Knowledge Graph Embeddings over Hundreds of Linked Datasets

Michalis Mountantonakis[0000−0002−1951−0241] and Yannis Tzitzikas[0000−0001−8847−2130]

Institute of Computer Science - FORTH-ICS, Greece
and Computer Science Department - University of Crete, Greece
{mountant, tzitzik}@ics.forth.gr

Abstract. There is an increasing trend of using Linked Datasets for creating embeddings from URI sequences, since such embeddings can be exploited for several tasks, i.e., for machine learning problems, tasks related to content-based similarity, and others. Existing techniques exploit either a single or a few datasets (or RDF graphs) for creating URI sequences for one or more entities. However, there are not available approaches, where data from multiple datasets are combined, for enriching the URI sequences for a given entity. For this reason, we introduce a prototype, called `LODVec`, that exploits `LODsyndesis` knowledge graph, which is the largest knowledge graph including all inferred equivalence relationships. `LODVec` exploits this graph for creating URI sequences for millions of entities by combining data from 400 datasets, whereas it offers several configurable options for creating such URI sequences that are based on metadata (e.g., provenance). Moreover, it uses as input the produced URI sequences for creating URI embeddings through *word2vec* model. We evaluate the gain of exploiting several datasets (instead of a single or few ones) and the impact of cross-dataset reasoning for machine-learning based tasks (i.e., classification and regression), and we compare the effectiveness of several configurations and machine learning models.

Keywords: URI embeddings, Multiple Datasets, Machine Learning

1 Introduction

There is an increasing trend of exploiting LOD (Linked Open Data) for creating embeddings for URIs (Uniform Resource Identifiers), which can be exploitable for a number of tasks. Indicatively, they can be exploited i) for machine learning-based tasks [17], such as classification, regression, etc., ii) for similarity-based tasks [4], e.g. "Give me the top-K related entities to a given one", iii) for link prediction purposes [14], and others. There have been proposed several novel methods [17] taking as input RDF (Resource Description Frameworks) knowledge graphs and producing URI sequences for a set of given entities, i.e., sequences starting from a focused entity (or URI) that contains a path of URIs which is reachable from that entity. These sequences are given as input for producing URI embeddings which can be exploited in the aforementioned tasks.

However, current approaches exploit usually a single dataset for creating URI embeddings for one or more URIs (or entities). Moreover, many approaches are difficult to be configured by non-experts, since they do not provide an interactive service. Our objective is to make it feasible to exploit hundreds of RDF datasets simultaneously, for creating URI sequences and URI embeddings for any given entity (i.e., a URI). Concerning our research hypothesis, we assume that it is better to exploit multiple datasets for creating URI sequences and embeddings, instead of using one or few datasets, whereas we assume that there is not a single knowledge graph that can outperform all the others for any possible task.

Generally, it is not easy to combine all the available information for a given entity, since (a) data are scattered in different places, and (b) datasets use different URIs and models for representing their entities and schema elements, respectively [13]. For example, each of the three datasets of Fig. 1 uses a different URI for representing the movie "Inception" and the schema element "actor", while these datasets are located in different places. Indeed, for combining information from several datasets, one should collect the desired data and find the equivalences in instance and schema level. For achieving this target, it is required to compute the transitive and symmetric closure of equivalence relationships, such as `owl:sameAs` and `owl:equivalentProperty`, which is quite expensive [11].

For tackling the above difficulties, we exploit `LODsyndesis` knowledge graph [11], which contains two billion triples from 400 datasets of 9 different domains, and it has pre-computed the transitive and symmetric closure of the equivalence relationships in instance and schema level. Due to cross-dataset reasoning, its' indexes offer direct and fast access to all the available information for any entity. In this way, it is feasible to create URI sequences for the same entity from multiple datasets, therefore, the number of possible URI sequences that can be created is highly increased (as we shall see in Section 4). For example, suppose that the required task is to predict the exact rating of a movie, e.g., "Inception" (see Fig.1), and for this reason we plan to create embeddings from URI sequences for using them in such a machine learning task. In Fig.1, by using only one dataset, say *DBpedia* (`http://dbpedia.org/`), we can find data such as the genres of this movie. However, it does not contain information about the awards won by this movie, e.g., in Fig. 1 such data occur in *Wikidata* (`http://wikidata.org/`). On the contrary, `LODsyndesis` (see the lower side of Fig. 1), a) pre-computes the equivalences in schema (e.g., dbp:actor≡wkd:actor≡frb:played) and instance level (e.g., dbp:Inception≡wkd:Inception≡frb:Inception), b) collects all the available triples containing an entity (e.g., Inception) either as a subject or as an object (by following both direct and indirect edges), and c) records data provenance (see the label near to each node in Fig. 1). Therefore, by using `LODsyndesis`, it is feasible to create URI sequences by using all the three datasets of Fig. 1.

