RDF2Vec: RDF Graph Embeddings for Data Mining

Petar Ristoski, Heiko Paulheim

Data and Web Science Group, University of Mannheim, Germany
{petar.ristoski,heiko}@informatik.uni-mannheim.de

Abstract. Linked Open Data has been recognized as a valuable source for background information in data mining. However, most data mining tools require features in propositional form, i.e., a vector of nominal or numerical features associated with an instance, while Linked Open Data sources are graphs by nature. In this paper, we present RDF2Vec, an approach that uses language modeling approaches for unsupervised feature extraction from sequences of words, and adapts them to RDF graphs. We generate sequences by leveraging local information from graph substructures, harvested by Weisfeiler-Lehman Subtree RDF Graph Kernels and graph walks, and learn latent numerical representations of entities in RDF graphs. Our evaluation shows that such vector representations outperform existing techniques for the propositionalization of RDF graphs on a variety of different predictive machine learning tasks, and that feature vector representations of general knowledge graphs such as DBpedia and Wikidata can be easily reused for different tasks.

Keywords: Graph Embeddings, Linked Open Data, Data Mining

1 Introduction

Linked Open Data (LOD) [29] has been recognized as a valuable source of background knowledge in many data mining tasks and knowledge discovery in general [25]. Augmenting a dataset with features taken from Linked Open Data can, in many cases, improve the results of a data mining problem at hand, while externalizing the cost of maintaining that background knowledge [18].

Most data mining algorithms work with a propositional *feature vector* representation of the data, i.e., each instance is represented as a vector of features $\langle f_1, f_2, ..., f_n \rangle$, where the features are either binary (i.e., $f_i \in \{true, false\}$), numerical (i.e., $f_i \in \mathbb{R}$), or nominal (i.e., $f_i \in S$, where S is a finite set of symbols). LOD, however, comes in the form of *graphs*, connecting resources with types and relations, backed by a schema or ontology.

Thus, for accessing LOD with existing data mining tools, transformations have to be performed, which create propositional features from the graphs in LOD, i.e., a process called *propositionalization* [10]. Usually, binary features (e.g., `true` if a type or relation exists, `false` otherwise) or numerical features

(e.g., counting the number of relations of a certain type) are used [20, 24]. Other variants, e.g., counting different graph sub-structures are possible [34].

In this work, we adapt language modeling approaches for latent representation of entities in RDF graphs. To do so, we first convert the graph into a set of sequences of entities using two different approaches, i.e., graph walks and Weisfeiler-Lehman Subtree RDF graph kernels. In the second step, we use those sequences to train a neural language model, which estimates the likelihood of a sequence of entities appearing in a graph. Once the training is finished, each entity in the graph is represented as a vector of latent numerical features.

Projecting such latent representations of entities into a lower dimensional feature space shows that semantically similar entities appear closer to each other. We use several RDF graphs and data mining datasets to show that such latent representation of entities have high relevance for different data mining tasks.

The generation of the entities' vectors is task and dataset independent, i.e., once the vectors are generated, they can be used for any given task and any arbitrary algorithm, e.g., SVM, Naive Bayes, Random Forests, Neural Networks, KNN, etc. Also, since all entities are represented in a low dimensional feature space, building machine learning models becomes more efficient. To foster the reuse of the created feature sets, we provide the vector representations of DBpedia and Wikidata entities as ready-to-use files for download.

The rest of this paper is structured as follows. In Section 2, we give an overview of related work. In Section 3, we introduce our approach, followed by an evaluation in section Section 4. We conclude with a summary and an outlook on future work.

2 Related Work

In the recent past, a few approaches for generating data mining features from Linked Open Data have been proposed. Many of those approaches are supervised, i.e., they let the user formulate SPARQL queries, and a fully automatic feature generation is not possible. LiDDM [8] allows the users to declare SPARQL queries for retrieving features from LOD that can be used in different machine learning techniques. Similarly, Cheng et al. [3] propose an approach feature generation after which requires the user to specify SPARQL queries. A similar approach has been used in the RapidMiner[1] semweb plugin [9], which preprocesses RDF data in a way that it can be further processed directly in RapidMiner. Mynarz et al. [16] have considered using user specified SPARQL queries in combination with SPARQL aggregates.

FeGeLOD [20] and its successor, the *RapidMiner Linked Open Data Extension* [23], have been the first fully automatic unsupervised approach for enriching data with features that are derived from LOD. The approach uses six different unsupervised feature generation strategies, exploring specific or generic relations. It has been shown that such feature generation strategies can be used in many data mining tasks [21, 23].

[1] http://www.rapidminer.com/

A similar problem is handled by *Kernel functions*, which compute the distance between two data instances by counting common substructures in the graphs of the instances, i.e. walks, paths and trees. In the past, many graph kernels have been proposed that are tailored towards specific applications [7], or towards specific semantic representations [5]. Only a few approaches are general enough to be applied on any given RDF data, regardless the data mining task. Lösch et al. [12] introduce two general RDF graph kernels, based on intersection graphs and intersection trees. Later, the intersection tree path kernel was simplified by Vries et al. [33]. In another work, Vries et al. [32, 34] introduce an approximation of the state-of-the-art Weisfeiler-Lehman graph kernel algorithm aimed at improving the computation time of the kernel when applied to RDF. Furthermore, the kernel implementation allows for explicit calculation of the instances' feature vectors, instead of pairwise similarities.

Our work is closely related to the approaches DeepWalk [22] and Deep Graph Kernels [35]. DeepWalk uses language modeling approaches to learn social representations of vertices of graphs by modeling short random-walks on large social graphs, like BlogCatalog, Flickr, and YouTube. The Deep Graph Kernel approach extends the DeepWalk approach, by modeling graph substructures, like graphlets, instead of random walks. The approach we propose in this paper differs from these two approaches in several aspects. First, we adapt the language modeling approaches on directed labeled RDF graphs, compared to the undirected graphs used in the approaches. Second, we show that task-independent entity vectors can be generated on large-scale knowledge graphs, which later can be reused on variety of machine learning tasks on different datasets.

