GEval: A Modular and Extensible Evaluation Framework for Graph Embedding Techniques

Pellegrino, Maria Angela; Altabba, Abdulrahman; Garofalo, Martina; Ristoski, Petar; Cochez, Michael

published in
The Semantic Web
2020

DOI (link to publisher)
10.1007/978-3-030-49461-2_33

document version
Publisher's PDF, also known as Version of record

GEval: A Modular and Extensible Evaluation Framework for Graph Embedding Techniques

Maria Angela Pellegrino[1(✉)], Abdulrahman Altabba[2], Martina Garofalo[1], Petar Ristoski[3], and Michael Cochez[4]

[1] Department of Computer Science, University of Salerno, Fisciano, Italy
mapellegrino@unisa.it, margar1994@gmail.com
[2] Information Systems and Databases, RWTH Aachen University, Aachen, Germany
abdulrahman.altabba@rwth-aachen.de
[3] IBM Research Almaden, San Jose, CA, USA
petar.ristoski@ibm.com
[4] VU Amsterdam, Amsterdam, The Netherlands
m.cochez@vu.nl

Abstract. While RDF data are graph shaped by nature, most traditional Machine Learning (ML) algorithms expect data in a vector form. To transform graph elements to vectors, several graph embedding approaches have been proposed. Comparing these approaches is interesting for 1) developers of new embedding techniques to verify in which cases their proposal outperforms the state-of-art and 2) consumers of these techniques in choosing the best approach according to the task(s) the vectors will be used for. The comparison could be delayed (and made difficult) by the choice of tasks, the design of the evaluation, the selection of models, parameters, and needed datasets. We propose *GEval*, an evaluation framework to simplify the evaluation and the comparison of graph embedding techniques. The covered tasks range from ML tasks (Classification, Regression, Clustering), semantic tasks (entity relatedness, document similarity) to semantic analogies. However, GEval is designed to be (easily) extensible. In this article, we will describe the design and development of the proposed framework by detailing its overall structure, the already implemented tasks, and how to extend it. In conclusion, to demonstrate its operating approach, we consider the parameter tuning of the KGloVe algorithm as a use case.

Keywords: Evaluation framework · Knowledge graph embedding · Machine Learning · Semantic tasks

Resource type Software Framework
GitHub link https://github.com/mariaangelapellegrino/Evaluation-Framework
Permanent URL https://doi.org/10.5281/zenodo.3715267.

A. Harth et al. (Eds.): ESWC 2020, LNCS 12123, pp. 565–582, 2020.
https://doi.org/10.1007/978-3-030-49461-2_33

1 Introduction

A graph embedding technique takes an RDF graph as its input and creates a low-dimensional feature vector representation of nodes and edges of the graph. Formally, a graph embedding technique aims to learn a function $f: G(V, E) \rightarrow \mathbb{R}^d$ which is a mapping from the graph $G(V, E)$, where V is the set of vertices and E is the set of edges, to a set of numerical embeddings for the vertices and edges, where d is the dimension of the embedding. The purpose of such a graph embedding technique is to represent each node and edge in a graph (or a subset of them) as a low-dimensional vector; often while preserving semantic properties (e.g., keeping similar entities close together) and/or topological features. If only nodes are embedded, it is called node embedding.

A desirable property for the obtained vectors is that they would be task-independent, meaning that they can be reused for other applications as they were created for. Therefore, it is useful to have an idea of how the vectors perform on different tasks to broaden the insight into the information the embedding algorithm is able to preserve. It is also important to know whether the vectors show very good performance on a given task while their performance degrades significantly on others. It is important to bear in mind that the *extrinsic* evaluation is not the only (and probably it is not the best) way to elect the best embedding approach. However, this kind of evaluation is extremely useful to choose the best set of vectors according to the tasks they will be used for. Besides the evaluation and comparison, a systematic evaluation is also useful in parameter tuning. In fact, many embedding algorithms have various parameters, which are difficult to set. Therefore, in this scenario, it is interesting to compare different versions of the same algorithm and check how the parameters affect extrinsic evaluations.

By considering extrinsic evaluation and comparison, one of the first aspects to take into account is the choice of the tasks. Systematic comparative evaluations of different approaches are scarce; approaches are rather evaluated on a handful of often project-specific data sets. Usually, they do not show how the algorithm performs on large and less regular graphs, such as DBpedia[1] or Wikidata[2].

To simplify the evaluation phase while providing a wider comparison, we present the design and open-source implementation of GEval (Graph Embeddings Evaluation), a software *evaluation framework* for knowledge graph (KG) embedding techniques. The provided tasks range from machine learning (ML) (classification, regression, and clustering) and semantic tasks (entity relatedness and document similarity) to semantic analogies. Furthermore, the framework is designed to be extended with additional tasks. It is useful both for embedding algorithm developers and users. On one side, when a new embedding algorithm is defined, there is the need to evaluate it upon tasks it was created for. On the other side, users can be interested in performing particular tests and choosing the embedding algorithm that performs best for their application. Our goal is

[1] https://dbpedia.org/.
[2] https://www.wikidata.org.

to address both situations providing a ready-to-use framework that can be customized and easily extended. This work is a continuation on our earlier work [13].

