





The DLCC Node Classification Benchmark for Analyzing Knowledge Graph Embeddings

Jan Portisch^{1,2}(✉)  and Heiko Paulheim² 

¹ SAP SE, Walldorf, Germany
jan.portisch@sap.com

² Data and Web Science Group, University of Mannheim, Mannheim, Germany
{jan,heiko}@informatik.uni-mannheim.de

Abstract. Knowledge graph embedding is a representation learning technique that projects entities and relations in a knowledge graph to continuous vector spaces. Embeddings have gained a lot of uptake and have been heavily used in link prediction and other downstream prediction tasks. Most approaches are evaluated on a single task or a single group of tasks to determine their overall performance. The evaluation is then assessed in terms of how well the embedding approach performs on the task at hand. Still, it is hardly evaluated (and often not even deeply understood) what information the embedding approaches are *actually* learning to represent.

To fill this gap, we present the DLCC (Description Logic Class Constructors) benchmark, a resource to analyze embedding approaches in terms of which kinds of classes they can represent. Two gold standards are presented, one based on the real-world knowledge graph DBpedia and one synthetic gold standard. In addition, an evaluation framework is provided that implements an experiment protocol so that researchers can directly use the gold standard. To demonstrate the use of DLCC, we compare multiple embedding approaches using the gold standards. We find that many DL constructors on DBpedia are actually learned by recognizing different correlated patterns rather than those defined in the gold standard; we further find that specific DL constructors, such as cardinality constraints, are particularly hard to be learned for most embedding approaches.

Keywords: Knowledge graph embedding · Node classification · Description logics · Benchmark · Evaluation framework

1 Introduction

Knowledge graph embeddings are projections of entities and relations to continuous vector spaces. They have been proposed for various purposes and are typically evaluated on task-specific gold standards such as FB15k and WN18 [3] for link prediction, kgbench for node classification [2], or GEval [9, 10] for machine learning tasks such as classification, regression, or clustering. The benchmarks frequently come with their own evaluation protocol.

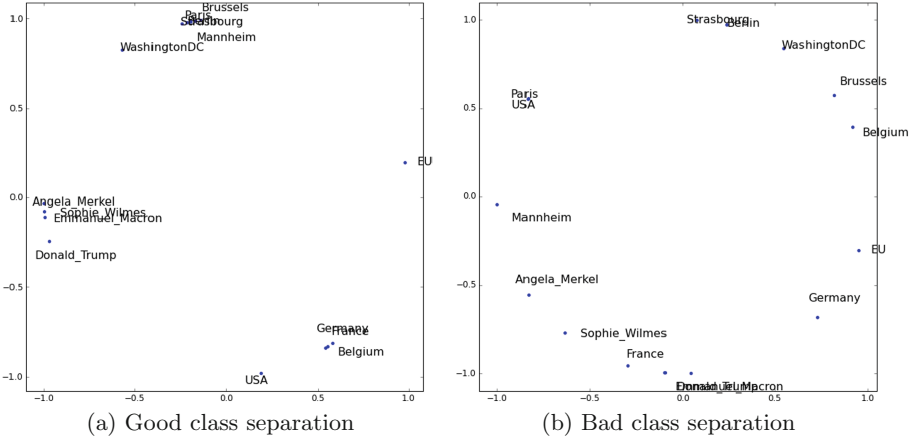


Fig. 1. Two example embeddings. The left-hand side embedding shows a good class separation of persons, countries, and cities, whereas the right-hand side one does not.

Independent of the original benchmark task, knowledge graph embeddings are generally versatile so that they can be used for multiple tasks [11]. While the performance of embeddings in downstream tasks is often superior to other entity representation techniques, most, if not all, embedding approaches have in common that it is not ultimately clear *what* is learned. For example, both for link prediction and for node classification, it is required that classes can be separated (e.g., persons, countries, and cities are clustered in the embedding space) [11], but so far, it has not been systematically evaluated which embedding methods can learn which kinds of class separations. Figure 1 shows an example of two embedding spaces with different qualities of class separation.

In this paper, we present the DLCC (for *Description Logic Class Constructors*) dataset and an evaluation framework that help to better analyze and understand embedding approaches for specific DL constructors. There are four contributions of this paper: (1) A framework for the DLCC gold standard creation is presented, (2) two concrete gold standards are provided – a real graph-based gold standard and one based on synthetic knowledge graphs, (3) an evaluation framework is provided to easily evaluate and compare the class separation capabilities of embeddings, and (4) a preliminary analysis for different state of the art embedding approaches is provided.

2 Related Work

In the area of link prediction (or knowledge base completion), the two well-known evaluation datasets FB15k and WN18 [3] are both based on real datasets: FB15k is based on the Freebase dataset, and WN18 is based on WordNet [5]. They were

presented in the context of link prediction: Given a triple in the form (*head*, *relation*, *tail*), two prediction tasks (*head*, *relation*, *?*) and (*?*, *relation*, *tail*) are created. The evaluation is performed by calculating the mean rank/HITS@10 for a list of proposals. Since it has been remarked that those datasets contain too many simple inferences due to inverse relations, the more challenging variants FB15k-237 [21] and WN18RR [4] have been proposed. More recently, evaluation sets based on larger knowledge graphs, such as YAGO3-10 [4] and DBpedia50k/DBpedia500k [19], have been introduced.

Bloem et al. [2] introduce *kgbench*, a node classification benchmark for knowledge graphs, which, like DLCC, comes with datasets in different sizes and pre-defined train/test splits. Unlike DLCC, *kgbench* is based on real-world datasets. Therefore, it is suitable to evaluate and compare the quality of different embedding approaches on real-world tasks but does not provide any insights into what these embedding approaches are capable of representing.

Alshagari et al. [1] present a framework for ontological concepts covering three aspects: (i) categorization, (ii) hierarchy, and (iii) logic validation. The framework can be used for language models and for knowledge graph embeddings. The work presented in this paper differs in that it goes beyond explicit DBpedia types. The evaluation of this paper is, therefore, of analytical rather than descriptive nature. Moreover, the task sets of DLCC are significantly larger and more comprehensive.

