Semantic Web 0 (0) 1
IOS Press

Knowledge Graph Embedding for Data Mining vs. Knowledge Graph Embedding for Link Prediction – Two Sides of the Same Coin?

Jan Portisch [a,b] and Nicolas Heist [a] and Heiko Paulheim [a,*]
[a] *Data and Web Science Group, University of Mannheim, Germany*
E-mails: jan@informatik.uni-mannheim.de, nico@informatik.uni-mannheim.de, heiko@informatik.uni-mannheim.de
[b] *SAP SE, Germany*
E-mail: jan.portisch@sap.com

Abstract. Knowledge Graph Embeddings, i.e., projections of entities and relations to lower dimensional spaces, have been proposed for two purposes: (1) providing an encoding for data mining tasks, and (2) predicting links in a knowledge graph. Both lines of research have been pursued rather in isolation from each other so far, each with their own benchmarks and evaluation methodologies. In this paper, we argue that both tasks are actually related, and we show that the first family of approaches can also be used for the second task and vice versa. In two series of experiments, we provide a comparison of both families of approaches on both tasks, which, to the best of our knowledge, has not been done so far. Furthermore, we discuss the differences in the similarity functions evoked by the different embedding approaches.

Keywords: Knowledge Graph Embedding, Link Prediction, Data Mining, RDF2vec

1. Introduction

In the recent past, the topic of knowledge graph embedding – i.e., projecting entities and relations in a knowledge graph into a numerical vector space – has gained a lot of traction. An often cited survey from 2017 [1] lists already 25 approaches, with new models being proposed almost every month, as depicted in Fig. 1.

Even more remarkably, two mostly disjoint strands of research have emerged in that vivid area. The first family of research works focus mostly on *link prediction* [2], i.e., the approaches are evaluated in a knowledge graph refinement setting [3]. The optimization goal here is to distinguish correct from incorrect triples in the knowledge graph as good as possible. The evaluations of this kind of approaches are always conducted within the knowledge graph, using the existing knowledge graph assertions as ground truth.

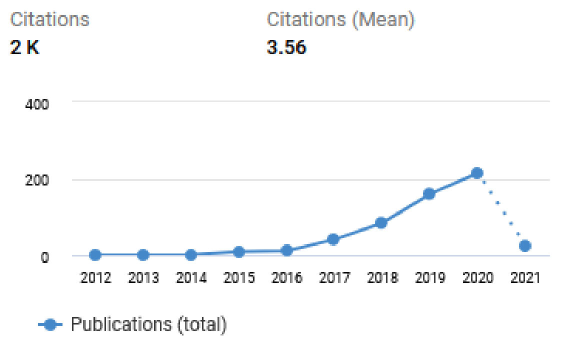

Fig. 1. Publications with *Knowledge Graph Embedding* in their title or abstract, created with dimensions.ai

[*]Corresponding author. E-mail: heiko@informatik.uni-mannheim.de.

A second strand of research focuses on the embedding of entities in the knowledge graph for downstream tasks outside the knowledge graph, which often come from the data mining field – hence, we coin this family of approaches *embeddings from data mining*. Examples include: the prediction of external variables for entities in a knowledge graph [4], information retrieval backed by a knowledge graph [5], or the usage of a knowledge graph in content-based recommender systems [6]. In those cases, the optimization goal is to create an embedding space which reflects *semantic similarity* as good as possible (e.g., in a recommender system, similar items to the ones in the user interest should be recommended). The evaluations here are always conducted outside the knowledge graph, based on external ground truth.

In this paper, we want to look at the commonalities and differences of the two approaches. We look at two of the most basic and well-known approaches of both strands, i.e., *TransE* [7] and *RDF2vec* [4], and analyze and compare their optimization goals in a simple example. Moreover, we analyze the performance of approaches from both families in the respective other evaluation setup: we explore the usage of link-prediction based embeddings for other downstream tasks based on similarity, and we propose a link prediction method based on RDF2vec. From those experiments, we derive a set of insights into the differences of the two families of methods, and a few recommendations on which kind of approach should be used in which setting.

2. Related Work

As pointed out above, the number of works on knowledge graph embedding is legion, and enumerating them all in this section would go beyond the scope of this paper. However, there have already been quite a few survey articles.

The first strand of research works – i.e., knowledge graph embeddings for link prediction – has been covered in different surveys, such as [1], and, more recently, [8], [9], and [10]. The systematic of those reviews is similar, as they distinguish different families of approaches: *translational distance models* [1] or *geometric models* [9] focus on link prediction as a geometric task, i.e., projecting the graph in a vector space so that a translation operation defined for relation r on a head h yields a result close to the tail t.

The second family among the link prediction embeddings are *semantic matching* [1] or *matrix factorization or tensor decomposition* [9] models. Here, a knowledge graph is represented as a three-dimensional tensor, which is decomposed into smaller matrices or tensors. The reconstruction operation can then be used for link prediction.

The third and youngest family among the link prediction embeddings are based on deep learning and graph neural networks. Here, neural network training approaches, such as convolutional neural networks, capsule networks, or recurrent neural networks, are adapted to work with knowledge graphs [9].

While most of those approaches only consider graphs with nodes and edges, most knowledge graphs also contain literals, e.g., strings and numeric values. [11] shows a survey of approaches which take such literal information into account. It is also one of the few review articles which considers embedding methods from both research strands.

Link prediction is typically evaluated on a set of standard datasets, and uses a within-KG protocol, where the triples in the knowledge graph are divided in a training, testing, and validation set. Prediction accuracy is then assessed on the validation set. Datasets commonly used for the evaluation are FB15k, which is a subset of Freebase, and WN18, which is derived from WordNet [7]. Since it has been remarked that those datasets contain too many simple inferences due to inverse relations, the more challenging variants FB15k-237 [12] and WN18RR [13] have been proposed. More recently, evaluation sets based on larger knowledge graphs, such as YAGO3-10 [13] and DBpedia50k/DBpedia500k [14] have been introduced.

The second strand of research works, focusing on the embedding for downstream tasks (which are often from the domain of data mining), is not as extensively reviewed, and the number of works in this area are still smaller. One of the more comprehensive evaluations is shown in [15], which is also one of the rare works which includes approaches from both strands in a common evaluation. They show that at least the three methods for link prediction used – namely TransE, TransR, and TransH – perform inferior on downstream tasks, compared to approaches developed specifically for optimizing for entity similarity in the embedding space.

For the evaluation of entity embeddings optimized for entity similarity, there are quite a few use cases at hand. The authors in [16] list a number of tasks, including classification and regression of entities based on external ground truth variables, entity clustering, as well as identifying semantically related entities.

Works which explicitly compare approaches from both research strands are still rare. In [17], an in-KG scenario, i.e., the detection and correction of erroneous links, is considered. The authors compare RDF2vec (with an additional classification layer) to TransE and DistMult on the link prediction task. The results are mixed: While RDF2vec outperforms TransE and DistMult in terms of Mean Reciprocal Rank and Precision@1, it is inferior in Precision@10. Since the results are only validated on one single dataset, the evidence is rather thin.

In [18], the authors analyze the vector spaces of different embedding models with respect to class separation, i.e., they fit the best linear separation between classes in different embedding spaces. According to their findings, RDF2vec achieves a better linear separation than the models tailored to link prediction.

Overall, while there are quite a few works using and evaluating approaches from both research strands, direct comparisons are rather rare. This paper aims at closing that gap.

3. Knowledge Graph Embedding Methods for Data Mining

Traditionally, most data mining methods are working on propositional data, i.e., each instance is a row in a table, described by a set of (binary, numeric, or categorical) features. For using knowledge graphs in data mining, one needs to either develop methods which work on graphs instead of propositional data, or find ways to represent instances of the knowledge graph as feature vectors [19]. The latter is often referred to as *propositionalization* [20].