For exploiting the above characteristics, we introduce a research prototype, i.e., `LODVec`, that (i) takes as input one or more entities (as URIs or in plain text), (ii) it offers several configurable options for creating URI sequences and embeddings for these entities from hundreds of datasets through `LODsyndesis`, and (iii) it produces URI sequences based on user's selections. Moreover, `LODVec` (iv)

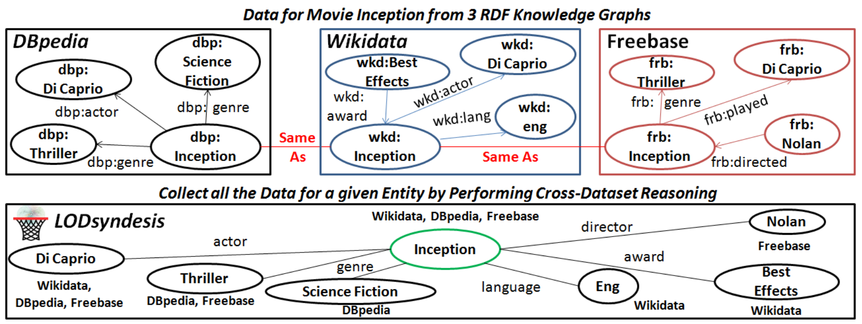

Fig. 1: Running example containing 3 knowledge graphs and `LODsyndesis`

converts the produced sequences into vector representations (i.e., embeddings) by exploiting *word2vec* approach [8,9] through *dl4j* API (`https://deeplearning4j.org/`). Finally, it can (v) exploit the produced vectors for several purposes, e.g., for performing classification and regression tasks by using *WEKA* API [20]. For testing the proposed approach, we report experimental results for machine learning classification and regression tasks by using two datasets containing movies and music albums, i.e., the target of classification task is to classify whether a movie or a music album has a high or low rating, whereas the target of regression task is to predict their exact rating. We introduce experiments showing the impact of using multiple datasets and cross-dataset reasoning in terms of effectiveness, whereas, we compare the performance of different configurations.

The rest of this paper is organized as follows: Section 2 introduces the background and related work. Section 3 provides the problem statement, the algorithm for creating URI sequences and all the steps and functionalities that our approach supports, whereas Section 4 includes the experimental evaluation about the effectiveness of our approach. Finally, Section 5 concludes the paper.

2 Background and Related Work

2.1 Background

Linked Data. Resource Description Framework (RDF) [1] is a model that can be represented as a graph, and uses Uniform Resource Identifiers (URIs), or anonymous nodes to denote resources, and literals to denote constants. Every statement in RDF can be represented as a triple. A triple is a statement of the form subject-predicate-object $\langle s, p, o \rangle$, and it is any element of $T = (U \cup B) \times (U) \times (U \cup B \cup L)$ where U, B and L are the sets of URIs, blank nodes and literals, respectively. Any finite subset of T corresponds to an RDF graph (or dataset). We divide the URIs in three disjoint sets, entities E (e.g., "Inception"), properties P (e.g., actor) and RDF classes C (e.g., Drama Movie). In this paper, we focus only on triples containing an entity as subject, and an RDF class or an entity as object, therefore we consider triples in $T' = U \times P \times (E \cup C) \subseteq T$.

Word2Vec. It is a shallow two-layer neural network model for producing word embeddings [8, 9]. It takes as input a text, and it produces a vector with several (usually hundreds of) dimensions for each unique word appearing in the text. The target of *word2Vec* [8,9] is to group the vectors of similar words closely in the vector space. In this paper, we will exploit this model for creating vectors for entities, by using the skip-gram model, which is a method that uses a specific word for predicting a target context, since "it produces more accurate results for large datasets" (`https://deeplearning4j.org/docs/latest/deeplearning4j-nlp-word2vec`). Our target is to use this model for placing similar entities (e.g., similar movies) to a close position in the vector space.