Ristoski et al. [17] provide a collection of benchmarking datasets for machine learning, including classification, clustering, and regression tasks. Later, the GEval framework [9, 10] was introduced to provide a standardized evaluation protocol for this dataset. The evaluation datasets are based on DBpedia. Internally, the embeddings are processed by different downstream classification, regression, or clustering algorithms. The evaluation framework presented in this paper is similar to GEval in that it also evaluates multiple classifiers given a concept vector input.

Melo and Paulheim [8] provide a method for synthesizing benchmark datasets for link and entity type prediction, which are used in conjunction with a fixed ontology. Their goal is to mimic the characteristic of existing knowledge graphs in terms of distributions and patterns.

3 Covered DL Constructors

The aim of this paper is to provide a benchmark for analyzing which kinds of constructs in a knowledge graph can be recognized by different embedding methods. To that end, we define class labels using different DL constructors. Later on, we apply classification algorithms to analyze how well the differently labeled classes can be separated using different embedding algorithms.

Ingoing and Outgoing Relations. All entities that have a particular outgoing or ingoing relations (e.g., *everything that has a location*).

$$\exists r. \top \tag{1}$$

$$\exists r^{-1}. \top \quad (2)$$

$$\exists r. \top \sqcup \exists r^{-1}. \top \quad (3)$$

where r is bound to a particular relation.¹

Relations to Particular Individuals. All entities that have a relation (in any direction) to a particular individual (e.g., *everything that is related to New York City*).

$$\exists R. \{e\} \sqcup \exists R^{-1}. \{e\} \quad (4)$$

where R is *not* bound to a particular relation. Those relations can also span two (or more² hops):

$$\exists R_1. (\exists R_2. \{e\}) \sqcup \exists R_1^{-1}. (\exists R_2^{-1} \{e\}) \quad (5)$$

Particular Relations to Particular Individuals. All entities that have a particular relation to a particular individual (e.g., *movies directed by Steven Spielberg*).

$$\exists r. \{e\} \quad (6)$$

Qualified Restrictions. All entities that have a particular relation to an individual of a given type (e.g., *all people married to soccer players*).

$$\exists r. T \quad (7)$$

$$\exists r^{-1}. T \quad (8)$$

If types are modeled as a normal relation in the graph (i.e., `rdf:type` is yet another relation), we can reformulate Eq. 7 and 8 to

$$\exists r. (\exists \text{rdf:type}. T) \quad (7a)$$

$$\exists r^{-1}. (\exists \text{rdf:type}. T) \quad (8a)$$

In that case, it behaves equally to a chained variant of Eq. 6.

Cardinality Restrictions of Relations. All entities that have at least or at most n relations of a particular kind (e.g., *people who have at least two citizenships*). Here, we depict only the *lower bound* variant because the corresponding decision problem is between the two variants (entities that fall below the bound, i.e., adhere to the upper bound, are in the negative example set).³

$$\geq 2r. \top \quad (9)$$

$$\geq 2r^{-1}. \top \quad (10)$$

¹ We use r to denote a particular relation, whereas R denotes *any* relation.

² For reasons of scalability, we restrict the provided gold standard to two hops.

³ The fact that most KGs follow the open-world assumption is neglected here since we test for the presence/absence of patterns.

Table 1. Overview of the test cases

Test case	DL expression
tc01	$\exists r. \top$
tc02	$\exists r^{-1}. \top$
tc03	$\exists r. \top \sqcup \exists r^{-1}. \top$
tc04	$\exists R. \{e\} \sqcup \exists R^{-1}. \{e\}$
tc05	$\exists R_1. (\exists R_2. \{e\}) \sqcup \exists R_1^{-1}. (\exists R_2^{-1} \{e\})$
tc06	$\exists r. \{e\}$
tc07	$\exists r. T$
tc08	$\exists r^{-1}. T$
tc09	$\geq 2r. \top$
tc10	$\geq 2r^{-1}. \top$
tc11	$\geq 2r. T$
tc12	$\geq 2r^{-1}. T$

Qualified Cardinality Restrictions. Qualified cardinality restrictions combine qualified restrictions with cardinalities (e.g., people who have published at least two science fiction novels).

$$\geq 2r. T \tag{11}$$

$$\geq 2r^{-1}. T \tag{12}$$

Table 1 summarizes the DL constructors for which test cases were built.

4 Approach

For the twelve test cases in Table 1, we create positive examples (i.e., those which fall into the respective class) and those which do not (under closed-world semantics). For example, for tc01, we would generate a set of positive instances for which $\exists r. \top$ holds and a set of negative instances for which $\nexists r. \top$ holds. We then evaluate how well these two classes can be separated, given the embedding vectors of the positive and negative instances. For that, we split the examples into a training and testing partition, we train binary classifiers on the training subset of the examples, and evaluate their performance on the test subset.

The approach is visualized in Fig. 2: A gold standard generator generates a set of positive and negative URIs, as well as a fixed train/test split. The approach presented in this paper allows to generate custom gold standards – however, a contribution of this paper is also to provide a pre-calculated gold standard. This pre-calculated gold standard can be used to guarantee reproducibility. Officially published gold standards are versioned to allow for future improvements. In this paper, we present version v1 of the gold standard.

A user provides embeddings in a simple textual format and provides them together with the training data as input to the evaluator. The evaluator trains

multiple classifiers and evaluates them on the selected gold standard using the provided vectors as classification input. The program then calculates multiple statistics in the form of CSV files that can be further analyzed in a spreadsheet program or through data analysis frameworks such as pandas⁴. These analyses help the user to understand how well the provided vectors are performing on a particular DL constructor.