RDF2vec [4] is a prominent example from the second family. It adapts the word2vec approach [21] for deriving word embeddings (i.e., vector representations for words) from a corpus of sentences. RDF2vec creates such sentences by performing *random walks* on an RDF graph and collecting the sequences of entities and relations, then trains a word2vec model on those sequences. It has been shown that this strategy outperforms other strategies of propositionalization. The relation between propositionalization and embedding methods has also recently been pointed out by [22].

3.1. Data Mining is based on Similarity

Predictive data mining tasks are predicting classes or numerical values for instances. A typical application are recommender systems: given the set of items (e.g., movies, books) a user liked, predict whether s/he likes a particular other item (i.e., make a binary prediction: yes/no), or, given the (numerical) ratings a user gave to items in the past, predict the rating s/he will give to other instances. Such ratings can then be used to create a sorted list of recommendations.

Content-based recommender systems are one family of recommender systems. They rely on *item similarity*, i.e., if a user liked item A in the past, the system will recommend items which are similar to A. To that end, each item is represented as a set of features, and items with similar features are considered similar.

RDF2vec has been shown to be usable for recommender systems, since the underlying method tends to create similar vectors for similar entities, i.e., position them closer in vector space [6]. Figure 2 illustrates this using a 2D PCA plot of RDF2vec vectors for movies in DBpedia. It can be seen that clusters of movies, e.g., Disney movies, Star Trek movies, and Marvel related movies are formed.

Similar to recommender systems, many techniques for predictive data mining rely on similarity in one or the other way. This is more obvious for, e.g., k-nearest neighbors, where the predicted label for an instance is the majority or average of labels of its closest neighbors (i.e., most similar instances), or Naive Bayes, where an instance is predicted to belong to a class if its feature values are most similar to the typical distribution of features for this class (i.e., it is similar to an average member of this class). A similar argument can be made for neural networks, where one can assume a similar output when changing the value of one input neuron (i.e., one feature value) by a small delta. Other classes of approaches (such as Support Vector Machines) use the concept of *class separability*, which is similar to exploiting similarity: datasets with well separable classes have similar instances (belonging to the same class) close to each other, while dissimilar instances (belonging to different classes) are further away from each other [23].

3.2. Creating Similar Embeddings for Similar Instances

To understand how (and why) RDF2vec creates embeddings that have the property of placing nearby vec-

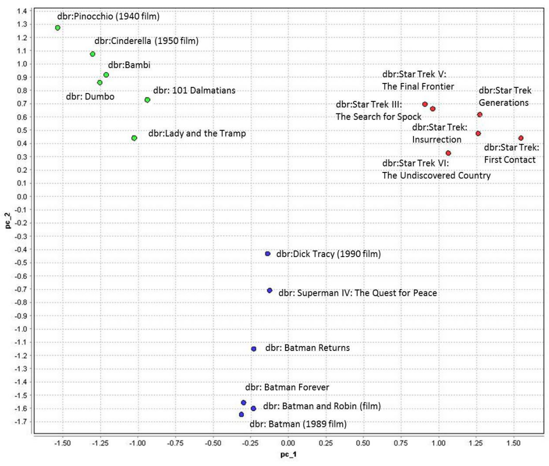

Fig. 2. RDF2vec embeddings for movies in DBpedia, from [6].

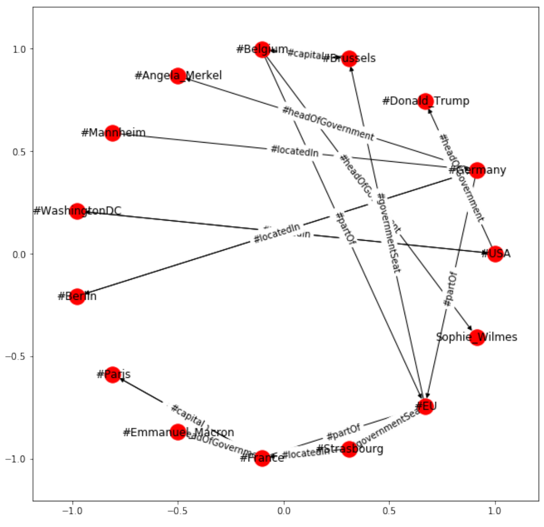

Fig. 3. Example graph used for illustration

tors, we use the running example depicted in Fig. 3, showing a number of European cities, countries, and heads of those governments.

As discussed above, the first step of RDF2vec is to create random walks on the graph. To that end, RDF2vec starts a fixed number of random walks of a fixed maximum length from each entity. Since the example above is very small, we will, for the sake of illustration, enumerate *all* walks of length 4 that can be created for the graph. Those walks are depicted in Fig. 5. It is notable that, since the graph has nodes without outgoing edges, some of the walks are actually shorter than 4.

In the next step, the walks are used to train a predictive model. Since RDF2vec uses word2vec, it can be trained with the two flavors of word2vec, i.e., CBOW (context back of words) and SG (skip gram). The first predicts a word, given its surrounding words, the second predicts the surroundings, given a word. For the sake of our argument, we will only consider the second variant, depicted in Fig. 6. Simply speaking, given training examples where the input is the target word (as a one-hot-encoded vector) and the output is the context words (again, one hot encoded vectors), a neural network is trained, where the hidden layer is typically of smaller dimensionality than the input. That hidden layer is later used to produce the actual embedding vectors.

To create the training examples, a window with a given size is slid over the input sentences. Here, we use a window of size 2, which means that the two words preceding and the two words succeeding a context word are taken into consideration. Fig. 7 shows the training examples generated for three instances.

A model that learns to predict the context given the target word would now learn to predict the *majority* of the context words for the target word at hand at the output layer called *output* in Fig. 6, as depicted in the lower part of Fig. 7. Here, we can see that Paris and Berlin share two out of four predictions, so do Mannheim and Berlin. Angela Merkel and Berlin share one out of four predictions.

Considering again Fig. 6, given that the activation function which computes the *output* from the *projection* values is continuous, it implies that similar activations on the output layer requires similar values on the projection layer. Hence, for a well fit model, the distance on the projection layer of Paris, Berlin, and Mannheim should be comparatively lower than the distance of the other entities, since they activate similar outputs.[1]

Fig. 8 depicts a two-dimensional RDF2vec embedding learned for the example graph.[2] We can observe that there are clusters of persons, countries, and cities. The grouping of similar objects also goes further – we

[1]Note that there are still weights learned for the individual connections between the projection and the output layer, which emphasize some connections more strongly than others. Hence, we cannot simplify our argumentation in a way like "with two common context words activated, the entities must be projected twice as close as those with one common context word activated".