2.2 Related work

Knowledge graph embedding approaches. *RDF2Vec* [17] is an approach that takes as input an RDF knowledge graph, produces URI sequences based on several strategies, such as random graph walks, and uses *word2vec* for creating vectors. They have also proposed strategies for performing biased graph walks [3], which are based on a number of metrics and statistics, such as the frequency of properties, objects, pagerank and others. They have tested these strategies for multiple tasks, such as classification and regression, by using two datasets *Wikidata* and *DBpedia*, whereas they have used the *GloVe* model [15] for creating RDF embeddings by exploiting global patterns. The authors in [5] used several bibliographic RDF datasets and *word2vec* for enriching the data of scientific publications with information from multiple data sources, while in [6] the authors exploited enriched ontology structures for producing RDF embeddings which were used for the task of Entity Linking. In [14], the authors combined embeddings from *DBpedia* and social network datasets for performing link prediction, whereas in [4] Wikipedia knowledge graph was exploited for finding the most similar entities to a given one for a specific time period. Concerning other graph-based models, such as TransH [19] and TransR [7], they use algorithms for creating entity and relation graph embeddings, i.e., the relationships between two entities are represented as translations in the embedding space.
Feature extraction approaches combining data from several datasets. In [10], the authors proposed a tool that can send SPARQL queries in several endpoints for creating features. However, it does not produce embeddings and it cannot collect all the data for a given entity (i.e., cross-dataset reasoning is required). Moreover, RapidMiner Semantic Web Extension [16] creates features by integrating data from a lot of datasets. However, it performs the integration task by traversing owl:sameAs paths on-the-fly (through SPARQL queries, which can be time-consuming), and not by exploiting pre-constructed indexes.
Novelty & Comparison with other approaches. To the best of our knowledge, this is the first work providing an interactive tool which can easily create URI embeddings for any set of entities by combining data from hundreds of datasets simultaneously. Since current approaches do not take into account the equivalences in schema and instance level, they have been mainly tested on a single dataset. At this point our objective is a) to offer a simple way for creating URI sequences for multiple datasets, and b) to investigate whether the creation of even simple URI sequences and embeddings from different datasets can improve

the effectiveness of several tasks (e.g., machine-learning tasks). Concerning the limitations of our approach, for the time being we do not support methods for creating longer URI sequences (e.g., through random walks [2]), and algorithms that have been successfully applied to knowledge graphs (e.g., [7, 19]).

3 `LODVec`: The Proposed Approach

Section 3.1 introduces the problem statement, Section 3.2 shows some useful notations and metadata, and Section 3.3 describes all the steps of `LODVec`.

3.1 Problem Statement

Creating URI Sequences. The input is a set of entities $E' \subseteq E$, and the first target for each $e \in E'$ is to create a finite set of URI sequences $Seq_U(e)$. Each URI sequence $s \in Seq_U(e)$ is of the form $\langle e, p, o \rangle$ where $p \in P$ and $o \in (C \cup E)$. In this paper, we exploit the neighborhood of an entity e by following both direct and indirect edges, therefore each URI sequence corresponds to a single triple containing e either as a subject or as an object. Our target is to collect all the produced $Seq_U(e)$ for each $e \in E'$, i.e., we construct the set $Seq_U E' = \bigcup_{e \in E'} Seq_U(e)$, where $Seq_U E' \subseteq T'$.
From URI Sequences to URI Embeddings. The target is to map each entity e to a vector space $v(e)$ by using the set $Seq_U E'$, and *word2vec* algorithm (by using skip-gram model). We expect that if two entities e and e' are similar, then their produced vectors $v(e)$ and $v(e')$ will be close to the vector space.
Output Exploitation. Our target is the output vectors to be directly exploitable for machine learning classification and regression tasks, and for finding the top-K similar entities to a given entity e. Concerning the first case, one should also provide as input the corresponding categorical or continuous variable $Y(e)$ for a given entity e. On the second case, there is no need for additional input.

3.2 Metadata & Notations

First, we denote as $D = \{D_i, ..., D_n\}$ a set of datasets, and as $triples(D_i)$ the set of triples for a given dataset D_i (e.g., DBpedia). Table 1 represents a set of notations and metadata that can aid users to select what type of URI sequences will be created. The first one corresponds to the datasets containing a triple t, while the second one indicates which datasets contain at least one triple for an entity e. The third one denotes the datasets that contain at least one triple, for one or more entities $e \in E'$. The fourth is used for showing how many of the entities in E' can be found in a single dataset D_i (i.e., a number in range $[0, |E'|]$), whereas the fifth and the sixth formulas denote all the properties of an entity e and of all the entities E', respectively. The seventh formula shows the number of entities, for whom there is at least one triple that contain a property p. Formula 8 shows all the objects of a triple whose subject is e and predicate is p (e.g., all the actors of a movie). The last formula shows the frequency (popularity) of a URI in the whole graph, i.e., how many triples contain a URI u.