This paper is structured as follows: in Sect. 2, we present related work; in Sect. 3, we describe our evaluation framework, detailing the implemented tasks in Sect. 4; in Sect. 5, we present a practical use case, i.e., how to exploit the proposed framework in parameter tuning; then, we conclude with considerations and some final observation.

2 Related Work

The easiest way to categorize software evaluation frameworks is related to the provided tasks. Moreover, frameworks can be distinguished according to the expected input. In the case of embedding algorithms, an evaluation framework can take as input the model and train it before starting the evaluation. In the alternative, it could expect pre-computed vectors. The input format can influence the type of covered tasks. For example, for a fair comparison in link prediction, it is important to know the input graph used to train the model. Only by bounding the training set, it is possible to fairly test unknown edges and verify the ability of the embedding algorithm to forecast only positive edges. In this section, we will focus on frameworks to evaluate graph embedding approaches by pointing out the covered tasks. In Table 1, we will list frameworks by reporting the publication year, the covered tasks, and if they expect the model or the pre-trained vectors.

Table 1. For each evaluation framework, we report 1) the publication year, 2) the available tasks, and 3) the input format. ***Task label***: *Clas* for Classification, *Clu* for Clustering, *DocSim* for Document Similarity, *EntRel* for Entity Relatedness, *LP* for Link Prediction, *Net Comp* for Network Compression, *Reg* for Regression, *SemAn* for Semantic Analogies, *Vis* for Visualization.

	Year	**Tasks**	**Embedding technique**
Bonner et al. [3]	2017	Topological structure	Model
GEM [7]	2018	Clas, Clu, LP, Net Comp, Vis	Model
Rulinda et al. [16]	2018	Clu, LP, Vis	Model
OpenNE	2019	Clas, Vis	Model
EvalNE [11]	2019	LP	Model
AYNEC [2]	2019	LP	-
Bogumil et al. [9]	2019	Clu	Model
GEval	2019	Clas, Clu, DocSim, EntRel, Reg, SemAn	Vectors

Goyal and Ferrara [7] released an open-source Python library, GEM (Graph Embedding Methods), which provides a unified interface to train many state-of-the-art embedding techniques on the Zachary's Karate graph and test them on

network compression, visualization, clustering, link prediction, and node classification. GEM modular implementation should help users to introduce new datasets. This library is bounded to the embedding methods provided by the authors, while the introduction of new embedding approaches requires compliance with an interface defined within the library. It focuses more on the implementation of embedding approaches than on the effective evaluation workflow.

Bonner et al. [3] provide a framework to assess the effectiveness of graph embeddings approaches in capturing detailed topological structure, mainly at the vertex level. For instance, they hypothesize that a good graph embedding should be able to preserve the vertex centrality. The evaluation is based on empirical and synthetic datasets. Also this task needs to be aware of the graph used during the training phase of the model to verify the presence of topological structure in the vectors. The authors do not state if further tasks can be added.

Rulinda et al. [16] implement a collection of graph embedding techniques and, once trained, they evaluate the resulting vectors on clustering, link prediction, and visualization. The framework focuses only on uniform graphs.

Even if OpenNE[3] is an open-source package to train and test graph embedding techniques on node classification and network visualization, it is more focused on the generation phase than on the evaluation aspect.

EvalNE [11] focuses on the Link Prediction task. It starts from an incomplete training graph along with a (more) complete version of the graph to test and verify the prediction power. EvalNE interprets the link prediction task as a binary classification task and it can be extended by adding other binary classifiers.

Also AYNEC [2] focuses on the link prediction task. It provides some incomplete graphs as a training set. The user, on his/her behalf, can train a graph embedding algorithm on these datasets and run the link prediction task on the testing datasets. AYNEC takes as input the forecast edges and evaluate them by considering the complete graph. It provides all the useful phases to evaluate the link prediction task, but the link prediction step is charged to the user.

Bogumil et al. [9] focus on the clustering task and they define a divergence score that can be used to distinguish good and bad embeddings. They test a pool of embeddings of synthetic and real datasets. From their work it appears that they plan to extend the framework to hypergraphs. They do not state how or whether the framework can be used and extended for other tasks.

We propose GEval, an evaluation framework that combines both ML and semantic tasks. The advantage of considering also semantic tasks in the evaluation is due to the recent trend to extend neural embedding techniques, traditionally introduced only for natural language word sequences, also to KGs. Besides the wider diffusion of this kind of embedding techniques, to the best of our knowledge, they are not incorporated in a KG evaluation framework. Our tool is designed to be modular and extensible. It takes as input already pre-trained vectors without constraint on the way these vectors are produced.

[3] https://github.com/thunlp/OpenNE.