4.1 Gold Standard Generator

The gold standard generator is publicly available⁵. It is implemented as a Java maven project. The generator can generate either a DBpedia benchmark (see Subject. 5.1) or a synthetic one (see Subject. 5.2). Any DBpedia version can be used; the user merely needs to provide a SPARQL endpoint. A comprehensive set of unit tests ensures a high code quality. The generator automatically generates a fixed train-test split for the evaluation framework or any other downstream application. The split is configurable; for the pre-generated gold standards, an 80–20 split is used. The resulting gold standard is balanced – i.e., the number of positives equals the number of negatives – and the train and test partitions are stratified. Hence, any classifier which achieves an accuracy significantly above 50% is capable of learning the test case’s problem type from the vectors to some extent.

It is important to note that the generator only needs to be run by users who want to build their own gold standards. The typical user would merely download⁶ the official gold standard files online. We recommend using the pre-calculated gold standards to ensure comparability across publications.

4.2 Evaluation Framework

The evaluator is publicly available⁷ as well together with usage examples. It is implemented in Python and can be easily used in a Jupyter notebook. A comprehensive set of unit tests ensures a high code quality.

The standard user can directly download the gold standard and use the evaluation framework. To test class separability, the evaluation framework currently runs six machine learning classifiers:⁸ (1) decision trees, (2) naïve Bayes, (3) KNN, (4) SVM, (5) random forest, and (6) a multilayer perceptron network. The framework uses the default configurations of the sklearn library⁹.

After training and evaluation, the framework persists multiple CSV files per test case as well as higher-level aggregate CSV files. Examples of such CSV files

⁴ <https://pandas.pydata.org/>.

⁵ <https://github.com/janothan/DL-TC-Generator>.

⁶ DOI: 10.5281/zenodo.6509715; GitHub link for the latest version. <https://github.com/janothan/DL-TC-Generator/tree/master/results>.

⁷ <https://github.com/janothan/dl-evaluation-framework>.

⁸ The evaluation framework is not restricted to the set of classifiers listed here. New classifiers can be easily added if desired.

⁹ <https://scikit-learn.org/stable/index.html>.

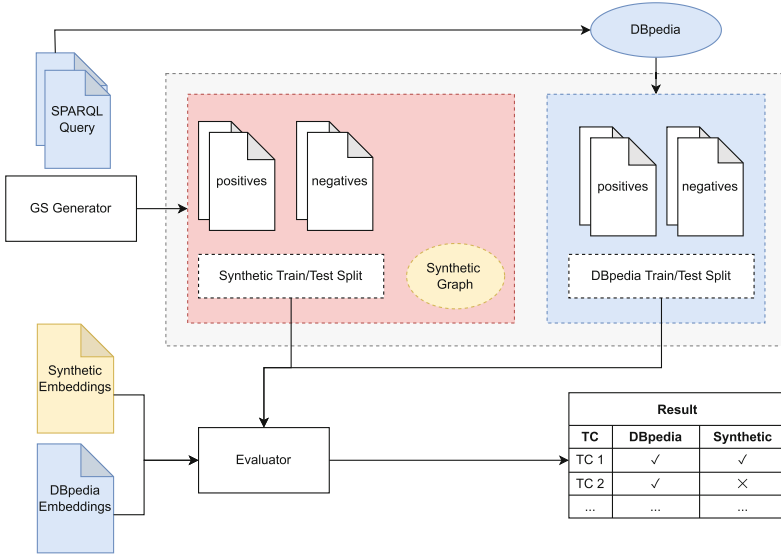


Fig. 2. Overview of the approach

are a file listing the accuracy per classifier and per test case or a file listing the accuracy of the best classifier per test case. In the case of DBpedia, test cases are created for multiple domains, and the results can be analyzed on the level of each domain separately or in an aggregated manner on the level of the test case.

5 Benchmarks

We currently provide two benchmarks, while the framework described above allows for generating customized benchmarks.

5.1 DBpedia Benchmark

We use the DBpedia knowledge graph to create test cases.¹⁰ We created SPARQL queries for each test case (see Table 1) to generate positives, negatives, and hard negatives. The latter are meant to be less easily distinguishable from the positives and are created by variations such as softening the constraints in the class constructor or switching subject and object in the constraint. For example, for qualified relations, a positive example would be a person playing in a team which is a basketball team. A simple negative example would be any person not playing in a basketball team, whereas a hard negative example would be any person playing in a team that is not a basketball team.

¹⁰ We used DBpedia version 2021-09. The generator can be configured to use any DBpedia SPARQL endpoint if desired.

Query examples for every test case in the people domain are provided in Table 2. The framework uses slightly more involved queries to vary the size of the result set and to better randomize results.

In total, we used six different domains: people (P), books (B), cities (C), music albums (A), movies (M), and species (S). This setup yields more than 200 hand-written SPARQL queries, which are used to obtain positives, negatives, and hard negatives; they are available online¹¹ and can be easily extended, e.g., to add an additional domain. For each test case, we created differently sized (50, 500, 5000) balanced test sets.¹²

5.2 Synthetic Benchmark

The previous benchmark is realistic and well suited to compare approaches on differently typed DL constructors.

However, the following aspects have to be considered: (1) DBpedia is a large knowledge graph; not every embedding approach can be used to learn an embedding for it (or not every researcher has the computational means to do so, respectively). (2) Depending on the DL constructor and the domain, not enough examples can be found on DBpedia. (3) It cannot be precluded that patterns correlate; therefore, the fact that an embedding approach can learn a particular class can only be an indicator that it *might* learn the underlying constructor pattern, but the results are not conclusive. Correlating properties, type biases for entities, etc., may lead to surprising results in some domains (see Sect. 6.3).

Therefore, we complement the DBpedia-based gold standard with a synthetic benchmark. The idea is to generate a graph that contains the DL constructors (positive and negative) of interest. The graph can be constructed to resemble the DBpedia graph statistically but can be significantly smaller (and contain a sufficient number of positives and negatives), and, by construction, side effects and correlations which exist in DBpedia can be mitigated to a large extent.