ID	Notation or Metadata	Formula
1	Provenance of Triple t	$prov(t) = \{D_i \in D \mid t = \langle s,p,o\rangle, t \in triples(D_i)\}$
2	Provenance of Entity e	$dsets(e) = \{D_i \in D \mid \exists\langle e,p,o\rangle \in triples(D_i)\}$
3	Provenance of Entities E'	$dsets_{E'} = \bigcup_{e \in E'} dsets(e)$
4	Coverage of a Dataset given E'	$covD(D_i, E') = \|\{e \in E' \mid D_i \in dsets(e)\}\|$
5	Coverage of Properties given E'	$covP(p, E') = \|\{e \in E', \mid \exists\langle e,p,o\rangle \in T'\}\|$
6	Properties of an Entity e	$Prop(e) = \{p \in P \mid \langle e,p,o\rangle \in T'\}$
7	Properties of Entities E'	$Prop_{E'} = \bigcup_{e \in E'} Prop(e)$
8	Objects of an entity-prop. pair	$Obj(e,p) = \{o \in U \| \langle e,p,o\rangle \in T'\}$
9	Frequency of a given URI u	$freq(u) = \|\{t \in T' \mid t = \langle s,p,o\rangle, s = u \text{ or } o = u\}\|$

Table 1: Notations-Metadata

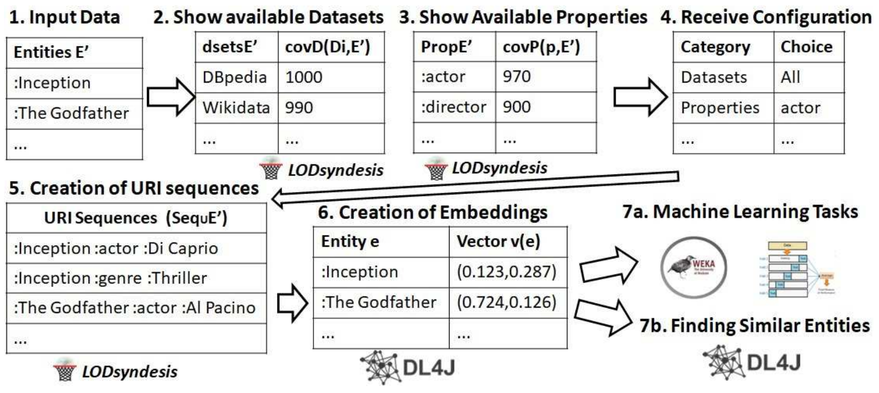

Fig. 2: The steps of `LODVec` approach.

3.3 The Steps & Algorithm for Creating URI Sequences

Here, we describe the functionality and all the steps of `LODVec` (see Fig. 2).

Steps 1-4. Input & Configuration. The first step (see Fig. 2) is to receive the input entities E' (we support over 400 million entities), which can be given a) as a list of URIs (e.g., dbp:Inception), b) as a list of entities in plain text (e.g., "Inception"), or c) just a URI that represents an RDF class or a category (e.g., dbc:Films_about_dreams)! In the latter case, `LODVec` retrieves automatically the desired URIs. By exploiting `LODsyndesis`, `LODVec` can optionally show to the user (i) the datasets containing the input entities (i.e., $dsets_{E'}$), in descending order according to their $covD(D_i, E')$, and (ii) the list of available properties for the given entities, i.e., $Prop_{E'}$, in descending order according to their $covP(p, E')$ (i.e., Steps 2 and 3). In Step 4 of Fig. 2, the user selects which configuration will be used, i.e., the input of Alg. 1. In particular, one can select the datasets D_{sel} that will be used ($D_{sel} \subseteq dsets_{E'}$), the desired properties $Prop_{sel}$ ($Prop_{sel} \subseteq Prop_{E'}$) and whether all the possible sequences or the top-K ones will be created. For creating the top-K URI sequences, one should give as input the exact number K (i.e., a positive integer), and the method to sort the triples according to the frequency of each triple's object (i.e., in ascending, descending or random order).

Step 5. Algorithm for Creating URI sequences. Alg. 1 creates URI sequences for a set of entities E'. It iterates over all the input entities and for

Algorithm 1: Creating *URI sequences* for a set of entities E'

```
Input: Entities $E'$, properties $Prop_{sel}$, datasets $D_{sel}$, cardinality $crd$ ({"All" | "Top-K"}), Integer $K$, sortType ({"asc"| "desc"| "rand"})
Output: URI Sequences $Seq_U E'$ for all the entities $E$
1  $Seq_U E' \leftarrow \emptyset$
2  forall $e \in E'$ do
3      $Seq_U(e) \leftarrow \emptyset$, $freqMap(e) \leftarrow \emptyset$
4      forall $p \in (Prop(e) \cap Prop_{sel})$ do
5          forall $o \in Obj(e,p), prov(\langle e,p,o \rangle) \cap D_{sel} \neq \emptyset$ do
6              if $crd \equiv$ "All" then
7                  $Seq_U(e) \leftarrow Seq_U(e) \cup \{\langle e,p,o \rangle\}$
8              else if $crd \equiv$ "Top-K" then
9                  $freqMap(e) \leftarrow freqMap(e) \cup \{\langle e,p,o \rangle, freq(o)\}$
10     if $crd \equiv$ "Top-K" then
11         $sortSeq(e) \leftarrow mergeSortByValue(freqMap(e))_{sortType}$
12         forall $\langle e,p,o \rangle \in Left(sortSeq(e))$ do
13             $Seq_U(e) \leftarrow Seq_U(e) \cup \{\langle e,p,o \rangle\}$
14             if $|Seq_U(e)| \equiv K$ then
15                 break
16     $Seq_U E' \leftarrow Seq_U E' \cup Seq_U(e)$
17 Return $Seq_U E'$
```