3 Approach

In our approach, we adapt neural language models for RDF graph embeddings. Such approaches take advantage of the word order in text documents, explicitly modeling the assumption that closer words in the word sequence are statistically more dependent. In the case of RDF graphs, we consider entities and relations between entities instead of word sequences. Thus, in order to apply such approaches on RDF graph data, we first have to transform the graph data into sequences of entities, which can be considered as sentences. Using those sentences, we can train the same neural language models to represent each entity in the RDF graph as a vector of numerical values in a latent feature space.

3.1 RDF Graph Sub-Structures Extraction

We propose two general approaches for converting graphs into a set of sequences of entities, i.e, graph walks and Weisfeiler-Lehman Subtree RDF Graph Kernels.

Definition 1 *An RDF graph is a graph G = (V, E), where V is a set of vertices, and E is a set of directed edges.*

The objective of the conversion functions is for each vertex $v \in V$ to generate a set of sequences S_v, where the first token of each sequence $s \in S_v$ is the

vertex v followed by a sequence of tokens, which might be edges, vertices, or any substructure extracted from the RDF graph, in an order that reflects the relations between the vertex v and the rest of the tokens, as well as among those tokens.

Graph Walks In this approach, for a given graph $G = (V, E)$, for each vertex $v \in V$ we generate all graph walks P_v of depth d rooted in the vertex v. To generate the walks, we use the breadth-first algorithm. In the first iteration, the algorithm generates paths by exploring the direct outgoing edges of the root node v_r. The paths generated after the first iteration will have the following pattern $v_r \rightarrow e_{1i}$, where $i \in E(v_r)$. In the second iteration, for each of the previously explored edges the algorithm visits the connected vertices. The paths generated after the second iteration will follow the following patter $v_r \rightarrow e_{1i} \rightarrow v_{1i}$. The algorithm continues until d iterations are reached. The final set of sequences for the given graph G is the union of the sequences of all the vertices $\bigcup_{v \in V} P_v$.

Weisfeiler-Lehman Subtree RDF Graph Kernels In this approach, we use the subtree RDF adaptation of the Weisfeiler-Lehman algorithm presented in [32, 34]. The Weisfeiler-Lehman Subtree graph kernel is a state-of-the-art, efficient kernel for graph comparison [30]. The kernel computes the number of sub-trees shared between two (or more) graphs by using the Weisfeiler-Lehman test of graph isomorphism. This algorithm creates labels representing subtrees in h iterations.

There are two main modifications of the original Weisfeiler-Lehman graph kernel algorithm in order to be applicable on RDF graphs [34]. First, the RDF graphs have directed edges, which is reflected in the fact that the neighborhood of a vertex v contains only the vertices reachable via outgoing edges. Second, as mentioned in the original algorithm, labels from two iterations can potentially be different while still representing the same subtree. To make sure that this does not happen, the authors in [34] have added tracking of the neighboring labels in the previous iteration, via the multiset of the previous iteration. If the multiset of the current iteration is identical to that of the previous iteration, the label of the previous iteration is reused.

The procedure of converting the RDF graph to a set of sequences of tokens goes as follows: (i) for a given graph $G = (V, E)$, we define the Weisfeiler-Lehman algorithm parameters, i.e., the number of iterations h and the vertex subgraph depth d, which defines the subgraph in which the subtrees will be counted for the given vertex; (ii) after each iteration, for each vertex $v \in V$ of the original graph G, we extract all the paths of depth d within the subgraph of the vertex v on the relabeled graph. We set the original label of the vertex v as the starting token of each path, which is then considered as a sequence of tokens. The sequences after the first iteration will have the following pattern $v_r \rightarrow T_1 \rightarrow T_1 \ldots T_d$, where T_d is a subtree that appears on depth d in the vertex's subgraph; (iii) we repeat step 2 until the maximum iterations h are reached. (iv) The final set of sequences is the union of the sequences of all the vertices in each iteration $\bigcup_{i=1}^{h} \bigcup_{v \in V} P_v$.

3.2 Neural Language Models – word2vec

Neural language models have been developed in the NLP field as an alternative to represent texts as a bag of words, and hence, a binary feature vector, where each vector index represents one word. While such approaches are simple and robust, they suffer from several drawbacks, e.g., high dimensionality and severe data sparsity, which limits the performances of such techniques. To overcome such limitations, neural language models have been proposed, inducing low-dimensional, distributed embeddings of words by means of neural networks. The goal of such approaches is to estimate the likelihood of a specific sequence of words appearing in a corpus, explicitly modeling the assumption that closer words in the word sequence are statistically more dependent.

While some of the initially proposed approaches suffered from inefficient training of the neural network models, with the recent advancements in the field several efficient approaches has been proposed. One of the most popular and widely used is the word2vec neural language model [13, 14]. Word2vec is a particularly computationally-efficient two-layer neural net model for learning word embeddings from raw text. There are two different algorithms, the Continuous Bag-of-Words model (CBOW) and the Skip-Gram model.

Continuous Bag-of-Words Model The CBOW model predicts target words from context words within a given window. The model architecture is shown in Fig. 1a. The input layer is comprised from all the surrounding words for which the input vectors are retrieved from the input weight matrix, averaged, and projected in the projection layer. Then, using the weights from the output weight matrix, a score for each word in the vocabulary is computed, which is the probability of the word being a target word. Formally, given a sequence of training words $w_1, w_2, w_3, ..., w_T$, and a context window c, the objective of the CBOW model is to maximize the average log probability:

$$\frac{1}{T}\sum_{t=1}^{T} logp(w_t|w_{t-c}...w_{t+c}), \tag{1}$$

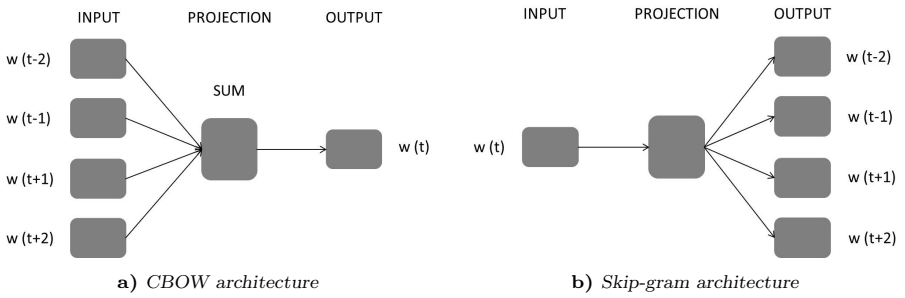
where the probability $p(w_t|w_{t-c}...w_{t+c})$ is calculated using the softmax function:

$$p(w_t|w_{t-c}...w_{t+c}) = \frac{exp(\bar{v}^T v'_{w_t})}{\sum_{w=1}^{V} exp(\bar{v}^T v'_w)}, \tag{2}$$

where v'_w is the output vector of the word w, V is the complete vocabulary of words, and $\bar{v}$ is the averaged input vector of all the context words:

$$\bar{v} = \frac{1}{2c}\sum_{-c\leq j\leq c, j\neq 0} v_{w_{t+j}} \tag{3}$$

Skip-Gram Model The skip-gram model does the inverse of the CBOW model and tries to predict the context words from the target words (Fig. 1b). More formally, given a sequence of training words $w_1, w_2, w_3, ..., w_T$, and a context



a) *CBOW architecture* **b)** *Skip-gram architecture*

Fig. 1: Architecture of the CBOW and Skip-gram model.

window c, the objective of the skip-gram model is to maximize the following average log probability:

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq c, j\neq 0} logp(w_{t+j}|w_t), \tag{4}$$

where the probability $p(w_{t+j}|w_t)$ is calculated using the softmax function:

$$p(w_o|w_i) = \frac{exp(v'^T_{wo} v_{wi})}{\sum_{w=1}^{V} exp(v'^T_w v_{wi})}, \tag{5}$$

where v_w and v'_w are the input and the output vector of the word w, and V is the complete vocabulary of words.

In both cases, calculating the softmax function is computationally inefficient, as the cost for computing is proportional to the size of the vocabulary. Therefore, two optimization techniques have been proposed, i.e., hierarchical softmax and negative sampling [14]. Empirical studies haven shown that in most cases negative sampling leads to a better performance than hierarchical softmax, which depends on the selected negative samples, but it has higher runtime.

Once the training is finished, all words (or, in our case, entities) are projected into a lower-dimensional feature space, and semantically similar words (or entities) are positioned close to each other.

4 Evaluation

We evaluate our approach on a number of classification and regression tasks, comparing the results of different feature extraction strategies combined with different learning algorithms.

4.1 Datasets

We evaluate the approach on two types of RDF graphs: (i) small domain-specific RDF datasets and (ii) large cross-domain RDF datasets. More details about the evaluation datasets and how the datasets were generated are presented in [28].

Small RDF Datasets These datasets are derived from existing RDF datasets, where the value of a certain property is used as a classification target:

- The *AIFB* dataset describes the AIFB research institute in terms of its staff, research groups, and publications. In [1], the dataset was first used to predict the affiliation (i.e., research group) for people in the dataset. The dataset contains 178 members of five research groups, however, the smallest group contains only four people, which is removed from the dataset, leaving four classes.
- The *MUTAG* dataset is distributed as an example dataset for the DL-Learner toolkit[2]. It contains information about 340 complex molecules that are potentially carcinogenic, which is given by the `isMutagenic` property. The molecules can be classified as "mutagenic" or "not mutagenic".
- The *BGS* dataset was created by the British Geological Survey and describes geological measurements in Great Britain[3]. It was used in [33] to predict the lithogenesis property of named rock units. The dataset contains 146 named rock units with a lithogenesis, from which we use the two largest classes.

Large RDF Datasets As large cross-domain datasets we use DBpedia [11] and Wikidata [31].

We use the English version of the 2015-10 DBpedia dataset, which contains $4,641,890$ instances and $1,369$ mapping-based properties. In our evaluation we only consider object properties, and ignore datatype properties and literals.

For the Wikidata dataset we use the simplified and derived RDF dumps from 2016-03-28[4]. The dataset contains $17,340,659$ entities in total. As for the DBpedia dataset, we only consider object properties, and ignore the data properties and literals.

We use the entity embeddings on five different datasets from different domains, for the tasks of classification and regression. Those five datasets are used to provide classification/regression targets for the large RDF datasets (see Table 1).

- The *Cities* dataset contains a list of cities and their quality of living, as captured by Mercer[5]. We use the dataset both for regression and classification.
- The *Metacritic Movies* dataset is retrieved from Metacritic.com[6], which contains an average rating of all time reviews for a list of movies [26]. The initial dataset contained around $10,000$ movies, from which we selected $1,000$ movies from the top of the list, and $1,000$ movies from the bottom of the list. We use the dataset both for regression and classification.
- Similarly, the *Metacritic Albums* dataset is retrieved from Metacritic.com[7], which contains an average rating of all time reviews for a list of albums [27].

[2] http://dl-learner.org
[3] http://data.bgs.ac.uk/
[4] http://tools.wmflabs.org/wikidata-exports/rdf/index.php?content=dump_download.php&dump=20160328
[5] https://www.imercer.com/content/mobility/quality-of-living-city-rankings.html
[6] http://www.metacritic.com/browse/movies/score/metascore/all
[7] http://www.metacritic.com/browse/albums/score/metascore/all

Table 1: Datasets overview. For each dataset, we depict the number of instances, the machine learning tasks in which the dataset is used (C stands for classification, and R stands for regression) and the source of the dataset.

Dataset	#Instances	ML Task	Original Source
Cities	212	R/C (c=3)	Mercer
Metacritic Albums	1600	R/C (c=2)	Metacritic
Metacritic Movies	2000	R/C (c=2)	Metacritic
AAUP	960	R/C (c=3)	JSE
Forbes	1585	R/C (c=3)	Forbes
AIFB	176	C (c=4)	AIFB
MUTAG	340	C (c=2)	MUTAG
BGS	146	C (c=2)	BGS

- The *AAUP* (American Association of University Professors) dataset contains a list of universities, including eight target variables describing the salary of different staff at the universities[8]. We use the average salary as a target variable both for regression and classification, discretizing the target variable into "high", "medium" and "low", using equal frequency binning.
- The *Forbes* dataset contains a list of companies including several features of the companies, which was generated from the Forbes list of leading companies 2015[9]. The target is to predict the company's market value as a regression task. To use it for the task of classification we discretize the target variable into "high", "medium", and "low", using equal frequency binning.