[2]Created with PyRDF2vec [24], using two dimensions, a walk length of 8, and standard configuration otherwise

```
Berlin locatedIn Germany .
Germany headOfGovernment Angela_Merkel .
Mannheim locatedIn Germany .
Belgium capital Brussels .
Germany partOf EU .
Belgium partOf EU .
Belgium headOfGovernment Sophie_Wilmes .
EU governmentSeat Brussels .
USA capital WashingtonDC .
WashingtonDC locatedIn USA .
France capital Paris .
France headOfGovernment Emmanuel_Macron .
Paris locatedIn France .
Strasbourg locatedIn France .
Germany capital Berlin .
Brussels locatedIn Belgium .
France partOf EU .
USA headOfGovernment Donald_Trump .
EU governmentSeat Strasbourg .
```

Fig. 4. Triples of the example Knowledge Graph

```
Belgium partOf EU governmentSeat Brussels
Belgium capital Brussels locatedIn Belgium
Belgium partOf EU governmentSeat Strasbourg
Belgium headOfGovernment Sophie_Wilmes
Berlin locatedIn Germany capital Berlin
Berlin locatedIn Germany headOfGovernment Angela_Merkel
Berlin locatedIn Germany partOf EU
Brussels locatedIn Belgium headOfGovernment Sophie_Wilmes
Brussels locatedIn Belgium partOf EU
Brussels locatedIn Belgium capital Brussels
EU governmentSeat Strasbourg locatedIn France
EU governmentSeat Brussels locatedIn Belgium
France headOfGovernment Emmanuel_Macron
France capital Paris locatedIn France
France partOf EU governmentSeat Brussels
France partOf EU governmentSeat Strasbourg
Germany partOf EU governmentSeat Brussels
Germany partOf EU governmentSeat Strasbourg
Germany capital Berlin locatedIn Germany
Germany headOfGovernment Angela_Merkel
Mannheim locatedIn Germany capital Berlin
Mannheim locatedIn Germany headOfGovernment Angela_Merkel
Mannheim locatedIn Germany partOf EU
Paris locatedIn France headOfGovernment Emmanuel_Macron
Paris locatedIn France partOf EU
Paris locatedIn France capital Paris
Strasbourg locatedIn France capital Paris
Strasbourg locatedIn France headOfGovernment Emmanuel_Macron
Strasbourg locatedIn France partOf EU
USA headOfGovernment Donald_Trump
USA capital Washington_DC locatedIn USA
Washington_DC locatedIn USA capital Washington_DC
Washington_DC locatedIn USA headOfGovernment Donald_Trump
```

Fig. 5. Walks extracted from the example graph

can, e.g., observe that European cities in the dataset are embedded closer to each other than to Washington D.C. This is in line with previous observations showing that RDF2vec is particularly well suited in creating clusters also for finer-grained classes [25]. A predictive model could now exploit those similarities, e.g., for type prediction, as proposed in [26] and [25].

3.3. *Usage for Link Prediction*

From Fig. 8, we can assume that link prediction should, in principle, be possible. For example, the predictions for heads of governments all point in a similar direction. This is in line with what is known about word2vec, which allows for computing analogies, like

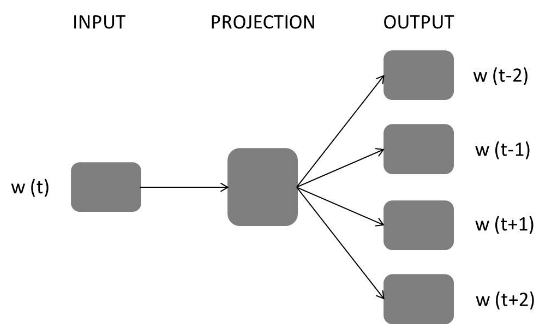

Fig. 6. The Skip Gram variant of word2vec [4]

the well-known example

$$v(King) - v(Man) + v(Woman) \approx v(Queen) \quad (1)$$

RDF2vec does not learn relation embeddings, only entity embeddings.[3] Hence, we cannot directly predict relations, but we can use those analogies. If we want to make a tail prediction like

$$< h, r, ? >, \quad (2)$$

we can identify another pair $< h', r, t' >$ and exploit the above analogy, i.e.,

$$t' - h' + h \approx t \quad (3)$$

To come to a stable prediction, we would use the average, i.e.,

$$t \approx \frac{\sum_{<h',r,t'>} t' - h' + h}{|< h', r, t' >|} \quad (4)$$

With the same idea, we can also average the relation vectors r for each relation that holds between all its head and tail pairs, i.e.,

$$r \approx \frac{\sum_{<h',r,t'>} t' - h'}{|< h', r, t' >|}, \quad (5)$$

and thereby reformulate the above equation to

$$t \approx h + r, \quad (6)$$

which is what we expect from an embedding model for link prediction. Those approximate relation vectors for the example at hand are depicted in Fig. 9. We can see that in some (not all) cases, the directions of the vectors are approximately correct: the *partOf* vector is roughly the difference between *EU* and *Germany*, *France*, and *Belgium*, and the *headOfGovernment* vector is approximately the vector between the countries and the politicians cluster. On the other hand, the *capital* vector seems to be less accurate.

It can also be observed that the vectors for *locatedIn* and *capitalOf* point in reverse directions, which makes sense because they form connections between two clusters (countries and cities) in opposite directions.

[3]Technically, we can also make RDF2vec learn embeddings for the relations, but they would not behave the way we need them.

4. Knowledge Graph Embedding Methods for Link Prediction

A larger body of work has been devoted on knowledge graph embedding methods for link prediction. Here, the goal is to learn a model which embeds entities and relations in the same vector space.

4.1. Link Prediction is based on Vector Operations

As the main objective is link prediction, most models, more or less, try to find a vector space embedding of entities and relations so that

$$t \approx h \oplus r \quad (7)$$

holds for as many triples $< h, r, t >$ as possible. $\oplus$ can stand for different operations in the vector space; in basic approaches, simple vector addition ($+$) is used. In our considerations below, we will also use vector addition.

In most approaches, negative examples are created by corrupting an existing triple, i.e., replace the head or tail with another entity from the graph (some approaches also foresee corrupting the relation). Then, a model is learned which tries to tell apart corrupted from non-corrupted triples. The formulation in the original TransE paper [7] defines the loss function L as follows:

$$L = \sum_{(h,r,t)\in S} \sum_{(h',r,t')\in S'} [\gamma + d(h + r, t) - d(h' + r, t')]_+ \quad (8)$$

where γ is some margin, and d is a distance function, i.e., the $L1$ or $L2$ norm. S is the set of statements that

Target Word	w_{-2}	w_{-1}	w_{+1}	w_{+2}
Paris	France	capital	locatedIn	France
Paris	–	–	locatedIn	France
Paris	–	–	locatedIn	France
Paris	–	–	locatedIn	France
Paris	France	capital	–	–
Paris	France	capital	–	–
Berlin	–	–	locatedIn	Germany
Berlin	Germany	capital	–	–
Berlin	–	–	locatedIn	Germany
Berlin	–	–	locatedIn	Germany
Berlin	Germany	capital	locatedIn	Germany
Berlin	Germany	capital	–	–
Mannheim	–	–	locatedIn	Germany
Mannheim	–	–	locatedIn	Germany
Mannheim	–	–	locatedIn	Germany
Angela Merkel	Germany	headOfGovernment	–	–
Angela Merkel	Germany	headOfGovernment	–	–
Angela Merkel	Germany	headOfGovernment	–	–
Donald Trump	USA	headOfGovernment	–	–
Donald Trump	USA	headOfGovernment	–	–
Belgium	–	–	partOf	EU
Belgium	–	–	capital	Brussels
Belgium	Brussels	locatedIn	–	–
Belgium	–	–	partOf	EU
Belgium	–	–	headOfGovernment	Sophie Wilmes
Belgium	Brussels	locatedIn	headOfGovernment	Sophie Wilmes
Belgium	Brussels	locatedIn	partOf	EU
Belgium	Brussels	locatedIn	capital	Brussels
Belgium	Brussels	locatedIn	–	–
Paris	France	capital	locatedIn	France
Berlin	Germany	capital	locatedIn	Germany
Mannheim	–	–	locatedIn	Germany
Angela Merkel	Germany	headOfGovernment	–	–
Donald Trump	USA	headOfGovernment	–	–
Belgium	Brussels	locatedIn	partOf	EU

Fig. 7. Training examples for instances *Paris*, *Berlin*, *Mannheim*, *Angela Merkel*, *Donald Trump*, and *Belgium* (upper part) and majority predictions (lower part).