each entity e (lines 2-3), it initializes its corresponding set of URI sequences and a frequency map (it is used for creating only the top-K sequences). The next step is to traverse each direct or indirect property p of entity e (i.e., $Prop(e)$) that belong also to the desired properties given by the user (i.e., $Prop_{sel}$). The indexes of `LODsyndesis` store in the same place all the objects for each entity-property pair (e.g., movie-actor) [11]. Therefore, for each p we traverse all the possible objects, and we check if at least one dataset containing that triple, belongs to the desired datasets D_{sel} (lines 4-5). In such a case, we check the cardinality variable, i.e., if the user desires to find all the possible URI sequences, we add the URI sequence (i.e., triple) to $Seq_u(e)$ (lines 6-7). Otherwise, we add the triple and its object's frequency, i.e., $freq(o)$ (retrieved by sending a request to `LODsyndesis` REST services [12]), to the frequency map (lines 8-9). After traversing all the desired triples for the entity e, if the user wants to create the top-K URI sequences, we sort the values according to the given sortType (lines 10-11), e.g., if sortType equals "desc", we will sort the K triples, which are found in the left side of $freqMap(e)$, in descending order with respect to their object's frequency. We iterate over the sorted map (i.e., $sortSeq(e)$), that contains in its left side (i.e., $Left$) a URI sequence and in its right side the $freq(o)$, and we add to $Seq_u(e)$ the top-K results (lines 12-15). Finally, we add the $Seq_U(e)$ to $Seq_U E'$, whereas after all the iterations, Alg. 1 returns all the URI sequences.

The time complexity of Alg. 1 is $\mathcal{O}(|triples_{E'}|)$, since in the worst case we read all the triples for the entities E'. The space complexity is $\mathcal{O}(Seq_U E')$, since we keep in memory all the produced sequences. We do not load in memory the triples of each entity, since we use random access methods for reading the desired part of the indexes. Finally, for creating the top-K sequences for each entity e we also use a sorting algorithm whose time complexity is $\mathcal{O}(n * log_n)$.

Steps 6-7. Converting URI Sequences to Embeddings & Exploitation. The next step (Step 6 of Fig. 2) is to exploit the produced set $Seq_U E'$ for creating one vector per entity e. For this reason, we use the *word2vec* implementation of *dl4j* library, which produces as output a vector $v(e)$ for each $e \in E'$. The produced vectors can be exploited for (i) machine-learning tasks and (ii) for similarity tasks. Regarding task (i), the vectors can be downloaded in ".arff" format, which can be used as input for *WEKA* API, for performing classification and regression (Step 7a of Fig. 2), while `LODVec` offers a service for producing automatically such results. Concerning task (ii), they can be downloaded in ".txt" format that is directly accessible from *dl4j* API, whereas `LODVec` offers a service for finding the top-K related entities to a given one (Step 7b of Fig. 2).

Demo & Video. A demo video of `LODVec` is accessible in the following link: `https://youtu.be/qR9RFZVs4TY`. An online demo (i.e., a JAVA web application) that will allow anyone to create URI embeddings for over 400 million entities will be released soon in `https://www.ics.forth.gr/isl/LODsyndesis/`.

4 Experimental Evaluation

In Section 4.1, we evaluate the impact of cross-dataset reasoning and of using multiple datasets, for four machine-learning tasks, while Section 4.2 shows a small example for the task of finding similar entities. All the experiments were performed on a single machine with an *i5 core*, *8GB RAM*, and *1 TB disc space*.

4.1 Machine Learning Tasks - The Gain of Using Multiple Datasets

Metacritic Movies & Music Albums Datasets. We use the Metacritic Movies and Music Albums datasets (derived from [18]), which contain the DBpedia URIs of 2,000 movies and of 1,600 music albums. Both datasets contain an average rating of all time reviews for each movie and music album. Concerning movies, 1,000 of them have high rating, i.e., > 60, and the remaining 1,000 ones have low rating, i.e., < 40. Regarding music albums, 800 of them have high rating, i.e., > 79, and the remaining 800 ones have low rating, i.e. < 63. The goal of (binary) classification is to predict whether a movie or a music album has a high or a low rating, whereas the target of regression is to find the exact average rating of each movie and music album.

