with different biased walk strategies. Third, all of the strategies evaluated in this work are unsupervised, but supervised approaches could be evaluated that sacrifice generality to gain predictive performance. At last, as already mentioned, each of the walking strategies are complementary to each other. Combining different strategies together will potentially result in increased predictive performances.

Code availability:
We provide a Python implementation of RDF2Vec which can be combined with any of the walking strategies discussed in this work[2]. Moreover, we provide all code required to reproduce the reported results[3].

Acknowledgements:
Gilles Vandewiele (1S31417N), Bram Steenwinckel (1SA0219N) and Michael Weyns (1SD8821N) are funded by a strategic base research grant of the Fund for Scientific Research Flanders (FWO). Pieter Bonte (1266521N) is funded by a postdoctoral fellowship of the FWO.

References

1. Blondel, V.D., et al.: Fast unfolding of communities in large networks. Journal of statistical mechanics: theory and experiment **2008**(10), P10008 (2008)
2. Cochez, M., et al.: Biased graph walks for rdf graph embeddings. In: Proceedings of the 7th International Conference on Web Intelligence, Mining and Semantics. pp. 1–12 (2017)
3. Fortunato, S.: Community detection in graphs. Physics reports **486** (2010)
4. Goldberg, Y., Levy, O.: word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method. arXiv preprint arXiv:1402.3722 (2014)
5. Grover, A., Leskovec, J.: node2vec: Scalable feature learning for networks. In: Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 855–864 (2016)
6. Hamilton, W.L., et al.: Representation Learning on Graphs: Methods and Applications. Preprint of article to appear in the IEEE Data Engineering Bulletin (2017)
7. Ivanov, S., Burnaev, E.: Anonymous walk embeddings. arXiv preprint arXiv:1805.11921 (2018)
8. Kriege, N.M., et al.: A survey on graph kernels. Applied Network Science **5**(1), 1–42 (2020)
9. Lambiotte, R., et al.: Laplacian dynamics and multiscale modular structure in networks. arXiv preprint arXiv:0812.1770 (2008)

[2] `github.com/IBCNServices/pyRDF2Vec`
[3] `github.com/GillesVandewiele/WalkExperiments`

10. Lehmann, J., et al.: Dbpedia–a large-scale, multilingual knowledge base extracted from wikipedia. Semantic Web **6**(2), 167–195 (2015)
11. Mikolov, T., et al.: Distributed representations of words and phrases and their compositionality. In: Advances in neural information processing systems (2013)
12. Nickel, M., et al.: A review of relational machine learning for knowledge graphs. Proceedings of the IEEE **104**(1), 11–33 (2015)
13. Pellegrino, M.A., et al.: Geval: a modular and extensible evaluation framework for graph embedding techniques. In: ESWC proceedings. Springer (2020)
14. Perozzi, B., et al.: Don't walk, skip! online learning of multi-scale network embeddings. In: Proceedings of the 2017 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining 2017. pp. 258–265 (2017)
15. Ristoski, P., et al.: A collection of benchmark datasets for systematic evaluations of machine learning on the semantic web. In: International Semantic Web Conference. pp. 186–194. Springer (2016)
16. Ristoski, P., Paulheim, H.: A comparison of propositionalization strategies for creating features from linked open data. Linked Data for Knowledge Discovery (2014)
17. Ristoski, P., Paulheim, H.: Semantic web in data mining and knowledge discovery: A comprehensive survey. Journal of Web Semantics **36**, 1–22 (2016)
18. Ristoski, P., et al.: Rdf2vec: Rdf graph embeddings and their applications. Semantic Web **10**(4), 721–752 (2019)
19. Schlichtkrull, M., et al.: Modeling relational data with graph convolutional networks. In: European Semantic Web Conference. pp. 593–607. Springer (2018)
20. Schlötterer, J., et al.: Investigating extensions to random walk based graph embedding. In: 2019 IEEE International Conference on Cognitive Computing (ICCC). pp. 81–89. IEEE (2019)
21. Sen, P., et al.: Collective classification in network data. AI magazine **29** (2008)
22. Shervashidze, N., et al.: Weisfeiler-lehman graph kernels. Journal of Machine Learning Research **12**(Sep), 2539–2561 (2011)
23. Vandewiele, G., et al.: Inducing a decision tree with discriminative paths to classify entities in a knowledge graph. In: SEPDA2019, the 4th International Workshop on Semantics-Powered Data Mining and Analytics. pp. 1–6 (2019)
24. Vishwanathan, S.V.N., et al.: Graph kernels. Journal of Machine Learning Research **11**(Apr), 1201–1242 (2010)
25. de Vries, G.K.: A fast approximation of the weisfeiler-lehman graph kernel for rdf data. In: Joint European Conference on Machine Learning and Knowledge Discovery in Databases. pp. 606–621. Springer (2013)
26. de Vries, G.K.D., de Rooij, S.: Substructure counting graph kernels for machine learning from rdf data. Web Semantics: Science, Services and Agents on the World Wide Web **35**, 71–84 (2015)
27. Weisfeiler, B., Lehman, A.A.: A reduction of a graph to a canonical form and an algebra arising during this reduction. Nauchno-Technicheskaya Informatsia (1968)
28. Wilcke, X., et al.: The Knowledge Graph as the Default Data Model for Machine Learning. Data Science **1** (2017). https://doi.org/10.3233/DS-170007
29. Yanardag, P., Vishwanathan, S.: Deep graph kernels. In: Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. pp. 1365–1374 (2015)