The configurable parameters are `numClasses`, `numProperties`, `numInstances`, `branchingFactor`, `maxTriplesPerNode`, and `numNodesInterest` (all parameters are integers). The overall process is depicted in Algorithm 1: First, a class tree with `numClasses` classes is constructed in a way that each class has at most `branchingFactor` children. Then, `numproperties` properties are generated. Each property is assigned to a range and domain from the class tree, whereby the first property has the root node as domain and range type so that every node can be involved in at least one triple statement. A skew can be introduced so that domain and range refer with a higher probability to a more general class than to a specific one. Lastly, we generate instances and assign them to a class as type, which is depicted in Algorithm 1.

Once the ontology is created, `numNodesInterest` positives and negatives are generated (adhering to domain/range restrictions). Each class constructor is

¹¹ <https://github.com/janothan/DL-TC-Generator/tree/master/src/main/resources/queries>.

¹² The desired size classes can be configured in the framework.

Table 2. Exemplary SPARQL queries for class Person

TC	Query Positive	Query Negative	Query Negative (hard)
tc01	SELECT DISTINCT(?x) WHERE { ?x a dbo:Person . ?x dbo:child ?y . }	SELECT DISTINCT(?x) WHERE { ?x a dbo:Person . FILTER(NOT EXISTS { ?x dbo:child ?z})}	SELECT DISTINCT(?x) WHERE { ?x a dbo:Person . ?y dbo:child ?x. FILTER(NOT EXISTS { ?x dbo:child ?z})}
tc02	Analogous to tc01 (inverse case).		
tc03	SELECT DISTINCT(?x) WHERE { ?x a dbo:Person . ?x dbo:child ?y} UNION { ?x a dbo:Person . ?y dbo:child ?x}}	SELECT COUNT(?x) WHERE { ?x a dbo:Person . FILTER(NOT EXISTS{ ?x dbo:child ?y} AND NOT EXISTS { ?z dbo:child ?x})}	–
tc04	SELECT DISTINCT(?x) WHERE { { ?x a dbo:Person . ?x ?y dbr:New_York_City} UNION { ?x a dbo:Person . dbr:New_York_City ?y ?x}}	SELECT DISTINCT(?x) WHERE { ?x a dbo:Person . FILTER(NOT EXISTS{ ?x ?y dbr:New_York_City} AND NOT EXISTS { dbr:New_York_City ?y ?x})}	SELECT DISTINCT(?x) WHERE {{ ?x a dbo:Person . ?x ?y1 ?z . ?z ?y2 dbr:New_York_City } UNION { ?x a dbo:Person . ?z ?y1 ?x . dbr:New_York_City ?y2 ?z } FILTER(NOT EXISTS {?x ?r dbr:New_York_City} AND NOT EXISTS {dbr:New_York_City ?s ?x})}
tc05	Analogous to tc04 (inverse case).		
tc06	SELECT DISTINCT(?x) WHERE { ?x a dbo:Person . ?x dbo:birthPlace dbr:New_York_City }	SELECT DISTINCT(?x) WHERE { ?x a dbo:Person . FILTER(NOT EXISTS{ ?x dbo:birthPlace dbr:New_York_City })}	SELECT DISTINCT(?x) ?r WHERE {{ ?x a dbo:Person . ?x dbo:birthPlace ?y . dbr:New_York_City ?r ?x . FILTER(?y!=dbr:New_York_City)} UNION { ?x a dbo:Person . ?x dbo:birthPlace ?y . ?x ?r dbr:New_York_City . FILTER(?y!=dbr:New_York_City)}
tc07	SELECT DISTINCT(?x) WHERE { ?x a dbo:Person . ?x dbo:team ?y . ?y a dbo:BasketballTeam }	SELECT DISTINCT(?x) WHERE { ?x a dbo:Person . FILTER(NOT EXISTS{ ?x dbo:team ?y . ?y a dbo:BasketballTeam})}	SELECT DISTINCT(?x) WHERE { ?x a dbo:Person . ?x dbo:team ?z1 . ?x ?r ?z2 . ?z2 a dbo:BaseballTeam FILTER(NOT EXISTS{ ?x dbo:team ?y . ?y a dbo:BasketballTeam })}
tc08	Analogous to tc07 (inverse case).		
tc09	SELECT DISTINCT(?x) WHERE { ?x a dbo:Person . ?x dbo:award ?y1. ?x dbo:award ?y2. FILTER(?y1!=?y2)}	SELECT DISTINCT(?x) WHERE { ?x a dbo:Person . FILTER(NOT EXISTS{ ?x dbo:award ?y1. ?x dbo:award ?y2. FILTER(?y1!=?y2)}}	SELECT DISTINCT(?x) WHERE { ?x a dbo:Person . ?x dbo:award ?y . FILTER(NOT EXISTS{ ?x dbo:award ?z. FILTER(?y1=?z)}}
tc10	Analogous to tc09 (inverse case).		
tc11	SELECT DISTINCT(?x) WHERE { ?x a dbo:Person . ?x dbo:recordLabel ?y1 . ?y1 a dbo:RecordLabel . ?x dbo:recordLabel ?y2 . ?y2 a dbo:RecordLabel . FILTER(?y1!=?y2)}	SELECT DISTINCT(?x) WHERE { ?x a dbo:Person . FILTER(NOT EXISTS{ ?x dbo:recordLabel ?y1 . ?y1 a dbo:RecordLabel . ?x dbo:recordLabel ?y2 . ?y2 a dbo:RecordLabel . FILTER(?y1!=?y2)}}	SELECT DISTINCT(?x) WHERE { ?x a dbo:Person . ?x dbo:recordLabel ?y1 . ?y1 a dbo:RecordLabel . FILTER(NOT EXISTS{ ?x dbo:recordLabel ?y2 . ?y2 a dbo:RecordLabel . FILTER(?y1!=?y2)}}
tc12	Analogous to tc11 (inverse case).		