4.2 Experimental Setup

The first step of our approach is to convert the RDF graphs into a set of sequences. For each of the small RDF datasets, we first build two corpora of sequences, i.e., the set of sequences generated from graph walks with depth 8 (marked as W2V), and set of sequences generated from Weisfeiler-Lehman subtree kernels (marked as K2V). For the Weisfeiler-Lehman algorithm, we use 4 iterations and depth of 2, and after each iteration we extract all walks for each entity with the same depth. We use the corpora of sequences to build both CBOW and Skip-Gram models with the following parameters: window size = 5; number of iterations = 10; negative sampling for optimization; negative samples = 25; with average input vector for CBOW. We experiment with 200 and 500 dimensions for the entities' vectors. The remaining parameters have the default value as proposed in [14].

As the number of generated walks increases exponentially [34] with the graph traversal depth, calculating Weisfeiler-Lehman subtrees RDF kernels, or all graph walks with a given depth d for all of the entities in the large RDF graph quickly becomes unmanageable. Therefore, to extract the entities embeddings for the large RDF datasets, we use only random graph walks entity sequences. More precisely, we follow the approach presented in [22] to generate limited number of random walks for each entity. For DBpedia, we experiment with 500 walks

[8] http://www.amstat.org/publications/jse/jse_data_archive.htm

[9] http://www.forbes.com/global2000/list/

per entity with depth of 4 and 8, while for Wikidata, we use only 200 walks per entity with depth of 4. Additionally, for each entity in DBpedia and Wikidata, we include all the walks of depth 2, i.e., direct outgoing relations. We use the corpora of sequences to build both CBOW and Skip-Gram models with the following parameters: window size = 5; number of iterations = 5; negative sampling for optimization; negative samples = 25; with average input vector for CBOW. We experiment with 200 and 500 dimensions for the entities' vectors. All the models, as well as the code, are publicly available[10].

We compare our approach to several baselines. For generating the data mining features, we use three strategies that take into account the direct relations to other resources in the graph [20], and two strategies for features derived from graph sub-structures [34]:

- Features derived from specific relations. In the experiments we use the relations *rdf:type* (types), and *dcterms:subject* (categories) for datasets linked to DBpedia.
- Features derived from generic relations, i.e., we generate a feature for each incoming (rel in) or outgoing relation (rel out) of an entity, ignoring the value or target entity of the relation.
- Features derived from generic relations-values, i.e, we generate feature for each incoming (rel-vals in) or outgoing relation (rel-vals out) of an entity including the value of the relation.
- Kernels that count substructures in the RDF graph around the instance node. These substructures are explicitly generated and represented as sparse feature vectors.
  - The Weisfeiler-Lehman (WL) graph kernel for RDF [34] counts full subtrees in the subgraph around the instance node. This kernel has two parameters, the subgraph depth d and the number of iterations h (which determines the depth of the subtrees). We use two pairs of settings, $d = 1, h = 2$ and $d = 2, h = 3$.
  - The Intersection Tree Path kernel for RDF [34] counts the walks in the subtree that spans from the instance node. Only the walks that go through the instance node are considered. We will therefore refer to it as the root Walk Count (WC) kernel. The root WC kernel has one parameter: the length of the paths l, for which we test 2 and 3.

We perform two learning tasks, i.e., classification and regression. For classification tasks, we use Naive Bayes, k-Nearest Neighbors (k=3), C4.5 decision tree, and Support Vector Machines. For the SVM classifier we optimize the parameter C in the range $\{10^{-3}, 10^{-2}, 0.1, 1, 10, 10^2, 10^3\}$. For regression, we use Linear Regression, M5Rules, and k-Nearest Neighbors (k=3). We measure accuracy for classification tasks, and root mean squared error (RMSE) for regression tasks. The results are calculated using stratfied 10-fold cross validation.

The strategies for creating propositional features from Linked Open Data are implemented in the RapidMiner LOD extension[11] [21, 23]. The experiments,

[10] http://data.dws.informatik.uni-mannheim.de/rdf2vec/

[11] http://dws.informatik.uni-mannheim.de/en/research/rapidminer-lod-extension

Table 2: Classification results on the small RDF datasets. The best results are marked in bold. Experiments marked with "\" did not finish within ten days, or have run out of memory.

Strategy/Dataset	AIFB				MUTAG				BGS			
	NB	KNN	SVM	C4.5	NB	KNN	SVM	C4.5	NB	KNN	SVM	C4.5
rel in	16.99	47.19	50.70	50.62	\	\	\	\	61.76	54.67	63.76	63.76
rel out	45.07	45.56	50.70	51.76	41.18	54.41	62.94	62.06	54.76	69.05	72.70	69.33
rel in & out	25.59	51.24	50.80	51.80	\	\	\	\	54.76	67.00	72.00	70.00
rel-vals in	73.24	54.54	81.86	80.75	\	\	\	\	79.48	83.52	86.50	68.57
rel-vals out	86.86	55.69	82.39	71.73	62.35	62.06	73.53	62.94	84.95	65.29	83.10	73.38
rel-vals in&out	87.42	57.91	88.57	85.82	\	\	\	\	84.95	70.81	85.80	72.67
WL_1_2	85.69	53.30	92.68	71.08	91.12	62.06	92.59	93.29	85.48	63.62	82.14	75.29
WL_2_2	85.65	65.95	83.43	89.25	70.59	62.06	94.29	93.47	90.33	85.57	91.05	87.67
WC_2	86.24	60.27	75.03	71.05	90.94	62.06	91.76	93.82	84.81	69.00	83.57	76.90
WC_3	86.83	64.18	82.97	71.05	92.00	72.56	86.47	93.82	85.00	67.00	78.71	76.90
W2V CBOW 200	70.00	69.97	79.48	65.33	74.71	72.35	80.29	74.41	56.14	74.00	74.71	67.38
W2V CBOW 500	69.97	69.44	82.88	73.40	75.59	70.59	82.06	72.06	55.43	73.95	74.05	65.86
W2V SG 200	76.76	71.67	87.39	65.36	70.00	71.76	77.94	68.53	66.95	69.10	75.29	71.24
W2V SG 500	76.67	76.18	89.55	71.05	72.35	72.65	78.24	68.24	68.38	71.19	78.10	63.00
K2V CBOW 200	85.16	84.48	87.48	76.08	78.82	69.41	86.47	68.53	93.14	95.57	94.71	88.19
K2V CBOW 500	90.98	88.17	86.83	76.18	80.59	70.88	90.88	66.76	93.48	95.67	94.82	87.26
K2V SG 200	85.65	87.96	90.82	75.26	78.53	69.29	95.88	66.00	91.19	93.24	95.95	87.05
K2V SG 500	88.73	88.66	**93.41**	69.90	82.06	70.29	**96.18**	66.18	91.81	93.19	**96.33**	80.76