Word2Vec Parameters. We exploit *word2vec* algorithm from *dl4j* library, since we expect that movies and music albums with similar rating will be placed closely in vector space. For building the model, we use the skip-gram model, we exclude URIs that exist less than 5 times in the produced sequences (i.e., $minWordFrequency = 5$), we use 10 iterations and we select the window size parameter to be 2, i.e., since we build small URI sequences. For each movie and music album we produce a single vector $v(e)$ having 100 dimensions.

Datasets	Prop.	*crd*	$\lvert Seq_U(e)\rvert$ Average	Creation Time	NB (CF)	SVM (CF)	LR (Reg)	SMOreg (Reg)
Freebase (FR)	All	All	112.0	4.3 min	79.71%	82.02%	16.37	16.56
DBpedia (DB)	All	All	23.8	4.0 min	68.42%	71.14%	20.02	20.71
Wikidata (WK)	All	All	22.5	4.2 min	66.88%	67.66%	20.81	21.40
DB,WK	All	All	38.3	4.4 min	68.62%	74.92%	19.18	19.90
DB,FR	All	All	132.0	4.5 min	79.75%	82.51%	16.11	16.35
FR,WK	All	All	129.0	4.6 min	81.20%	83.32%	16.25	16.55
DB,FR,WK	All	All	144.7	4.7 min	82.11%	84.10%	16.01	16.31
All 14 $dsets(E')$	All	All	**170.1**	5.7 min	**82.41%**	**84.70%**	**15.57**	**15.65**
FR	type	All	5.5	3.5 min	67.49%	71.40%	19.18	19.59
All 14 $dsets(E')$	type	All	18.3	4.0 min	67.53%	72.92%	18.90	19.42
All 14 $dsets(E')$	ct+tp	All	30.9	4.7 min	73.13%	74.90%	18.80	19.04
All 14 $dsets(E')$	All	30 asc	30.0	120 min	66.95%	72.50%	19.58	20.05
All 14 $dsets(E')$	All	30 rand	30.0	6.0 min	69.24%	73.10%	19.86	20.26
All 14 $dsets(E')$	All	30 desc	30.0	120 min	71.43%	75.86%	19.07	19.46

Table 2: Classification and regression experiments on Movies dataset

Machine Learning Models & Metrics. The produced vectors are given as input in WEKA API [20], for performing classification and regression by using a 10-fold cross validation [20]. Regarding classification (CF), we use the default implementation of *Naive Bayes* (NB) and *Support Vector Machine* (SMO) of WEKA, and we measure the accuracy percentage of each model (percentage of correct predictions), i.e., the goal is to maximize that percentage. Concerning regression (Reg), we use the default implementation of *Linear Regression* (LR) and *Support Vector Machine for Regression* (SMOreg) of WEKA, and we measure the root mean squared error (RMSE), i.e., the target is to minimize the RMSE value. Finally, we measure the accuracy and the RMSE value for the trivial *Vote* method, that selects randomly a class and a rating for each entity.

Results for both Tasks and Datasets. Tables 2 and 3 show several statistics and experiments for movies and music albums, respectively. In each Table, the first column shows the used RDF datasets, the second the properties, the third one the cardinality, the fourth one the average URI sequences per entity, and the fifth one the total creation time of all URI sequences. The columns 6 and 7 show the accuracy of the classification task (SF) and the last two columns the RMSE value for the regression task (Reg), by using different models.

Results for Movies. In rows 2-9 of Table 2, we show experiments by using all the possible properties and by creating all the possible URI sequences, however, in each case we use a different subset of datasets. We can see how the average number of URI sequences increases as we add more datasets (see the fourth column of Table 2), e.g., by using *DBpedia* we created 23.8 URI sequences per movie, whereas with all the datasets we created on average 170.1 URI sequences. Moreover, as we add more datasets, the total time for creating all the URI sequences did not increase so much, i.e., by using only *DBpedia* we needed 4 minutes (i.e., 0.12 seconds per entity), whereas by using all datasets the corresponding time was 5.7 minutes (i.e., 0.17 seconds per entity).