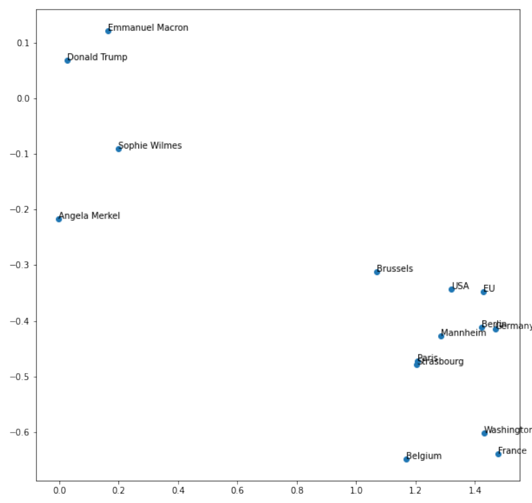

Fig. 8. The example graph embedded with RDF2vec

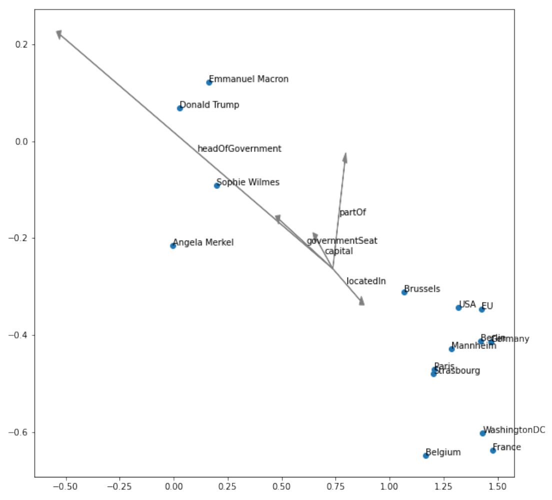

Fig. 9. Average relation vectors for the example

are in the knowledge graph, and S' are the corrupted statements derived from them. In words, the formula states for a triple $< h, r, t >$, $h + r$ should be closer to t than to t' for some corrupted tail, similarly for a corrupted head. However, a difference of γ is accepted.

Fig. 10 shows the example graph from above, as embedded by TransE.[4] Looking at the relation vectors, it

[4]Created with PyKEEN [27], using 128 epochs, a learning rate of 0.1, the softplus loss function, and default parameters otherwise,

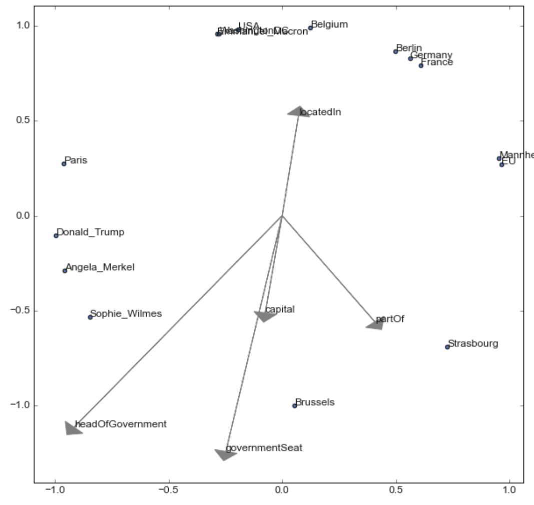

Fig. 10. Example graph embedded by TransE

can be observed that they seem approximately accurate in some cases, e.g.,

$$Germany + headOfGovernment \approx Angela\_Merkel,$$

but not everywhere.[5]

Like in the RDF2vec example above, we can observe that the two vectors for *locatedIn* and *capital* point in opposite directions. Although less obviously than in the RDF2vec example, we can also see that entities in similar classes form (weak) clusters: cities are mostly in the right half of the space, people in the left, countries in the upper part.

4.2. Usage for Data Mining

As discussed above, positioning similar entities close in a vector space is an essential requirement for using entity embeddings in data mining tasks. To understand why an approach tailored towards link prediction can also, to a certain extent, cluster similar instances together, we first rephrase the approximate link prediction equation (8) as

$$t = h + r + \eta_{h,r,t}, \tag{9}$$

where $\eta_{h,r,t}$ can be considered an error term for the triple $< h, r, t >$. Moreover, we define

$$\eta_{max} = \max_{<h,r,t> \in S} \eta_{h,r,t} \tag{10}$$

Next, we consider two triples $< h_1, r, t >$ and $< h_2, r, t >$, which share a relation to an object – e.g., in our example, France and Belgium, which both share the relation *partOf* to *EU*.[6] In that case,

$$t = h_1 + r + \eta_{h_1,r,t} \tag{11}$$

and

$$t = h_2 + r + \eta_{h_2,r,t} \tag{12}$$

hold. From that, we get[7]

$$\begin{aligned} h_1 - h_2 &= \eta_{h_2,r,t} - \eta_{h_1,r,t} \\ \Rightarrow |h_1 - h_2| &= |\eta_{h_2,r,t} - \eta_{h_1,r,t}| \\ &= |\eta_{h_2,r,t} + (-\eta_{h_1,r,t})| \\ &\leqslant |\eta_{h_2,r,t}| + |-\eta_{h_1,r,t}| \\ &= |\eta_{h_2,r,t}| + |\eta_{h_1,r,t}| \\ &\leqslant 2 \cdot \eta_{max} \end{aligned} \tag{13}$$

In other words, η_{max} also imposes an upper bound of two entities sharing a relation to an object. As a consequence, the lower the error in relation prediction, the closer are entities which share a common statement.

This also carries over to entities sharing the same two-hop connection. Consider two further triples $< h_{1a}, r_a, h_1 >$ and $< h_{2a}, r_a, h_2 >$. In our example, this could be two cities located in the two countries, e.g., *Strasbourg* and *Brussels*. In that case, we would have

$$h_1 = h_{1a} + r_a + \eta_{h_{1a},r_a,h_1} \tag{14}$$

$$h_2 = h_{2a} + r_a + \eta_{h_{2a},r_a,h_2} \tag{15}$$

Substituting this in (11) and (12) yields

$$t = h_{1a} + r_a + \eta_{h_{1a},r_a,h_1} + r + \eta_{h_1,r,t} \tag{16}$$

$$t = h_{2a} + r_a + \eta_{h_{2a},r_a,h_2} + r + \eta_{h_2,r,t}. \tag{17}$$

as advised by the authors of PyKEEN: https://github.com/pykeen/pykeen/issues/97

[5]This does not mean that TransE does not work. The training data for the very small graph is rather scarce, and two dimensions might not be sufficient to find a good solution here.

[6]Although, at first glance in Fig. 10, this relation does not seem to hold for *Belgium*, we can see that *EU* is still closer to *Belgium* + *partOf* than most other instances, which is in line with TransE's optimization goal.

[7]Using the triangle inequality for the first inequation.

Consequently, using similar transformations as above, we get

$$\begin{aligned} h_{1a} - h_{2a} &= \eta_{h_{2a},r_a,h_2} - \eta_{h_{1a},r_a,h_1} + \eta_{h_2,r,t} - \eta_{h_1,r,t} \\ \Rightarrow |h_{1a} - h_{2a}| &\leqslant 4 \cdot \eta_{max} \end{aligned} \tag{18}$$

Again, η_{max} constrains the proximity of the two entities h_{1a} and h_{2a}, but only half as strictly as for the case of h_1 and h_2.