including the feature generation and the evaluation, were performed using the RapidMiner data analytics platform.[12] The RapidMiner processes and the complete results can be found online.[13]

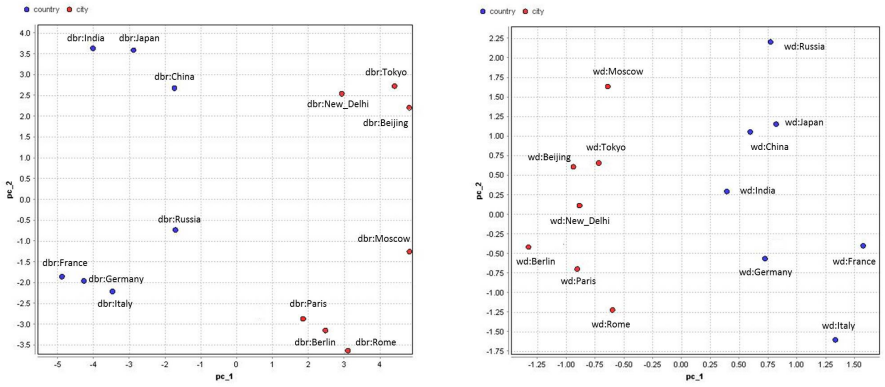
4.3 Results

The results for the task of classification on the small RDF datasets are given in Table 2. From the results we can observe that the K2V approach outperforms all the other approaches. More precisely, using the skip-gram feature vectors of size 500 in an SVM model provides the best results on all three datasets. The W2V approach on all three datasets performs closely to the standard graph substructure feature generation strategies, but it does not outperform them. K2V outperforms W2V because it is able to capture more complex substructures in the graph, like sub-trees, while W2V focuses only on graph paths.

The results for the task of classification on the five different datasets using the DBpedia and Wikidata entities' vectors are given in Table 3, and the results for the task of regression on the 5 different dataset using the DBpedia and Wikidata entities' vectors are given in Table 4. We can observe that the latent vectors extracted from DBpedia and Wikidata outperform all of the standard feature generation approaches. In general, the DBpedia vectors work better than the Wikidata vectors, where the skip-gram vectors with size 200 or 500 built on graph walks of depth 8 on most of the datasets lead to the best performances. An exception is the AAUP dataset, where the Wikidata skip-gram 500 vectors outperform the other approaches.

[12] https://rapidminer.com/

[13] http://data.dws.informatik.uni-mannheim.de/rmlod/LOD_ML_Datasets/



a) *DBpedia vectors* **b)** *Wikidata vectors*

Fig. 2: Two-dimensional PCA projection of the 500-dimensional Skip-gram vectors of countries and their capital cities.

On both tasks, we can observe that the skip-gram vectors perform better than the CBOW vectors. Also, the vectors with higher dimensionality and paths with bigger depth on most of the datasets lead to a better representation of the entities and better performances. However, for the variety of tasks at hand, there is no universal approach, i.e., embedding model and a machine learning method, that consistently outperforms the others.

4.4 Semantics of Vector Representations

To analyze the semantics of the vector representations, we employ Principal Component Analysis (PCA) to project the entities' feature vectors into a two dimensional feature space. We selected seven countries and their capital cities, and visualized their vectors as shown in Figure 2. Figure 2a shows the corresponding DBpedia vectors, and Figure 2b shows the corresponding Wikidata vectors. The figure illustrates the ability of the model to automatically organize entities of different types, and preserve the relationship between different entities. For example, we can see that there is a clear separation between the countries and the cities, and the relation "capital" between each pair of country and the corresponding capital city is preserved. Furthermore, we can observe that more similar entities are positioned closer to each other, e.g., we can see that the countries that are part of the EU are closer to each other, and the same applies for the Asian countries.

4.5 Features Increase Rate

Finally, we conduct a scalability experiment, where we examine how the number of instances affects the number of generated features by each feature generation strategy. For this purpose we use the *Metacritic Movies* dataset. We start with a random sample of 100 instances, and in each next step we add 200 (or 300)

Table 3: Classification results. The first number represents the dimensionality of the vectors, while the second number represent the value for the depth parameter. The best results are marked in bold. Experiments marked with "\" did not finish within ten days, or have run out of memory.