3 GEval: Evaluation Framework for Graph Embeddings

GEval is a software framework to perform evaluation and comparison of graph embedding techniques. It takes as input a file containing pre-computed vectors. More in detail, the input file must provide pairs of an embedded node (represented by its IRI) and the related vector. For each task, ground truth is modeled as a gold standard, which will be further referred to as *gold standard datasets*. They contain the tested entities and its ground truth. In Fig. 1, there is a diagrammatic representation of the involved parts in the framework and their interactions. The starting point is the *Evaluation Manager* which is the orchestrator of the whole evaluation and it is in charge of 1) verifying the correctness of the parameters set by the user, 2) instantiating the correct data manager according to the data format provided by the user, 3) determining which task(s) the user asked for, and 4) managing the storage of the results.

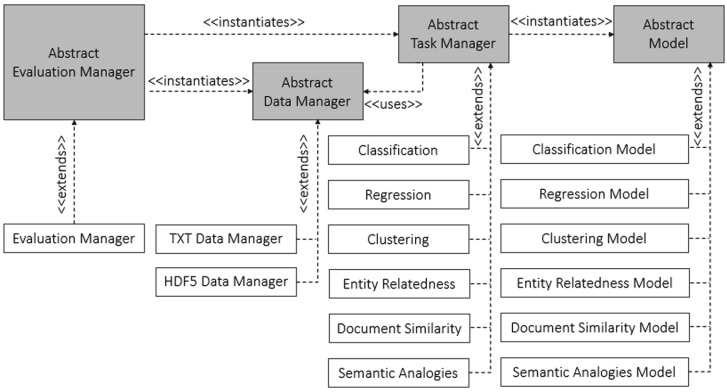


Fig. 1. The diagram represents the components of the framework. The blue boxes represent abstract classes, while the white boxes represent concrete classes. If A <<`extends`>> B, A is the concrete class which extends and makes the abstract behaviour of A concrete. If A <<`instantiates`>> B, A creates an instance of B. If A <<`uses`>> B, A is dependent on B.

Running Details. GEval can be run from the command line and by APIs. As stressed before, most of the actions performed by the evaluator strictly depend on the user settings and preferences. Users can customize the evaluation settings by: i) specifying them on the command line (useful when only a few settings must be specified and the user desires to use the default value for most of the parameters); ii) organizing them in an XML file (especially useful when there is the need to define most of the parameters); iii) passing them to a function that starts the evaluation. In the `example` folder of the project on GitHub, there are examples for the different ways to provide the parameters. The parameters are:

vectors_file path of the file where the embedded vectors are stored;
vector_file_format data format of the input file;
vectors_size length of the embedded vectors;
tasks list of the tasks to execute;
parallel task execution mode;
debugging_mode `True` to run the tasks by reporting all the information collected during the run, `False` otherwise;
similarity_metric metric used to compute the distance among vectors. When an embedding technique is created, there is often also a specific distance metric which makes sense to measure similarity in the created space. This measure is a proxy for the similarity between the entities in the graph;
analogy_function function to compute the analogy among vectors. By specifying `None`, the default function is used. To customize it, the programmatically provided function handler must take 3 parameters and return a number.

```
def default_analogy_function(a, b, c){return b - a + c}
```

top_k it is used to look for the *right* answer among the top_k values. The vector returned by the analogy function (that will be referred to as *predicted vector*) gets compared with the k most similar ones. If the predicted vector is among the k most similar ones, the answer is considered correct;
compare_with list of the runs to compare the results with. Each run is identified uniquely and the user can refer to specific runs to compare with by using these IDs. It is auto-generated by the framework and it corresponds to *vectorFilename_vectorSize_similarityMetric_topK* and a progressive number to disambiguate runs with the same parameters.

In Table 2, we will detail for each parameter the default value, the accepted options, if the parameter is mandatory, and which component uses it.

Data Management. The input file can be provided either as a plain text (also called TXT) file or as a HDF5. In particular, the TXT file must be a white-space separated value file with a line for each embedded entity. Each row must contain the IRI of the embedded entity and its vector representation. Since most of the tasks implemented in the evaluation framework need to intersect (inner join) the data set(s) used as gold standard and the input file, we also work with an indexed file format to speed up the merging phase. Indeed, the direct access to the entities of interest gives us the chance to save time during the merging step and also to save space since we do not read the complete vectors file into the memory. Among the available formats, we decided to work with HDF5[4]. The HDF5 vectors file must provide one group called *vectors*. In this group there must be a dataset for each entity with the base32 encoding of the entity name as the dataset name and the embedded vector as its value. Depending on the file format, the data manager decides to read the whole content or not. For instance, the TXT file will be completely read. HDF5, instead, provides an immediate access to vectors

[4] https://www.hdfgroup.org/solutions/hdf5/.

Table 2. The table reports details for each parameter: the parameter name, the default value, the accepted options, if it is mandatory, and which component/task uses it. The * means that the parameter is used by all the tasks.

Parameter	Default	Options	Mandatory	Used_ by
vectors_file			✓	*
vector_file_format	TXT	TXT, HDF5		data_manager
vectors_size	200	numeric value		data_manager
tasks	_all	Class, Reg, Clu, EntRel, DocSim, SemAn		evaluation_manager
parallel	False	boolean		evaluation_manager
debugging_mode	False	boolean		*
similarity_metric	cosine	Sklearn affinity metrics[a]		Clu, DocSim
analogy_function	None	handler to function		semantic_analogy
top_k	2	numeric value		SemAn
compare_with	_all	list of run IDs		evaluation_manager

[a]https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics.pairwise

of interest. Each data manager has to i) read the gold standard datasets, ii) read the input file and iii) determine how to merge each gold standard dataset and the input file. The behaviour of the data manager is modelled by the `abstract data manager`, implemented by a concrete data manager based on the input file format and it refined by task data managers.