4.3. Comparing the Two Notions of Similarity

In the examples above, we can see that embeddings for link prediction have a tendency to project similar instances close to each other in the vector space. Here, the notion of similarity is that two entities are similar if they share a relation to another entity, i.e., e_1 and e_2 are considered similar if there exist two statements $< e_1, r, t >$ and $< e_2, r, t >$ or $< h, r, e_1 >$ and $< h, r, e_2 >$,[8] or, less strongly, if there exists a chain of such statements. More formally, we can write the notion of similarity between two entities in link prediction approaches as

$$e_1 \approx e_2 \leftarrow \exists\, t, r : r(e_1, t) \wedge r(e_2, t) \tag{19}$$

$$e_1 \approx e_2 \leftarrow \exists\, h, r : r(h, e_1) \wedge r(h, e_2) \tag{20}$$

In other words: two entities are similar if they share a common connection to a common third entity.

RDF2vec, on the other hand, covers a wider range of such similarities. Looking at Fig. 7, we can observe that two entities sharing a common relation to two different objects are also considered similar (*Berlin* and *Mannheim* both share the fact that they are *located in Germany*, hence, their predictions for w_{+1} and w_{+2} are similar).

However, there in RDF2vec, similarity can also come in other notions. For example, *Angela Merkel* and *Donald Trump* are also considered similar, because they both share the relation *headOfGovernment*, albeit with different subjects (i.e., their prediction for w_{-1} is similar). In contrast, such similarities do not lead to close projections for link prediction embeddings. In fact, in Fig. 10, it can be observed that *Donald Trump* and *Angela Merkel* are not very close. In other words, the following two notions of similarity also hold for RDF2vec:

$$e_1 \approx e_2 \leftarrow \exists\, t_1, t_2, r : r(e_1, t_1) \wedge r(e_2, t_2) \tag{21}$$

$$e_1 \approx e_2 \leftarrow \exists\, h_1, h_2, r : r(h_1, e_1) \wedge r(h_2, e_2) \tag{22}$$

On a similar argument, RDF2vec also positions entities closer which share *any* relation to another entity. Although this is not visible in the two-dimensional embedding depicted in Fig. 8, RDF2vec would also create vectors with some similarity for *Angela Merkel* and *Berlin*, since they both have a (albeit different) relation to Germany (i.e., their prediction for w_{-2} is similar). Hence, the following notions of similarity can also be observed in RDF2vec:

$$e_1 \approx e_2 \leftarrow \exists\, t, r_1, r_2 : r_1(e_1, t) \wedge r_2(e_2, t) \tag{23}$$

$$e_1 \approx e_2 \leftarrow \exists\, h, r_1, r_2 : r_1(h, e_1) \wedge r_2(h, e_2) \tag{24}$$

The example with *Angela Merkel* and *Berlin* already hints at a slightly different notion of the interpretation of proximity in the vector space evoked by RDF2vec: not only *similar*, but also *related* entities are positioned close in the vector space. This means that to a certain extent, RDF2vec mixes the concepts of similarity and relatedness in its distance function. We will see examples of this in later considerations, and discuss how they interfere with downstream applications.

5. Experiments

To compare the two sets of approaches, we use standard setups for evaluating knowledge graph embedding methods for data mining as well as for link prediction.

5.1. Experiments on Data Mining Tasks

In our experiments, we follow the setup proposed in [28] and [16]. Those works propose the use of data mining tasks with an external ground truth, e.g., predicting certain indicators or classes for entities. Those entities are then linked to a knowledge graph. Different feature extraction methods – which includes the generation of embedding vectors – can then be compared using a fixed set of learning methods.

The setup of [16] comprises six tasks using 20 datasets in total:

- Five classification tasks, evaluated by accuracy.
- Five regression tasks, evaluated by root mean squared error.
- Four clustering tasks (with ground truth clusters), evaluated by accuracy.

[8]The argument in section 4.2 would also work for shared relations to common heads.

- A document similarity task (where the similarity is assessed by computing the similarity between entities identified in the documents), evaluated by the harmonic mean of Pearson and Spearman correlation coefficients. The dataset is based on the LP50 dataset [29].
- An entity relatedness task (where semantic similarity is used as a proxy for semantic relatedness), evaluated by Kendall's Tau. The dataset is based on the KORE dataset [30].
- Four semantic analogy tasks (e.g., *Athens is to Greece as Oslo is to X*), which are based on the original datasets on which word2vec was evaluated [21].

We follow the evaluation protocol suggested in [16]. This protocol foresees the usage of different algorithms on each task for each embedding (e.g., Naive Bayes, Decision Tree, k-NN, and SVM for classification), and also performs parameter tuning in some cases. In the end, we report the best results per task and embedding method. Those results are depicted in Table 1.

For generating the different embedding vectors, we use the DGL-KE framework [31], and we use the RDF2vec vectors provided by the KGvec2go API [32]. We compare RDF2vec [4], TransE (with L1 and L2 norm) [7], TransR [33], RotatE [34], DistMult [35], RESCAL [36], and ComplEx [37]. All embeddings are trained on DBpedia 2016-10.[9] To create the embedding vectors with DGL-KE we use the parameter configurations recommended by the framework, a dimension of 200, and a step maximum of 1,000,000.

From the table, we can observe a few expected and a few unexpected results. First, since RDF2vec is tailored towards classic data mining tasks like classification and regression, it is not much surprising that those tasks are solved better by using RDF2vec vectors. Still, some of the link prediction methods (in particular TransE and RESCAL) perform reasonably well on those tasks.

Referring back to the different notions of similarity that these families of approaches imply (cf. section 4.3), this behavior can be explained by the tendency of RDF2vec to positioning entities closer in the vector space which are more similar to each other (e.g., two cities that are similar). Since it is likely that some of those dimensions are also correlated with the target variable at hand (in other words: they encode some dimension of similarity that can be used to predict the target variable), classifiers and regressors can pick up on those dimensions and exploit them in their prediction model.

What is also remarkable is the performance on the entity relatedness task. While RDF2vec embeddings reflect entity relatedness to a certain extent, this is not given for any of the link prediction approaches. According to the notions of similarity discussed above, this is reflected in the RDF2vec mechanism: RDF2vec has an incentive to position two entities closer in the vector space if they share relations to a common entity, as shown in equations 21-24. One example is the relatedness of *Apple Inc.* and *Steve Jobs* – here, we can observe the two statements

$$product(AppleInc., IPhone)$$

$$knownfor(SteveJobs, IPhone)$$

in DBpedia, among others. Those lead to similar vectors in RDF2vec according to equation 23.

The same property of also assigning closer embedding vectors to related entities explains the comparatively bad results of RDF2vec on the first two clustering tasks. Here the task is to separate cities and countries in two clusters, but since a city is also related to the country it is located in, RDF2vec may position city and country rather closely together. Hence, that city has a certain probability of ending up in the same cluster as the country. The latter two clustering tasks are different: the third one contains five clusters (cities, albums, movies, universities, and companies), which are less likely to be strongly related (except universities and companies to cities) and therefore are more likely to be projected in different areas in the vector space. Here, the difference of RDF2vec to the best performing approaches (i.e., TransE-L1 and TransE-L2) is not that severe.

The problem of relatedness being mixed with similarity does not occur so strongly for *homogeneous* sets of entities, as in the classification and regression tasks, where all entities are of the same kind (cities, companies, etc.) – here, two companies which are related (e.g., because one is a holding of the other) can also be considered similar to a certain degree (in that case, they are both operating in the same branch). This also explains why the forth clustering task (where the task is to assign sports teams to clusters by the type of sports) works well for RDF2vec – here, the entities are again homogeneous.

[9]The code for the experiments as well as the resulting embeddings can be found at https://github.com/nheist/KBE-for-Data-Mining

At the same time, the test case of clustering teams can also be used to explain why link prediction approaches work well for that kind of tasks: here, it is likely that two teams in the same sports share a relation to a common entity, i.e., they fulfill equations 19 and 20. Examples include participation in the same tournaments or common former players.