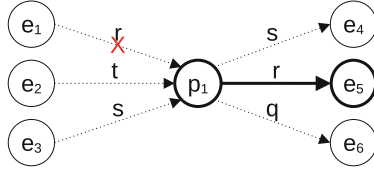


Fig. 3. Illustration of the instance generation, using the class constructor $\exists r.T$. First, the pattern is instantiated for the positive example p_1 with the edge (p_1, r, e_5) . Then, random edges are inserted (dashed lines). The edge (e_1, r, p_1) is removed, because it would turn e_1 into an additional positive example.

first initialized explicitly for the positive examples. Then, for each entity e in the graph (i.e., positive and negative examples), $rand(n) \in [1, maxTriplesPerNode]$ random triples are generated, which have e as a subject and adhere to the domain and range definitions, whereby it is checked that no additional positives are created, and no negatives are turned into positives accidentally (see Fig. 3).

For version v1 of the gold standard, `numClasses` = 760, `numProperties` = 1355, `numInstances` = 10,000, `branchingFactor` = 5, `maxTriplesPerNode` = 11, and `numNodesInterest` = 1000 were chosen. The parameters were chosen to form graphs which are smaller than DBpedia but resemble the DBpedia graph statistically. Therefore, the statistical properties of the DBpedia ontology calculated by Heist et al. [6] were used.

6 Exemplary Analysis

In order to demonstrate the use of the DLCC benchmark, we compare two flavors of RDF2vec [16], two flavors of TransE [3], as well as TransR [7] and ComplEx [22] embeddings with respect to their capability of separating the classes in the different datasets.

6.1 Configurations

For DBpedia, we use version 2021-09. We train RDF2vec in the variants SG and its order-aware counterpart SG_{oa} [14]. The embedding files are available via KGvec2go [12].¹³ For the DBpedia embeddings, we used 500 random, duplicate free walks per entity, with a depth of 4, a window of 5, 5 epochs, and a dimension of 200. We used the same parameters for the synthetic gold standard with the exception of $dimension = 100$ and $walks = 100$ to account for the smaller gold standard size. The embeddings were trained using the jRDF2vec¹⁴ framework [13].

For TransE, we use the variants using the L1 and L2 norm [3]. TransE, TransR, and ComplEx were trained using the DGL-KE framework¹⁵ [23], using

¹³ <http://data.dws.informatik.uni-mannheim.de/kgvec2go/dbpedia/2021-09/>.

¹⁴ <https://github.com/dwslab/jRDF2Vec>.

¹⁵ <https://github.com/awslabs/dgl-ke>.

Algorithm 1. Ontology Creation

```

procedure GENERATECLASSTREE(numClasses, branchingFactor)
  clsURIs ← GENERATEURIS(numClasses)
  root ← RANDOMDRAW(clsURIs)
  i ← 0
  workList ← NEWLIST( )
  result ← NEWTREE( )
  currentURI ← root
  for clsURI in clsURIs do
    if clsURI = root then
      CONTINUE
    end if
    if i = branchingFactor then
      currentURI ← workList.removeFirst()
      i ← 0
    end if
    result.addLeaf(currentURI, clsURI)
    i ← i + 1
    workList.add(clsURI)
  end for
  return result
end procedure

procedure GENERATEPROPERTIES(numProperties, classTree)
  properties ← GENERATEURIS(numProperties)
  for property in properties do
    property.addDomain( DRAWDOMAINRANGE(classTree, 0.25) )
    property.addRange( DRAWDOMAINRANGE(classTree, 0.25) )
  end for
  return properties
end procedure

procedure DRAWDOMAINRANGE(classTree, p)
  result ← classTree.randomClass()
  while Random.nextDouble > p ∧ ¬(classTree.getChildren(result) == ∅) do
    result ← randomDraw(classTree.getChildren(result))
  end while
end procedure

procedure POPULATECLASSES(numInstances, classTree)
  instances ← GENERATEURIS(numInstances)
  for instance in instances do
    instance.type(classTree.randomClass())
  end for
  return instances
end procedure

```

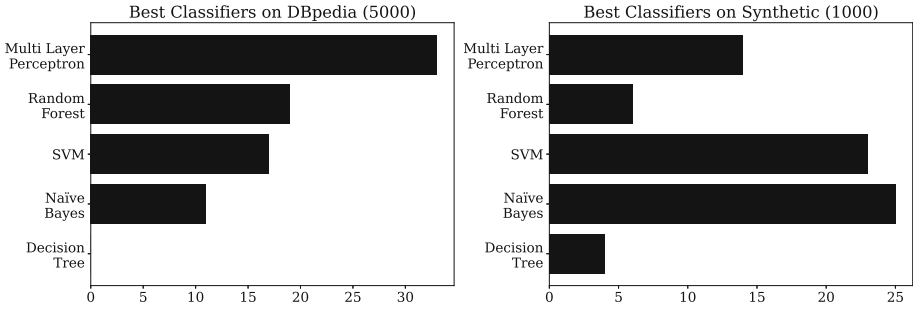


Fig. 4. Best classifiers on the DBpedia and synthetic gold standards. It is important to note that the total number of test cases varies between the two gold standards – therefore, two separate plots were drawn.

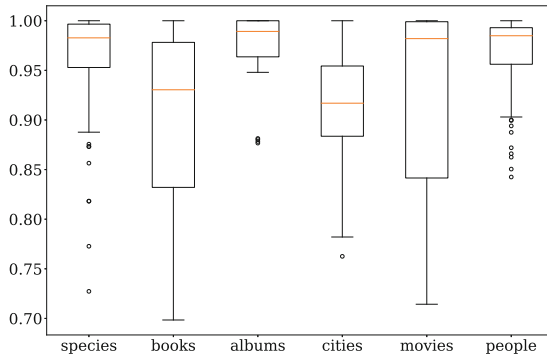


Fig. 5. Domain complexity of the DBpedia gold standard (size class 5000)

the respective default parameters, with 200 dimensions for DBpedia and 100 for the synthetic datasets, as for RDF2vec. The models are publicly available.¹⁶

6.2 Results and Interpretation

The results on the DBpedia gold standard (class size 5,000) and the synthetic gold standard (class size 1,000) are depicted in Tables 3 and 4. For each model and test case, six classifiers were trained (192 classifiers in total). The tables present the results of the best classifiers. We performed significance tests (approximated one-sided binomial test) for each test case and approach with $\alpha = 0.05$ to determine whether the accuracy is significantly higher than 0.5 (random guessing). Since multiple classifiers were trained for each test case, we applied a Bonferroni correction [18] of α to account for the multiple testing problem. On the DBpedia gold standard, all results are significant; on the synthetic gold standard, more insignificant results are observed, particularly for TransR and ComplEx.