Strategy/Dataset	Cities				Metacritic Movies				Metacritic Albums				AAUP				Forbes			
	NB	KNN	SVM	C4.5	NB	KNN	SVM	C4.5	NB	KNN	SVM	C4.5	NB	KNN	SVM	C4.5	NB	KNN	SVM	C4.5
types	55.71	56.17	63.21	59.05	68.00	57.60	71.40	70.00	66.50	50.75	62.31	54.44	41.00	85.62	91.67	92.78	55.08	75.84	75.67	75.85
categories	55.74	49.98	62.39	56.17	75.25	62.70	76.35	69.50	67.40	54.13	64.50	56.62	48.00	85.83	90.78	91.87	60.38	76.11	75.70	75.70
rel in	60.41	58.46	71.70	60.35	52.75	49.90	60.35	60.10	51.13	62.19	65.25	60.75	45.63	85.94	90.62	92.81	50.24	76.49	75.16	76.10
rel out	47.62	60.00	66.04	56.71	52.90	58.45	66.40	62.70	58.75	63.75	62.25	64.50	41.15	85.83	89.58	91.35	64.73	75.84	75.73	75.92
rel in & out	59.44	58.57	66.04	56.47	52.95	59.30	67.75	62.55	58.69	64.50	67.38	61.56	42.71	85.94	89.67	92.50	22.27	75.96	76.34	75.98
rel-vals in	\	\	\	\	50.60	50.00	50.60	50.00	50.88	50.00	50.81	50.00	54.06	84.69	89.51	\	14.95	76.15	76.97	75.73
rel-vals out	53.79	35.91	55.66	64.13	78.50	54.78	78.71	\	74.06	52.56	76.99	\	57.81	85.73	91.46	91.78	67.09	75.61	75.74	76.74
rel-vals in&out	\	\	\	\	77.90	55.75	77.82	\	74.25	51.25	75.85	\	63.44	84.69	91.56	\	67.20	75.88	75.96	76.75
WL_1_2	70.98	49.31	65.34	75.29	75.45	66.90	79.30	70.80	73.63	64.69	76.25	62.00	58.33	91.04	91.46	92.40	64.17	75.71	75.10	76.59
WL_2_3	65.48	53.29	69.90	69.31	\	\	\	\	\	\	\	\	\	\	\	\	\	\	\	\
WC_2	72.71	47.39	66.48	75.13	75.39	65.89	74.93	69.08	72.00	60.63	76.88	63.69	57.29	90.63	93.44	92.60	64.23	75.77	76.22	76.47
WC_3	65.52	52.36	67.95	65.15	74.25	55.30	78.40	\	72.81	52.87	77.94	\	57.19	90.73	90.94	92.60	64.04	75.65	76.22	76.59
DB2vec CBOW 200 4	59.32	68.84	77.39	64.32	65.60	79.74	82.90	74.33	70.72	71.86	76.36	67.24	73.36	89.65	29.00	92.45	89.38	80.94	76.83	84.81
DB2vec CBOW 500 4	59.32	71.34	76.37	66.34	65.65	79.49	82.75	73.87	69.71	71.93	75.41	65.65	72.71	89.65	29.11	92.01	89.02	80.82	76.95	85.17
DB2vec SG 200 4	60.34	71.82	76.37	65.37	65.25	80.44	83.25	73.87	68.95	73.89	76.11	67.87	71.20	89.65	28.90	92.12	88.78	80.82	77.92	85.77
DB2vec SG 500 4	58.34	72.84	76.87	67.84	65.45	80.14	**83.65**	72.82	70.41	74.34	78.44	67.49	71.19	89.65	28.90	92.23	88.30	80.94	77.25	84.81
DB2vec CBOW 200 8	69.26	69.87	67.32	63.13	57.83	70.08	65.25	67.47	67.91	64.44	72.42	65.39	68.18	85.33	28.90	90.50	77.35	80.34	28.90	85.17
DB2vec CBOW 500 8	62.26	69.87	76.84	63.21	58.78	69.82	69.46	67.67	67.53	65.83	74.26	63.42	62.90	85.22	29.11	90.61	89.86	80.34	78.65	84.81
DB2vec SG 200 8	73.32	75.89	78.92	60.74	79.94	79.49	83.30	75.13	77.25	76.87	**79.72**	69.14	78.53	85.12	29.22	91.04	**90.10**	80.58	78.96	84.68
DB2vec SG 500 8	**89.73**	69.16	84.19	72.25	80.24	78.68	82.80	72.42	73.57	76.30	78.20	68.70	75.07	**94.48**	29.11	94.15	88.53	80.58	77.79	86.38
WD2vec CBOW 200 4	68.76	57.71	75.56	61.37	51.49	52.20	51.64	49.01	50.86	50.29	51.44	50.09	50.54	90.18	89.63	88.83	49.84	81.08	76.77	79.14
WD2vec CBOW 500 4	68.24	57.75	85.56	64.54	49.22	48.56	51.04	50.98	53.08	50.03	52.33	53.28	48.45	90.39	89.74	88.31	51.95	80.74	78.18	80.32
WD2vec SG 200 4	72.58	57.53	75.48	52.32	69.53	70.14	75.39	67.00	60.32	62.03	64.76	58.54	60.87	90.50	89.63	89.98	65.45	81.17	77.74	77.03
WD2vec SG 500 4	83.20	60.72	79.87	61.67	71.10	70.19	76.30	67.31	55.31	58.92	63.42	56.63	55.85	90.60	89.63	87.69	58.95	81.17	79.00	79.56

Table 4: Regression results. The first number represents the dimensionality of the vectors, while the second number represent the value for the depth parameter. The best results are marked in bold. Experiments that did not finish within ten days, or that have run out of memory are marked with "\".