The semantic analogies task also reveals some interesting findings. First, it should be noted that the relations which form the respective analogies (capital, state, and currency) is contained in the knowledge graph used for the computation. That being said, we can see that most of the link prediction results (except for RotatE and RESCAL) perform reasonably well here. Particularly, the first cases (capitals and countries) can be solved particularly well in those cases, as this is a 1:1 relation, which is the case in which link prediction is a fairly simple task.

On the other hand, the currency case is solved particularly bad by most of the link prediction results. This relation is an n:m relation (there are countries with more than one official, inofficial, or historic currency, and many currencies, like the Euro, are used across many countries. Moreover, looking into DBpedia, this relation contains a lot of mixed usage and is not maintained with very high quality. For example, DBpedia lists 33 entities whose currency is US Dollars[10] – the list contains historic entities (e.g., West Berlin), errors (e.g., Netherlands), and entities which are not countries (e.g., OPEC), but the United States are not among those. For such kind of relations which contain a certain amount of noise and heterogeneous information, many link prediction approaches are obviously not well suited.

RDF2vec, in contrast, can deal reasonably well with that case. Here, two effects interplay when solving such tasks: (i) as shown above, relations are encoded by the proximity in RDF2vec to a certain extent, i.e., the properties in equations (3) and (4) allow to perform analogy reasoning in the RDF2vec space in general. Moreover, (ii) we have already seen the tendency of RDF2vec to position *related* entities in relative proximity. Thus, for RDF2vec, it can be assumed that the following holds:

$$UK \approx PoundSterling \tag{25}$$

$$USA \approx USDollar \tag{26}$$

[10]http://dbpedia.org/page/United_States_dollar

Since we can rephrase the first equation as

$$PoundSterling - UK \approx 0 \tag{27}$$

we can conclude that analogy reasoning in RDF2vec would yield

$$PoundSterling - UK + USA \approx USDollar \tag{28}$$

Hence, in RDF2vec, two effects – the preservation of relation vectors as well as the proximity of related entities – are helpful for analogy reasoning, and the two effects also work for rather noisy cases. However, for cases which are 1:1 relations in the knowledge graph with rather clean training data available, link prediction approaches are better suited for analogy reasoning.

5.2. Experiments on Link Prediction Tasks

In a second series of experiments, we analyze if we can use embedding methods developed for similarity computation, like RDF2vec, also for link prediction. We use the two established tasks WN18 and FB15k for a comparative study.

While link prediction methods are developed for the task at hand, RDF2vec is not. Although RDF2vec computes vectors for relations, they do not necessarily follow the same notion as relation vectors for link prediction, as discussed above. Hence, we investigate two approaches:

1. We average the difference for each pair of a head and a tail for each relation r, and use that as average as a proxy for a relation vector for prediction, as shown in equation (4). The predictions are the entities whose embedding vectors are the closest to the approximate prediction. This method is denoted as *avg*.
2. For predicting the tail of a relation, we train a neural network to predict an embedding vector of the tail based embedding vectors, as shown in Fig. 11. The predictions for a triple $< h, r, ? >$ are the entities whose embedding vectors are closest to the predicted vector for h and r. Two separate networks are trained for predicting heads and tails. This method is denoted as *ANN*.

We trained the RDF2vec embeddings with 2,000 walks, a depth of 4, a dimension of 200, a window of 5, and 25 epochs in SG mode. For the second prediction approach, the two neural networks use two hidden

Task	Metric	Dataset	RDF2vec	TransE-L1	TransE-L2	TransR	RotatE	DistMult	RESCAL	ComplEx
Classification	ACC	AAUP	**0.676**	0.628	0.651	0.607	0.617	0.597	0.623	0.602
		Cities	**0.810**	0.676	0.752	0.757	0.581	0.666	0.740	0.637
		Forbes	**0.610**	0.550	0.601	0.561	0.526	0.601	0.563	0.578
		Albums	**0.774**	0.637	0.746	0.728	0.550	0.666	0.678	0.693
		Movies	**0.739**	0.603	0.728	0.715	0.567	0.668	0.693	0.655
Clustering	ACC	Cities and Countries (2K)	0.758	0.982	**0.994**	0.962	0.510	0.957	0.991	0.955
		Cities and Countries	0.696	0.953	0.979	0.952	0.691	0.909	**0.990**	0.591
		Cities, Albums, Movies, AAUP, Forbes	0.926	**0.946**	0.944	0.908	0.860	0.878	0.936	0.914
		Teams	0.917	0.887	**0.977**	0.844	0.853	0.883	0.881	0.881
Regression	RMSE	AAUP	**68.745**	81.503	69.728	88.751	80.177	78.337	72.880	73.665
		Cities	15.601	19.694	14.455	**13.558**	26.846	19.785	15.137	19.809
		Forbes	36.459	37.589	38.398	39.803	38.343	38.037	**35.489**	37.877
		Albums	**11.930**	14.128	12.589	12.789	14.890	13.452	13.537	13.009
		Movies	**19.648**	23.286	20.635	20.699	23.878	22.161	21.362	22.229
Semantic Analogies	ACC	(All) capitals and countries	0.685	0.709	0.675	**0.938**	0.377	0.782	0.211	0.814
		Capitals and countries	0.648	0.840	0.792	**0.937**	0.640	0.802	0.312	0.864
		Cities and State	0.342	0.335	0.209	**0.392**	0.294	0.379	0.089	0.309
		Currency (and Countries)	**0.339**	0.005	0.285	0.143	0.000	0.001	0.000	0.000
Document Similarity	HarmonicMean	LP50	0.348	0.343	0.397	**0.434**	0.326	0.360	0.344	0.341
Entity Relatedness	KendallTau	KORE	**0.504**	0.002	-0.081	0.139	-0.039	0.147	0.087	0.115

Table 1
Results of the different data mining tasks

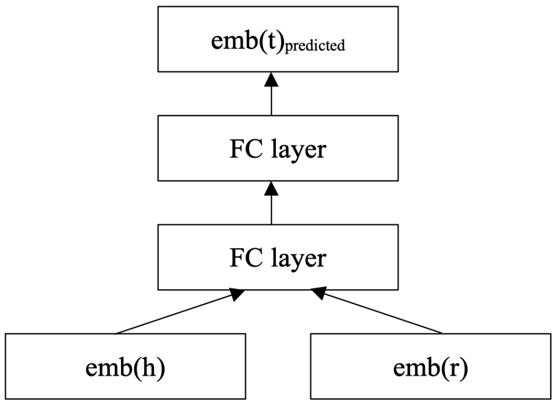

Fig. 11. Training a neural network for link prediction with RDF2vec

layers of size 200, and we use 15 epochs, a batch size of 1,000, and mean squared error as loss.

The results of the link prediction experiments are shown in Table 2.[11] We can observe that the RDF2vec based approaches perform at the lower end of the spectrum. The avg approach outperforms DistMult and RESCAL on WN18, and both approaches are about en par with RESCAL on FB15k.

While the results are not overwhelming, they show that similarity of entities, as RDF2vec models it, is at least a useful signal for implementing a link prediction approach.

5.3. Discussion

As already discussed above, the notion of similarity which is conveyed by RDF2vec mixes *similarity* and *relatedness*. This can be observed, e.g., when querying for the 10 closest concepts to *Angela Merkel* in DBpedia in the different spaces, as shown in Table 3. The approach shows a few interesting effects:

- While most of the approaches (except for RotatE) provide a clean list of people, RDF2vec brings up a larger variety of results, containing also *Germany* and *Berlin* (and also a few results which are not instances, but relations; however, those could be filtered out easily in downstream applications if necessary). This demonstrates the property of RDF2vec of mixing *similarity* and *relatedness*.
- The approaches at hand have different foci in determining similarity. For example, TransE-L2 outputs a list of German and Austrian chancellors, TransE-L1 outputs mostly leaders from different parties and countries, TransR focuses on German politicians in various parties and functions, etc. In all of those cases, the persons share some property with the query entity *Angela Merkel* (profession, role, nationality, etc.), but similarity is usually affected only by one of those properties. In other words: one notion of similarity *dominates* the others.
- In contrast, the persons in the output list of RDF2vec are *related to the query entity* in *different* respects. In particular, they played different roles during Angela Merkel's chancellorship (Gauck was the German president, Lammert was the chairman of the parliament, and Voßkuhle was the chairman of the federal court). Here, there is no *dominant* property, instead, similarity (or rather: relatedness) is encoded along various properties.