Task Management. Once data have been accessed, the task(s) can be run. Each task is modelled as a pair of task manager and model. The task manager is in charge of 1) merging the input file and each gold standard file (if more than one is provided) (by exploiting the data manager), 2) instantiating and training a model for each configuration to test, and 3) collecting and storing results computed by the model. Therefore, the framework is in charge of retrieving entities of interest, i.e., entities listed in gold standard datasets, and the related vectors. Only the intersection of entities provided by the input file and the ones required by gold standard datasets will be considered into the evaluation. Each task can decide if the missing entities (i.e., the entities required into the gold standard file, but absent into the input file) will affect the final result of the task or not. According to the user preferences, tasks can be run in sequential or in parallel. The parallelization is trivially handled: by asking for the parallel execution, a new process is created for each task and it is immediately run. Once results are returned, they are collected and stored by the Evaluation Manager.

Out-of-the-Box Tasks and Extension Points. The provided tasks range from ML tasks (Classification, Regression, Clustering), semantic tasks (entity relatedness and document similarity) to semantic analogies. Each task is kept separate, by satisfying the modularity requirement. By the usage of abstraction, it is easy to add new tasks and/or data manager. The `abstract data manager` defines the interface of a data manager, while `abstract task manager` and `abstract model` define the interface of a new task. Extending the framework with new data formats and/or new tasks is as simple as creating a class implementing these interfaces. To further enrich an already implemented task, it is easy to retrieve the exact point to modify since each task is limited to its task manager and model. Moreover, to extend the evaluation also to edges, it is enough to create gold standard dataset containing edges and related ground truth.

Results Storage. For each task and for each file used as gold standard, GEval will create i) an output file that contains a reference to the file used as gold standard and all the information related to evaluation metric(s) provided by each task, ii) a file containing all the *missing* entities, iii) a log file reporting extra information, occurred problems, and execution time, and iv) information related to the comparison with previous runs. In particular, about the comparison, it reports the values effectively considered in the comparison and the ranking of the current run upon the other ones. The results of each run are stored in the directory `results/result_<starting time of the execution>` generated by the evaluation manager in the local path.

4 Out of the Box Available Tasks

The available tasks are Classification, Regression and Clustering that belong to the ML field, and Entity Relatedness, Document Similarity and Semantic Analogies, more related to the semantic field. Each task is implemented as a concrete task manager that implements functionalities modelled by the `Abstract Task Manager`. Each task follows the same workflow:

1. the task manager asks data manager to merge each gold standard dataset and the input file and keeps track of both the retrieved vectors and the *missing* entities, i.e., entities required by the gold standard dataset, but absent in the input file;
2. a model for each configuration is instantiated and trained;
3. the missing entities are managed: it is up to the task to decide if they should affect the final result or they can be simply ignored;
4. the scores are calculated and stored.

We will separately analyse each task, by detailing the gold standard datasets, the configuration of the model(s), and the computed evaluation metrics.

4.1 Classification

Table 3 contains details related to the gold standard datasets used for the Classification task, the trained models and its parameter(s) (if any), and the evaluated metric. The gold standard datasets have been designed for use in quantitative performance testing and systematic comparisons of approaches. They can be freely downloaded from the author's website.[5] The missing entities are simply ignored. The results are calculated using stratified 10-fold cross-validation.

Table 3. Details of the Classification task.

	Dataset	**Semantic of classes**	**Classes**	**Size**	**Source**
INPUT	Cities	Living style	3	212	Mercer
	AAUP	Salary of professors	3	960	JSE
	Forbes	Agency income	3	1,585	Forbes
	Albums	Album popularity	2	1,600	Metacritic
	Movies	Movie popularity	2	2,000	Metacritic
	Model			**Conf**	
MODEL	Naive Bayes (NB)			-	
	C4.5 decision tree			-	
	k-NN			k=3	
	SVM			$C \in \{10^{-3}, 10^{-2}, 0.1, 1, 10, 10^2, 10^3\}$	
OUTPUT	**Metric**		**Range**	**Optimum**	
	Accuracy		[0,1]	Highest	

4.2 Regression

Table 4 contains details related to the gold standard datasets used for the Regression task, the trained models and its parameter(s) (if any), and the evaluated metric. The gold standard datasets used for the Regression tasks are the same used for the Classification task. The missing entities are simply ignored. The results are calculated using stratified 10-fold cross-validation.

4.3 Clustering

Table 5 contains details related to the gold standard datasets used for the Clustering task, the trained models and its parameter(s) (if any), and the evaluated metrics.

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

Table 4. Details of the Regression task.