Strategy/Dataset	Cities			Metacritic Movies			Metacritic Albums			AAUP			Forbes		
	LR	KNN	M5	LR	KNN	M5	LR	KNN	M5	LR	KNN	M5	LR	KNN	M5
types	24.30	22.16	18.79	77.80	30.68	22.16	16.45	18.36	13.95	9.83	34.95	6.28	29.22	21.07	18.32
categories	18.88	22.68	22.32	84.57	23.87	22.50	16.73	16.64	13.95	8.08	34.94	6.16	19.16	21.48	18.39
rel in	49.87	18.53	19.21	22.60	41.40	22.56	13.50	22.06	13.43	9.69	34.98	6.56	27.56	20.93	18.60
rel out	49.87	18.53	19.21	21.45	24.42	20.74	13.32	14.59	13.06	8.82	34.95	6.32	21.73	21.11	18.97
rel in & out	40.80	18.21	18.80	21.45	24.42	20.74	13.33	14.52	12.91	12.97	34.95	6.36	26.44	20.98	19.54
rel-vals in	\	\	\	21.46	24.19	20.43	13.94	23.05	13.95	\	34.96	6.27	\	20.86	19.31
rel-vals out	20.93	23.87	20.97	25.99	32.18	22.93	\	15.28	13.34	\	34.95	6.18	\	20.48	18.37
rel-vals in&out	\	\	\	\	25.37	20.96	\	15.47	13.33	\	34.94	6.18	\	20.20	18.20
WL_1_2	20.21	24.60	20.85	\	21.62	19.84	\	13.99	12.81	\	34.96	6.27	\	19.81	19.49
WL_2_3	17.79	20.42	17.04	\	\	\	\	\	\	\	\	\	\	\	\
WC_2	20.33	25.95	19.55	\	22.80	22.99	\	14.54	12.87	9.12	34.95	6.24	\	20.45	19.26
WC_3	19.51	33.16	19.05	\	23.86	19.19	\	19.51	13.02	\	35.39	6.31	\	20.58	19.04
DB2vec CBOW 200 4	14.37	12.55	14.33	15.90	17.46	15.89	11.79	12.45	11.59	12.13	45.76	12.00	18.32	26.19	17.43
DB2vec CBOW 500 4	14.99	12.46	14.66	15.90	17.45	15.73	11.49	12.60	11.48	12.44	45.67	12.30	18.23	26.27	17.62
DB2vec SG 200 4	13.38	12.54	15.13	15.81	17.07	15.84	11.30	12.36	11.42	12.13	45.72	12.10	17.63	26.13	17.85
DB2vec SG 500 4	14.73	13.25	16.80	15.66	17.14	15.67	11.20	12.11	11.28	12.09	45.76	11.93	18.23	26.09	17.74
DB2vec CBOW 200 8	16.17	17.14	17.56	21.55	23.75	21.46	13.35	15.41	13.43	6.47	55.76	6.47	24.17	26.48	22.61
DB2vec CBOW 500 8	18.13	17.19	18.50	20.77	23.67	20.69	13.20	15.14	13.25	6.54	55.33	6.55	21.16	25.90	20.33
DB2vec SG 200 8	12.85	14.95	12.92	15.15	17.13	**15.12**	10.90	11.43	10.90	6.22	56.95	6.25	18.66	21.20	18.57
DB2vec SG 500 8	11.92	12.67	**10.19**	15.45	17.80	15.50	**10.89**	11.72	10.97	6.26	56.95	6.29	18.35	21.04	**16.61**
WD2vec CBOW 200 4	20.15	17.52	20.02	23.54	25.90	23.39	14.73	16.12	14.55	16.80	42.61	6.60	27.48	22.60	21.77
WD2vec CBOW 500 4	23.76	18.33	20.39	24.14	22.18	24.56	14.09	16.09	14.00	13.08	42.89	6.08	50.23	21.92	26.66
WD2vec SG 200 4	20.47	18.69	20.72	19.72	21.44	19.10	13.51	13.91	13.67	6.86	42.82	6.52	23.69	21.59	20.49
WD2vec SG 500 4	22.25	19.41	19.23	25.99	21.26	19.19	13.23	14.96	13.25	8.27	42.84	**6.05**	21.98	21.73	21.58

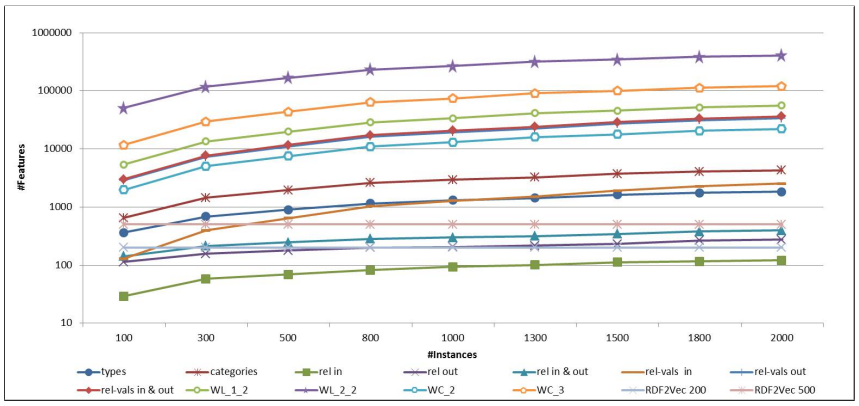


Fig. 3: Features increase rate per strategy (log scale).

unused instances, until the complete dataset is used, i.e., 2,000 instances. The number of generated features for each sub-sample of the dataset using each of the feature generation strategies is shown in Figure 3.

From the chart, we can observe that the number of generated features sharply increases when adding more samples in the datasets, especially for the strategies based on graph substructures. However, the number of features remains the same when using the RDF2Vec approach, independently of the number of samples in the data. Thus, by design, it scales to larger datasets without increasing the dimensionality of the dataset.

5 Conclusion

In this paper, we have presented RDF2Vec, an approach for learning latent numerical representations of entities in RDF graphs. In this approach, we first convert the RDF graphs in a set of sequences using two strategies, Weisfeiler-Lehman Subtree RDF Graph Kernels and graph walks, which are then used to build neural language models. The evaluation shows that such entity representations could be used in two different machine learning tasks, outperforming standard feature generation approaches.

So far we have considered only simple machine learning tasks, i.e, classification and regression, but in the future work we would extend the number of applications. For example, the latent representation of the entities could be used for building content-based recommender systems [4]. The approach could also be used for link predictions, type prediction, graph completion and error detection in knowledge graphs [19], as shown in [15, 17]. Furthermore, we could use this approach for the task of measuring semantic relatedness between two entities, which is the basis for numerous tasks in information retrieval, natural language processing, and Web-based knowledge extractions [6]. To do so, we could easily calculate the relatedness between two entities as the probability of one entity being the context of the other entity, using the softmax function given in Equation 2 and 5, using the input and output weight matrix of the neural model.

Similarly, the approach can be extended for entity summarization, which is also an important task when consuming and visualizing large quantities of data [2].

Acknowledgements The work presented in this paper has been partly funded by the German Research Foundation (DFG) under grant number PA 2373/1-1 (Mine@LOD).