¹⁶ <http://data.dws.informatik.uni-mannheim.de/kgvec2go/dbpedia/2021-09/non-rdf2vec/>.

Table 3. Results on the DBpedia gold standard. The best result for each test case is printed in bold. Listed are the results of the best classifier for each task and model.

TC	RDF2vec	RDF2vec _{oa}	TransE-L1	TransE-L2	TransR	ComplEx
tc01	0.915	0.937	0.842	0.947	0.858	0.862
tc01 hard	0.681	0.891	0.799	0.916	0.744	0.651
tc02	0.953	0.961	0.852	0.970	0.832	0.853
tc02 hard	0.637	0.780	0.780	0.849	0.693	0.608
tc03	0.949	0.958	0.821	0.933	0.856	0.874
tc04	0.960	0.968	0.934	0.986	0.973	0.990
tc04 hard	0.963	0.984	0.814	0.912	0.855	0.935
tc05	0.986	0.992	0.867	0.948	0.881	0.905
tc06	0.957	0.963	0.929	0.985	0.976	0.991
tc06 hard	0.863	0.936	0.823	0.779	0.964	0.933
tc07	0.938	0.955	0.930	0.987	0.978	0.966
tc08	0.961	0.966	0.898	0.964	0.870	0.888
tc09	0.902	0.901	0.884	0.938	0.879	0.883
tc09 hard	0.785	0.793	0.749	0.848	0.758	0.776
tc10	0.947	0.958	0.957	0.984	0.898	0.931
tc10 hard	0.740	0.737	0.775	0.774	0.656	0.739
tc11	0.932	0.897	0.917	0.960	0.930	0.946
tc11 hard	0.725	0.737	0.712	0.806	0.753	0.723
tc12	0.955	0.938	0.961	0.984	0.879	0.894
tc12 hard	0.714	0.717	0.762	0.765	0.659	0.710

Figure 4 shows the aggregated number of the best classifiers for each embedding on each test case. It is visible that on DBpedia, MLPs work best, followed by random forests and SVMs. On the synthetic gold standard, naïve Bayes works best most of the time, followed by SVMs and MLPs. The differences can partly be explained by the different size classes of the training sets (MLPs and random forests typically work better on more data).

Figure 5 depicts the complexity per domain of the DBpedia gold standard in a box-and-whisker plot. The complexity was determined by using the accuracy of the best classifier of each embedding model without hard test cases (since not every domain has an equal amount of hard test cases). We observe that all domain test cases are similarly hard to solve, whereby the albums, people, and species domain are a bit simpler to solve than the books and cities domain.

In general, we can observe that the results on the DBpedia gold standard are much higher than on the synthetic gold standard. While on the DBpedia gold standard, all but five tasks can be solved with an accuracy above 0.9 (although the cases with hard variants are actually harder than the non-hard ones, and all the five problems with a best accuracy below 0.9 are hard cases), the synthetic gold standard has quite a few tasks (tc07–tc12) which are obviously much harder. For example, it is hardly possible for any of the approaches to learn classes whose

Table 4. Results on the synthetic gold standard. The best result for each test case is printed in bold; statistically insignificant results are printed in italics. Listed are the results of the best classifier for each task and model.

TC	RDF2vec	RDF2vec _{oa}	TransE-L1	TransE-L2	TransR	Complex
tc01	0.882	0.867	0.767	0.752	0.712	0.789
tc02	0.742	0.737	0.677	0.677	<i>0.531</i>	<i>0.549</i>
tc03	0.797	0.812	<i>0.531</i>	0.581	<i>0.554</i>	<i>0.536</i>
tc04	1.000	0.998	0.790	0.898	0.685	<i>0.553</i>
tc05	0.892	0.819	0.691	0.774	0.631	0.726
tc06	0.978	0.963	0.898	0.978	0.888	1.000
tc07	0.583	0.583	<i>0.540</i>	0.615	0.673	<i>0.518</i>
tc08	0.563	0.585	0.585	0.613	<i>0.540</i>	<i>0.523</i>
tc09	0.610	0.628	0.588	<i>0.543</i>	<i>0.525</i>	<i>0.545</i>
tc10	0.638	0.623	0.588	0.573	<i>0.518</i>	<i>0.510</i>
tc11	0.633	0.580	0.583	0.590	0.573	0.590
tc12	0.644	0.614	0.618	<i>0.550</i>	<i>0.513</i>	<i>0.540</i>

definitions involve cardinalities. RDF2vec can produce results slightly above the baseline here because the frequencies of properties appearing in random walks can reflect cardinalities to a certain extent.

Furthermore, we can observe that it seems easier to predict patterns involving outgoing edges than those involving ingoing edges (cf. tc02 vs. tc01, tc08 vs. tc07, tc10 vs. tc09, tc12 vs. tc11), at least for the DBpedia case. Even though the tasks are very related, this can be explained by the learning process, which often emphasizes outgoing directions: In RDF2vec, random walks are performed in forward direction; similarly, TransE is directed in its training process.

For constructors involving a particular entity (tc04 and tc05), we can observe that RDF2vec is clearly better than embedding approaches for link prediction, at least on the synthetic gold dataset. Those tasks refer to *entity relatedness*, for which RDF2vec has been shown to be more adequate [14, 15]. The picture is more diverse for the other cases.

6.3 DBpedia Gold Standard vs. Synthetic Gold Standard

The results reveal great differences between the gold standards. Many class constructors that are easily learnable on the DBpedia gold standard are hard on the synthetic one. Moreover, the previously reported superiority of RDF2vec_{oa} over standard RDF2vec [11, 14] cannot be observed on the synthetic data.