	Dataset	Semantic of values	Size	Source
INPUT	Cities	Living style	212	Mercer
	AAUP	Salary of professors	960	JSE
	Forbes	Agency income	1,585	Forbes
	Albums	Album popularity	1,600	Metacritic
	Movies	Movie popularity	2,000	Metacritic
MODEL	**Model**		**Conf**	
	Linear Regression		-	
	M5Rules		-	
	k-NN		k=3	
OUTPUT	**Metric**		**Range**	**Optimum**
	Root Mean Squared Error (RMSE)		[0,1]	Lowest

The gold standard datasets encompass different domains:

- the *Cities, Metacritic Movies, Metacritic Albums, AAUP* and *Forbes datasets* are the datasets already used for the Classification and Regression task, here used as a single dataset. Since these datasets contain resources belonging to distinct class (City, Music Album, Movie, University, and Company), the goal of the clustering approach on this dataset is verifying the ability to distinguish elements belonging to completely different classes. Therefore, the entities from each set are considered member of the same cluster;
- *Cities* and *Countries* are retrieved by SPARQL queries over DBpedia, asking for all `dbo:City`[6] and `dbo:PopulatedPlace`, respectively.
- the small version of the dataset *Cities* and *Countries* is defined as before, but balancing the clusters by retrieving only 2,000 Cities. The balancing operation has been performed since the majority of clustering approaches (k-means is an example in this direction) attempt to balance the size of the clusters while minimising the interaction between dissimilar nodes [17]. Therefore, unbalanced clusters could strongly affect the final results.
- *Football* and *Basketball teams* are retrieved by SPARQL queries run against the DBpedia SPARQL endpoint, asking for all `dbo:SportsTeam` whose identifier contains respectively `football_team` or `basketball_team`.

All the models but k-Means allow to customize the distance function. Therefore, we exploit the similarity_metric given in input by the user. Only k-Means is bounded (due to its implementation) to the euclidean distance.

For each missing entities a *singleton cluster* is created, i.e., a cluster which contains only the current entity. Further, soft clustering approaches, such as DBscan, do not cluster all entities. We call these entities *miss-clustered entities*

[6] `dbo` is the prefix of http://dbpedia.org/ontology/.

Table 5. Details of the Clustering task.

	Dataset	Interpretation of clusters	Clusters	Size
INPUT	Teams	{Football T., Basketball T.}	2	4,206
	Cities and Countries	{Cities, Countries}	2	4,344
	Cities, Albums, Movies, AAUP, Forbes	{Cities, Albums, Movies, Universities, Societies}	5	6,357
	Cities and Countries	{Cities, Countries}	2	11,182
MODEL	**Model**		**Conf**	
	Agglomerative Clu.		*similarity_metric*	
	Ward Hierarchical Clu.		*similarity_metric*	
	DBscan		*similarity_metric*	
	k-Means		-	
OUTPUT	**Metric**		**Range**	**Optimum**
	adjusted rand score		[-1,1]	Highest
	adjusted mutual info score		[0,1]	Highest
	Fowlkes Mallow index		[0,1]	Highest
	v_measure score		[0,1]	Highest
	homogeneity score		[0,1]	Highest
	completeness score		[0,1]	Highest

and manage them exactly as the missing entities, i.e., we create a singleton cluster for each of them. The evaluation metrics are applied to the combination of the clusters returned by the clustering algorithm and all the *singleton clusters*.

4.4 Entity Relatedness

In the entity relatedness task we assume that two entities are related if they often appear in the same context [15]. The goal of this task is to check if embedded vectors are able to preserve the semantic relatedness which can be detected from the original entities. The relatedness between vectors is brought back to the computation of the similarity metric among them.

Table 6 contains details related to the gold standard dataset used for the Entity Relatedness task, the model, and the evaluated metrics. The original version of the gold standard dataset KORE [8] consists of 420 pairs of words: for each of 21 main words, there are 20 words whose relatedness has been manually assessed. The dataset has been adapted by manually resolving each word as DBpedia entities. The main entities belong to four distinct categories: Actors, Companies, TV series, and Video-games. Missing entities are managed as follows:

- if a main entity is missing, it is simply ignored;
- if one or more related entities attached to the same main entity are missing, first, the task compute the similarity among the available entities as reported in the model described in the Table 6; then, all the missing related entities are randomly put in the tail of the sorted list, and, finally, the evaluation metric

Table 6. Details of the Entity Relatedness task.

INPUT	**Dataset**	**Structure**	**Size**
	KORE [8]	*main entity* with a *sorted list of 20 related entities*	420
MODEL	**Model**		**Conf**
	sim_scores = [ ] for each main entity as *me*: for each related entity as *re*: sim_scores.add(similarity(*me, re*)) sort(sim_scores) //from more to less similar		*similarity_metric*
OUTPUT	**Metric**	**Range**	**Interpretation**
	Kendall's tau correlation coefficient	[-1,1]	Extreme values: correlation Values close to 0: no correlation

is calculated on the ranking obtained by the similarity score among all the available pairs concatenated with the missing entities.