Classification and Regression Results. First, for the trivial *Vote* method, we obtained 50% accuracy, whereas the *RMSE* value was 23.1. The single dataset with the highest accuracy and the lowest RMSE was *FreeBase* (`http://freebase.com`), i.e., we obtained 82% accuracy through *SVM* model, whereas its RMSE value was 16.37 (through *LR* model). The corresponding percentages for *DBpedia* and *Wikidata* were much smaller. However, by taking each pair of these 3 datasets, the accuracy increased, and the RMSE value decreased in all cases (versus using only one dataset from each pair). Certainly, by using only *FreeBase*, we achieved better results comparing to use both *DBpedia* and *Wikidata*, which seems rational, since from *Freebase* we created a large number of sequences. However, by combining *Freebase* with either *DBpedia* or *Wikidata*, or by using all 3 datasets (these combinations are feasible due to cross-dataset reasoning), we obtained better results. By using all the 14 available datasets (out of 400 datasets) having data about these movies, we achieved the highest accuracy (84.7%) and the lowest RMSE (15.57). However, we should note that 8 of these 14 datasets offered very few URI sequences for this task. Finally, *SVM* outperformed *NB* in all classification cases, while *LR* was more effective in regression tasks.

Other Configurations. In rows 10-15 of Table 2, we show indicatively some experiments with different configurations. By creating URI sequences containing only the property *rdf:type*, we obtained better results by using all datasets in comparison to use only *Freebase*, while by creating sequences that contain both *rdf:type* and *dcterms:subject* (ct+tp) the results improved. Regarding experiments containing the top-K sequences, by creating only the top-30 URI sequences according to objects frequency for each movie in descending order (i.e., triples with the 30 most popular objects per movie), we achieved the highest accuracy and the lowest RMSE. On the contrary, a random order was more effective versus an ascending one. Concerning the creation time of *desc* and *asc*, is it slower versus the other configurations (i.e., 3.6 seconds per entity), since we send several requests to `LODsyndesis` [12] for retrieving the frequency of objects.

Results for Music Albums. Table 3 shows experiments by creating all the URI sequences, each time by using a different subset of datasets. By using only *DBpedia* (see the fourth column), we created on average 16.3 URI sequences per album, whereas by using all the datasets, we created 35.9. Concerning the creation time of URI sequences, it is again low (i.e., 0.11 seconds per entity).

Classification and Regression Results. For the trivial *Vote* method, we obtained 50% accuracy, whereas the *RMSE value* was 13.95. The single dataset having the best performance was *DBpedia* (see Table 3), whereas *Freebase* was not so accurate for this task. Therefore, even by selecting to use exactly one dataset for both movies and music albums (i.e., the same dataset in both cases), we will not be able to obtain good results for both tasks. Similarly to movies, as we add more datasets, the results are better for both regression and classification, except for the pairs containing the dataset *Wikidata*. By including all the 5 available datasets for music albums, we obtain the highest accuracy (i.e., 71.31%) and the lowest *RMSE* value (i.e., 12.41). Finally, similarly to the case of movies, the best model was the *SVM* for classification and the *LR* for regression.

Datasets	Prop.	*crd*	$\lvert Seq_U(e)\rvert$ Average	Creation Time	NB (CF)	SVM (CF)	LR (Reg)	SMOreg (Reg)
Freebase (FR)	All	All	9.2	2.1 min	52.75%	56.57%	13.74	14.08
DBpedia (DB)	All	All	16.3	2.3 min	64.01%	68.21%	12.75	13.07
Wikidata (WK)	All	All	6.7	1.9 min	60.38%	61.37%	13.89	14.68
DB,WK	All	All	20.5	2.3 min	63.42%	67.61%	12.85	13.10
DB,FR	All	All	25.5	2.3 min	64.30%	68.64%	12.60	13.03
FR,WK	All	All	16.0	2.2 min	54.10%	57.65%	14.00	14.50
DB,FR,WK	All	All	29.8	2.4 min	64.73%	69.02%	12.55	13.01
All 5 $dsets(E')$	All	All	**35.9**	2.5 min	**67.21%**	**71.31%**	**12.41**	**12.72**

Table 3: Classification and regression experiments on Music Albums dataset

Datasets	Position 1	Position 2	Position 3	Position 4	Position 5
DB	Pink Panther 2	**Ratatouille**	Shrek 2	Rain Mant	**Space Chimps**
DB,FR	**Finding Nemo**	**Toy Story 2**	**Incredibles**	**Toy Story**	Princess and the Frog
All	**Finding Nemo**	**Ratatouille**	**Incredibles**	**Toy Story 3**	**Toy Story**

Table 4: Top-5 related movies to “Wall-E” movie by using different datasets

4.2 Finding Similar Entities

`LODVec` can produce the top-K similar entities for a given one, since it uses *word2vec* from *dl4j* API. Table 4 shows an indicative example, i.e., finding the 5 most similar movies (from the set of 2,000 movies) to the animated movie “WALL-E”, by creating all the URI sequences and by using a) only *DBpedia*, b) *DBpedia* and *Freebase* and c) all datasets. For evaluating the results, we typed in *Google Search Engine* the keywords “WALL-E related movies”, and we retrieved a list with related movies that “People also search for WALL-E”. By using only *DBpedia*, 2 of 5 movies were in the list of related movies (the bold ones in Table 4), whereas by using both *Freebase* and *DBpedia*, 4 of 5 movies were at that list. Finally, by exploiting all the datasets, all the 5 movies were part of the list.