Figure 6 shows an excerpt of DBpedia, which we will use to illustrate these deviations. The instance `dbr:LeBron_James` is a positive example for task tc07 in Table 2. At the same time, 95.6% of all entities in DBpedia fulfilling the positive query for positive examples also fall in the class `∃dbo:position.⊤` (which is

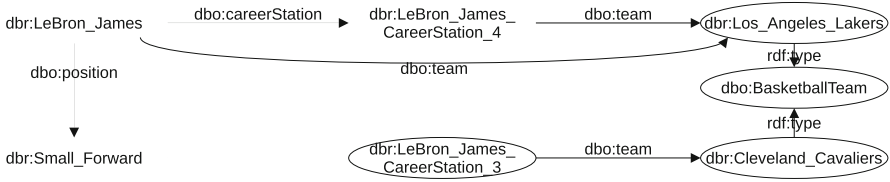


Fig. 6. Excerpt of DBpedia

a tc01 problem), but only 13.6% of all entities fulfilling the query for trivial negatives. Hence, on a balanced dataset, this class can be learned with an accuracy of 0.91 by any approach that can learn classes of type tc01. As a comparison to the synthetic dataset shows, the results on the DBpedia test set for tc07 actually overestimate the capability of many embedding approaches to learn classes constructed with a tc07 class constructor. Such correlations are quite frequent in DBpedia but vastly absent in the synthetic dataset.

The example can also explain the advantage of RDF2vec_{oa} on DBpedia. Unlike standard RDF2vec, this approach would distinguish the appearance of `dbo:team` as a direct edge of `dbr:LeBron_James` as well as an indirect edge connected to `dbr:LeBron_James_CareerStation_N`, where the former denotes the current team, whereas the latter also denotes all previous teams. Those subtle semantic differences of distinctive usages of the same property in various contexts also do not exist in the synthetic gold standard. Hence, the order-aware variant of RDF2vec does not have an advantage here.

7 Conclusion and Future Work

In this paper, we presented DLCC, a resource to analyze embedding approaches in terms of which kinds of classes they are able to represent. DLCC comes with an evaluation framework to easily evaluate embeddings using a reproducible protocol. All DLCC components, i.e., the gold standard, the generation framework, and the evaluation framework, are publicly available.¹⁷

We have shown that many patterns using DL class constructors on DBpedia are actually learned by recognizing patterns with other constructors correlating with the pattern to be learned, thus yielding misleading results. This effect is less prominent in the synthetic gold standard. We showed that certain DL constructors, such as cardinality constraints, are particularly hard to learn.

In the future, we plan to extend the systematic evaluation to more embedding approaches, including the flavors of RDF2vec, which were published more recently [14, 15, 20]. The synthetic dataset generator also allows for more interesting experiments: We can systematically analyze the scalability of existing approaches or study how variations in the synthetic gold standard (e.g., larger and smaller ontologies) influence the outcome.

¹⁷ Dataset DOI: 10.5281/zenodo.6509715.

References

1. Alshargi, F., Shekarpour, S., Soru, T., Sheth, A.P.: Metrics for evaluating quality of embeddings for ontological concepts. In: Martin, A., Hinkelmann, K., Gerber, A., Lenat, D., van Harmelen, F., Clark, P. (eds.) Proceedings of the AAAI 2019 Spring Symposium on Combining Machine Learning with Knowledge Engineering (AAAI-MAKE 2019) Stanford University, Palo Alto, California, USA, March 25–27, 2019, Stanford University, Palo Alto, California, USA, 25–27 March 2019. CEUR Workshop Proceedings, vol. 2350. CEUR-WS.org (2019). <https://ceur-ws.org/Vol-2350/paper26.pdf>
2. Bloem, P., Wilcke, X., van Berkel, L., de Boer, V.: **kgbench**: a collection of knowledge graph datasets for evaluating relational and multimodal machine learning. In: Verborgh, R., et al. (eds.) ESWC 2021. LNCS, vol. 12731, pp. 614–630. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-77385-4_37
3. Bordes, A., Usunier, N., García-Durán, A., Weston, J., Yakhnenko, O.: Translating embeddings for modeling multi-relational data. In: Burges, C.J.C., Bottou, L., Ghahramani, Z., Weinberger, K.Q. (eds.) Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5–8, 2013, Lake Tahoe, Nevada, United States. pp. 2787–2795 (2013). <https://proceedings.neurips.cc/paper/2013/hash/1cecc7a77928ca8133fa24680a88d2f9-Abstract.html>
4. Dettmers, T., Minervini, P., Stenetorp, P., Riedel, S.: Convolutional 2D knowledge graph embeddings. In: McIlraith, S.A., Weinberger, K.Q. (eds.) Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-2018), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-2018), New Orleans, Louisiana, USA, 2–7 February 2018, pp. 1811–1818. AAAI Press (2018). <https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/17366>
5. Fellbaum, C. (ed.): WordNet: An Electronic Lexical Database. Language, Speech, and Communication, MIT Press, Cambridge (1998). <https://doi.org/10.7551/mitpress/7287.001.0001>, <https://doi.org/10.7551/mitpress/7287.001.0001>
6. Heist, N., Hertling, S., Ringler, D., Paulheim, H.: Knowledge graphs on the web - an overview. In: Tiddi, I., Lécué, F., Hitzler, P. (eds.) Knowledge Graphs for eXplainable Artificial Intelligence: Foundations, Applications and Challenges, Studies on the Semantic Web, vol. 47, pp. 3–22. IOS Press (2020). <https://doi.org/10.3233/SSW200009>, <https://doi.org/10.3233/SSW200009>
7. Lin, Y., Liu, Z., Sun, M., Liu, Y., Zhu, X.: Learning entity and relation embeddings for knowledge graph completion. In: Bonet, B., Koenig, S. (eds.) Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, 25–30 January 2015, Austin, Texas, USA, pp. 2181–2187. AAAI Press (2015). <https://www.aaai.org/ocs/index.php/AAAI/AAAI15/paper/view/9571>
8. Melo, A., Paulheim, H.: Synthesizing knowledge graphs for link and type prediction benchmarking. In: Blomqvist, E., Maynard, D., Gangemi, A., Hoekstra, R., Hitzler, P., Hartig, O. (eds.) ESWC 2017. LNCS, vol. 10249, pp. 136–151. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-58068-5_9
9. Pellegrino, M.A., Altabba, A., Garofalo, M., Ristoski, P., Cochez, M.: GEval: a modular and extensible evaluation framework for graph embedding techniques. In: Harth, A., et al. (eds.) ESWC 2020. LNCS, vol. 12123, pp. 565–582. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-49461-2_33