4.5 Document Similarity

Table 7 contains details related to the gold standard datasets used for the Document Similarity task and the evaluated metric. The original dataset used as gold standard is the LP50 data set [10], a collection of 50 news articles from the Australian Broadcasting Corporation. It were pairwise annotated manually by 8 to 12 different university students who evaluated the similarity among documents assigning to each pair a point in the range [1, 5] where 5 means maximum similarity. To create the gold standard dataset, we worked as follows. For each pair of documents, the average of the manually assessed rates is computed. Then, we the extract the entities from the documents using the annotator xLisa[7].

Model. The algorithm takes two documents d_1 and d_2 as its input and calculates their similarity as follows:

- For each document, the related set of entities is retrieved. The output of this step are the sets E_1 and E_2, respectively.
- For each pair of entities (i.e., for the cross product of the sets), the similarity score is computed.
- Only the maximum value is preserved for determining the document similarity evaluation. Therefore, for each entity in E_1 the maximum similarity to an entity in E_2 is kept and vice versa.

[7] http://km.aifb.kit.edu/sites/xlisa/.

Table 7. Details of the Document Similarity task.

INPUT	**Dataset**	**Structure**	**Size**
	LP50 [10]	doc1 doc2 avg	50 docs
MODEL	**Model**		**Conf**
	it is described into the Doc. Sim. section		*similarity_metric*
OUTPUT	**Metric**	**Range**	**Interpretation**
	Pearson correlation (P_cor)	[-1,1]	Extreme: correlation Close to 0: no correlation
	Spearman correlation (S_cor)	[-1,1]	Extreme: correlation Close to 0: no correlation
	Harmonic mean of P_cor and S_cor	[-1,1]	Extreme: correlation Close to 0: no correlation

– The similarity score between the two documents is calculated by averaging the sum of all these maximum similarities.

The annotators also provided weights. Hence, the previous procedure is repeated by considering the weights to normalise the distances. The Document Similarity task simply ignores any missing entities and computes the similarity only on entities that both occur in the gold standard dataset and in the input file.

4.6 Semantic Analogies

The Semantic Analogies task is based on quadruplets of words ($word_1, word_2$, $word_3, word_4$) and it checks whether it is possible to predict the last word based on the first three ones, given that the same analogy exists between $word_1$ and $word_2$ as between $word_3$ and $word_4$. A practical example [12] is the quadruplet (king, queen, man, woman). Then, one can compute X = vector("queen")-vector("woman") + vector("man") and check if X is near to the embedding of "king". In Word2Vec both syntactic and semantic analogies are considered. However, in our evaluation framework we consider only semantic analogies as KGs do generally not provide conjugated verbs, female and male nouns, singular and plural words, which are required information to perform the syntactic analogy evaluation. The original datasets used as gold standard can be freely be downloaded[8]. To create the gold standard datasets for the Semantic Analogies task, all the words have been manually substituted with DBpedia entities (Table 8).

Model. The task takes the quadruplets (v_1, v_2, v_3, v_4) and works on the first three vectors to predict the fourth one. Among all the vectors, the nearest to the predicted one is retrieved, where the *nearest* is computed by the dot product.

[8] https://sites.google.com/site/semeval2012task2/download.

Table 8. Details of the Semantic Analogies task.

INPUT	**Dataset**	**Structure**	**Size**	**Source**
	Capitals and countries	ca1 co1 ca2 co2	505	Word2Vec [12]
	Currency (and Countries)	cu1 co1 cu2 co2	866	Word2Vec [12]
	Cities and State	ci1 st1 ci2 st2	2,467	Word2Vec [12]
	(All) capitals and countries	ca1 co1 ca2 co2	4,523	Word2Vec [12]
MODEL	**Model**			**Conf**
	it is described into the Sem. An. section			*analogy_function*
OUTPUT	**Metric**		**Range**	**Optimum**
	accuracy		[0,1]	Highest

The analogy function to compute the predicted vector can be customised. The vector returned by the function (the *predicted vector*) gets compared with the *top_k* most similar ones. If the actual fourth vector is among the *top_k* most similar ones, the answer is considered correct. *top_k* can be customised by the user.

5 Evaluation and Use Case

Execution Time Evaluation. We have already tested the execution time of each task both in sequential and in parallel [13]. We are interested in estimating how the vector size affects the computational time for the Classification and Regression tasks. The experiments are performed on a system with an Intel(R) Core(TM) i7-8700T CPU at 2.40 GHz and 16 GB RAM. We evaluated vectors produced by RDF2Vec [6] and by KGloVe [5]. Here, we report only results related to KGloVe since in both cases we observed the same trend. We extrapolated only vectors required by the Classification and Regression tasks, because of memory limitations. Then, we crop the filtered vectors by considering $[10, 20, 50, 100, 150, 180, 200]$ as size. We perform the Classification and the Regression tasks on all the obtained vectors. In Fig. 2, you can observe the actual execution times of the ML tasks and you can note that the execution time of Classification and Regression tasks is linearly correlated with the vector size.