References

1. Bloehdorn, S., Sure, Y.: Kernel methods for mining instance data in ontologies. In: Proceedings of the 6th International The Semantic Web and 2Nd Asian Conference on Asian Semantic Web Conference. pp. 58–71. ISWC'07/ASWC'07 (2007)
2. Cheng, G., Tran, T., Qu, Y.: Relin: relatedness and informativeness-based centrality for entity summarization. In: The Semantic Web–ISWC, pp. 114–129 (2011)
3. Cheng, W., Kasneci, G., Graepel, T., Stern, D., Herbrich, R.: Automated feature generation from structured knowledge. In: CIKM (2011)
4. Di Noia, T., Ostuni, V.C.: Recommender systems and linked open data. In: Reasoning Web. Web Logic Rules, pp. 88–113. Springer (2015)
5. Fanizzi, N., d'Amato, C.: A declarative kernel for alc concept descriptions. In: Foundations of Intelligent Systems, pp. 322–331 (2006)
6. Hoffart, J., Seufert, S., Nguyen, D.B., Theobald, M., Weikum, G.: Kore: keyphrase overlap relatedness for entity disambiguation. In: Proceedings of the 21st ACM international conference on Information and knowledge management. pp. 545–554. ACM (2012)
7. Huang, Y., Tresp, V., Nickel, M., Kriegel, H.P.: A scalable approach for statistical learning in semantic graphs. Semantic Web (2014)
8. Kappara, V.N.P., Ichise, R., Vyas, O.: Liddm: A data mining system for linked data. In: LDOW (2011)
9. Khan, M.A., Grimnes, G.A., Dengel, A.: Two pre-processing operators for improved learning from semanticweb data. In: RCOMM (2010)
10. Kramer, S., Lavrač, N., Flach, P.: Propositionalization approaches to relational data mining. In: Džeroski, S., Lavrač, N. (eds.) Relational Data Mining, pp. 262–291. Springer Berlin Heidelberg (2001)
11. Lehmann, J., Isele, R., Jakob, M., Jentzsch, A., Kontokostas, D., Mendes, P.N., Hellmann, S., Morsey, M., van Kleef, P., Auer, S., Bizer, C.: DBpedia – A Large-scale, Multilingual Knowledge Base Extracted from Wikipedia. Semantic Web Journal (2013)
12. Lösch, U., Bloehdorn, S., Rettinger, A.: Graph kernels for rdf data. In: The Semantic Web: Research and Applications, pp. 134–148. Springer (2012)
13. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781 (2013)
14. Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J.: Distributed representations of words and phrases and their compositionality. In: Advances in neural information processing systems. pp. 3111–3119 (2013)
15. Minervini, P., Fanizzi, N., d'Amato, C., Esposito, F.: Scalable learning of entity and predicate embeddings for knowledge graph completion. In: 2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA). pp. 162–167. IEEE (2015)
16. Mynarz, J., Svátek, V.: Towards a benchmark for LOD-enhanced knowledge discovery from structured data. In: The Second International Workshop on Knowledge Discovery and Data Mining Meets Linked Open Data (2013)

17. Nickel, M., Murphy, K., Tresp, V., Gabrilovich, E.: A review of relational machine learning for knowledge graphs: From multi-relational link prediction to automated knowledge graph construction. arXiv preprint arXiv:1503.00759 (2015)
18. Paulheim, H.: Exploiting linked open data as background knowledge in data mining. In: Workshop on Data Mining on Linked Open Data (2013)
19. Paulheim, H.: Knowlegde Graph Refinement: A Survey of Approaches and Evaluation Methods. Semantic Web Journal (2016), to appear
20. Paulheim, H., Fümkranz, J.: Unsupervised generation of data mining features from linked open data. In: Proceedings of the 2nd international conference on web intelligence, mining and semantics. p. 31. ACM (2012)
21. Paulheim, H., Ristoski, P., Mitichkin, E., Bizer, C.: Data mining with background knowledge from the web. RapidMiner World (2014)
22. Perozzi, B., Al-Rfou, R., Skiena, S.: Deepwalk: Online learning of social representations. In: Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining. pp. 701–710. ACM (2014)
23. Ristoski, P., Bizer, C., Paulheim, H.: Mining the web of linked data with rapidminer. Web Semantics: Science, Services and Agents on the World Wide Web 35, 142–151 (2015)
24. Ristoski, P., Paulheim, H.: A comparison of propositionalization strategies for creating features from linked open data. In: Linked Data for Knowledge Discovery (2014)
25. Ristoski, P., Paulheim, H.: Semantic web in data mining and knowledge discovery: A comprehensive survey. Web Semantics: Science, Services and Agents on the World Wide Web (2016)
26. Ristoski, P., Paulheim, H., Svátek, V., Zeman, V.: The linked data mining challenge 2015. In: KNOW@LOD (2015)
27. Ristoski, P., Paulheim, H., Svátek, V., Zeman, V.: The linked data mining challenge 2016. In: KNOWLOD (2016)
28. Ristoski, P., de Vries, G.K.D., Paulheim, H.: A collection of benchmark datasets for systematic evaluations of machine learning on the semantic web. In: International Semantic Web Conference (To Appear). Springer (2016)
29. Schmachtenberg, M., Bizer, C., Paulheim, H.: Adoption of the linked data best practices in different topical domains. In: The Semantic Web–ISWC (2014)
30. Shervashidze, N., Schweitzer, P., Van Leeuwen, E.J., Mehlhorn, K., Borgwardt, K.M.: Weisfeiler-lehman graph kernels. The Journal of Machine Learning Research 12, 2539–2561 (2011)
31. Vrandečić, D., Krötzsch, M.: Wikidata: a free collaborative knowledgebase. Communications of the ACM 57(10), 78–85 (2014)
32. de Vries, G.K.D.: A fast approximation of the Weisfeiler-Lehman graph kernel for RDF data. In: ECML/PKDD (1) (2013)
33. de Vries, G.K.D., de Rooij, S.: A fast and simple graph kernel for rdf. In: DMLOD (2013)
34. 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)
35. 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. ACM (2015)