With that observation in mind, we can come up with an initial set of recommendations for choosing embedding approaches:

- RDF2vec works well when dealing with sets of *homogeneous entities*. Here, the problem of confusing related entities (like Merkel and Berlin) is negligible, because all entities are of the same kind anyways. In those cases, RDF2vec captures the finer distinctions between the entities better than embeddings for link prediction, and it encodes a larger variety of semantic relations.
- For problems where *heterogeneous sets of entities* are involved, embeddings for link prediction often do a better job in telling different entities apart.

Link prediction is a problem of the latter kind: in embedding spaces where different types are properly separated, link prediction mistakes are much rarer. Given an embedding space where entities of the same type are always closer than entities of a different type, a link prediction approach will always rank all "compatible" entities higher than all incompatible ones. Consider the following example in FB15k:

$$instrument(GilScottHeron, ?)$$

Here, music instruments are expected in the object position. However, the RDF2vec based approach predicts, among plausible candidates such as *electric guitar* and *acoustic guitar*, also *guitarist* and *Jimmy Page*

[11]The code for the experiments can be found at https://github.com/janothan/kbc_rdf2vec

Dataset	Metric	RDF2vec(avg)	RDF2vec(ANN)	TransE	TransR	RotatE	DistMult	RESCAL	ComplEx
WN18	Mean Rank Raw	147	353	263	232	-	-	1180	-
	Mean Rank Filtered	135	342	251	219	309	-	1163	-
	HITS@10 Raw	64.39	49.68	75.4	78.3	-	-	37.2	-
	HTIS@10 Filtered	71.33	55.43	89.2	91.7	**95.9**	57.7	52.8	94.7
FB15K	Mean Rank Raw	399	349	243	226	-	-	828	-
	Mean Rank Filtered	347	303	125	78	40	-	683	-
	HITS@10 Raw	35.31	34.25	34.9	43.8	-	-	28.4	-
	HTIS@10 Filtered	40.54	41.81	47.1	65.5	88.4	**94.2**	44.1	84.0

Table 2

Results of the link prediction tasks on WN18 and FB15K. Results for TransE and RESCAL from [7], results for RotatE from [34], results for DistMult from [35], results for TransR from [33].

RDF2vec	TransE-L1	TransE-L2	TransR
Joachim Gauck	Gerhard Schröder	Gerhard Schröder	Sigmar Gabriel
Norbert Lammert	James Buchanan	Helmut Kohl	Frank-Walter Steinmeier
Stanislaw Tillich	Neil Kinnock	Konrad Adenauer	Philipp Rösler
Andreas Voßkuhle	Nicolas Sarkozy	Helmut Schmidt	Gerhard Schröder
Berlin	Joachim Gauck	Werner Faymann	Joachim Gauck
German language	Jacques Chirac	Alfred Gusenbauer	Christian Wulff
Germany	Jürgen Trittin	Kurt Georg Kiesinger	Guido Westerwelle
federalState	Sigmar Gabriel	Philipp Scheidemann	Helmut Kohl
Social Democratic Party	Guido Westerwelle	Ludwig Erhard	Jürgen Trittin
deputy	Christian Wulff	Wilhelm Marx	Jens Böhrnsen
RotatE	**DistMult**	**RESCAL**	**ComplEx**
Pontine raphe nucleus	Gerhard Schröder	Gerhard Schröder	Gerhard Schröder
Jonathan W. Bailey	Milan Truban	Kurt Georg Kiesinger	Diána Mészáros
Zokwang Trading	Maud Cuney Hare	Helmut Kohl	Francis M. Bator
Steven Hill	Tristan Matthiae	Annemarie Huber-Hotz	William B. Bridges
Chad Kreuter	Gerda Hasselfeldt	Wang Zhaoguo	Mette Vestergaard
Fred Hibbard	Faustino Sainz Muñoz	Franz Vranitzky	Ivan Rosenqvist
Mallory Ervin	Joachim Gauck	Bogdan Klich	Edward Clouston
Paulinho Kobayashi	Carsten Linnemann	İrsen Küçük	Antonio Capuzzi
Fullmetal Alchemist and the Broken Angel	Norbert Blüm	Helmut Schmidt	Steven J. McAuliffe
Archbishop Dorotheus of Athens	Neil Hood	Mao Zedong	Jenkin Coles

Table 3

Closest concepts to *Angela Merkel* in the different embedding approaches used.

(who is a well-known guitarist). While *electric guitar*, *guitarist*, and *Jimmy Page* are semantically related, not all of them are sensible predictions here, and the fact that RDF2vec reflects that semantic relatedness is a drawback in link prediction.

The same argument underlies an observation made by Zouaq and Martel [18]: the authors found that RDF2vec is particularly well suited for distinguishing fine-grained entity classes (as opposed to coarse-grained entity classification). For fine-grained classification (e.g., distinguishing guitar players from singers), all entities to be classified are already of the same coarse class (e.g., musician), and RDF2vec is very well suited for capturing the finer differences. However, for coarse classifications, misclassifications by mistaking relatedness for similarity become more salient.

From the observations made in the link prediction task, we can come up with another recommendation:

– For relations which come with rather clean data quality, link prediction approaches work well. However, for more noisy data, RDF2vec has a higher tendency of creating useful embedding vectors.

For the moment, this is a hypothesis, which should be hardened, e.g., by performing controlled experiments on artificially noised link prediction tasks.

6. Conclusion and Outlook

In this paper, we have compared two use cases and families of knowledge graph embeddings which have, up to today, not undergone any thorough direct comparison: approaches developed for data mining, such as RDF2vec, and approaches developed for link prediction, such as TransE and its descendants.

We have argued that the two approaches actually do something similar, albeit being designed with different goals in mind. To support this argument, we have run two sets of experiments which examined how well the different approaches work if applied in the respective other setup. We show that, to a certain extent, embedding approaches designed for link prediction can be applied in data mining as vice versa, however, there are differences in the outcome.

From the experiments, we have also seen that proximity in the embedding spaces works differently for the two approaches: in RDF2vec, proximity encodes both similarity and relatedness, while TransE and its descendants rather encode similarity alone. On the other hand, for entities that are of the same type, RDF2vec covers finer-grained similarities better. Moreover, RDF2vec seems to work more stably in cases where the knowledge graphs are rather noisy and weakly adherent to their schema.

These findings give rise both for a recommendation and some future work. First, in use cases where relatedness plays a role next to similarity, or in use cases where all entities are of the same type, RDF2vec may yield better results. On the other hand, for cases with mixed entity types where it is important to separate the types, link prediction embeddings might yield better results.

Moreover, the open question remains whether it is possible to develop embedding methods that combine the best of both worlds – e.g., that provide both the coarse type separation of TransE and its descendants and the fine type separation of RDF2vec, or that support competitive link prediction while also representing relatedness. We expect to see some interesting developments along these lines in the future.

References

[1] Q. Wang, Z. Mao, B. Wang and L. Guo, Knowledge graph embedding: A survey of approaches and applications, *IEEE Transactions on Knowledge and Data Engineering* **29**(12) (2017), 2724–2743.