Use Case. In this use case, we focus on parameter tuning and we will use results produced by ML tasks to detect the best combination of hyper-parameters. In this evaluation, we consider a modified version of KGloVe [4] where the difference with the original algorithm lies in the parallel implementation (GPU based) of the underlying GloVe [1]. Our goal is to optimize KGloVe parameters to find out the values that produce vectors which gain the best results in ML tasks. In Fig. 3, the entire pipeline is visible. Starting from DBpedia 2016, the *graph walks* produces a co-occurrence matrix for the nodes of the graph [4]. The parameters that affect the co-occurrence matrix are α, ϵ, and the *weighting function* which is applied once on the graph (forward weighting function) and once

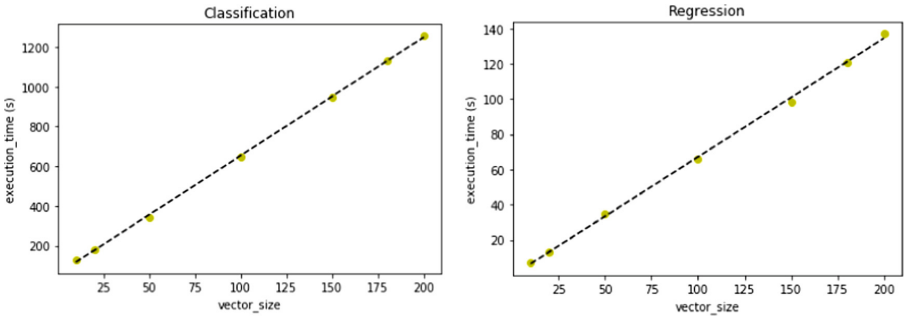


Fig. 2. It represents how vector size affects the execution time of the Classification and Regression tasks: they are linearly correlated.

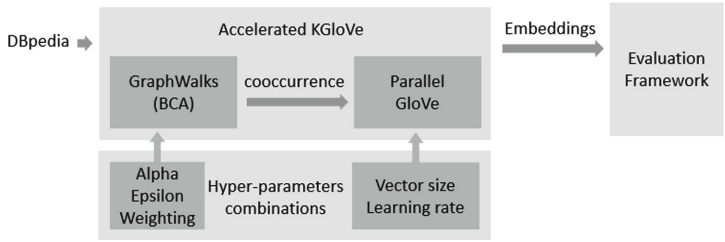
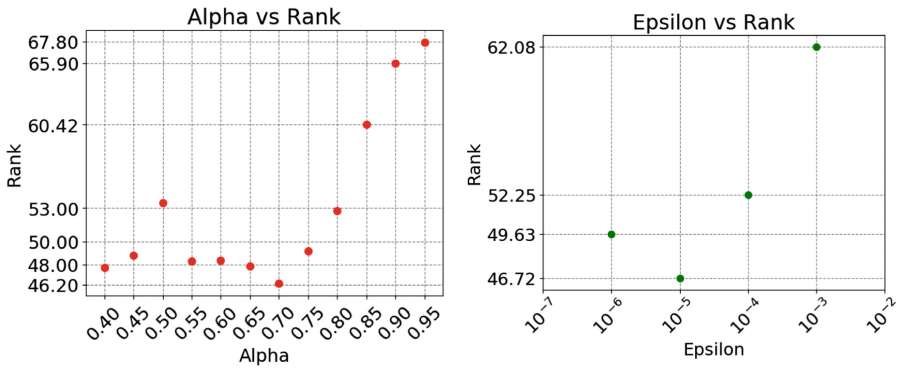


Fig. 3. Pipeline of hyper-parameter tuning

on the graph with reversed edges (backward weighting function). The *Parallel GloVe* [1] implementation takes the co-occurrences matrix as input and trains the vectors in parallel by minimising the loss function defined by GloVe [14]. The produced embeddings are affected by GloVe parameters, i.e., the vector size and the learning rate. To reduce the employed resources in finding the optimum parameters combination, we opt for a random search. We performed the evaluation by considering a set of 105 uniformly random generated combinations: we tested $\alpha \in \{0.4, 0.45, 0.5, 0.55, 0.6, 0.65, 0.7, 0.75, 0.8, 0.85, 0.9, 0.95\}$; *learning rate*$= 0.01$; *vector size*$= 50$; $\epsilon \in \{10^{-6}, 10^{-5}, 10^{-4}, 10^{-3}\}$; *weighting functions* (forward, backward) $|\text{weighers}| \times |\text{weighers}| = 12$ options $\times$ 12 options. Once produced vectors, we evaluate them on the Classification and Regression tasks implemented by GEval. GEval runs 10 times both the Classification and the Regression task and returns the average result. By considering the combination of models and their configurations (see Table 3), the classification task produces 10 accuracy scores, while the regression task produces 2 RMSE scores (k-NN and LR). For each run, we take the average of the results produced by the 5 datasets used as a gold standard.

Then, we rank the runs (and therefore the parameter combination) according to the 12 different scores. The average rank is taken for evaluating the corresponding parameter combination. To find out the performance according to a given parameter y, we plot the performance for each run of y (if there is a value

of y which is used multiple times, we compute the average). In Figs. 4a and b the ranked values of α and ϵ are presented. We observe that $\alpha = 0.7$ and $\epsilon = 10^{-5}$ produce the best embeddings for ML tasks.