10. Pellegrino, M.A., Cochez, M., Garofalo, M., Ristoski, P.: A configurable evaluation framework for node embedding techniques. In: Hitzler, P., et al. (eds.) ESWC 2019. LNCS, vol. 11762, pp. 156–160. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-32327-1_31
11. Portisch, J., Heist, N., Paulheim, H.: Knowledge graph embedding for data mining vs. knowledge graph embedding for link prediction - two sides of the same coin? *Seman. Web* **13**(3), 399–422 (2022). <https://doi.org/10.3233/SW-212892>, <https://doi.org/10.3233/SW-212892>
12. Portisch, J., Hladik, M., Paulheim, H.: Kgvec2go - knowledge graph embeddings as a service. In: Calzolari, N., et al. (eds.) Proceedings of The 12th Language Resources and Evaluation Conference, LREC 2020, Marseille, France, 11–16 May 2020. pp. 5641–5647. European Language Resources Association (2020). <https://aclanthology.org/2020.lrec-1.692/>
13. Portisch, J., Hladik, M., Paulheim, H.: Rdf2vec light - a lightweight approach for knowledge graph embeddings. In: Taylor, K.L., Gonçalves, R.S., Lécué, F., Yan, J. (eds.) Proceedings of the ISWC 2020 Demos and Industry Tracks: From Novel Ideas to Industrial Practice co-located with 19th International Semantic Web Conference (ISWC 2020), Globally online, November 1–6, 2020 (UTC). CEUR Workshop Proceedings, vol. 2721, pp. 79–84. CEUR-WS.org (2020). <https://ceur-ws.org/Vol-2721/paper520.pdf>
14. Portisch, J., Paulheim, H.: Putting RDF2vec in order. In: Seneviratne, O., Pesquita, C., Sequeda, J., Etcheverry, L. (eds.) Proceedings of the ISWC 2021 Posters, Demos and Industry Tracks: From Novel Ideas to Industrial Practice co-located with 20th International Semantic Web Conference (ISWC 2021), Virtual Conference, 24–28 October 2021. CEUR Workshop Proceedings, vol. 2980. CEUR-WS.org (2021). <https://ceur-ws.org/Vol-2980/paper352.pdf>
15. Portisch, J., Paulheim, H.: Walk this way! entity walks and property walks for RDF2vec. *CoRR abs/2204.02777* (2022). 10.48550/arXiv.2204.02777, <https://doi.org/10.48550/arXiv.2204.02777>
16. Ristoski, P., Rosati, J., Noia, T.D., Leone, R.D., Paulheim, H.: Rdf2vec: RDF graph embeddings and their applications. *Seman. Web* **10**(4), 721–752 (2019). <https://doi.org/10.3233/SW-180317>, <https://doi.org/10.3233/SW-180317>
17. Ristoski, P., de Vries, G.K.D., Paulheim, H.: A collection of benchmark datasets for systematic evaluations of machine learning on the semantic web. In: Groth, P., et al. (eds.) The Semantic Web - ISWC 2016–15th International Semantic Web Conference, Kobe, Japan, 17–21 October 2016, Proceedings, Part II. LNCS, vol. 9982, pp. 186–194 (2016). https://doi.org/10.1007/978-3-319-46547-0_20, https://doi.org/10.1007/978-3-319-46547-0_20
18. Salzberg, S.: On comparing classifiers: Pitfalls to avoid and a recommended approach. *Data Min. Knowl. Discov.* **1**(3), 317–328 (1997). <https://doi.org/10.1023/A:1009752403260>, <https://doi.org/10.1023/A:1009752403260>
19. Shi, B., Weninger, T.: Open-world knowledge graph completion. In: McIlraith, S.A., Weinberger, K.Q. (eds.) Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-2018), the 30th Innovative Applications of Artificial Intelligence (IAAI-2018), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-2018), New Orleans, Louisiana, USA, 2–7 February 2018, pp. 1957–1964. AAAI Press (2018). <https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/view/16055>

20. Steenwinckel, E., et al.: Walk extraction strategies for node embeddings with RDF2vec in knowledge graphs. In: Kotsis, G., et al. (eds.) DEXA 2021. CCIS, vol. 1479, pp. 70–80. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-87101-7_8
21. Toutanova, K., Chen, D.: Observed versus latent features for knowledge base and text inference. In: Proceedings of the 3rd Workshop on Continuous Vector Space Models and Their Compositionality, pp. 57–66 (2015). <https://doi.org/10.18653/v1/W15-4007>, <https://www.doi.org/10.18653/v1/W15-4007>
22. Trouillon, T., Welbl, J., Riedel, S., Gaussier, É., Bouchard, G.: Complex embeddings for simple link prediction. In: Balcan, M., Weinberger, K.Q. (eds.) Proceedings of the 33rd International Conference on Machine Learning, ICML 2016, New York City, NY, USA, June 19–24, 2016. JMLR Workshop and Conference Proceedings, vol. 48, pp. 2071–2080. JMLR.org (2016). <https://proceedings.mlr.press/v48/trouillon16.html>
23. Zheng, D., et al.: DGL-KE: training knowledge graph embeddings at scale. In: Huang, J., et al. (eds.) Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval, SIGIR 2020, Virtual Event, China, July 25–30, 2020. pp. 739–748. ACM (2020). <https://doi.org/10.1145/3397271.3401172>, <https://doi.org/10.1145/3397271.3401172>