5 Conclusion

There is a lack of approaches that create URI embeddings from multiple RDF datasets. For this reason, we introduced a prototype called `LODVec` that exploits the semantically enriched indexes of `LODsyndesis` knowledge graph, and offers configurable options for creating URI sequences and embeddings through *word2vec* algorithm, for over 400 million entities from 400 RDF datasets. The produced embeddings can be exploited in several tasks. In our case, we evaluated the gain of using multiple datasets (and cross-dataset reasoning) for four machine-learning tasks, e.g., for classifying whether a movie has a high or low rating. Indicatively, by using data from *DBpedia* we created 23.8 URI sequences per movie, while by combining information from 14 datasets, the corresponding number was 170.1. We identified even 13% increase in the accuracy of predicting if a movie is highly rated by using multiple datasets, instead of using only *DBpedia*. As a future work, we plan (a) to create longer URI sequences, (b) to create vectors through other models, such as *GloVe* [15], and (c) to apply graph-based techniques, such as [3,19]. Finally, our goal is to train the whole knowledge graph for retrieving the top-K similar entities for any given entity quickly.

Acknowledgements. The research work was supported by the Hellenic Foundation for Research and Innovation (HFRI) and the General Secretariat for Research and Technology (GSRT), under the HFRI PhD Fellowship grant (GA. No. 166).

References

1. Antoniou, G., van Harmelen, F.: A Semantic Web Primer, 2nd Edition. The MIT Press (2008)
2. Cochez, M., Ristoski, P., Ponzetto, S.P., Paulheim, H.: Biased graph walks for RDF graph embeddings. In: WIMS. p. 21. ACM (2017)
3. Cochez, M., Ristoski, P., Ponzetto, S.P., Paulheim, H.: Global RDF vector space embeddings. In: ISWC. pp. 190–207. Springer (2017)
4. Dietze, S., Mohapatra, N., Iosifidis, V., Ekbal, A., Fafalios, P.: Time-aware and corpus-specific entity relatedness. pp. 33–39 (2018)
5. Hajra, A., Tochtermann, K.: Linking science: approaches for linking scientific publications across different LOD repositories. IJMSO **12**(2-3), 124–141 (2017)
6. Inan, E., Dikenelli, O.: Effect of enriched ontology structures on RDF embedding-based entity linking. In: MTSR Conference. pp. 15–24. Springer (2017)
7. Lin, Y., Liu, Z., Sun, M., Liu, Y., Zhu, X.: Learning entity and relation embeddings for knowledge graph completion. In: AAAI conference (2015)
8. Mikolov, T., Chen, K., Corrado, G., Dean, J.: Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781 (2013)
9. 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)
10. Mountantonakis, M., Tzitzikas, Y.: How linked data can aid machine learning-based tasks. In: TPDL International conference. pp. 155–168. Springer (2017)
11. Mountantonakis, M., Tzitzikas, Y.: High performance methods for linked open data connectivity analytics. Information **9**(6), 134 (2018)
12. Mountantonakis, M., Tzitzikas, Y.: LODsyndesis: Global scale knowledge services. Heritage **1**(2), 335–348 (2018)
13. Mountantonakis, M., Tzitzikas, Y.: Large scale semantic integration of linked data: A survey. ACM Computing Surveys (accepted for publication) (2019)
14. Nechaev, Y., Corcoglioniti, F., Giuliano, C.: Type prediction combining linked open data and social media. In: CIKM. pp. 1033–1042. ACM (2018)
15. Pennington, J., Socher, R., Manning, C.: Glove: Global vectors for word representation. In: Proceedings of EMNLP conference. pp. 1532–1543 (2014)
16. Ristoski, P., Bizer, C., Paulheim, H.: Mining the web of linked data with rapidminer. Journal of Web Semantics **35**, 142–151 (2015)
17. 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)
18. Ristoski, P., et al.: A collection of benchmark datasets for systematic evaluations of machine learning on the semantic web. In: ISWC. pp. 186–194. Springer (2016)
19. Wang, Z., Zhang, J., Feng, J., Chen, Z.: Knowledge graph embedding by translating on hyperplanes. In: AAAI conference on artificial intelligence (2014)
20. Witten, I.H., Frank, E., Hall, M.A., Pal, C.J.: Data Mining: Practical machine learning tools and techniques. Morgan Kaufmann (2016)