(a) α Ranking (b) ϵ Ranking

Fig. 4. Parameter tuning of KGloVe. Lower results are the best ones.

6 Conclusion

GEval aims to simplify the evaluation phase of KG embedding techniques providing tasks ranging from ML to semantic ones. To the best of our knowledge, our proposal is one of the most comprehensive frameworks to evaluate KG embedding techniques for heterogeneous graphs. GEval can be used in evaluation and comparison over multiple tasks. Moreover, it can be also used in parameters tuning, as shown in the presented use case. The modularity of GEval is achieved by keeping each task separated, but still abstracting away the commonalities.

Our software framework can be used to perform benchmarks, but it is not designed as a benchmark itself. We provide the framework as a command-line tool and by APIs[9]. We do not provide server-side execution, since the computation of tasks and the memory requirements can be onerous and can not be determined apriori. In our opinion, it is more beneficial to provide the software and give the opportunity of choosing the hardware requirements adapt to the size of the managed vectors. GEval is not bounded to evaluate only node embeddings. By incorporating also edges into the gold standard datasets, it is possible to consider graph embeddings which embed both nodes and edges. Our default gold standard datasets contain DBpedia entities. However, this is not a framework requirement; it is possible to evaluate different sets of entities (and embeddings of other KGs) by adding gold standard datasets.

[9] https://pypi.org/project/evaluation-framework/.

The framework has been published with an open-source licence in order to be used by the whole community. GEval is already of interest for experimentation with graph embedding techniques by the authors' institutes (Fraunhofer FIT, the RWTH Aachen University, the University of Salerno, and IBM research). Moreover, other institutes show an interest in collaborating to this project. The Télécom ParisTech is interested in extending the already available tasks to incorporate gold standard datasets related to (French) museums. We are now working to create the gold standard of interest. Moreover, we are working with the University of Madrid to incorporate the Link Prediction task in GEval. We are certain that also others will benefit from this valuable resource.

References

1. Altabba, A.: Accelerating KGloVe graph embedding (2019). Unpublished thesis
2. Ayala, D., Borrego, A., Hernández, I., Rivero, C.R., Ruiz, D.: AYNEC: all you need for evaluating completion techniques in knowledge graphs. In: Hitzler, P., et al. (eds.) ESWC 2019. LNCS, vol. 11503, pp. 397–411. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-21348-0_26
3. Bonner, S., Brennan, J., Kureshi, I., Theodoropoulos, G., McGough, A.S., Obara, B.: Evaluating the quality of graph embeddings via topological feature reconstruction. In: 2017 IEEE International Conference on Big Data, pp. 2691–2700 (2017)
4. Cochez, M., Ristoski, P., Ponzetto, S.P., Paulheim, H.: Global RDF vector space embeddings. In: d'Amato, C., et al. (eds.) ISWC 2017. LNCS, vol. 10587, pp. 190–207. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68288-4_12
5. Cochez, M., Ristoski, P., Ponzetto, S.P., Paulheim, H.: KGloVe DBpedia uniform embeddings (2017). https://doi.org/10.5281/zenodo.1320148
6. Cochez, M., Ristoski, P., Ponzetto, S.P., Paulheim, H.: RDf2Vec DBpedia uniform embeddings (2017). https://doi.org/10.5281/zenodo.1318146
7. Goyal, P., Ferrara, E.: Graph embedding techniques, applications, and performance: a survey. Knowl. Based Syst. **151**, 78–94 (2018)
8. Hoffart, J., Seufert, S., Nguyen, D.B., Theobald, M., Weikum, G.: KORE: keyphrase overlap relatedness for entity disambiguation. In: Proceedings of the 21st ACM CIKM, pp. 545–554 (2012)
9. Kaminski, B., Pralat, P., Théberge, F.: An unsupervised framework for comparing graph embeddings. CoRR abs/1906.04562 (2019)
10. Lee, M.D., Welsh, M.: An empirical evaluation of models of text document similarity. In: XXVII Annual Conference of the Cognitive Science Society (2005)
11. Mara, A., Lijffijt, J., Bie, T.D.: EvalNE: a framework for evaluating network embeddings on link prediction. In: Reproducibility in Machine Learning, ICLR (2019)
12. Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J.: Distributed representations of words and phrases and their compositionality. In: 27th Annual Conference on Neural Information Processing Systems, pp. 3111–3119 (2013)
13. 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
14. Pennington, J., Socher, R., Manning, C.D.: GloVe: global vectors for word representation. In: Proceedings of the Conference on Empirical Methods in Natural Language Processing, pp. 1532–1543 (2014)

15. Ristoski, P., Rosati, J., Noia, T.D., Leone, R.D., Paulheim, H.: RDF2Vec: RDF graph embeddings and their applications. Semant. Web **10**(4), 721–752 (2019)
16. Rulinda, J., de Dieu Tugirimana, J., Nzaramba, A., Aila, F.O., Langat, G.K.: An integrated platform to evaluate graph embedding. Int. J. Sci. Eng. Res. **9**, 665–676 (2018)
17. White, S., Smyth, P.: A spectral clustering approach to finding communities in graph. In: Proceedings of the SIAM International Conference on Data Mining (2005)