[2] X. Han, S. Cao, X. Lv, Y. Lin, Z. Liu, M. Sun and J. Li, Openke: An open toolkit for knowledge embedding, in: *Proceedings of the 2018 conference on empirical methods in natural language processing: system demonstrations*, 2018, pp. 139–144.

[3] H. Paulheim, Knowledge graph refinement: A survey of approaches and evaluation methods, *Semantic web* **8**(3) (2017), 489–508.

[4] P. Ristoski and H. Paulheim, Rdf2vec: Rdf graph embeddings for data mining, in: *International Semantic Web Conference*, Springer, 2016, pp. 498–514.

[5] B. Steenwinckel, G. Vandewiele, I. Rausch, P. Heyvaert, R. Taelman, P. Colpaert, P. Simoens, A. Dimou, F. De Turck and F. Ongenae, Facilitating the Analysis of COVID-19 Literature Through a Knowledge Graph, in: *International Semantic Web Conference*, Springer, 2020, pp. 344–357.

[6] P. Ristoski, J. Rosati, T. Di Noia, R. De Leone and H. Paulheim, RDF2Vec: RDF graph embeddings and their applications, *Semantic Web* **10**(4) (2019), 721–752.

[7] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston and O. Yakhnenko, Translating embeddings for modeling multi-relational data, in: *Advances in neural information processing systems*, 2013, pp. 2787–2795.

[8] Y. Dai, S. Wang, N.N. Xiong and W. Guo, A Survey on Knowledge Graph Embedding: Approaches, Applications and Benchmarks, *Electronics* **9**(5) (2020), 750.

[9] A. Rossi, D. Firmani, A. Matinata, P. Merialdo and D. Barbosa, Knowledge Graph Embedding for Link Prediction: A Comparative Analysis, *arXiv preprint arXiv:2002.00819* (2020).

[10] S. Ji, S. Pan, E. Cambria, P. Marttinen and P.S. Yu, A survey on knowledge graphs: Representation, acquisition and applications, *arXiv preprint arXiv:2002.00388* (2020).

[11] G.A. Gesese, R. Biswas, M. Alam and H. Sack, A Survey on Knowledge Graph Embeddings with Literals: Which model links better Literal-ly?, *arXiv preprint arXiv:1910.12507* (2019).

[12] K. Toutanova, D. Chen, P. Pantel, H. Poon, P. Choudhury and M. Gamon, Representing text for joint embedding of text and knowledge bases, in: *Proceedings of the 2015 conference on empirical methods in natural language processing*, 2015, pp. 1499–1509.

[13] T. Dettmers, P. Minervini, P. Stenetorp and S. Riedel, Convolutional 2d knowledge graph embeddings, *arXiv preprint arXiv:1707.01476* (2017).

[14] B. Shi and T. Weninger, Open-world knowledge graph completion, *arXiv preprint arXiv:1711.03438* (2017).

[15] M. Cochez, P. Ristoski, S.P. Ponzetto and H. Paulheim, Global RDF vector space embeddings, in: *International Semantic Web Conference*, Springer, 2017, pp. 190–207.

[16] M.A. Pellegrino, A. Altabba, M. Garofalo, P. Ristoski and M. Cochez, GEval: A Modular and Extensible Evaluation Framework for Graph Embedding Techniques, in: *European Semantic Web Conference*, Springer, 2020, pp. 565–582.

[17] J. Chen, X. Chen, I. Horrocks, E. B. Myklebust and E. Jimenez-Ruiz, Correcting Knowledge Base Assertions, in: *Proceedings of The Web Conference 2020*, 2020, pp. 1537–1547.

[18] A. Zouaq and F. Martel, What is the schema of your knowledge graph? leveraging knowledge graph embeddings and clustering for expressive taxonomy learning, in: *Proceedings of The International Workshop on Semantic Big Data*, 2020, pp. 1–6.

[19] P. Ristoski and H. Paulheim, Semantic Web in data mining and knowledge discovery: A comprehensive survey, *Journal of Web Semantics* **36** (2016), 1–22.

[20] P. Ristoski and H. Paulheim, A comparison of propositionalization strategies for creating features from linked open data, *Linked Data for Knowledge Discovery* **6** (2014).

[21] T. Mikolov, K. Chen, G. Corrado and J. Dean, Efficient estimation of word representations in vector space, *arXiv preprint arXiv:1301.3781* (2013).

[22] N. Lavrač, B. Škrlj and M. Robnik-Šikonja, Propositionalization and Embeddings: Two Sides of the Same Coin, *arXiv preprint arXiv:2006.04410* (2020).

[23] P.-N. Tan, M. Steinbach and V. Kumar, *Introduction to data mining*, Pearson Education India, 2016.

[24] G. Vandewiele, B. Steenwinckel, M. Weyns, P. Bonte, F. Ongenae and F.D. Turck, pyRDF2Vec: A python library for RDF2Vec, 2020, https://github.com/IBCNServices/pyRDF2Vec.

[25] R. Sofronova, R. Biswas, M. Alam and H. Sack, Entity Typing based on RDF2Vec using Supervised and Unsupervised Methods.

[26] M. Kejriwal and P. Szekely, Supervised typing of big graphs using semantic embeddings, in: *Proceedings of The International Workshop on Semantic Big Data*, 2017, pp. 1–6.

[27] M. Ali, M. Berrendorf, C.T. Hoyt, L. Vermue, S. Sharifzadeh, V. Tresp and J. Lehmann, PyKEEN 1.0: A Python Library for Training and Evaluating Knowledge Graph Emebddings, *arXiv preprint arXiv:2007.14175* (2020).

[28] P. Ristoski, G.K.D. De Vries and H. Paulheim, A collection of benchmark datasets for systematic evaluations of machine learning on the semantic web, in: *International Semantic Web Conference*, Springer, 2016, pp. 186–194.

[29] M.D. Lee, B. Pincombe and M. Welsh, An empirical evaluation of models of text document similarity, in: *Proceedings of the annual meeting of the cognitive science society*, Vol. 27, 2005.

[30] J. Hoffart, S. Seufert, D.B. Nguyen, M. Theobald and G. Weikum, KORE: keyphrase overlap relatedness for entity disambiguation, in: *Proceedings of the 21st ACM international conference on Information and knowledge management*, 2012, pp. 545–554.

[31] D. Zheng, X. Song, C. Ma, Z. Tan, Z. Ye, J. Dong, H. Xiong, Z. Zhang and G. Karypis, DGL-KE: Training Knowledge Graph Embeddings at Scale, in: *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '20, Association for Computing Machinery, New York, NY, USA, 2020, pp. 739–748–.

[32] J. Portisch, M. Hladik and H. Paulheim, KGvec2go–Knowledge Graph Embeddings as a Service, *arXiv preprint arXiv:2003.05809* (2020).

[33] Y. Lin, Z. Liu, M. Sun, Y. Liu and X. Zhu, Learning entity and relation embeddings for knowledge graph completion, in: *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 29, 2015.

[34] Z. Sun, Z.-H. Deng, J.-Y. Nie and J. Tang, Rotate: Knowledge graph embedding by relational rotation in complex space, *arXiv preprint arXiv:1902.10197* (2019).

[35] B. Yang, W.-t. Yih, X. He, J. Gao and L. Deng, Embedding entities and relations for learning and inference in knowledge bases, *arXiv preprint arXiv:1412.6575* (2014).

[36] M. Nickel, V. Tresp and H.-P. Kriegel, A three-way model for collective learning on multi-relational data, in: *Icml*, 2011.

[37] T. Trouillon, J. Welbl, S. Riedel, É. Gaussier and G. Bouchard, Complex embeddings for simple link prediction, in: *International Conference on Machine Learning*, PMLR, 2016, pp. 2071–2080.